

WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI
POLITECHNIKA WROCŁAWSKA

APLIKACJA MOBILNA
DO KLASYFIKACJI WYDATKÓW
NA PODSTAWIE PARAGONÓW

KAROL SZYMOŃCZYK

NR INDEKSU: 243434

Praca inżynierska napisana
pod kierunkiem
dra Marcina Michalskiego



Politechnika
Wrocławskiego

WROCŁAW 2020

Spis treści

1 Wstęp	1
2 Analiza problemu	3
2.1 Paragon fiskalny	3
2.2 Przetrzymywanie paragonów	4
2.3 Prowadzenie statystyk wydatków	4
3 Projekt systemu	5
3.1 Architektura systemu	5
3.2 Preprocessing zdjęcia	6
3.3 Sczytywanie paragonu	13
3.3.1 OCR	13
3.4 Dataset	14
3.4.1 Kategorie	14
3.5 Wydobycie informacji z tekstu	16
3.6 Automatyczna klasyfikacja	17
3.6.1 Uczenie maszynowe	17
3.6.2 Przygotowanie danych	18
3.6.3 Klasyfikator	19
3.7 API	20
3.8 Baza danych	20
3.9 Aplikacja mobilna	21
4 Implementacja systemu	27
4.1 Główne technologie	27
4.2 Omówienie kodów źródłowych	29
4.2.1 Preprocessing zdjęcia	29
4.2.2 Sczytanie paragonu	31
4.2.3 Klasyfikacja paragonu	31
4.2.4 API	33
4.2.5 Aplikacja mobilna	34
5 Instalacja i uruchomienie	37
5.1 Backend	37
5.2 Aplikacja mobilna	37
5.3 Model	38
6 Testy	39
6.1 Klasyfikator	39
7 Podsumowanie	43
7.1 Realizacja projektu	43
7.2 Przyszły rozwój	43

Bibliografia 46

A Zawartość płyty CD 47

Wstęp

W dzisiejszych czasach, kiedy płatności bezgotówkowe stały się niezwykle popularne, ludzie najczęściej posiadają więcej niż jedno konto, często w wielu różnych bankach. Nadal jednak istnieją miejsca, w których płatność kartą nie jest możliwa i wówczas wymagana jest gotówka. Znacznie utrudnia to ludziom śledzenie swoich codziennych wydatków, ponieważ wiąże się to z koniecznością analizy wielu różnych źródeł. Jednakże istnieje element łączący zakupy, niezależnie od środka płatniczego - paragon, który otrzymujemy przy większości dokonywanych transakcji. Nie jest on jednak pozbawiony wad. W formie fizycznej podatny jest na zniszczenia oraz nietrudno o jego zagubienie. Ponadto analiza wydatków na ich podstawie może być czasochłonna i łatwo w niej o błęd.

Celem pracy jest zaprojektowanie oraz implementacja aplikacji mobilnej do przechowywania i klasyfikacji paragonów fiskalnych. Ma ona umożliwiać wykonanie zdjęcia paragonu dostępnym w urządzeniu aparatem, a następnie automatycznie przydzielić go do odpowiedniej kategorii. Aplikacja powinna pozwalać także na manualne dodawanie własnych tagów do danego paragonu. Użytkownik będzie miał możliwość oznaczenia za ich pomocą konkretnej wartości zakupów, co pozwoli mu oddzielić interesujące go produkty od całego rachunku. Na podstawie tak dodanych paragonów, aplikacja przedstawić będzie w formie graficznej prowadzone statystyki wydatków, zarówno według automatycznie przydzielanych kategorii ogólnych, jak i spersonalizowanych znaczników.

Praca swoim zakresem obejmuje wstępnią obróbkę oraz przygotowanie zdjęcia paragonu, a następnie sczytanie z niego tekstu za pomocą programu OCR (ang. Optical Character Recognition). Obejmuje także stworzenie własnej bazy polskich paragonów oraz wytrenowanie na jej podstawie zaprojektowanego modelu umożliwiającego przydzielenie paragonu do jednej z wcześniej zdefiniowanych kategorii. W jej zakres wchodzi także stworzenie aplikacji mobilnej oraz interfejsu programistycznego pozwalającego na korzystanie za jej pomocą z wcześniej opisanych modułów. Aplikacja ta pozwala na wykorzystanie znajdującego się w urządzeniu mobilnym aparatu w celu wykonania zdjęcia paragonu, a następnie wyświetlenia wyników działania odczytania informacji z paragonów oraz przydzielonej mu kategorii. Umożliwia ona także manualne dodanie do zeskanowanego paragonu wybranych przez użytkownika znaczników oraz określenia za ich pomocą wybranej kwoty zakupów. Na podstawie tak zebranych paragonów prezentuje ona wydatki całkowite użytkownika oraz z podziałem na kategorie lub tagi w formie wykresów oraz tabel w wybranym odstępie czasu.

Na rynku istnieją już aplikacje pozwalające na skanowanie oraz przechowywanie paragonów, ale niewiele z nich pozwala na ich automatyczne klasyfikowanie oraz prowadzenie na ich podstawie statystyk. Dodatkową funkcją, pomagającą prowadzić dokładniejsze statystyki, jest zastosowanie personalnych tagów, dodawanych manualnie przez użytkownika do zeskanowanych paragonów. Wszystkie dostępne aplikacje o zblążonej funkcjonalności pozwalają na dodanie jedynie jednego takiego znacznika do konkretnego paragonu, przez co nie jest możliwy podział jego kwoty na kilka różnych dziedzin. W wyniku tego użytkownik nie może dodać do osobnych statystyk wyłącznie interesującej go części zakupów.

Praca składa się z sześciu rozdziałów.

Rozdział pierwszy zawiera wstęp. W rozdziale drugim przedstawiono analizę problemu prowadzenia statystyk na podstawie paragonów, ich klasyfikacji oraz przechowywania. W rozdziale trzecim przedstawiono projekt systemu. Szczegółowo omówiono architekturę poszczególnych warstw aplikacji oraz sposoby komunikacji między nimi. W czwartym rozdziale opisano zastosowane podczas implementacji technologie oraz omówiono najważniejsze fragmenty kodów źródłowych. Znajdują się tam wybrane języki programowania oraz użyte biblioteki z podziałem na części projektu, w których zostały wykorzystane. Rozdział piąty został poświęcony instalacji oraz sposobom wdrożenia systemu w środowisku docelowym. W rozdziale szóstym opisane zostały prowadzone testy. Końcowy rozdział stanowi podsumowanie całej pracy oraz wyników testów wybranych rozwiązań.



Analiza problemu

Dokładne analizowanie oraz spisywanie wydatków często kojarzone jest z dużym nakładem pracy. Ludzie często rezygnują z tej aktywności z powodu jej czasochłonności oraz przekonania, że jest to zbędne. Pomimo wrażenia stałej kontroli wydatków, bez ich systematycznej analizy trudno jest zauważać schematy oraz określić, na co najczęściej przeznaczane są pieniądze oraz jakie są to kwoty. Jedną z najlepszych metod jest zbieranie paragonów ze wszystkich zakupów oraz podsumowywanie wydatków z podziałem na wybrane kategorie. Już po pierwszym miesiącu można dostrzec, na jakie rzeczy wydano najwięcej pieniędzy oraz czy było to niezbędne. Po kilku kolejnych miesiącach, wydatki można porównywać z poprzednimi okresami oraz wyciągać wnioski na kolejne. W ten sposób jest się w stanie regularnie oszczędzać pieniędzy przez zwiększenie świadomości bez utraty komfortu.

2.1 Paragon fiskalny

Paragon fiskalny jest jednym z dokumentów potwierdzających dokonanie zakupu, wydrukowany dla nabywcy przez kasę rejestrującą. Zgodnie z Rozporządzeniem Ministra Finansów w sprawie kas rejestrujących paragon fiskalny drukowany przez kasę musi zawierać następujące informacje [1]:

- Imię i nazwisko lub nazwę podatnika, adres punktu sprzedaży, a w przypadku sprzedaży prowadzonej w miejscowościach niestałych – adres siedziby lub miejsca zamieszkania podatnika,
- Numer identyfikacji podatkowej (NIP) podatnika,
- Numer kolejny wydruku,
- Datę oraz godzinę i minutę sprzedaży,
- Oznaczenie „PARAGON FISKALNY”,
- Nazwę towaru lub usługi pozwalającą na jednoznaczna ich identyfikację,
- Cenę jednostkową towaru lub usługi,
- Ilość i wartość sumaryczną sprzedaży danego towaru lub usługi z oznaczeniem literowym przypisanej stawki podatku,
- Wartość opustów, obniżek lub narzutów, o ile występują,
- Wartość sprzedaży brutto i wysokość podatku według poszczególnych stawek podatku z oznaczeniem literowym po uwzględnieniu opustów, obniżek lub narzutów,
- Wartość sprzedaży zwolnionej od podatku z oznaczeniem literowym,
- Łączną wysokość podatku,
- Łączną wartość sprzedaży brutto,
- Oznaczenie waluty, w której jest zapisywana sprzedaż, przynajmniej przy łącznej wartości sprzedaży brutto,
- Kolejny numer paragonu fiskalnego,



- Numer kasy i oznaczenie kasjera – przy więcej niż jednym stanowisku kasowym,
- Logo fiskalne i numer unikatowy.

Ponadto uregulowany jest także jego układ. Wysokość znaków na paragonie fiskalnym nie może być mniejsza niż 2,50 mm, szerokość taśmy paragonowej nie może być mniejsza niż 28 mm.

2.2 Przetrzymywanie paragonów

Głównym problemem w przypadku przechowywania fizycznych paragonów jest ich trwałość, ponieważ przetrzymywane nawet w odpowiednich warunkach przez długi czas stają się niemożliwe do odczytania. Sporym problemem okazać się może także ich objętość przy dużej ilości oraz długi czas przeszukiwania w celu odnalezienia konkretnego paragonu, który interesuje nas w danym momencie. Dobrym pomysłem wydaje się zatem ich digitalizacja umożliwiająca ich zachowanie na znacznie dłuższy czas niż w wersji analogowej oraz prostą edycję i przeszukiwanie. Problemem w tym wypadku może być stan, w jakim są zachowane oraz struktura ich powierzchni. Dodatkowo jakość ich poprawnego odczytu ze zdjęcia wpływa aparat, którym wykonano fotografię oraz tło, na jakim została wykonana.

2.3 Prowadzenie statystyk wydatków

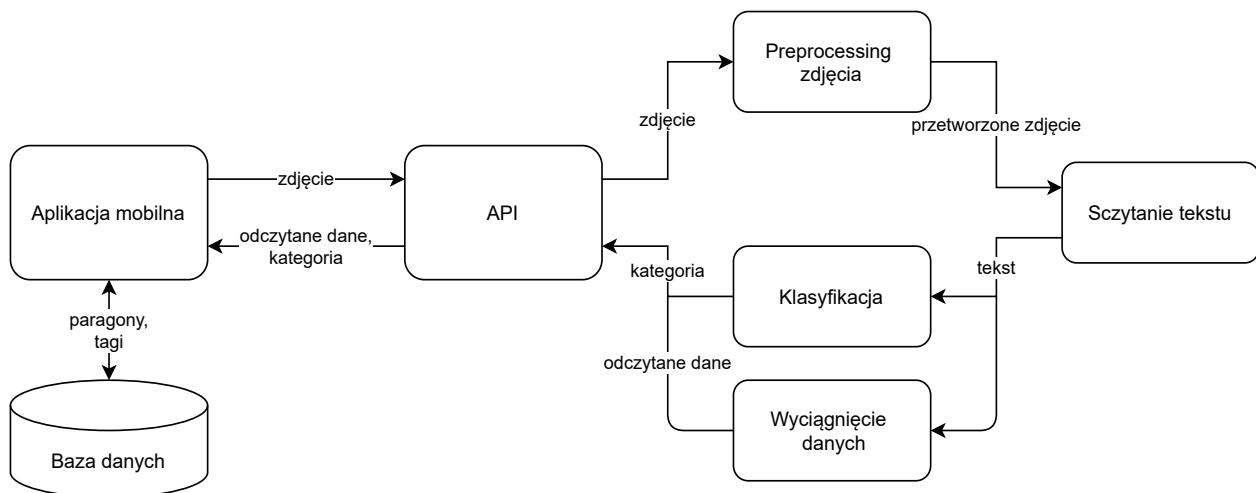
W przypadku statystyk wydatków prowadzonych ręcznie na podstawie paragonów łatwo jest o błąd, który staje się bardzo trudny do wykrycia. Prowadzone notatki mogą także ulec zniszczeniu lub zagubieniu, co prowadzi do utraty wszystkich zebranych dotychczas danych. Dodatkowo należy trzymać się przyjętych na początku oznaczeń, ponieważ edycja wcześniej spisanych danych może okazać się czasochłonna i prowadzić do dodatkowych błędów. Trudność może także sprawić uzyskanie statystyk z konkretnego okresu, ponieważ wymaga to dodatkowych obliczeń.

Projekt systemu

W tym rozdziale przedstawiono szczegółowy projekt systemu, opis jego modułów oraz zachodzących między nimi relacji.

3.1 Architektura systemu

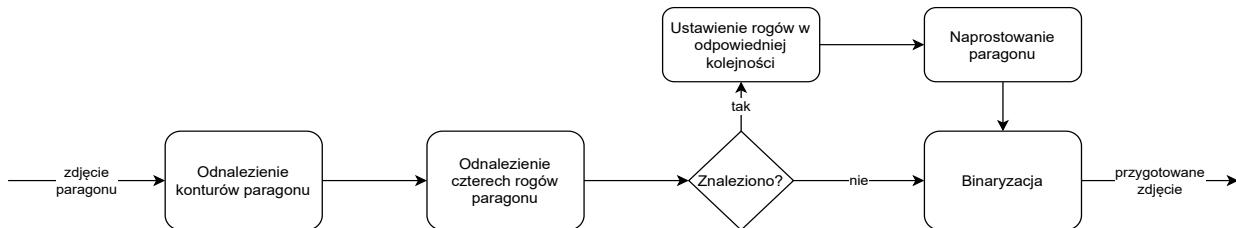
Projekt został podzielony na cztery części: backend, aplikacja mobilna, baza danych oraz API. Na część backendową składają się moduły odpowiadające za przetworzenie otrzymanego zdjęcia, odczytanie z niego informacji oraz klasyfikację paragonu na ich podstawie. Warstwa aplikacji mobilnej posiada także moduł umożliwiający użytkownikowi zalogowanie się lub utworzenie nowego konta. Połączona jest ona bezpośrednio z bazą danych co umożliwia synchronizację jej globalnego stanu z zawartością kolekcji przypisanej danemu użytkownikowi. Poniżej przedstawiono schemat relacji zachodzących w systemie. Moduły zostały szczegółowo opisane w dalszej części rozdziału.



Rysunek 3.1: Schemat relacji między modułami projektu

3.2 Preprocessing zdjęcia

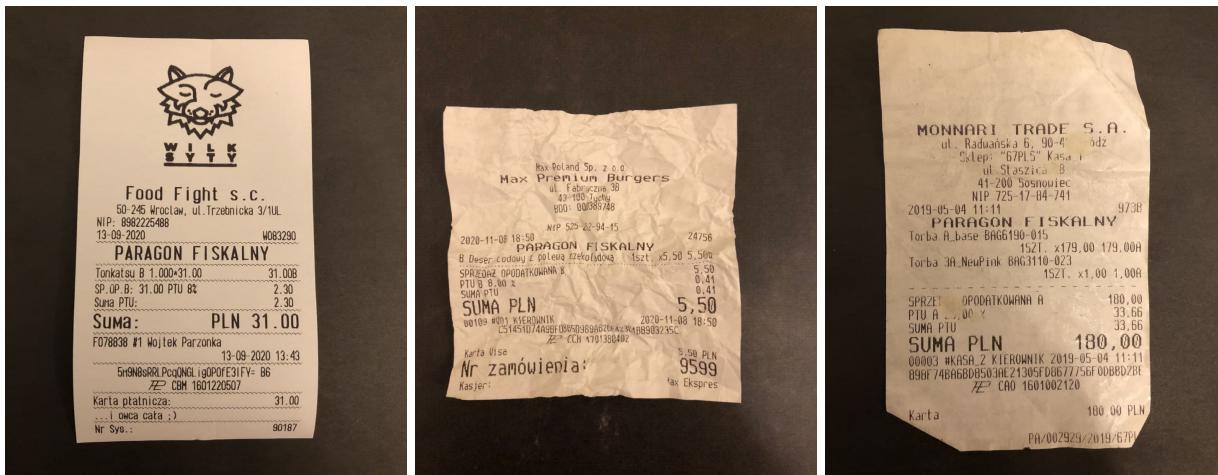
Preprocessing, czyli wstępna obróbka, był pierwszym krokiem po otrzymaniu zdjęcia paragonu od użytkownika. Miał on na celu odpowiednie przygotowanie dokumentu, aby zwiększyć efektywność działania opisanego w następnej części programu OCR poprzez usunięcie zakłóceń z obrazu oraz zbędnego tła.



Rysunek 3.2: Schemat obróbki zdjęcia

Sprawdzanie efektywności działania programu oraz dobór parametrów odbywało się na 7 różnych paragonech. Każdy z nich dotyczył innego, często napotykanego problemu.

Pierwszy z nich był paragonem równym i prosto ustawnionym oraz pełnił rolę przykładu, z którym porównywane były wyniki uzyskane na pozostałych obrazach. Drugi paragon był zgnieciony w kulkę, a następnie rozprostowany, aby zimitować jego przechowywanie w kieszeni lub reklamówce z zakupami. Paragon trzeci posiadał uszkodzone rogi - zgniecone lub zagięte do tyłu. Kolejny był zdecydowanie dłuższy od reszty, przez co zdjęcie obejmowało więcej tła. Czwarty paragon był stosunkowo stary i wyblakły z racji jego długiego przechowywania. Piąte zdjęcie przedstawiało paragon ułożony pod kątem, a ostatnie zdjęcie zostało wykonane na jasnym tle.



Prosty

Pognieciony

Uszkodzone rogi



Biale tło

Rysunek 3.4: Zdjęcia testowe z różnymi problemami przed obróbką

Naprostopowanie paragonu

Aby dokonać odpowiedniego ustawnienia paragonu na obrazie, tak by tekst znajdował się na nim poziomo oraz był widoczny z góry, należało najpierw odnaleźć jego kontury. Obróbka rozpoczęła się od przekształcenia obrazu do szarości oraz odfiltrowania niewielkich szumów. Następnie obraz poddany został wyostrzeniu, aby uwypunktować krawędzie oraz rogi paragonu. Później zdjęcie przekształcone było do czerni i bieli, tak by otrzymać czarny paragon na białym tle.



Rysunek 3.5: Zdjęcia testowe z różnymi problemami po obróbce wstępnej

Dopiero na tak przekształconym obrazie odnajdywane były kontury paragonu. Kolejnym krokiem było wyznaczenie jego rogów. Odbywało się to poprzez selekcję trzech największych konturów znalezionych na zdjęciu, a następnie badaniu, czy któryś z nich przypomina czworokąt.



Prosty

Pognieciony

Uszkodzone rogi



Dlugi

Wyblakły

Przekrzywiony



Białe tło

Rysunek 3.7: Zdjęcia testowe z różnymi problemami z zaznaczonymi konturami paragonów

W przypadku, gdy na skanowanym paragonie nie uda się odnaleźć takiego kształtu, tak jak wystąpiło to na przykładzie z białym tłem, do etapu binaryzacji zostaje przekazywany obraz przekształcony do szarości. W kolejnym kroku odnalezione rogi paragonu układane są zgodnie z ruchem wskazówek zegara, rozpoczynając od lewego górnego rogu. Korzystając ustawionych w odpowiedniej kolejności rogów, dokonywana jest transformacja obrazu z użyciem perspektywy. Przekształcenie to pozwalało także pozbyć się zbędnego tła i pozostawić na zdjęciu jedynie paragon. Aby dokonać tego przekształcenia, wyliczana była macierz transformacji rozmiaru 3×3 .

Można ją przedstawić w następującej formie:

$$M_T = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}$$

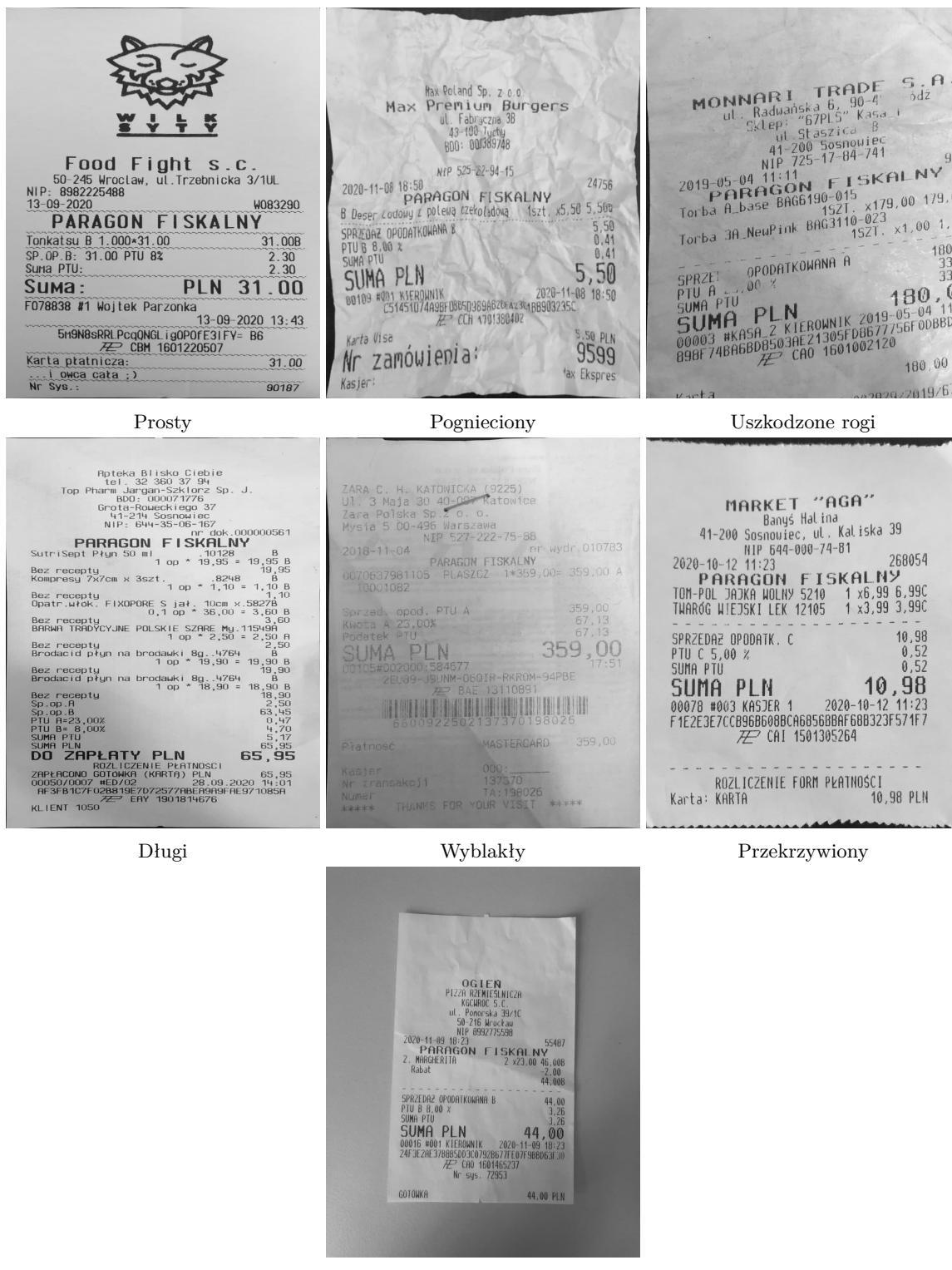
Można z niej wyróżnić składową $\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$ odpowiadającą za obrót oraz skalowanie obrazu, składową $[b_1 \ b_2]$ definiującą wektor translacji oraz składową $\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ określającą wektor projekcji [2].

Macierz transformacji wyliczana jest na podstawie podanych rogów dokumentu oraz punktów docelowych. Składa się ona z ośmiu stałych, które wyliczane są z ośmiu układów równań z ogólnego równania transformacji:

$$M_T \cdot \begin{bmatrix} x_{src} \\ y_{src} \\ 1 \end{bmatrix} = \begin{bmatrix} t \cdot x_{dst} \\ t \cdot y_{dst} \\ t \end{bmatrix},$$

gdzie x_{src} oraz y_{src} są współrzednymi punktu należącego do obrazu wejściowego. Aby uzyskać współrzędne punktu wyjściowego, należało podzielić wartości x oraz y przez wartość t nazywaną czynnikiem podziału.

Następnie każdy z pikseli wejściowego obrazu zostaje zmapowany na podstawie równania transformacji, korzystając z wyliczonej macierzy.



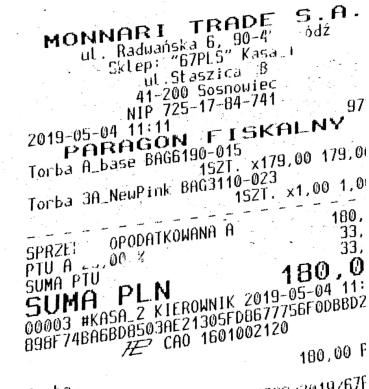
Rysunek 3.8: Zdjęcia testowe z różnymi problemami po przetransformowaniu



Binaryzacja

Ostatnim etapem preprocessingu obrazu była jego binaryzacja, czyli podział na interesujące nas obiekty oraz tło. Polega ona na wyznaczeniu dla danego obszaru progu jasności, a następnie piksele jaśniejsze od wyznaczonego progu otrzymują jedną wartość, a ciemniejsze drugą. Wyróżnia się binaryzację globalną oraz lokalną, w zależności od obszaru, na jakim obowiązuje ten sam próg.

Obróbka ta pozwala na usunięcie obecnych defektów, takich jak różnokolorowe tło czy cienie, tak aby pozostawić jedynie znajdujący się na paragonie czarny tekst na białym tle. W projekcie wykorzystano binaryzację lokalną z metodą średniej, polegającą na wyliczeniu progu jasności dla każdego piksela na podstawie średniej wartości sąsiadujących bloków.



Prosty

Pognieciony

Uszkodzone rogi



Długi

Wyblakły

Przekrzywiony



Białe tło

Rysunek 3.10: Zdjęcia testowe z różnymi problemami po zakończonym preprocessingu

Jednym z głównych problemów okazały się paragony z uszkodzonymi rogami. Niektóre z nich przez te wady pomijały proces transformacji i poddawane były tylko binaryzacji. Zdarzały się jednak przypadki, w których uszkodzenia powodowały niepoprawne odnalezienie rogów, przez co część paragonu zostawała ucięta.

Drugim problemem były przypadki, gdy paragon został sfotografowany na jasnym tle. Kontrast między paragonem a jego otoczeniem był zbyt mały, przez co odnajdywanie jego konturów kończyło się niepowodzeniem.

3.3 Sczytywanie paragonu

Odczytywanie tekstu z przygotowanego zdjęcia paragonu odbywa się za pomocą programu OCR. Etap ten ma na celu jego digitalizację poprzez zapisanie wszystkich widniejących na nim informacji w pliku tekstowym.

3.3.1 OCR

OCR (z ang. *Optical Character Recognition*), Optyczne Rozpoznawanie Znaków, czyli zestaw programów i technik służących do rozpoznawania znaków oraz tekstów w plikach graficznych [3].

Programy te umożliwiają komputerowi automatyczne przekształcanie tekstów zapisanych w formie graficznej do ciągu znaków, które są dla nich łatwe do przetwarzania, zakodowanych na przykład za pomocą unikodu czy ASCII. W celu odnalezienia tekstu program dokonuje segmentacji obrazu, czyli podziału na części, które są jednorodne pod względem wybranych własności. Takimi obszarami są zbiory pikseli tworzące znaki. Linie tekstu zostają zidentyfikowane przez znalezienie rzędów białych pikseli, między którymi występują piksele czarne. W podobny sposób zostają oddzielone kolejne litery z linii. W tym wypadku odszukane zostają kolumny pikseli białych, między którymi znajdują się kolumny pikseli czarnych. Tak wydzielone fragmenty są normalizowane. Zostają obcięte ze zbędnego tła, często odchudzając znaki do grubości 1 piksela oraz zostają zmniejszone do odpowiednich rozmiarów. Następnie tak przygotowane znaki przyporządkowywane są do wzorców.

Dodatkowo wiele oprogramowań OCR stosuje także postprocessing. Polega on na złożeniu rozpoznanych liter w słowa, dzięki czemu na podstawie słownika zwiększa on swoją efektywność korygując w razie niepewności litery tak, by tworzyły istniejące słowa. Czasami wykorzystywana jest także analiza słów często występujących obok siebie (neighbourhood analysis), która potrafi skorygować dwa istniejące w słowniku słowa na podstawie ich częstości występowania obok siebie.



Podczas odczytywania tekstu występuły problemy z poprawnością odczytywanych liter oraz liczb. Najczęściej mylone znaki to 0 z 8, 9 z 8, O z 0, 5 z S czy 6 z G. Spowodowane to było nietypową czcionką oraz różnego rodzaju skrótami pojawiającymi się na paragonach. Dodatkowym utrudnieniem były często wyblakłe lub nadmiernie zniszczone paragony, które mimo wstępnej obróbki stanowiły duże wyzwanie dla OCR.

3.4 Dataset

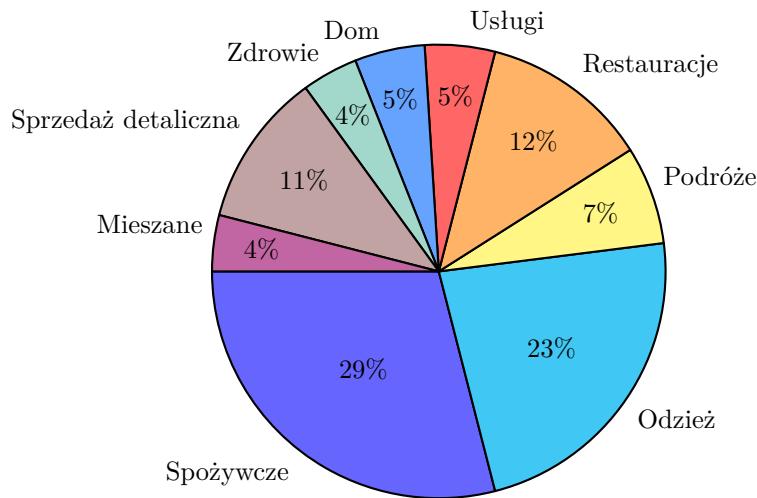
Tworzenie datasetu rozpoczęto od zebrania jak największej ilości paragonów z różnych miejsc oraz różnego rodzaju sklepów. Wszystkie zebrane paragony pochodzą z Polski, a najstarsze z nich są z 2018 roku. Składają się na nie paragony przechowywane w celach reklamacji, jak i te zbierane na bieżąco z lokalnych miejsc i sklepów. Na koniec okresu kolekcjonowania baza liczyła 469 paragonów.

3.4.1 Kategorie

Na podstawie zebranych paragonów oraz głównych obszarów wydatków wydzielono 9 kategorii. Prezentują się one następująco:

- Spożywcze (ang. *Grocery*) - oprócz zwykłych sklepów spożywczych zaliczane były tutaj także paragony pochodzące z marketów typu Biedronka czy Lidl, z racji tego, że zakupione tam towary to głównie artykuły spożywcze,
- Odzież (ang. *Clothes*) - kategoria, w której mieszą się paragony ze sklepów odzieżowych wszelkiego rodzaju, również obuwie,
- Podróże (ang. *Travel*) - do tej kategorii zaliczane bilety na różne środki transportu, parkingu oraz paragony ze stacji benzynowych
- Restauracje (ang. *Restaurants*) - do tej kategorii przydzielane są paragony pochodzące z lokali gastronomicznych,
- Usługi (ang. *Services*) - kategoria łącząca w sobie wszelkiego rodzaju świadczenia usług, takie jak wizyta u fryzjera czy przejazd taksówką oraz płatności związane z rozrywką, takie jak wejścia na obiekty sportowe czy do kina,
- Dom (ang. *Home*) - obejmuje paragony ze sklepów budowlanych, takich jak Ikea oraz sklepów związanych z wystrojem wnętrz, jak Home&You,
- Zdrowie (ang. *Health*) - zawiera głównie paragony pochodzące z aptek,
- Sprzedaż detaliczna (ang. *Retail*) - kategoria, do której trafiają paragony ze sklepów ze sprzedażą detaliczną nie zaliczające się do żadnej z powyższych kategorii, przykładowo Empik lub Rossmann,
- Mieszane (ang. *Miscellaneous*) - mieścią się w niej paragony z hipermarketów, w których można nabyć rzeczy z wielu różnych kategorii, takich jak Real czy Auchan.

Baza paragonów rozkłada się następująco na wyżej wymienione kategorie:



Rysunek 3.11: Diagram kołowy opisujący stosunek kategorii

Zdjęcia

Następnie każdemu z zebranych paragonów wykonano zdjęcie w jak najkorzystniejszych warunkach, czyli przy umiarkowanym oświetleniu na czarnym tle, aby zmaksymalizować efektywność odczytywania. Do wykonania zdjęć użyto aparatu dostępnego w smartfonie firmy Apple - iPhone X.



Rysunek 3.12: Przykładowe zdjęcie paragonu wykonane w najkorzystniejszych warunkach



Digitalizacja

Zapisane zdjęcia poddano obróbce za pomocą wcześniej opisanego skanera oraz sczytano z nich tekst przy użyciu OCR. Tekst zostawał zapisywany w pliku o nazwie odpowiadającej identyfikatorowi zdjęcia, z którego został odczytany oraz trafiał do folderu z danymi. Dodatkowo z odczytanego tekstu wydobywany był także NIP sprzedawcy, który okazał się pomocny na etapie przypisywania kategorii do poszczególnych paragonów. Następnie w pliku xlsx zapisywany był identyfikator zdjęcia wraz z odczytanym NIPem oraz kategorią odpowiadającą poszczególnemu paragonowi. Kategorie były uzupełniane ręcznie z pomocą numerów NIP, ponieważ paragony należące do tej samej kategorii często pochodziły z tych samych sklepów.

Największą trudnością okazało się zebranie zbliżonej liczby paragonów ze wszystkich kategorii. Było to spowodowane stosunkowo krótkim okresem kolekcjonowania oraz obostrzeniami powiązanymi z pandemią COVID-19. Znacznie utrudniło to zebranie paragonów z nieobecnej kategorii rozrywka, dlatego została ona wcielona do powiększonej kategorii usługi. Największą część stanowią paragony spożywcze, ponieważ były najłatwiejsze do pozyskania. Drugą dużą część stanowią paragony ze sklepów z odzieżą z racji często zachowywanych paragonów, w celu ewentualnej reklamacji towaru. Problemem było także odpowiednie dobranie kategorii na podstawie posiadanych paragonów, aby nie były one zbyt ogólne.

3.5 Wydobycie informacji z tekstu

Moduł ten odpowiedzialny był za wydobycie z tekstu za pomocą wyrażeń regularnych danych takich jak: NIP, cena całkowita oraz data i godzina dokonania zakupu.

Wyrażenia regularne

Wyrażenia regularne (ang. *Regular expression*) z technicznego punktu widzenia to wzorce opisujące łańcuchy znaków pozwalające na zaawansowaną pracę z tekstem. Mogą one określać zbiór pasujących łańcuchów, jak również wyszczególniać istotne jego części [4].

Wyrażenia regularne dobrano na podstawie analizy zebranych paragonów. Po ich przeglądzie wyszczególniono formaty poszukiwanych danych.

NIP zawsze składa się z dziesięciu cyfr i pojawiał się w dwóch formatach: XXX-XXX-XX-XX nadawany osobom fizycznym oraz XXX-XX-XX-XXX dla pozostałych podmiotów, gdzie X są cyframi z zakresu od 0 do 9.

Cena całkowita pojawiała się w postaci liczby złotówek oraz następującej po niej liczbie groszy. Wartości te oddzielone były kropką lub przecinkiem. Aby cenę całkowitą od cen produktów zanjdujących się na paragonie założono, że jest to ostatnia znaleziona wartość występująca w tym formacie.

Godzina zakupu występowała w formacie HH:MM lub rzadziej HH:MM:SS, gdzie HH oznacza godzinę, MM minutę, a SS sekundę dnia. Każda z wartości zawsze zapisana była za pomocą dwóch liczb. W przypadku liczb mniejszych od 10 była ona poprzedzana cyfrą 0. Godziny zapisane były według konwencji zegara 24-godzinnego.

Data zakupu występowała na paragonach w siedmiu różnych postaciach: DD-MM-YYYY, DD.MM.YYYY, DD/MM/YYYY, YYYY-MM-DD, YYYY.MM.DD, YYYY/MM/DD oraz dn.YYrMM.DD, gdzie D oznacza dzień, M miesiąc, a Y rok według kalendarza gregoriańskiego. Liczba symboli mówi o tym za pomocą ilu cyfr zapisana była dana wartość. Podobnie jak w przypadku godziny, gdy liczby były mniejsze od 10, a były zapisane na większej liczbie miejsc - zostawały poprzedzone cyfrą 0. Największy problem stanowił ostatni format daty, ponieważ występujący w nim rok zapisany jedynie na dwóch miejscach nie jest jednoznaczny. W tym wypadku przyjęto, że wprowadzone paragony nie będą starsze niż z 2000 roku. Lata zapisane w postaci YY zostały uzupełniane o poprzedzającą liczbę 20.

Największy problem stanowiły teksty odczytane z paragonów z różnego rodzaju defektami. W szukane informacje wkradały się niepożądane znaki przez co formaty danych nie były stałe. Zdarzały się także przypadki, w których dodatkowe znaki stwarzały frazy pasujące do wzorców w innych miejscach niż znajdowały się prawdziwe informacje.

3.6 Automatyczna klasyfikacja

Moduł klasyfikacyjny odpowiedzialny jest za automatyczne przydzielenie jednej z wymienionych wcześniej kategorii do paragonu. Klasyfikacja ta odbywa się z wykorzystaniem modelu wytrenowanego za pomocą algorytmów uczenia maszynowego.

3.6.1 Uczenie maszynowe

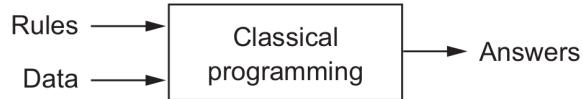
Uczenie maszynowe znane powszechnie pod angielską nazwą *machine learning* to obszar sztucznej inteligencji poświęcony algorytmom, które poprawiają się automatycznie poprzez trenowanie według wybranego algorytmu. Podejście to zrodziło się z pytania: czy komputer potrafi wykroczyć poza znane nam sposoby rozwiązania konkretnego problemu i nauczyć się samodzielnie wyznaczyć metodę analizując dane.

W projekcie uczenie maszynowe zostało wykorzystane w celu automatycznego przydzielania paragonu do jednej z kategorii. Problem ten nazywany jest **multiclass classification**, czyli klasyfikacją wieloklasową. Polega to na przydzieleniu badanego obiektu do jednej z trzech lub więcej klas.

Problem klasyfikacji wieloklasowej należy do dziedziny uczenia nadzorowanego.

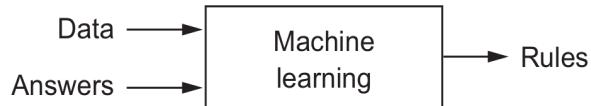
Uczenie nadzorowane

Uczenie nadzorowane (ang. *Supervised Learning*) jest rodzajem uczenia maszynowego, które zakłada nadzór ludzki nad tworzeniem funkcji odwzorowującej wejściowe dane na wyjściowe. W klasycznym programowaniu człowiek dostarcza zbiór zasad - czyli program - oraz dane, które są przetwarzane według tych zasad, abytrzymać wyniki.



Rysunek 3.13: Schemat klasycznego programowania [5]

W nadzorowanym uczeniu maszynowym, człowiek podaje dane oraz oczekiwane wyniki, aby otrzymać zbiór zasad, dlatego proces ten nazywany jest trenowaniem modelu.



Rysunek 3.14: Schemat uczenia maszynowego [5]

System ma za zadanie wyuczenia się na podstawie dostarczonych danych przewidywania prawidłowej odpowiedzi. Następnie zasady te mogą zostać wykorzystane na innym zbiorze, aby otrzymać wyniki odpowiadające nowym danym. Na podstawie wielu przykładów danych, zwanych zbiorem treningowym, algorytm buduje model matematyczny, który jest stale poprawiany wraz z kolejnymi przetwarzanymi danymi.



Trenowanie modelu odbywa się za pomocą **funkcji straty**. Przyjmuje ona wartość przewidzianą przez model oraz wartość prawdziwą, przypisaną do obiektu w zbiorze uczącym. Następnie oblicza wynik określający poprawność znalezionej rozwiązania. Minimalizacja tej funkcji podczas trenowania pozwala na wyznaczenie modelu dającego najlepsze predykcje według wybranej miary sukcesu dla dostarczonych danych.

Oprócz uczenia nadzorowanego można wyróżnić:

Uczenie częściowo nadzorowane (ang. *Semi - Supervised Learning*) zasada jego działania jest bardzo podobna do uczenia nadzorowanego. Jednak w tym wypadku dla zbioru uczącego zamiast odpowiedzi dostarczonych przez człowieka, generowane są one na podstawie danych wejściowych. Najczęściej w tym celu wykorzystywane są algorytmy heurystyczne, czyli algorytmy niedające gwarancji znalezienia rozwiązania optymalnego, pozwalające jednak na znalezienie wystarczająco dobrego rozwiązania w rozsądny czasie.

Uczenie nienadzorowane (ang. *Unsupervised Learning*) przebiega bez docelowych odpowiedzi. Model na podstawie danych wejściowych znajduje poprzez różnego rodzaju transformacje powiązania i relacje między obiektami. Najczęściej stosowane jest ono podczas analizy danych w celu lepszego zrozumienia, odszumienia oraz kompresji zbioru.

Uczenie wspomagane (ang. *Reinforcement Learning*) odbywa się na podstawie dostarczonych dozwolonych reguł oraz działań. Na ich podstawie model podczas szkolenia wybiera rozwiązanie maksymalizujące zyski, obserwując oraz analizując skutki po konkretnych akcjach. Rozwiążanie to stosowane jest najczęściej w branży gier komputerowych.

W projekcie jako dane i odpowiedzi zostały wykorzystane zebrane paragony i przyporządkowane im kategorie w celu uzyskania zasad klasyfikacji, którymi posługuje się wytrenowany model. Jednak aby rozpocząć jego szkolenie, niezbędne było odpowiednie przygotowanie zbioru uczącego.

3.6.2 Przygotowanie danych

Jako input wykorzystano cały tekst odczytany z paragonu za pomocą OCR. Pracę rozpoczęto od przygotowania danych, tak by dostosować je do wytrenowania modelu.

Kategorie

Wektoryzacja kategorii polegała na zakodowaniu ich w postaci liczbowej. Każda kategoria otrzymała kolejno własny identyfikator z zakresu od zera do liczby kategorii pomniejszonej o jeden. Następnie przypisane słownie do paragonów kategorie zostały zmapowane na liczby odpowiadające ich identyfikatorom.

Tekst

Przygotowanie tekstu rozpoczęło się od jego wektoryzacji. Jej pierwszym etapem był podział tekstu na tokeny. W tym wypadku tokenami były ciągi znaków oddzielone znakiem białym. Następnie każdemu unikalnemu tokenowi został nadany identyfikator. Kolejnym etapem było zliczanie wystąpień każdego z tokenów w każdym tekście. W ten sposób powstawał wektor z liczbą wystąpień każdego tokenu na pozycji odpowiadającej jego identyfikatorowi.

Zwektoryzowany tekst poddawany był transformacji TFIDF (z ang. *TF - term frequency, IDF - inverse document frequency*), która odpowiedzialna była za przydzielenie odpowiednich wag wydzielonym wcześniej tokenom na podstawie liczby ich wystąpień. Było to niezbędne do prawidłowego działania modelu, ponieważ tokeny pojawiające się w większości tekstów wnoszą o wiele mniej informacji, niż te specyficzne dla danych kategorii. Aby tego dokonać wektor z liczbami wystąpień tokenów został przekonwertowany do wektora liczb zmiennoprzecinkowych zgodnie z zasadami TFIDF:

$$\text{tfidf}(t, d) = \text{tf}(t, d) \cdot \text{idf}(t),$$



gdzie wartość $\text{tf}(t, d)$ wyliczana jest ze wzoru:

$$\text{tf}(t, d) = \frac{n_{td}}{\sum_k n_{kd}},$$

gdzie n_{td} jest liczbą wystąpień tokenu t w dokumencie d , natomiast mianownik jest sumą liczby wystąpień wszystkich tokenów w dokumencie d . Wartość wyrażenia $\text{idf}(t)$ wyliczana jest według wzoru:

$$\text{idf}(t) = \log\left(\frac{D}{\text{df}(t)}\right),$$

gdzie D jest liczbą wszystkich dokumentów, a $\text{df}(t)$ jest liczbą dokumentów w których pojawia się token t . Tak otrzymany wektor był normalizowany za pomocą normy euklidesowej:

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \dots + v_n^2}},$$

gdzie n jest liczbą współrzędnych wektora. Dzięki tej transformacji tekst reprezentowany był przez wektor składający się z wyliczonych na podstawie liczby wystąpień tokenów wartości. Słowa występujące najczęściej otrzymały najmniejsze wagi, natomiast te najbardziej unikalne otrzymały największe.

3.6.3 Klasyfikator

Tak przygotowane dane trafiają do klasyfikatora. Po testach przeprowadzonych na wybranych klasyfikatorach przedstawionych w rozdziale [Testy 6.1](#) do klasyfikacji wybrano LinearSVC, ponieważ wykazał największą dokładność osiągając średni wynik 75%.

LinearSVC

LinearSVC (z ang. *Linear Support Vector Classification*) jest modelem nadzorowanego uczenia maszynowego pozwalającego na klasyfikację danych. Odbywa się ona na zasadzie wyznaczenia hiperpłaszczyzny w n -wymiarowej przestrzeni rozdzielającej obiekty dwóch klas. n jest liczbą atrybutów, czyli mierzalnych właściwości użytych do opisu obiektów. Ponadto granica ta wyznaczana jest z zachowaniem maksymalnego marginesu miękkiego. Obiekty zostają przydzielone do odpowiedniej klasy w zależności od tego, po której stronie wyznaczonej granicy się znajdują.

Hiperpłaszczyzna w przestrzeni euklidesowej n -wymiarowej to zbiór rozwiązań równania postaci: $a_1 \cdot x_1 + a_2 \cdot x_2 + \dots + a_n \cdot x_n = A_n$ gdzie nie wszystkie współczynniki a_i są zerami. Hiperpłaszczyzna ma wymiar o 1 mniejszy niż przestrzeń, w której się zawiera [6].

Margines miękki jest maksymalnym marginesem oddzielającym obiekty dwóch klas, dopuszczającym błąd klasyfikacji niektórych danych. Dzięki temu dane znacząco odbiegające od normy nie wpływają aż tak drastycznie na klasyfikację przeciętnych obiektów. Pozwala to na uogólnienie klas, co umożliwia poprawną klasyfikację nieznanych danych. Wyznaczenie takiej granicy jest możliwe dzięki zastosowaniu metody cross-validation.

Cross-validation jest metodą walidacji polegającą na podziale danych na podzbiory oraz przeprowadzeniu analiz na niektórych z nich i zbadaniu wyników na pozostałych. Wyróżniony zostaje zbiór uczący oraz testowy. Celem metody jest wyznaczenie granicy dającą najdokładniejsze wyniki klasyfikacji. Wynik ten wyliczany jest za pomocą funkcji straty.

Funkcja straty

Jako funkcję straty wybrano **squared hinge**. Jest ona często stosowana podczas wyznaczania maksymalnej granicy podczas klasyfikacji binarnej. Wzór funkcji przedstawiono poniżej:

$$l(y, \hat{y}) = (\max(0, 1 - y_i \cdot \hat{y}_i))^2,$$



gdzie \hat{y} jest wynikiem przewidywania modelu. Wartość y jest równa -1 lub 1, w zależności od prawdziwej klasy obiektu. Wartość funkcji przyjmuje więc wartości:

- 0 - kiedy przewidziana została prawdziwa klasa obiektu i klasyfikator jest pewny swojego wyniku - $|\hat{y}| \geq 1$,
- wartość błędu podniesiona do kwadratu - kiedy przewidziana klasa różni się od prawdziwej lub w wypadku, gdy klasyfikator nie jest pewny swojego wyniku (nawet jeśli jest zgodny z prawdziwą klasą obiektu) - $|\hat{y}| < 1$.

Aby ocenić wynik modelu podczas jego trenowania, sumowane są wartości funkcji straty dla każdego obiektu.

Multiklasyfikacja

W przypadku klasyfikatorów SVC istnieją dwa podejścia do problemu multiklasyfikacji pozwalające sprowadzić ją do klasyfikacji binarnej.

One vs Rest

W podejściu One vs Rest (OvR), czyli jeden przeciwko reszcie problem sprowadzany jest do klasyfikacji binarnej dla każdej klasy. Polega to na porównaniu danej kategorii z resztą klas zagregowanych w jedną super-klasę. Proces ten powtarzany jest dla każdej kategorii, co oznacza wytrenowanie tylu modeli, jaka jest liczba klas. Przydzieloną na koniec klasą jest ta, która osiągnie najwyższy wynik prawdopodobieństwa przynależenia podczas porównań.

One vs One

Podejście One vs One (OvO), czyli jeden przeciwko jednemu polega na sprowadzeniu multiklasyfikacji do klasyfikacji binarnej dla każdej pary klas. Odbywa się przez porównanie każdej klasy z każdą, dlatego wymaga wytrenowania $\frac{n \cdot (n-1)}{2}$ modeli, gdzie n jest liczbą klas. Podobnie jak poprzednim podejściu przydzieloną na koniec klasą jest ta z najwyższym wynikiem prawdopodobieństwa przynależenia uzyskanym podczas porównań.

W projekcie wykorzystano podejście jeden przeciwko reszcie, ponieważ wykazało lepsze rezultaty w testach oraz wymagało wytrenowania mniejszej liczby modeli. W tym przypadku wystarczy ich 9, a przy podejściu jeden przeciwko jednemu liczba potrzebnych modeli wynosi 36.

3.7 API

API (z ang. *Application Programming Interface*), czyli interfejs programistyczny aplikacji to zbiór reguł opisujących, w jaki sposób programy lub podprogramy komunikują się ze sobą. API jest przede wszystkim specyfikacją wytycznych, jak powinna przebiegać interakcja między komponentami programowymi. Implementacja API jest zestawem rutyn, protokołów i rozwiązań informatycznych do budowy aplikacji komputerowych [7].

W tym projekcie API pełni rolę pośrednika pomiędzy warstwą frontendową oraz backendową. Ograniczone jest tylko do jednego punktu dostępu - `/classify`. Jego głównym zadaniem jest przyjęcie przesłanego z urządzenia mobilnego zdjęcia oraz zwrócenia wyników działania modułu klasyfikującego oraz wyciągającego informacje z tekstu w formacie JSON (ang. *JavaScript Object Notation*). Poniżej przedstawiona jest forma odpowiedzi.

3.8 Baza danych

Baza danych w projekcie odpowiedzialna jest za przechowywanie danych dotyczących paragonów oraz tagów dodanych przez użytkowników.



Kod źródłowy 3.1: Przykładowa odpowiedź w formacie JSON

```
{  
    "category": "travel",  
    "date": "2020.12.04",  
    "time": "23:50",  
    "total": 98.99  
}
```

Struktura bazy

Wykorzystana baza to tak naprawdę jeden duży obiekt w formacie JSON przetrzymywany w chmurze. Dzieli się ona na dwie główne kolekcje.

Receipts

Jest to kolekcja zawierająca paragony użytkowników. Podzielona jest ona według identyfikatorów, które są im nadawane podczas rejestracji. Paragony przetrzymywane są w postaci JSON. Zawierają pola:

- *Category* - kategoria paragonu,
- *Company* - firma dodana do paragonu,
- *Date* - data zakupu wraz z jego godziną w formie obiektu **Date**,
- *Photo* - ścieżka do zdjęcia paragonu wykonanego przez użytkownika,
- *Tags* - podkolekcja zawierająca taki przypisane do paragonu. Oprócz znacznika zawiera także jego wartość wprowadzoną przez użytkownika,
- *Title* - tytuł paragonu przypisany przez użytkownika,
- *Total* - cena całkowita zakupu.

Zdjęcia paragonów zapisywane są w osobnym miejscu w chmurze przeznaczonym do przetrzymywania plików - **Storage**. Każde z nich umieszczane jest pod unikalną ścieżką złożoną z identyfikatora użytkownika oraz nazwą zdjęcia. Umożliwia to ich późniejsze rozpoznanie oraz dopasowanie do danych paragonu zapisanych w bazie.

Tags

Kolekcja zawierająca personalne znaczniki. Podobnie jak w przypadku paragonów, kolekcja podzielona jest według identyfikatorów użytkowników. Kolekcja każdego użytkownika składa się z identyfikatorów jego własnych tagów, do których przypisana jest wprowadzona przez niego nazwa znacznika w polu *key*.

3.9 Aplikacja mobilna

Warstwa aplikacji mobilnej była ostatnim etapem projektu. Posiada ona interfejs graficzny umożliwiający użytkownikowi zdigitalizowanie oraz klasyfikację paragonu wykorzystując do tego omówione wcześniej moduły.



Uwierzytelnienie użytkownika

Korzystanie z aplikacji wymaga posiadania własnego konta. Dzięki temu możliwe jest jej używanie przez więcej niż jedną osobę na jednym urządzeniu mobilnym. Możliwa jest także zmiana urządzenia bez utraty dotychczas dodanych paragonów oraz tagów. Utworzenie takiego konta lub zalogowanie się na już istniejące, dostępne jest na pierwszym ekranie ukazującym się po uruchomieniu aplikacji. Uwierzytelnienie użytkownika odbywa się przy użyciu adresu e-mail oraz hasła. Dodatkowym zabezpieczeniem konta jest wymagana minimalna długość hasła wynosząca sześć znaków.

Uwierzytelnienie użytkownika odbywa się poprzez API. Jeśli użytkownik poprawnie się zaloguje, otrzymuje token w postaci łańucha znaków, który przetrzymywany jest w stanie aplikacji. Umożliwia on wykonywanie zapytań do bazy danych oraz dostęp do kolekcji danego użytkownika. Token ważny jest przez sześćdziesiąt minut od jego otrzymania. Po upływie tego czasu wymaga on odświeżenia poprzez ponowne zalogowanie się użytkownika do aplikacji. Wylogowywanie następuje poprzez usunięcie przetrzymywanego tokenu, co powoduje automatyczne przekierowanie do ekranu logowania.

Po udanej autentykacji, użytkownik ma wgląd do trzech głównych ekranów aplikacji:

- Ekran statystyk,
- Ekran dodawania paragonu,
- Ekran historii zakupów.

Miedzy ekranami można się poruszać za pomocą paska nawigacyjnego umieszczonego w dolnej części ekranu.

Personalne znaczniki

Tagi, czyli znaczniki umożliwiają użytkownikowi oznaczenie jedynie wybranej kwoty zakupów własną podkategorią. Do paragonu można dodawać wiele znaczników, jednak każdy może być przydzielony do niego tylko raz. Przykładowo użytkownik może utworzyć tag **słodycze**. Następnie przy dodawaniu lub edytowaniu paragonu ma możliwość użycia go poprzez wprowadzenie kwoty, jaką przeznaczył na produkty tego typu podczas zakupów. Kwota ta wprowadzana jest manualnie oraz nie musi być zgodna z cenami produktów widniejących na danym paragonie. Jedynym ograniczeniem jest suma wszystkich znaczników przypisanych do jednego paragonu, która nie może być większa niż widniejąca na nim cena całkowita. Tak dodawane tagi rozszerzają automatycznie przydzielaną kategorię oraz pozwalają na śledzenie wydatków przeznaczanych na wybrane przez siebie dziedziny.

Dodawanie paragonu

Paragon może zostać dodany do kolekcji po naciśnięciu na środkową ikonę paska nawigacyjnego. Gdy aplikacja zostaje uruchomiona po raz pierwszy na danym urządzeniu, zostaje wyświetlony okno dialogowe w celu udzielenia pozwolenia na dostęp do aparatu oraz galerii urządzenia. Gdy pozwolenia zostaną przyznane, uruchomiony zostaje ekran umożliwiający użytkownikowi szybkie wykonanie fotografii paragonu za pomocą aparatu dostępnego w smartfonie lub wybranie wcześniej zrobionego zdjęcia z galerii.

Po wykonaniu lub wybraniu zdjęcia użytkownik zostaje przekierowany na ekran oczekiwania. W tym czasie aplikacja wysyła zdjęcie do części backendowej poprzez API oraz oczekuje na odpowiedź. Po otrzymaniu informacji zwrotnej, dane zawarte w odpowiedzi zostają wyświetlane na ekranie. Użytkownik ma teraz możliwość ich weryfikacji oraz edycji w przypadku błędów odczytania lub klasyfikacji. W sytuacji nie odnalezienia szukanej informacji na zdjęciu pole zostaje wyświetcone puste. Po uzupełnieniu pól obowiązkowych, czyli daty, godziny oraz ceny całkowitej zakupów użytkownik może dodać taki paragon lub uzupełnić pola dodatkowe. Wśród nich znajduje się tytuł oraz firma. Ułatwiają one późniejsze wyszukiwanie konkretnego paragonu z całej kolekcji. Dostępna też jest opcja zaznaczenia, że produkt widniejący na paragonie podlega reklamacji oraz określenia daty okresu końca jej ważności. Dodatkowo na tym etapie ma możliwość dodania do paragonu własnych tagów oraz określenia ich wartości.



Statystyki

Na podstawie tak dodawanych paragonów aplikacja prowadzi statystyki wydatków użytkownika. Do ekranu statystyk można się dostać wybierając lewą ikonę znajdująca się na dolnym pasku nawigacji.

Okres czasu

Na samej górze ekranu znajdują się przyciski umożliwiające wybór okresu czasu, z którego przedstawione będą statystyki. Dostępne opcje to:

- Bieżący miesiąc,
- Bieżący rok,
- Cały okres kolekcjonowania,
- Własny okres czasu.

Opcja **Cały okres kolekcjonowania** obejmuje okres od najstarszej daty widniejącej na dostępnych paragonach do obecnego dnia. Po wybraniu ostatniej opcji **Własny okres czasu** na ekranie zostaje wyświetlane okno modalne, w którym użytkownik ma możliwość wyboru daty rozpoczęcia oraz zakończenia okresu, z którego chce uzyskać statystyki.

Tryby statystyk

Pod opcjami wyboru okresu czasu znajdują się przyciski z możliwymi trybami statystyk:

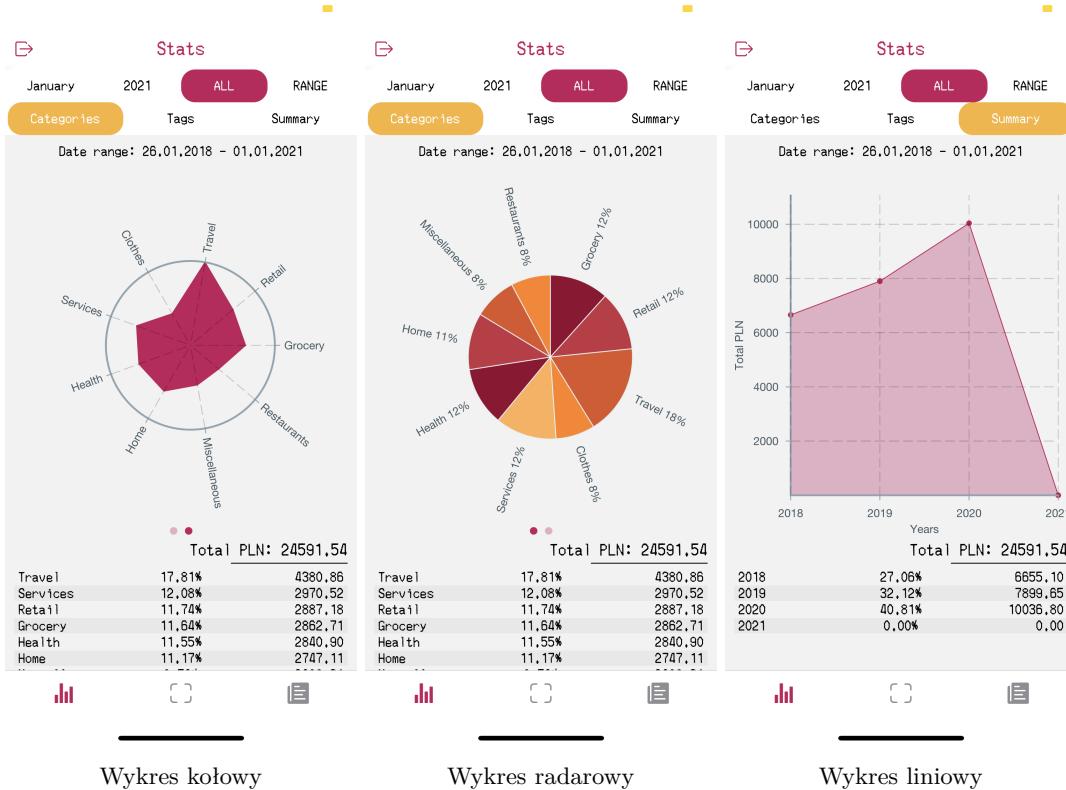
- Kategorie (ang. *categories*),
- Tagi (ang. *tags*),
- Podsumowanie (ang. *summary*).

Po wybraniu trybu **Kategorie** na ekranie wyświetlany jest wykres kołowy [3.15a](#) przedstawiający procentowe wartości każdej kategorii w całkowitych wydatkach. Po przeciągnięciu w lewo ukazuje się wykres radarowy [3.15b](#) obrazujący za pomocą wielokąta, na które kategorie użytkownik przeznacza najwięcej pieniędzy. Pod wykresami znajduje się informacja o całkowitej kwocie wydanej przez wybrany wcześniej okres czasu oraz tabela zawierająca nazwy kategorii, dokładniejsze wartości procentowe oraz konkretną kwotę w złotówkach przeznaczoną na daną kategorię.

Podobne wykresy można zobaczyć po przejściu do trybu **Tagi**. Dane prezentowane są na wykresie kołowym, pokazując wartość procentową kwoty oznaczonej konkretnym znacznikiem. Analogicznie istnieje możliwość prezentacji danych na wykresie radarowym oraz w tabeli znajdującej się pod wykresami.

W trybie **podsumowanie** wyświetlany jest wykres liniowy [3.15c](#) prezentujący rozkład ogólnych wydatków w wybranym okresie czasu. Pod wykresem znajduje się również tabela, jak w pozostałych trybach, zawierająca szczegóły naniesionych na wykres punktów. W przypadku wybrania opcji własnego okresu czasu w tym trybie, użytkownik dodatkowo ma możliwość wyboru najmniejszej jednostki czasu, do której sumowane są kwoty z paragonów. Sumy te wyświetlane są na wykresie jako punkty. Możliwe do wyboru jednostki to dzień, miesiąc lub rok.

Przykładowo użytkownik wybiera okres czasu odpowiadającą bieżącemu rokowi. Przełączając się między trybami, uzyska statystyki odpowiednio dla kategorii, tagów lub wydatków całkowitych dla bieżącego roku. Podobnie mając zaznaczony jeden z trybów, przykładowo **Kategorie**, może zmieniać okresy czasu otrzymując statystyki dotyczące kategorii wydatków dla zaznaczonego okresu.



Rysunek 3.15: Typy wykresów wydatków

Historia zakupów

Ekran historii zakupów prezentuje wszystkie dodane paragony w postaci listy kafelków 3.16a. Każdy kafelek zawiera najistotniejsze dane pozwalające zidentyfikować dany paragon, czyli datę, godzinę, cenę całkowitą oraz nazwę lub logo firmy, w której został dokonany zakup. Wyświetlone logo na kafelku zależy jego dostępności dla konkretnej firmy w bazie clearbit.com. W przeciwnym wypadku nazwa firmy zostaje wyświetlona w formie tekstowej. Lista posortowana jest w kolejności chronologicznej, a na jej szczytce znajduje się pasek wyszukiwania oraz informacje o ilości oraz całkowitej wartości odfiltrowanych paragonów. Searchbar pozwala użytkownikowi na szybkie wyszukanie paragonu po jego tytule lub firmie.

Po naciśnięciu w kafelek użytkownik zostaje przeniesiony na ekran ze szczegółowymi danymi paragonu 3.16b. Można tam znaleźć dodatkowe informacje takie jak przydzielona mu kategoria, dodane do niego znaczniki wraz z ich wartościami oraz ewentualna data końca okresu gwarancji. Widnieje tam także podgląd zdjęcia paragonu co umożliwia jego ponowne przeanalizowanie lub poprawienie błędów popełnionych przy jego dodawaniu.

W prawym górnym rogu ekranu, znajduje się przycisk przenoszący do ekranu edycji paragonu. Użytkownik ma możliwość modyfikowania wszystkich informacji z nim związanych oprócz zdjęcia. Możliwe jest także dodanie nowych tagów lub usunięcie obecnych.



The screenshot displays two main screens from a mobile application:

History Screen: Shows a grid of purchase receipts. Each receipt includes a logo, total amount, receipt number, date, and a small image of the receipt document.

Receipt Type	Total PLN	Receipt Number	Date
Zabka	408,85	Tytuł paragonu 93	Mon 04.01.2021
LIVE IN Levi's	342,69	Tytuł paragonu 82	Sun 03.01.2021
DECATHLON	180,66	Tytuł paragonu 83	Sun 03.01.2021
empik	321,92	Tytuł paragonu 91	Sat 02.01.2021
K	598,95	Tytuł paragonu 86	Sat 02.01.2021
BP	72,02	Tytuł paragonu 81	Sat 02.01.2021

Receipt Details Screen: Provides a detailed view of a single receipt. It shows the original receipt document with a breakdown of items and their costs.

Original Receipt Data:

Field	Value
Title:	Originalny
Company:	BP
Date:	Wednesday 24.06.2020
Time:	15:09
Category:	Travel
Total:	262,02 PLN
Tags:	Sweets
	6,99 PLN

Original Receipt Document (Textual Representation):

BP Europe S. r.o. Bielsko w Polsce
Jasnorzecka 1, 31-358 Kraków
BP Stan 239, kategoria: 40-511
ul. Górniodolska 55
BODI 000001598
nr wydr. 077045/0225
PARAGON FISKALNY
BENZ95 ACT Pomp # 1 55,83L * 4,21 235,04 zł
MONSTER ESPRE U9A250HL 1E * 6,99 6,99 zł
AMBI PUR CAR LEMUR 2ML 1szt * 19,99 19,99 zł

Sprzed. opod. PTU A 255,03
Sprzed. opod. PTU C 6,99
Kwota PTU A 23,08 zł 47,69
Kwota PTU C 0,00 zł 0,33
SUMA PTU 48,02 zł
SUMA PLN 262,02
ROZLICZENIE PŁATNOŚCI
Płatność: Visa 262,02
Wypełniono reżim: 262,02

000202/0225 #2 1992
2020-06-24 15:09
2198052682F023900139E2ED5FB0868985938926
ZERU1901768621 19221
Nr transakcji: 19221
Nowe Paragon z Technologią ACTIVE

Historia zakupów

Szczegóły paragonu

Rysunek 3.16: Ekrany historii zakupów



Implementacja systemu

4.1 Główne technologie

Poniżej znajduje się opis głównych technologii użytych w projekcie oraz motywacje ich wyboru.

Python

Python [8] - wysokopoziomowy język programowania ogólnego przeznaczenia poprzez rozbudowany zestaw bibliotek. Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią. Rozwijany jest jako projekt Open Source przez Python Software Foundation, która jest organizacją non-profit.

Język Python został użyty w backendowej części projektu. Został wybrany z powodu wielu kompatybilnych z tym językiem bibliotek, używanych do rozwiązywanych w tej części problemów. Jest także jednym z wiodących języków programowania używanych do przetwarzania obrazu oraz uczenia maszynowego. Dodatkowym atutem była znana autorowi składnia.

OpenCV

OpenCV (Open Computer Vision) [9] - otwartoźródłowa biblioteka posiadająca ponad 2500 zoptymalizowanych algorytmów wykorzystywanych w uczeniu maszynowym oraz przetwarzaniu obrazów. Została napisana w C, jednak istnieją nakładki pozwalające wykorzystywać ją w różnych językach programowania, w których zanajduje się język Python.

OpenCV jest jedną z najpopularniejszych bibliotek kompatybilnych z językiem python używanych do obróbki obrazu. Posiada wiele zoptymalizowanych algorytmów przydatnych podczas przygotowania paragonów przed użyciem programu OCR. Atutem przy wyborze była także rozbudowana dokumentacja oraz wiele dostępnych źródeł pomocnych w przyswojeniu technologii.

Pytesseract

Pytesseract [10] - biblioteka do optycznego rozpoznawania znaków przeznaczona dla języka Python. Umożliwia ona pracę z Tesseract-OCR Engine [11] obecnie rozwijanego przez firmę Google. Tesseract jest darmowym oprogramowaniem do optycznego rozpoznawania znaków, które wspiera ponad 100 języków. Posiada także możliwość wyboru rodzaju segmentacji oraz analizy układu tekstu na obrazie.

Pytesseract został wybrany, ponieważ umożliwia on pracę z Tesseract-OCR Engine w pythonowym kodzie. Tesseract jest najczęściej wybieranym programem OCR przez programistów z racji jego kompatybilności z wieloma językami programowania. Wspiera dużą liczbę różnego rodzaju czcionek oraz rozpoznawanych języków, w tym język Polski. Ponadto jest łatwo konfigurowalny oraz cechuje się dużą dokładnością.

Scikit-learn

Scikit-learn (sklearn) [12] - otwartoźródłowa biblioteka do uczenia maszynowego przeznaczona dla języka Python. Oferuje wiele algorytmów klasyfikacji, regresji oraz grupowania. W większości została stworzona w języku Python z użyciem biblioteki Numpy do wysokowydajnej algebry liniowej i operacji tablicowych.



Dodatkowo, niektóre jej moduły zostały napisane w Cython w celu poprawy ich wydajności. Projekt został zapoczątkowany na w 2007 roku na Google Summer of Code przez Davida Cournapeau.

Biblioteka Scikit-learn została wybrana z powodu wielu wbudowanych algorytmów uczenia maszynowego oraz narzędzi walidacyjnych. Wspiera dużą liczbę klasyfikatorów, co znacznie ułatwia wybór właściwego dla danego projektu. Ponadto biblioteka ta posiada rozbudowaną dokumentację zawierającą dokładne opisy oraz przykładowe użycia, co znacznie ułatwia jej wykorzystanie.

FastAPI

FastAPI [13] - szybki i wysokowydajny framework do tworzenia interfejsów programistycznych w języku Python. Dzięki prostej oraz intuicyjnej składni pozwala na szybkie stworzenie podstawowego API w kilku liniach kodu. Ponadto oferuje także automatycznie generowaną, interaktywną dokumentację.

Aplikacja nie posiada rozbudowanego interfejsu programistycznego, dlatego ważnym kryterium doboru narzędzia była jego prostota. FastAPI pozwala na stworzenie API w zaledwie kilku liniach kodu oraz cechuje się bardzo dużą szybkością działania. Była to także bardzo ważna cecha decydująca o jego wyborze, ponieważ w projekcie poprzez API wysyłane są zdjęcia z urządzenia mobilnego do części backendowej.

React Native

React Native [14] - otwartoźródłowy framework do tworzenia aplikacji mobilnych rozwijany przez firmę Facebook. Umożliwia tworzenie aplikacji na urządzeniu z systemami: Android, Android TV, iOS, macOS, tvOS, Web, Windows oraz UWP za pomocą frameworku React wraz z natywnymi elementami platform. Multiplatformowe zastosowanie jednego kodu jest możliwe dzięki przekładaniu języka JavaScript na kod natywny urządzeń.

Z racji tego, że aplikacja przeznaczona była zarówno na urządzenia mobilne z systemem Android, jak i iOS pula możliwych do wyboru frameworków była ograniczona. Decydującą przewagą frameworku React Native nad innymi dostępnymi rozwiązaniami była składnia zbliżona do tej wykorzystywanej w javascriptowej bibliotece React, która była znana autorowi. Jest ona także kompatybilna z często stosowanymi przy użyciu React bibliotekami, w tym biblioteką Redux do zarządzania globalnym stanem, co znacznie ułatwiło pracę.

Expo

Expo [15] - zestaw narzędzi oraz usług oparty na React Native oraz platformach natywnych. Pomagają one tworzyć oraz wdrażać aplikacje iOS, Android oraz internetowe z tego samego kodu JavaScript. Expo pozwala także na szybkie uruchamianie aplikacji tworzonej w React Native na fizycznych urządzeniach, poprzez aplikację Expo Client dostępną w sklepach na urządzeniach mobilnych. Dzięki temu możliwe jest szybkie uruchomienie tworzonej aplikacji na dowolnym urządzeniu poprzez zeskanowanie kodu QR.

Narzędzie Expo umożliwiło szybkie stworzenie aplikacji mobilnej oraz dostarczyło wiele interfejsów ułatwiających jej rozwijanie. Umożliwia ono także darmową dystrybucję kodu oraz działa multiplatformowo. Ponadto zaletą była także możliwość testowania aplikacji w czasie rzeczywistym na fizycznym urządzeniu. Pozwoliło to na sprawny proces implementacji oraz testowania bez konieczności korzystania z emulatorów smartfonów, które zużywają dużo pamięci RAM.

React Redux

React Redux [16] - niewielka, otwartoźródłowa biblioteka do zarządzania stanem JavaScriptowych aplikacji. Jej idea opiera się na założeniu, że stan aplikacji jest wynikiem stanu poprzedniego, przekształconego za pomocą akcji rozpoczynając od zdefiniowanego stanu początkowego.

Biblioteka React Redux jest jedną z najpopularniejszych bibliotek przeznaczonych do zarządzania globalnym stanem aplikacji. Używana jest w aplikacji mobilnej do wymiany informacji między komponentami oraz stanowi zestaw dobrych praktyk programistycznych. Jej wybór umotywowany był znaną składnią oraz prostotą działania dostosowaną do potrzeb aplikacji.



Victory Native

Victory Native [17] - zestaw komponentów do tworzenia wykresów w aplikacjach stworzonych za pomocą React Native. Dostarcza wiele typów wykresów z możliwością ich personalizacji oraz dostosowania do potrzeb aplikacji.

Wybór Victory Native w celu graficznego przedstawienia statystyk w formie wykresów dokonany był na podstawie testów różnych dostępnych rozwiązań. Okazała się najbardziej rozbudowaną oraz zaawansowaną biblioteką do tworzenia wykresów kompatybilną z React Native ze wszystkich testowanych rozwiązań.

Firebase

Firebase [18] - platforma przeznaczona do tworzenia aplikacji webowych oraz mobilnych rozwijana przez firmę Google. Posiada wiele gotowych rozwiązań używanych w większości aplikacji, takich jak autoryzacja użytkowników, baza danych czy różnego rodzaju analizy i testy.

Firebase jest często stosowanym rozwiązaniem w aplikacjach webowych oraz mobilnych, ponieważ udostępnia najważniejsze narzędzia stosowane podczas ich tworzenia. Została wykorzystana w projekcie w celu uwierzytelniania użytkowników, przetrzymywania ich kont oraz związanych z nimi danych. Głównym powodem jej wyboru była prostota wykorzystania wbudowanych modułów bezpiecznej autoryzacji oraz bazy danych poprzez REST API. Dodatkową zaletą była jej kompatybilność ze stanem aplikacji mobilnej oraz zaprojektowana autorowi składnia.

4.2 Omówienie kodów źródłowych

W tej sekcji przedstawione zostały wybrane fragmenty kodu źródłowego wraz z ich opisem. Kompletne kody źródłowe znajdują się na płycie CD dołączonej do niniejszej pracy.

4.2.1 Preprocessing zdjęcia

Wstępna obróbka

Kod źródłowy 4.1: Zlokalizowanie konturów oraz rogów paragonu na zdjęciu

```
def preprocess_img(original):
    img = original.copy()
    grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    blured = cv2.medianBlur(grayscaled, 5)
    sharpen_kernel = np.array([[-1, -1, -1],
                               [-1, 9, -1],
                               [-1, -1, -1]])
    sharpened = cv2.filter2D(blured, -1, sharpen_kernel)
    inverted = cv2.bitwise_not(sharpened)
    _, thresholded = cv2.threshold(inverted, 160, 255, cv2.THRESH_BINARY)
```

Wstępna obróbka obrazu rozpoczyna się od przekształcenia go do szarości oraz lekkiego rozmycia w celu usunięcia zakłóceń. Wykorzystany został filtr medianowy wyliczający dla każdego piksela medianę z otaczających go bloków pikseli określonej wielkości. Następnie obraz poddawano zastrzeniu przy użyciu funkcji `filter2D` dostępnej w bibliotece OpenCV ze zdefiniowanym powyżej jądrem. Kolejnym etapem było odwrócenie kolorów obrazu za pomocą operatora `bitwise_not`, który zmienia każdy bit na przeciwny. Na koniec zdjęcie poddawane było binaryzacji. W ten sposób uzyskiwany był obraz czarnego paragonu na białym tle, aby ułatwić odnalezienie na nim jego konturów.



Lokalizacja konturów oraz rogów

Kod źródłowy 4.2: Zlokalizowanie konturów oraz rogów paragonu na zdjęciu

```
def get_receipt_corners(img, thr):
    contours = imutils.grab_contours(cv2.findContours(
        thr, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE))

    biggest_contours = sorted(contours, key=cv2.contourArea, reverse=True)[1:4]

    for contours in biggest_contours:
        perimeter = cv2.arcLength(contours, True)
        corners = cv2.approxPolyDP(contours, 0.03 * perimeter, True)
        if len(corners) == 4:
            return corners

    return None
```

Przedstawiona powyżej funkcja odpowiada za odnalezienie na obrazie rogów paragonu. Aby tego dokonać, najpierw znajdują się odnalezione kontury występujące na obrazie za pomocą funkcji `findContours` z biblioteki OpenCV. Jest to możliwe dzięki wykorzystaniu relacji tła po obu stronach szukanej granicy na odpowiednio zbinaryzowanym obrazie [19]. Ustawienie trybu wyszukiwania na `RETR_LIST` powoduje zwrócenie konturów bez zachowania hierarchii ich zagnieżdżeń, ponieważ jest ona zbędna w tym przypadku. Ustawienie trybu przybliżania koturów na `CHAIN_APPROX_SIMPLE` pozwala na zwrócenie jedynie punktów początku oraz końca linii horyzontalnych, wertykalnych oraz ukośnych, z których składa się dany kontur. Funkcja `grab_contours` z biblioteki `imutils` pozwala wydobyć listę wierzchołków ze zwracanych przez funkcję obiektów. Zapobiega ona różnicom w ilości zwracanych obiektów przez metodę `findContours` w zależności od zainstalowanej na urządzeniu wersji biblioteki OpenCV. Aby wyznaczyć właściwe kontury paragonu, ze wszystkich znalezionych na obrazie wybierane znajdują się trzy największe. Następnie w pętli dla każdego z nich wyliczany jest obwód, który wykorzystywany jest do wyliczenia maksymalnego, dopuszczalnego błędu podczas przybliżenia konturów wielokątem. Jeśli kontur udało się przybliżyć czworokątem, można stwierdzić z dużym prawdopodobieństwem, że jest to szukany paragon. Zostają wtedy zwrócone jego cztery wierzchołki. W przypadku niepowodzenia zwaracana jest wartość `None`.

Ułożenie rogów

Kod źródłowy 4.3: Ułożenie rogów w odpowiedniej kolejności

```
def get_corners_in_order(corners):
    top, bottom = np.split(corners[corners[:, 1].argsort()], 2)
    tl, tr = top[top[:, 0].argsort()]
    bl, br = bottom[bottom[:, 0].argsort()]

    return [tl, tr, br, bl]
```

Powyższy kod odpowiada za ułożenie rogów znalezionej paragonu w kolejności zgodnej z kierunkiem ruchu wskazówek zegara, rozpoczynając od lewego górnego rogu. Na początku, rogi dzielone są na górne oraz dolne odpowiednio sortując tablicę po drugiej współrzędnej punktów. Odpowiada ona za ich odległość od górnej krawędzi obrazu. Początkowy podział na rogi górne oraz dolne zamiast na lewe oraz prawe jest umotywowany podłużnym kształtem paragonu w przeważającej liczbie przypadków. Górnego rogi będą nadal miały mniejszą drugą współrzędną pod kątem, w którym prawy dolny róg przekroczy linię lewego górnego. Następnie rogi dolne oraz górne zostają podzielone na rogi lewe oraz prawe za pomocą sortowania po pierwszej współrzędnej. W przypadku, gdy paragon nie jest odwrócony do góry nogami, rogi lewe będą miały mniejszą pierwszą współrzędną odpowiadającą za odległość od lewej krawędzi obrazu. Rogi zwracane są w tablicy ułożone w wymaganej kolejności.

Transformacja obrazu

Kod źródłowy 4.4: Transformacja obrazu z użyciem

```
def transform_img(original, corners):
    img = original.copy()
    height, width = img.shape[0], img.shape[1]
    ordered_corners = get_corners_in_order(corners.reshape(4, 2))
    src = np.float32(ordered_corners)
    dst = np.float32([
        [0, 0],
        [width - 1, 0],
        [width - 1, height - 1],
        [0, height - 1]])

    matrix = cv2.getPerspectiveTransform(src, dst)
    transformed_image = cv2.warpPerspective(
        img, matrix, (width, height))

    return transformed_image
```

Transformacja obrazu odbywa się z użyciem wcześniej ułożonych rogów. Następnie wyznaczane są docelowe punkty transformacji na podstawie rozmiaru oryginalnego obrazu. Macierz przekształcenia wyliczana jest za pomocą metody `getPerspectiveTransform` 3.2 dostępnej w bibliotece OpenCV. Następnie z jej wykorzystaniem obraz zostaje przetransformowany za pomocą funkcji `warpPerspective`, która odpowiednio mapuje każdy piksel obrazu wejściowego. Zwracany zostaje przetransformowany obraz o takim samym rozmiarze jak oryginalne zdjęcie.

4.2.2 Sczytanie paragonu

Użycie Pytesseract

Kod źródłowy 4.5: Odczytanie tekstu ze zdjęcia

```
def extract_text_from_image(img):
    config = '--psm 1'
    presprocessed_img = preprocess(img)
    text = pytesseract.image_to_string(
        presprocessed_img, config=config, lang='pol')
    clear_text = remove_blank(text)

    return clear_text
```

Przedstawiony powyżej fragment kodu odpowiada za odczytanie tekstu ze zdjęcia paragonu. Aby zwiększyć poprawność odczytywania znaków, zmieniono domyślną segmentację, na segmentację z użyciem OSD (z ang. Orientation and script detection) za pomocą flagi `-psm 1`. Umożliwia ona dodatkowe uwzględnienie położenia paragonu oraz wykrycie tekstu znajdującego się na obrazie. Zwiększa to prawdopodobieństwo poprawnego odczytania tekstu, kiedy na zdjęciu podczas wstępnej obróbki nie zostaną odnalezione wszystkie cztery rogi paragonu i zdjęcie pozostanie nieobrócone. Dodatkowo do odczytywania został wykorzystany język polski, z racji na polskie pochodzenie paragonów. Dzięki temu są większe szanse na poprawne odczytanie danych sprzedawcy oraz słów występujących w całości na paragonach. Odczytany tekst zostaje także pozbawiony zbędnych białych znaków, takich jak znaki nowej linii czy wielokrotne tabulatory lub spacje.

4.2.3 Klasyfikacja paragonu

Budowa modelu



Kod źródłowy 4.6: Budowa modelu

```
def get_linearSVC_pipeline():
    return Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
                    ('clf', LinearSVC())])
```

Przedstawiony kod, za pomocą funkcji `Pipeline` dostępnej bibliotece w `Scikit-learn`, umożliwia zbudowanie modelu składającego się z funkcji wektoryzującej, transformującej oraz klasyfikatora. Do wektoryzacji tekstu na podstawie zliczenia wystąpień poszczególnych tokenów opisanej w części Projektu Systemu 3.6.2 został wybrany `CountVectorizer`. Następnie do transformacji wektora z liczb całkowitych na zmienoprzecinkowe poprzez przypisanie odpowiednich wag tokenom opisany w rozdziale 3.6.2 został wybrany `TfidfTransformer`. Ostatnim etapem jest klasyfikacja obiektu na podstawie wcześniejszej zwektoryzowanych danych. W oparciu o przeprowadzone testy, opisane w części 6.1, został wybrany klasyfikator `LinearSVC` 3.6.3.

Walidacja modelu

Kod źródłowy 4.7: Walidacja modelu

```
print(classification_report(y_test, predicted,
                            target_names=[categories[i] for i in labels.classes_]))
```

Powyższy kod umożliwiał walidację wytrenowanego modelu za pomocą funkcji `classification_report` z biblioteki `Scikit-learn`. Funkcja ta przyjmuje zbiór testowy klas oraz przewidywane wyniki zaproponowane przez model. Trzeci argument jest opcjonalny. Pozwala wyświetlić wyniki w czytelniejszy sposób z użyciem tekstowych nazw kategorii. Poniżej przedstawiono przykładowy wynik działania kodu:

Kod źródłowy 4.8: Przykładowy wynik walidacji

	precision	recall	f1-score	support
grocery	0.65	1.00	0.79	37
retail	0.88	0.58	0.70	12
travel	0.83	0.71	0.77	7
home	1.00	0.09	0.17	11
health	0.67	1.00	0.80	2
clothes	0.74	0.93	0.82	27
service	0.33	0.50	0.40	2
restaurant	0.92	0.55	0.69	22
miscellaneous	1.00	0.17	0.29	6
accuracy				126
macro avg	0.78	0.61	0.60	126
weighted avg	0.79	0.72	0.68	126

Na podstawie tak otrzymywanych wyników można przeanalizować dane dotyczące ogólnego działania modelu oraz dla każdej kategorii z osobna. Uzyskane wyniki prezentowane są w postaci wartości dokładności (accuracy), średniej bez użycia wag (macro avg) oraz średniej ważonej (weighted avg). Dokładność jest miarą określającą liczbę poprawnie sklasyfikowanych obiektów w stosunku do ich całkowitej liczby. Kolejne kolumny oznaczają:

- Precision - precyza, czyli wartość proporcji prawdziwie pozytywnych wyników wśród wszystkich wyników pozytywnych. W uproszczeniu, oznacza to wartość mówiącą jak dużo obiektów przydzielonych do danej kategorii naprawdę do niej należy,
- Recall - jest to stosunek wyników prawdziwie pozytywnych do sumy prawdziwie pozytywnych i fałszywie negatywnych. W uproszczeniu, oznacza to wartość mówiącą jak duża część obiektów naprawdę należących do danej klasy została do niej przydzielona,

- F1-score - miara f1 pozwala równomiernie uwzględnić wyniki precision oraz recall danego modelu w jednej wartości. Uzyskiwana jest poprzez wyliczenie ich średniej harmonicznej,
- Support - wartość określająca liczbę obiektów danej kategorii użytych podczas walidacji.

4.2.4 API

Odbieranie zdjęcia

Kod źródłowy 4.9: Odbieranie zdjęcia

```
@app.post("/classify")
async def photo(photo: UploadFile = File(...)):
    tmp_path = save_upload_file_tmp(photo)
    try:
        img = cv2.imread(str(tmp_path))
        text = getTextFromPhoto(img)
        data = getDataFromText(text)
        category = classifyReceipt(text)
    except:
        return {"date": "", "time": "", "total": "", "category": ""}
    finally:
        tmp_path.unlink()
    print(data)
    return {
        "date": data["date"],
        "time": data["time"],
        "total": data["total"],
        "category": category
    }
```

Powyższy kod umożliwia odebranie zdjęcia wysłanego na punkt dostępu `/classify` za pomocą metody POST. Odbywa się to z użyciem funkcji `UploadFile` dostępnej we frameworku `FastAPI`. Po jego odebraniu, zostaje ono zapisane w pliku tymczasowym, aby przeprowadzić na nim operację odczytania tekstu z użyciem OCR oraz jego klasyfikację. Następnie zwracana jest odpowiedź w formacie JSON zawierająca informacje o jego cenie całkowitej, dacie i godzinie zakupu oraz wynik modułu klasyfikującego. W przypadku jakichkolwiek błędów zostaje zwrocony obiekt JSON z pustymi polami.

Kod źródłowy 4.10: Zapisanie zdjęcia w tymczasowym pliku

```
def save_upload_file_tmp(file: UploadFile):
    try:
        suffix = Path(file.filename).suffix
        with NamedTemporaryFile(delete=False, suffix=suffix) as tmp:
            shutil.copyfileobj(file.file, tmp)
            tmp_path = Path(tmp.name)
    except:
        return None
    finally:
        file.file.close()
    return tmp_path
```

Zdjęcie zostaje zapisane w pliku tymczasowym o nazwie zgodnej z nazwą zdjęcia zawartej w wysłanym obiekcie. Plik, na którym zakończone zostały operacje, uznawany jest za zbędny, co pozwala na jego usunięcie w celu zaoszczędzenia pamięci.



4.2.5 Aplikacja mobilna

Loga firm

Kod źródłowy 4.11: Wyświetlanie logo firmy

```
<View>
  {logoError ? (
    <View style={styles.textLogoContainer}>
      <StyledText style={styles.textLogo} numberOfLines={2}>
        {props.company}
      </StyledText>
    </View>
  ) : (
    <Image
      style={styles.logo}
      source={{
        uri: `https://logo.clearbit.com/${props.company}.com?size=500`,
      }}
      onError={() => setLogoError(true)}
    />
  )}
</View>
```

Powyższy fragment kodu przedstawia odpowiednie wyświetlenie firmy na kafelku każdego paragonu. Źródło znacznika Image pobierane jest z zewnętrznego API - <https://logo.clearbit.com/>, które jako parametr przyjmuje nazwę firmy oraz oczekiwany rozmiar obrazu. Nazwa firmy pochodzi z obiektu `props` komponentu, gdzie znajdują się wszystkie informacje dotyczące danego paragonu. API w przypadku nieodnalezienia pasującej do zapytania firmy zwraca błąd z kodem 404, przez co istnieje możliwość jego łatwej obsługi we własności `onError`. W tym wypadku w stanie komponentu wartość własności `logoError` zostaje ustawiona na `true` i zamiast loga zostaje wyświetlona nazwa firmy w formie tekstu.

Wysyłanie zdjęcia

Kod źródłowy 4.12: Wysyłanie zdjęcia

```
try {
  const res = await fetch(`.${HOST}/calssify`, {
    method: 'POST',
    body: formData,
    headers: {
      'content-type': 'multipart/form-data',
    },
  });
  const resData = await res.json();
  setData(resData);
} catch (error) {
  setError('Something went wrong... Please check your internet connection');
}
```

Przedstawiony kod źródłowy wykonuje żądanie typu POST do API na punkt dostępu `/calssify`. W nagłówku zawiera ono informacje o przesyłanym typie danych. Podany typ `multipart/form-data` świadczy o przesyłaniu pliku w kilku częściach z racji na jego rozmiar. W swoim ciele żądanie zawiera obiekt `FormData`:

Kod źródłowy 4.13: Obiekt FormData

```
const formData = new FormData();
formData.append('photo', { uri: localUri, name: filename, type });
```



Obiekt ten składa się z jednej pary klucza oraz wartości. Pod kluczem 'photo' zostaje przesłany obiekt typu JSON z informacjami o wykonanym na urządzeniu mobilnym zdjęciu w formacie JSON. Zawiera on jego typ, nazwę oraz uri (ang. Uniform Resource Identifier), czyli ujednolicony łańcuch znaków pozwalający na identyfikację zasobów w sieci. Całość znajduje się w bloku try/catch, aby zapobiec przerwaniu działania aplikacji w przypadku napotkanego błędu. Jeśli zapytanie przebiegnie pomyślnie, odpowiedź zostaje zapisana w stanie komponentu. W przeciwnym razie zostaje ustawiony odpowiedni komunikat o błędzie.



Instalacja i uruchomienie

5.1 Backend

Wymagania

Aby uruchomić część backendową na maszynie, niezbędny jest zainstalowany język Python w wersji 3.8.x lub wyższej.

Instalacja

Aby zainstalować niezbędne do działania programu biblioteki, należy uruchomić terminal w folderze `backend`, a następnie wywołać komendę `pip install`.

Uruchomienie

W celu uruchomienia serwera backendowego należy uruchomić terminal w folderze `backend`. Następnie należy wprowadzić komendę `uvicorn main:app --host $IPv4 --reload`. W miejscu `$IPv4` należy wprowadzić adres IPv4 sieci, do której obecnie podłączona jest maszyna z uruchomionym serwerem.

5.2 Aplikacja mobilna

Wymagania

Aby uruchomić aplikację, na maszynie niezbędne jest posiadanie środowiska uruchomieniowego NodeJS oraz urządzenia mobilnego z systemem Android lub iOS (istnieje również możliwość uruchomienia aplikacji na emulatorze).

Dodatkowo, aby umożliwić poprawne połączenie z API, niezbędne jest odpowiednie ustawienie zmiennej `HOST`. Znajduje się ona w folderze `app/constants/Hosts`. Jej wartość powinna być zgodna z adresem IPv4 sieci (tym samym, z którym uruchomiony został serwer).

Instalacja

Aby zainstalować niezbędne do działania programu biblioteki, należy uruchomić terminal w folderze `app`, a następnie wywołać komendę `npm install`.

Na urządzeniu mobilnym należy zainstalować aplikację Expo dostępną w Google Play Store na urządzeniach z systemem Android oraz w sklepie AppStore na urządzeniach z systemem iOS.

Uruchomienie

Urządzenie mobilne musi być połączone z tą samą siecią, z którą połączona jest maszyna z uruchomionym serwerem. Następnie należy uruchomić terminal w folderze `app`. Po wpisaniu komendy `npm start` wyświetlony zostanie kod QR.



Wygenerowany kod należy zeskanować używając urządzenia mobilnego z zainstalowaną aplikacją Expo. Na urządzeniu mobilnym z systemem Android należy uruchomić aplikację Expo, wybrać opcję **Scan QR Code** znajdująca się w zakładce **Projects** oraz zeskanować wygenerowany wcześniej kod QR. W przypadku urządzeń z systemem iOS do zeskanowania kodu należy wykorzystać aplikację natywną **Aparat**. Po zeskanowaniu kodu, na górze ekranu pojawi się baner, który automatycznie przekieruje do aplikacji Expo z uruchomionym projektem.

Konto testowe

Na potrzeby demonstracji działania aplikacji oraz pełnego obrazu statystyk zostało utworzone konto testowe. Znajduje się na nim sto wygenerowanych losowo paragonów. Wszystkie posiadają to samo zdjęcie, które nie jest zgodne z danymi przypisanymi do paragonów. Można się na nie zalogować podając w ekranie logowania:

email: test@user.com

hasło: testuser

5.3 Model

Wymagania

Do wyszkolenia modelu niezbędny jest zainstalowany język Python w wersji 3.8.x lub wyższej.

Instalacja

Tak jak w przypadku części backendowej, aby zainstalować niezbędne do działania programu biblioteki, należy wywołać komendę `pip install` w uruchomionym terminalu w folderze `model`.

Wytrenowanie modelu

Wykorzystywany w projekcie model, można wytrenować używając załączonych danych. W tym celu należy uruchomić terminal w folderze `model`. Następnie należy wykonać komendę `python train_model.py`. Po zakończeniu działa programu, wytrenowany model będzie znajdował się w pliku `model.pkl`. Aby wykorzystać go w aplikacji, należy podmienić istniejący plik `model.pkl` znajdujący się w folderze `backend/classifier` na otrzymany w wyniku działania programu plik o tej samej nazwie.

Testy

6.1 Klasyfikator

Testy zostały przeprowadzone na przedstawionych poniżej klasyfikatorach dostępnych w bibliotece przeznaczonej do uczenia maszynowego `Scikit-learn`. Miały one na celu wyłonienie odpowiedniego klasyfikatora, dającego najlepsze wyniki klasyfikacji dla dostępnego zbioru danych.

Dummy classifier (dummy)

Klasyfikator stosowany jako punkt odniesienia dla wyników działania pozostałych klasyfikatorów. Wyniki generowane są losowo, uwzględniając jedynie rozkład klas w zbiorze uczącym [20].

K-nearest neighbours (KNN)

Klasyfikator ten jest trenowany na podstawie K najbliższych sąsiadów każdego obiektu, gdzie K jest liczbą całkowitą podawaną jako parametr. Końcowa klasa przydzielana obiekowi jest wyznaczana na podstawie klasy przyjmowanej przez największą liczbę sąsiadujących obiektów. W przypadku, gdy liczby obiektów należących do danych klas są równe, decyduje kolejność obiektów w zbiorze uczącym [21]. Podczas testów liczba uwzględnianych sąsiadów wynosiła pięć.

Klasyfikatory SVC

Działaniem klasyfikatory te zblizone są do Linear SVC opisanego w rozdziale Opis systemu 3.6.3. Różnią się wyznaczaną granicą oddzielającą obiekty dwóch klas. W zależności od jądra przyjmuje ona konkretny kształt. W testach użyto czterech klasyfikatorów SVC:

- Linear SVC (`linearSVC`) [22],
- Sigmoid SVC (`simgSVC`) [23],
- Polynomial SVC (`polySVC`) [23],
- Gaussian SVC (`rbfSVC`) [23].

Decision tree

Podczas uczenia klasyfikator tworzy drzewo decyzyjne na podstawie atrybutów zawartych w danych należących do zbioru uczącego. Następnie jest ono wykorzystywane w celu sklasyfikowania obiektu. W każdym węźle dokonywany jest wybór następnego na zasadzie `if-then-else` przybliżając rozpatrywany obiekt do jednej z klas. Złożoność decyzji w węzłach zależy od głębokości zbudowanego drzewa [24].

Random forest

Klasyfikator ten tworzy kilka drzew `Decision tree` na różnych podzbiorach danych ze zbioru. Następnie uśrednia wyniki wszystkich drzew w celu zwiększenia dokładności oraz kontroli przetrenowania modelu [25].



Multi-layer perceptron classifier (MLP)

Klasyfikator wykorzystujący wielowarstwowe sieci neuronowe. Do trenowania używa algorytmu opartego na stochastycznym spadku wzdłuż gradientu wyliczanych za pomocą propagacji wstecznej. Wspiera on klasifikację wieloklasową. Dokonywana jest ona przy użyciu `softmax` jako funkcji wyjściowej [26].

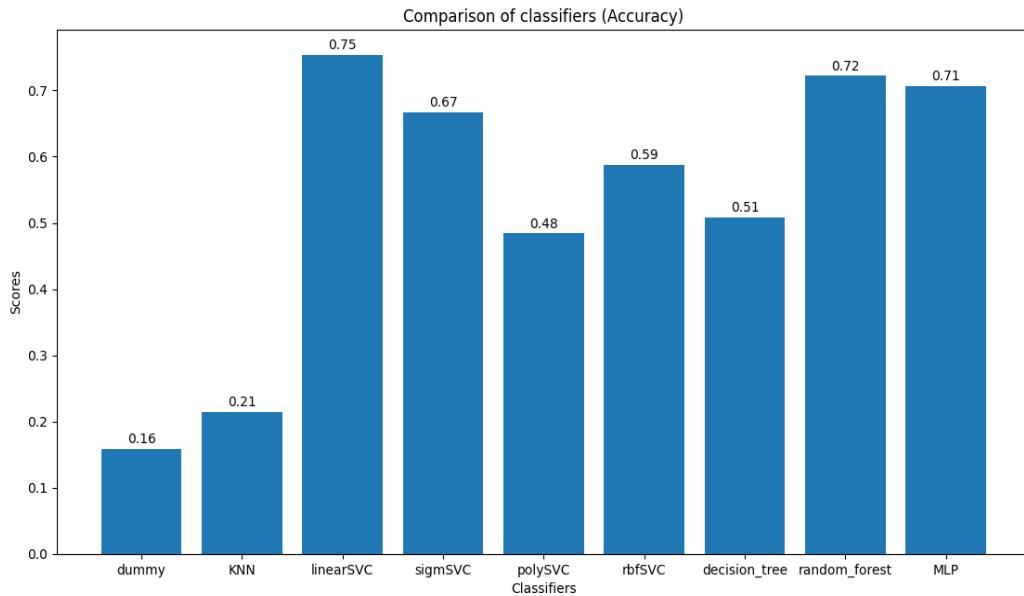
Z użyciem każdego klasyfikatora został wytrenowany model na tym samym zbiorze uczącym. W każdym przypadku teksty poddane były wektoryzacji z użyciem `CountVectorizer` oraz transformacji TFIDF za pomocą `TfidfTransformer` 3.6.2.

W przypadku Sigmoid SVC, Polynomial SVC oraz Gaussian SVC wykorzystany został klasyfikator SVC z odpowiednio jądrem ustawianym za pomocą parametru `kernel`.

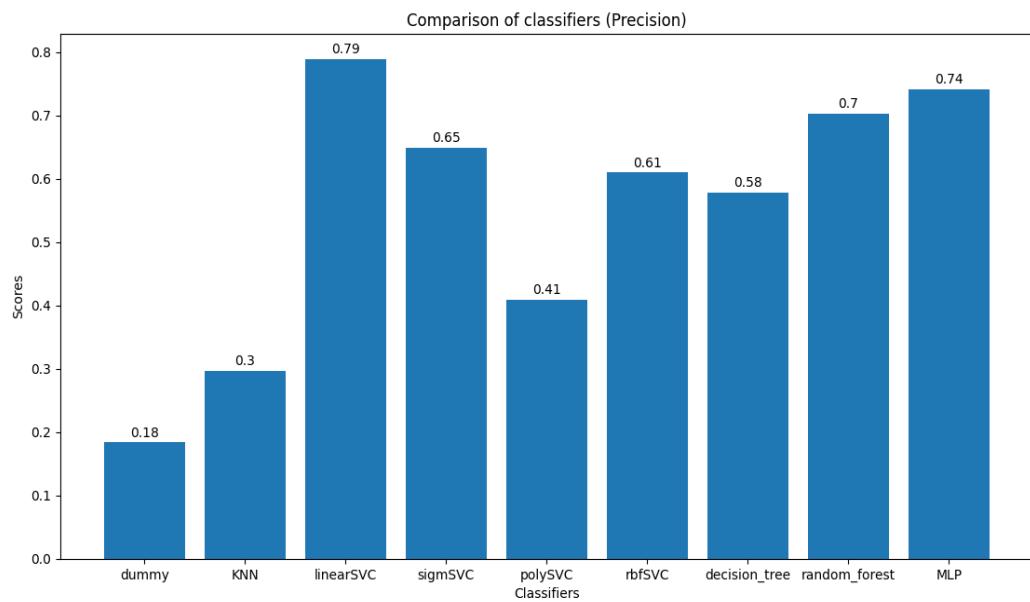
Poniżej zamieszczone zostały wykresy przedstawiające uzyskane wyniki wymienionych klasyfikatorów z podziałem na:

- Accuracy,
- Precision,
- Recall,
- F1.

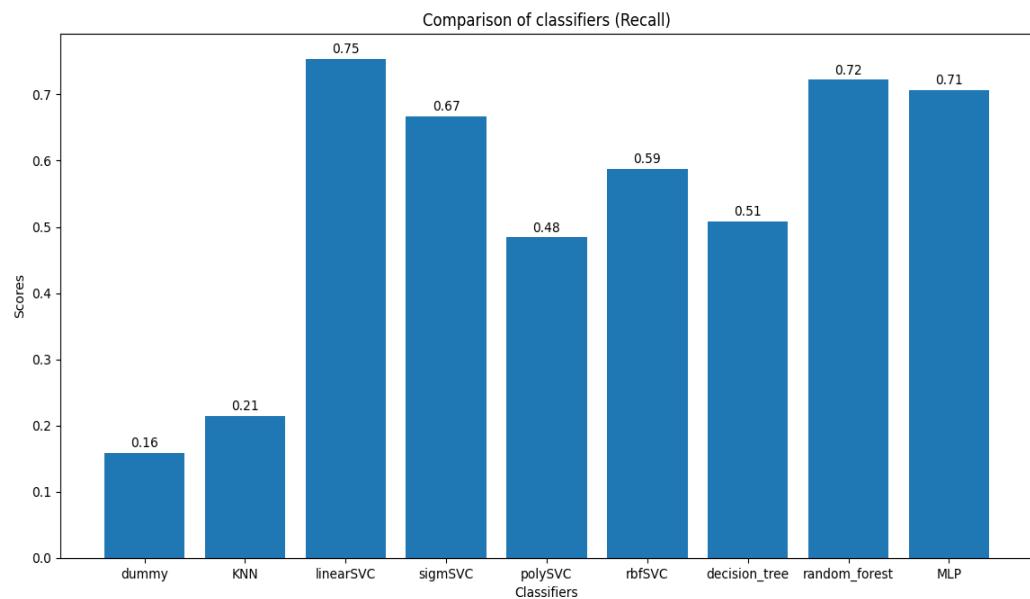
Wymienione miary zostały opisane w rozdziale **Implementacja** podczas opisu metod walidacji modelu 4.2.3.



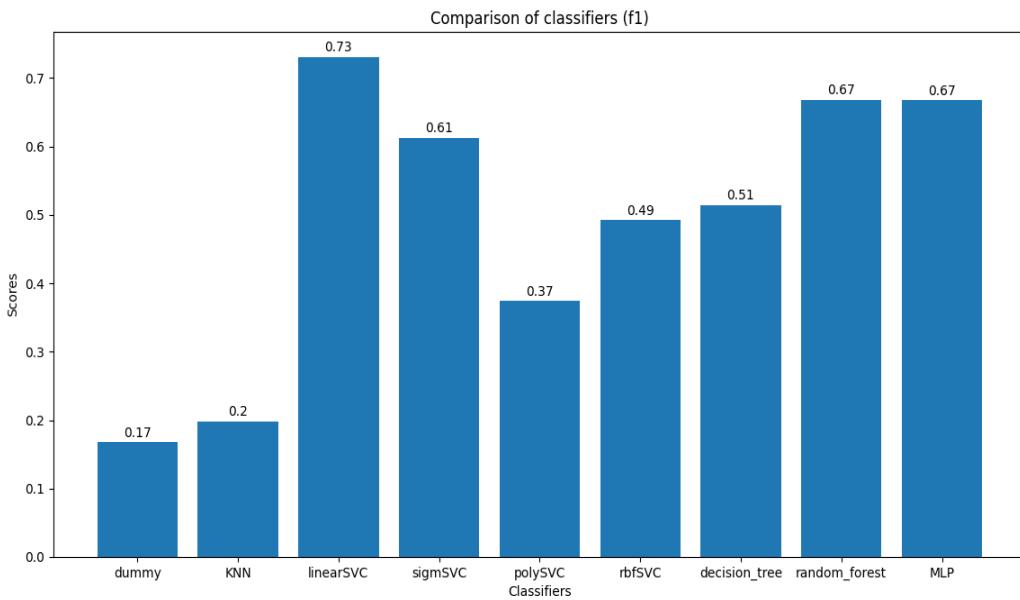
Rysunek 6.1: Porównanie wybranych klasyfikatorów na podstawie dokładności



Rysunek 6.2: Porównanie wybranych klasyfikatorów na podstawie precyzji



Rysunek 6.3: Porównanie wybranych klasyfikatorów na podstawie wartości recall



Rysunek 6.4: Porównanie wybranych klasyfikatorów na podstawie miary f1

Najodpowiedniejszą miarą sukcesu wydaje się **miara f1**. Uwzględnia ona zarówno wyniki **precision** oraz **recall** w odpowiednim stopniu. Zbiór danych nie jest na tyle zbalansowany, by można było polegać jedynie na mierze **accuracy**, jednak jest ona również istotna.

Wszystkie klasyfikatory uzyskały lepsze wyniki niż **Dummy classifier**. Był to oczekiwany rezultat, ponieważ stosuje on najbardziej prymitywne sposoby predykcji. Random forest dzięki zastosowaniu kilku drzew uzyskał lepsze wyniki od decision tree. Z klasyfikatorów SVC wyróżnił się **LinearSVC** uzyskując najlepsze wyniki we wszystkich testach. Można stwierdzić, że jest on najodpowiedniejszy z testowanych klasyfikatorów dla posiadanej zbioru danych.

Podsumowanie

7.1 Realizacja projektu

Patrząc na założenia oraz wymagania funkcjonalne projektu można stwierdzić, że jego realizacja przebiegła pomyślnie. Aplikacja umożliwia przeprowadzenie procesu digitalizacji paragonu z wykorzystaniem aparatu dostępnego w urządzeniu mobilnym oraz jego automatyczną klasyfikację. Umożliwia także zarządzanie dodanymi wcześniej paragonami oraz ich edycję. Dodatkowo aplikacja pozwala na dodawanie własnych tagów, co znacznie zwiększa jakość oraz personalizację prowadzonych statystyk. Dane dotyczące wydatków prezentowane są za pomocą wykresów oraz tabel na podstawie wybranego przez użytkownika okresu czasu. Cel został osiągnięty, umożliwiając prowadzenie statystyk swoich wydatków na podstawie paragonów za pomocą aplikacji mobilnej.

7.2 Przyszły rozwój

Patrząc na obecny stan oraz działanie systemu, można wyróżnić kilka możliwości przyszłego rozwoju. Do najważniejszych z nich należą udoskonalenia wymagające większej ilości danych, niż te zebrane w ciągu stosunkowo krótkiego okresu kolekcjonowania. Ponadto było ono utrudnione przez wprowadzane w tym okresie obostrzenia związane z pandemią COVID-19. Należą do nich:

- Podział istniejących kategorii na bardziej szczegółowe - dodatkowe kategorie pozwoliłyby na zwiększenie dokładności oraz jakości prowadzonych statystyk,
- Zwiększenie dokładności modelu - w tym celu można by wykorzystać dodawane przez użytkowników paragony oraz ich walidację automatycznie przydzielonej kategorii. Pozwoliłoby to na uzyskanie większej ilości danych, aby doszkolić model.

Oprócz tego, udoskonaleniu można by poddać moduł OCR, w celu zwiększenia jego dokładności. W tym celu najlepszym rozwiązaniem byłoby stworzenie własnego OCR, dedykowanego do odczytywania informacji z paragonów fiskalnych.



Bibliografia

- [1] Rozporządzenie ministra finansów w sprawie kas rejestrujących. Web pages: <https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20190000816>.
- [2] Perspective transform. Web pages: <https://theailearner.com/tag/cv2-getperspectivetransform/>.
- [3] Ocr. Web pages: https://pl.wikipedia.org/wiki/Optyczne_rzozpoznawanie_znak%C3%B3w.
- [4] Wyrażenia regularne. Web pages: https://pl.wikipedia.org/wiki/Wyra%C5%BCenie_regularne.
- [5] Francois Chollet. *Deep Learning with Python*. Manning Publications, New York, United States, 2017.
- [6] Hiperpłaszczyzna. Web pages: <https://pl.wikipedia.org/wiki/Hiperp%C5%82aszczyna>.
- [7] Api. Web pages: <https://www.gov.pl/web/popcwsparcie/standard-api-dla-udostepniania-danych>.
- [8] Python technology. Web pages: <https://www.python.org/>.
- [9] Opencv technology. Web pages: <https://opencv.org/>.
- [10] Pytesseract technology. Web pages: <https://pypi.org/project/pytesseract/>.
- [11] Tesseract technology. Web pages: <https://github.com/tesseract-ocr/tesseract>.
- [12] Scikit-learn technology. Web pages: <https://scikit-learn.org/stable/>.
- [13] Fastapi technology. Web pages: <https://fastapi.tiangolo.com/>.
- [14] React native technology. Web pages: <https://reactnative.dev/>.
- [15] Expo technology. Web pages: <https://expo.io/>.
- [16] React redux technology. Web pages: <https://react-redux.js.org/>.
- [17] Victory native technology. Web pages: <https://formidable.com/open-source/victory/docs/native/>.
- [18] Firebase. Web pages: <https://firebase.google.com/>.
- [19] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.
- [20] Dummy classifier. Web pages: <https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html>.
- [21] K-nearest neighbour classifier. Web pages: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>.
- [22] Linear svc. Web pages: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html>.
- [23] Svc. Web pages: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>.
- [24] Decision tree classifier. Web pages: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>.

- [25] Random forest classifier. Web pages: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [26] Mlp classifier. Web pages: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

Zawartość płyty CD

Na płycie CD znajduje się katalog z projektem o nazwie `project`. Zawiera on trzy podkatalogi:

- `backend` - zawierający programy odpowiedzialne za preprocessing obrazu, odczytanie z niego informacji oraz wydobycie danych z tekstu. Wszystkie te programy są wykorzystywane w głównym pliku `main.py`, który stanowi interfejs programistyczny aplikacji. W folderze tym znajduje się także wytrenowany model używany w aplikacji,
- `app` - zawierający aplikację mobilną,
- `model` - zawierający dane oraz programy użyte do wytrenowania modelu.

