Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Systemów Informatycznych

Mobilny system przetwarzania obrazu

Antoni Charchuła
Numer albumu 283713

promotor
mgr inż. Krzysztof Gracki

WARSZAWA 2020

Mobilny system przetwarzania obrazu

Streszczenie. Przetwarzanie obrazu odgrywa istotną rolę w dzisiejszym świecie. Niestety, operacje mogą być bardzo wymagające sprzętowo ze względu na dużą ilość obliczeń i przekształceń, co sprawia że stają się one mniej dostępne dla urządzeń z gorszymi jednostkami CPU, a przede wszystkim urządzeń mobilnych. Pomimo stałego rozwoju, ich wydajność nadal istotnie odbiega od komputerów dostępnych na rynku. Z tego względu, w pracy przedstawiany jest system, umożliwiający urządzeniom mobilnym, użycie skomplikowanych algorytmów z dziedziny przetwarzania obrazów, na swoim CPU, jak i na zewnętrznym serwerze. Stworzona została aplikacja mobilna na system Android oraz serwer typu REST, które korzystają z biblioteki OpenCV. W pracy prezentowana jest między innymi architektura systemu, sposób komunikacji jego elementów, rozwiązania napotkanych problemów, interfejs użytkownika, implementacja oraz przeprowadzone zostały także testy przedstawiające jakość działania stworzonych programów.

Słowa kluczowe: przetwarzanie obrazu, OpenCV, Java, Android

Mobile image processing system

Abstract. Image processing has a vital role in today's world. Unfortunately, operations can be very hardware-intensive due to the large number of computations and transformations, making them less accessible to devices with inferior CPUs, especially mobile devices. Despite constant development, their performance is still significantly lower than what personal computers can offer. This work presents a system that allows mobile devices to use complex image processing algorithms on their CPU as well as on an external server. A mobile application was created for the Android system and REST server, which both use OpenCV library. The work also presents the architecture of the system, the way of communication of its elements, solutions to the encountered problems, user interface, implementation and tests which were carried out in order to show the quality of created programs.

Keywords: image processing, OpenCV, Java, Android

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

……..
miejscowość i data

……..
imię i nazwisko studenta

……..
numer albumu

…….…….
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp

Celem pracy inżynierskiej było przeprowadzenie badań dotyczących wydajności przetwarzania obrazu na urządzeniach mobilnych. Wobec tego, stworzona została aplikacja mobilna umożliwiająca obróbkę zdjęć i plików wideo. Kluczowym aspektem było to, żeby użytkownik miał możliwość wyboru dowolnej ilości operacji oraz zmiany ich argumentów. Dodatkowo, wiedząc że przetwarzanie obrazu jest wymagające sprzętowo, a wiele telefonów znacząco odbiega pod względem wydajnościowym od komputerów dostępnych na rynku [1], należało stworzyć rozwiązanie tego problemu. Dlatego w ramach pracy stworzono zewnętrzny serwer, który ma wspomagać działanie aplikacji. W związku z duża popularnością i ilością gotowych rozwiązań, użyto w projekcie darmowej biblioteki OpenCV [2], która również posiada implementację na systemy mobilne. W przypadku rozwiązań niedostępnych dla telefonów, należało zaimplementować je na serwerze.

Pojęcie przetwarzania obrazu jest stosunkowo młode. Przyczyną tego było późne opracowanie efektywnych systemów komputerowych, posiadających odpowiednio duże zasoby pamięci i moc obliczeniową, zdolnych do przetworzenia plików danych reprezentujących próbki obrazu. Po raz pierwszy cyfrowego obrazu użyto w 1920 roku [3]. Nastąpiło to po wprowadzeniu pięciobitowego kodowania szarości przez system Bartlane. Skanowanie odbywało się element po elemencie, czego efektem było produkowanie przez system taśmy papierowej, na której rejestrowane były odpowiednie poziomy szarości. Zdjęcia przesyłano podmorskim kablem pomiędzy Londynem, a Nowym Yorkiem. Pod koniec lat dwudziestych zwiększono ilość tonów szarości w tym systemie oraz użyto technik znanych z fotografii, aby obrazy były lepszej jakości. Po tych osiągnięciach nastąpiła długa przerwa. Wraz ze zwiększeniem popularności komputerów mainframowych w dużych przedsiębiorstwach, dopiero w latach sześćdziesiątych zaistniały pierwsze podejścia do przetwarzania obrazów. Zostały one przeprowadzone w ramach misji kosmicznej Ranger 7, kiedy to poprawiono jakość zdjęć, na których widniał księżyc. Pod koniec lat siedemdziesiątych zaczęto używać przetwarzania obrazów w dziedzinie medycyny. Wielkim osiągnięciem było otrzymanie nagrody Nobla w 1979 roku przez Sir Godfrey'a N. Hounsfield'a oraz Profesora Allan'a M. Cormack'a za stworzenie powszechnie dziś znanej tomografii.

Jak widać, algorytmy przetwarzania obrazów mają istotny wpływ na większość dziedzin nauk oraz dzisiejszą rzeczywistość. Wszechobecne kamery w telefonach, czy te umieszczone w przestrzeni publicznej, zbierają dane, które poddawane są obróbce, używając do tego zaawansowanych technologii i metod. Dzięki nim zdjęcia stają się wyraźniejsze, wolne od szumów i niedoskonałości. Bardzo popularne staje się wykorzystywanie ich w sztucznej inteligencji przy rozpoznawaniu tekstu, gestów czy obiektów. Można je znaleźć także w medycynie, fotografii kosmicznej i satelitarnej, a nawet w kryminologii [3]. Jak widać stał się to nieodzowny element współczesnego świata, który stale się rozwija i zyskuje na popularności.

Dlatego, jednym z założeń całego systemu było stworzenie go tak, aby był łatwy do dalszego rozwinięcia. Miała zostać stworzona baza, którą dałoby się wykorzystać w kolejnych pracach i programach. Aktualna platforma posiada jedynie zalążek funkcjonalności jakie oferuje biblioteka. Jednak bez trudu, zarówno aplikację mobilną jak i serwer, da się rozbudować o kolejne funkcje. Stworzone programy i sposób ich komunikacji miały być pewnym wyznacznikiem oraz propozycją tworzenia takich systemów, a zarazem pokazaniem jak radzić sobie z problemami związanymi z przetwarzaniem obrazów. Dodatkowo, serwer może służyć do przetwarzania w innych programach. Miał być on na tyle uniwersalny, aby można było z niego korzystać przy użyciu dowolnej platformy, która posiada dostęp do internetu.

W powstałej pracy inżynierskiej przedstawione zostały etapy tworzenia aplikacji mobilnej oraz serwera, zaczynając od ich koncepcji, a kończąc na możliwościach ich rozszerzenia. Na samym początku, w rozdziałach 2 i 3, przedstawiono ogólne cele przetwarzania obrazu oraz opis problemu, którego dotyczy praca. W rozdziale 4 i 5 postawiono wymagania jakie system miał spełniać, opisano całą jego architekturę oraz zaprezentowano stos technologiczny i narzędzia, które pozwoliły na szybkie tworzenie kodu. W części szóstej, zobrazowano także działanie systemu opisując sposób komunikacji pomiędzy jego modułami oraz przedstawiono kilka wyników operacji przetwarzających obraz. W kolejnych dwóch, opisany został interfejs użytkownika oraz problemy jakie można napotkać w trakcie implementacji wraz z proponowanymi rozwiązaniami. Przeprowadzono także testy wydajnościowe, które potwierdzają problem jakim jest mała wydajność na urządzeniach mobilnych oraz słuszność wprowadzenia zewnętrznego serwera wspomagającego przeprowadzanie operacji na obrazach. Ich wyniki zostały umieszczone w tabelach i rysunkach w rozdziale 9. Wynikiem całej pracy jest w pełni działający system, przedstawiający główne operacje przetwarzania obrazów zawarte w bibliotece OpenCV. Jest on zarazem gotowy na dalsze modyfikacje, których propozycje można znaleźć w rozdziale 10 dotyczącym rozwoju systemu.

2. Cele przetwarzania obrazu

Przetwarzanie obrazu głównie odnosi się do cyfrowego przetwarzania obrazu, jako że operuje się na obrazach cyfrowych, czyli takich które są zapisane w postaci binarnej. Występują one najczęściej w dwóch różnych formatach: jako grafika wektorowa lub rastrowa. W grafice wektorowej obrazy tworzone są za pomocą określonych wcześniej punktów, które po połączeniu tworzą figury, krzywe i proste. Połączenia są opisywane za pomocą równań algebraicznych. W przypadku grafiki rastrowej obrazy są przechowywane jako dwuwymiarowa macierz pikseli, gdzie piksel to najmniejszy element obrazu, wypełniony w całości jednolitym kolorem. Najczęściej wykorzystuje się system RGB (z ang. *Red Green Blue*), w którym zakłada się, że piksel przechowuje informacje o intensywności koloru czerwonego, zielonego i niebieskiego. Model ten wynika z budowy i cech ludzkiego oka. Jakąkolwiek widzialną barwę można stworzyć poprzez kombinację tych trzech kolorów w odpowiednich proporcjach. W przypadku zdjęć czarno-białych przechowują one tylko dane o poziomie szarości.

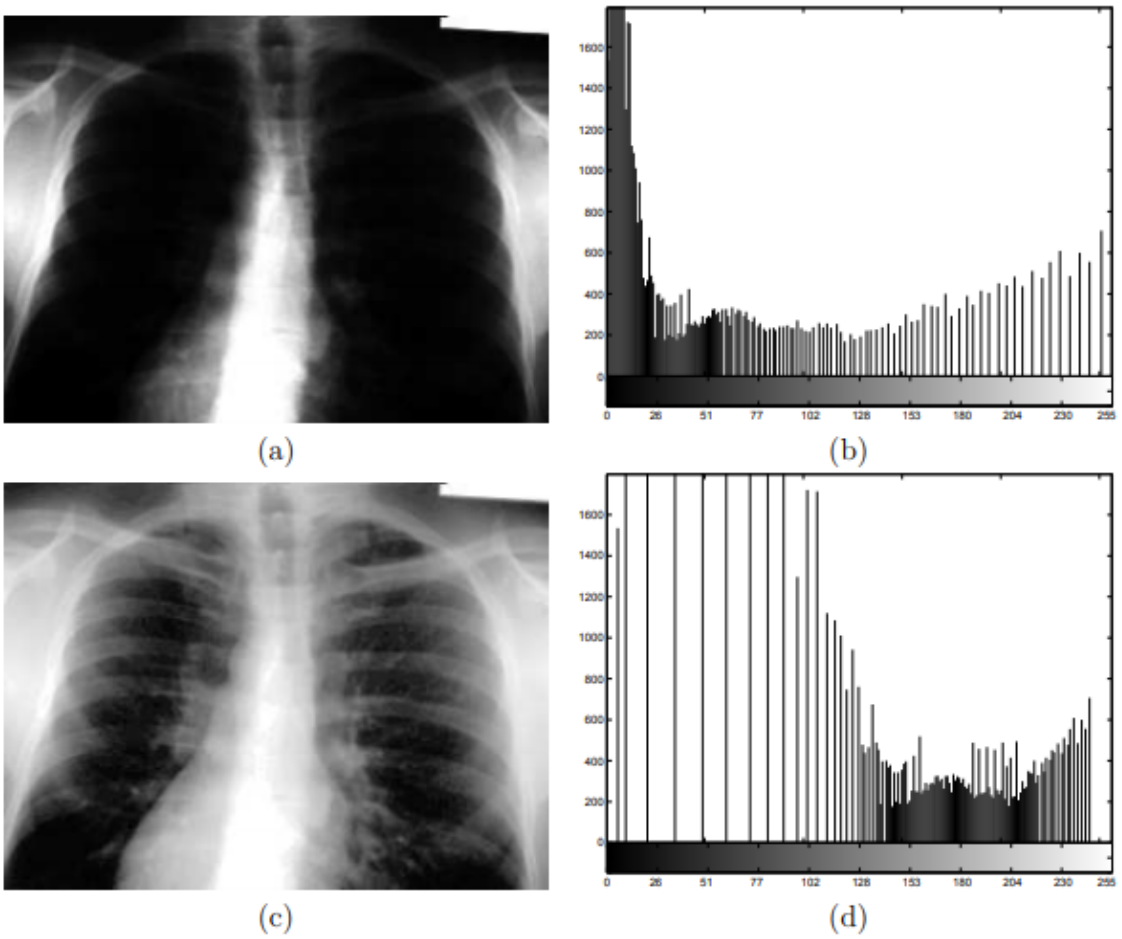
Przetwarzanie obrazu to dziedzina, której celem jest poprawienie jakości wejściowego zdjęcia lub jego przetworzenie według określonych reguł. Ma ona duży wpływ na rozwój dzisiejszego świata oraz stan w jakim aktualnie znajduje się nauka i współczesna technologia. Operacje służące do przetwarzania obrazu nie są proste, ze względu na dużą ilość działań na dużych zbiorach danych, jednak ich wyniki są na tyle efektywne i zadowalające, że stale się ich używa i je rozwija. Przetwarzanie obrazu można podzielić na kilka obszarów, którymi się zajmuje [4]:

Poprawa jakości – operacje mające na celu poprawę jakości obrazu dla odbiorcy lub dla kolejnych operacji, które dzięki lepszej jakości będą mogły dokładniej działać. Często uzyskiwana kosztem innych właściwości obrazu lub jego cech. Przykładem takiej operacji jest wyrównanie histogramu, którego wyniki można zobaczyć na rysunku 1. Polega ono na wyrównaniu wartości sąsiadujących ze sobą poziomów jasności, tak aby rozciągnęły się na szerszy zakres, czego wynikiem jest znacznie lepszy kontrast wynikowego obrazu.

Odbudowa – operacje mające na celu stworzenie poprawnego zdjęcia z wersji posiadającej uszkodzenia bądź artefakty. Korzystają z matematycznego modelu defektu, dzięki czemu możliwe jest jego usunięcie z wejściowego obrazu. Mogą być nimi na przykład szumy związane z ruchem obiektu, bądź złe wyostrzenie obiektu. Na rysunku 2 przedstawiony jest wynik nałożenia filtrów na defekt związany z widocznym rozmazaniem zdjęcia. Ich działanie polega na tzw. przetwarzaniu kontekstowym, które opiera się na analizowaniu nie tylko aktualnie branego pod uwagę piksela, ale również pikseli go otaczających.

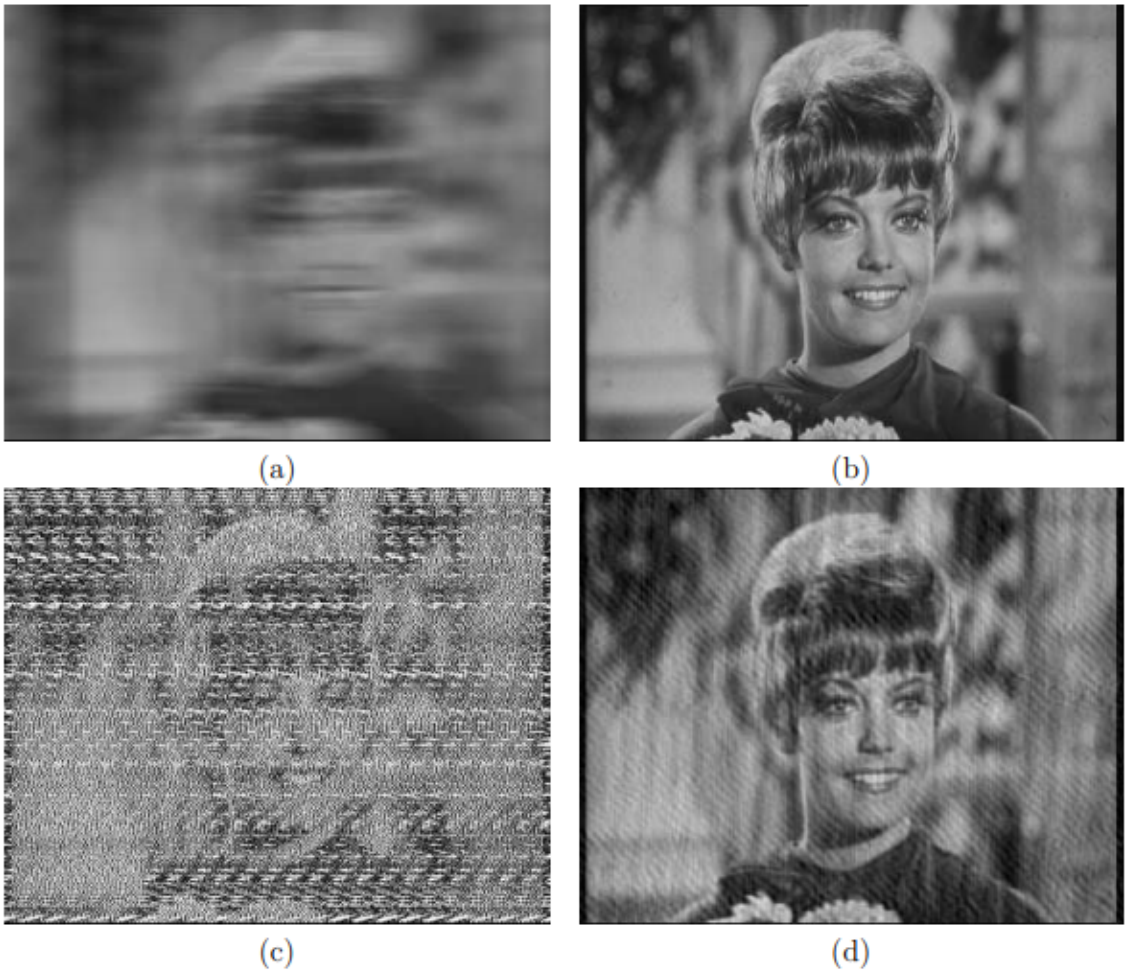
Analiza – operacje mające na celu wydobycie ze zdjęcia pożądanych informacji, na przykład tekstu, wzoru lub elementu. Szczególnie używa się ich w sztucznej inteligencji lub przy rozpoznawaniu obiektów.

Kompresja – operacje mające na celu zmniejszenie redundancji danych reprezentujących zdjęcie, czego skutkiem jest spadek wielkości pliku, w którym się ono znajduje. Można wyróżnić dwa typy kompresji: bezstratną i stratną. Pierwsza z nich modyfikuje dane tak, aby ostateczna postać obrazu była identyczna do wejściowego. W przypadku stratnej taka gwarancja już nie jest zapewniona, jednak dąży się, aby wynik był jak najbardziej podobny do pierwowzoru.



Rysunek 1. Przykład poprawy jakości zdjęcia – wyrównanie histogramu [4]

(a) Obraz oryginalny; (b) Histogram obrazu oryginalnego; (c) Obraz z wyrównanym histogramem; (d) Histogram obrazu po operacji wyrównania histogramu



Rysunek 2. Przykład odbudowania zdjęcia – filtracja [4]

(a) Obraz oryginalny; (b) Obraz odbudowany przy użyciu filtru pseudo-odwrotnego;
(c) Obraz skwantowany w 256 poziomach, na którym użyto filtru pseudo-odwrotnego;
(d) Obraz skwantowany w 256 poziomach, na którym użyto filtru Wienera

Każdy z obszarów porusza inna tematykę, jednak zawarte w nich rozwiązania razem tworzą zbiór bardzo zaawansowanych narzędzi do obróbki obrazów. Dodatkowo, operacje z dziedziny poprawy jakości i odbudowy, mogą mieć istotny wpływ na dalsze modyfikacje lub wyniki analizy przetwarzanego zdjęcia, co wskazuje na istniejącą zależność pomiędzy obszarami i możliwość ich współdziałania.

3. Opis problemu

3.1. Cel projektu

Głównym celem projektu jest umożliwienie użytkownikowi użycie operacji z dziedziny przetwarzania obrazu na urządzeniu mobilnym i przedstawienie mu ich wyników. W dobie coraz bardziej zaawansowanych smartfonów, niegdyś niedostępne dla tych urządzeń funkcje, dziś są bardzo łatwo osiągalne. Niestety, wydajność nadal znacząco odbiega od tego co oferują komputery posiadające bardziej zaawansowane procesory, a szczególnie procesory graficzne. Taki stan stawia przed programistami problem, jakim jest przeprowadzanie skomplikowanych oraz wymagających operacji, którym telefony mogą nie podołać. Dlatego, z góry założono, że należy stworzyć dodatkowy element systemu, który wspomagałby aplikację mobilną. Rozwiązanie powinno być na tyle wydajne, aby jego zastosowanie było widoczne i uzasadnione. Dodatkowo, powinno ono być łatwe w rozszerzaniu o dodatkowe funkcje, nawet takie które nie są dostępne dla telefonów. Najlepszym rozwiązaniem prawdopodobnie jest stworzenie serwera, z którym komunikacja odbywa się poprzez sieć HTTP. Pozwoli to także na użycie tego środka do kolejnych projektów, badań czy eksperymentów. Zarówno aplikacja mobilna jak i serwer nie powinny być programami zamkniętymi na zmiany i modyfikacje. Ich postać końcowa powinna mieć taką formę, aby umożliwić innym programistom ich łatwe rozszerzenie oraz użycie we własnych programach i problemach naukowych. Dodatkowo, implementacja powinna być zgodna z powszechnie znanymi standardami i wzorcami, które ułatwią jej utrzymanie i zrozumienie przez inne osoby.

3.2. Wyzwania

Przed rozpoczęciem tworzenia jakiegokolwiek projektu, warto wcześniej przemyśleć sposób działania aplikacji i przeszkody jakie mogą pojawić się w trakcie implementacji. Pozwala to na uniknięcie koniecznych, dużych zmian w kodzie, które wymagałyby dużego nakładu czasu i pracy. Istotne kwestie i problemy, których rozwiązanie pojawi się w kolejnych rozdziałach zostały przedstawione poniżej:

- określenie wymagań funkcjonalnych, czyli ustalenie jakie są oczekiwania wobec aplikacji – jest to wskaźnik na konkretne oczekiwania od systemu i usług, które ma udostępniać użytkownikowi. Wymagania te definiują między innymi co aplikacja ma robić, na co pozwala użytkownikowi, jak system ma działać oraz jakie ma funkcje;
- określenie wymagań niefunkcjonalnych, czyli określenie właściwości systemu oraz jego ograniczeń, na przykład wydajność czy sposób komunikacji jego elementów;
- znalezienie odpowiednich wzorców architektury dla aplikacji mobilnej i serwera, które pozwolą stworzyć programy zgodne ze znanymi standardami oraz ułatwią testowanie kodu, poprawią jego czytelność i możliwości rozbudowania;

- przemyślenie rodzaju serwera zewnętrznego, wspomagającego aplikację mobilną. Jest to bardzo ważna kwestia, od której zależy sposób komunikacji oraz generyczność rozwiązania, co jest istotne ze względu na to, że serwer może zostać wykorzystany w przyszłości do innych programów i prac, a jego funkcjonalność znacznie rozszerzona. Rozwiązanie powinno być na tyle wydajne, aby przy dostatecznie szybkim łączu, dodatkowy narzut czasu związany z przesłaniem i odesłaniem zdjęcia do urządzenia był opłacalny;
- ustalenie stosu technologicznego, dzięki któremu osiągnięte zostanie rozwiązanie optymalne, wydajne oraz przyjazne użytkownikowi i programiście. Nie mogą to być technologie przestarzałe. Trzeba brać pod uwagę jedynie te biblioteki i języki programowania, które są sprawdzone i popularne w użyciu;
- określenie operacji przetwarzania obrazu niedostępnych na system mobilny. W przypadku takiej funkcjonalności należy ją zaimplementować na zewnętrznym serwerze i pozwolić użytkownikowi na wykorzystanie tej operacji tylko przy jego użyciu;
- przemyślenie interfejsu użytkownika aplikacji mobilnej, która powinna być intuicyjna i łatwa w obsłudze nawet dla osoby, która wcześniej jej nie używała. Dodatkowo, powinien być on na tyle uniwersalny, aby nie ograniczał możliwości dodawania kolejnych funkcji do aplikacji;
- określenie możliwości dalszego rozwoju aplikacji, która z założenia ma być łatwa do rozszerzania o kolejne funkcje. Podczas implementacji warto o tym pamiętać, ze względu tworzenie kodu czystego i łatwego do zrozumienia, aby funkcjonalności, których system nie będzie posiadał, były możliwe do dodania w sposób bezproblemowy dla programisty.

4. Wymagania i realizacja

4.1. Wymagania funkcjonalne

Wymagania funkcjonalne określają elementy, które tworzą w pełni działający system – jego funkcje, zachowanie w konkretnych sytuacjach, w tym reakcję na błędy, oraz usługi dostępne dla użytkownika. Powinny one być kompletne i spójne, czyli oczekuje się, że będą zawierać opis wszystkich wymagań jakie system ma spełniać, które nawzajem nie będą się wykluczać.

Tabela 1. Wymagania funkcjonalne

Lp	Nazwa	Opis
F1	Wybór zdjęć	Aplikacja mobilna umożliwia wybór jednego lub większej ilości zdjęć z galerii systemowej
F2	Zrobienie zdjęcia	Aplikacja mobilna pozwala na zrobienie zdjęcia aparatem, które zostanie potem użyte do przetwarzania
F3	Wybór plików wideo	Aplikacja mobilna umożliwia wybór jednego lub większej ilości plików wideo z galerii systemowej
F4	Podgląd wybranych plików	Aplikacja mobilna pozwala na podgląd wybranych plików w dodatkowym oknie
F5	Anulowanie wybranych plików	Aplikacja mobilna pozwala na anulowanie wcześniej wybranych plików i ponowne wyświetlenie ekranu wyboru
F6	Wybór operacji oraz ich argumentów dla plików	Aplikacja mobilna pozwala na wybór dowolnej ilości wyświetlonych operacji wraz z jej argumentami, które zostaną zaaplikowane na pliki
F7	Osobne przetwarzanie	Aplikacja mobilna umożliwia wybór różnych operacji dla każdego pliku jak i miejsca przetwarzania
F8	Przetwarzanie obrazu na urządzeniu mobilnym	Aplikacja mobilna umożliwia przetworzenie obrazu na urządzeniu mobilnym
F9	Przetwarzanie zdalne	Aplikacja mobilna umożliwia wysłanie plików na zewnętrzny serwer oraz odebranie wyniku
F10	Wyświetlenie wyników	Aplikacja mobilna pozwala na wyświetlenie wszystkich przetworzonych obrazów lub plików wideo, które użytkownik wybrał

F11	Anulowanie przetwarzania	W przypadku za długiego przetwarzania, użytkownik może anulować aktualnie przeprowadzane procesy przetwarzające obraz
F12	Obsługa błędów	Aplikacja mobilna informuje użytkownika np. o braku połączenia z serwerem, czy przekroczeniu czasu przetwarzania
F13	Wyświetlenie czasu przetwarzania	Aplikacja mobilna wyświetla ile czasu zajęło przetworzenie wszystkich wybranych plików
F14	Test połączenia	Aplikacja mobilna umożliwia sprawdzenie jakości połączenia z zewnętrznym serwerem
F15	Zaawansowane wyświetlanie zdjęć	Aplikacja mobilna umożliwia dokładniejsze wyświetlenie przetworzonych zdjęć, umożliwiając ich zbliżanie
F16	Przetworzenie plików przez serwer	Serwer umożliwia odebranie pliku wraz z operacjami i ich argumentami, przetworzenie go, a następnie odesłanie wyniku

4.2. Wymagania niefunkcjonalne

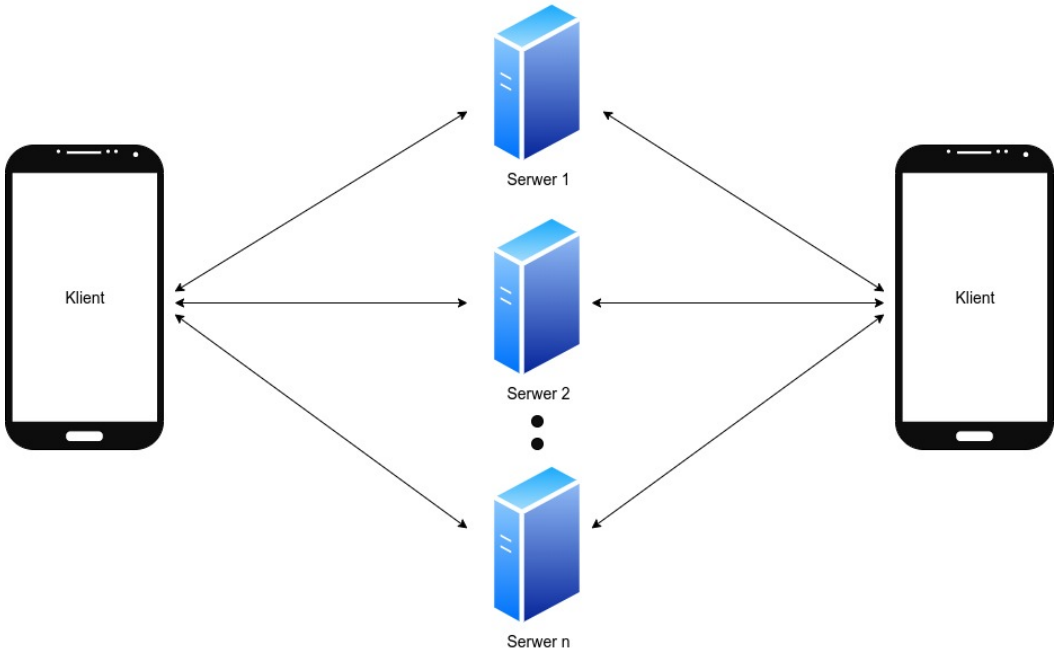
Wymagania niefunkcjonalne określają dodatkowe cechy tworzonego systemu. Nie definiują one funkcji programu, a przedstawiają zalecenia, wytyczne, a czasami ograniczenia, które są istotne podczas projektowania i tworzenia systemu.

Tabela 2. Wymagania niefunkcjonalne

Lp	Nazwa	Opis
NF1	Wielowątkowość	Zarówno aplikacja mobilna i serwer przetwarzają zdjęcia asynchronicznie na osobnych wątkach
NF2	Intuicyjność obsługi	Aplikacja mobilna posiada intuicyjny interfejs użytkownika, pozwalający niezaznajomionym z systemem użytkownikom, bezproblemową obsługę
NF3	Jakość kodu	Kod systemu jest przejrzysty i zrozumiały dla osób trzecich.
NF4	Utrzymanie	System korzysta z najnowszych wersji bibliotek i narzędzi
NF5	Komunikacja	Komunikacja z serwerem przebiega poprzez protokół HTTP

4.3. Architektura systemu

W pracy wielokrotnie już zostały przedstawione elementy systemu, czyli aplikacja stworzona dla urządzeń mobilnych oraz program pełniący funkcję serwera, który uruchamiany jest na komputerze lub serwerze. Na poniższym rysunku przedstawiony jest poglądowy schemat architektury:



Rysunek 3. Schemat architektury

Jest to przykład architektury typu klient-serwer, gdzie użytkownicy wysyłają zapytania do serwera. Komunikują się z nim za pomocą protokołu HTTP. Zapytania przetwarzane są na wielu wątkach, po to aby była możliwość obsługi wielu żądań w tym samym momencie. Wraz ze wzrostem liczby użytkowników, może się okazać, że jeden serwer nie będzie spełniał oczekiwań, dlatego docelowa postać systemu ma posiadać możliwość wysłania plików na więcej niż jedną instancję.

4.3.1. Aplikacja mobilna

W projekcie zdecydowano się stworzyć aplikację skierowaną na system Android. Jest to otwarty system rozwijany przez firmę Google. Cieszy się on ogromną popularnością na świecie, przez co ma duże zaplecze programistyczne, w tym wiele poradników, a nawet osobne IDE ułatwiające implementację programów [5].

Tworząc aplikacje na ten system, programista musi znać pewne wzorce, które ułatwiają implementację oraz pozwalają na tworzenie intuicyjnego i przejrzystego kodu. Popularnym podejściem jest oparcie się o model MVC (z ang. *Model View Controller*), którego założenia wyglądają następująco [6]:

- Model – zawiera dane, które pobierane są z bazy, bądź są tworzone w trakcie działania aplikacji. Oparty na stworzonych klasach określających atrybuty i możliwe do wykonania operacje na danym obiekcie.
- Widok – część aplikacji określająca interfejs użytkownika – między innymi wygląd okien, położenie kontrolek i ich atrybuty. Może zawierać odwołanie do elementów modelu, których dane są aktualnie wyświetlane.
- Kontroler – część aplikacji, która łączy widok i model. Kontroler zmienia widok gdy zmienia się model i na odwrót – na podstawie akcji wywołanej w warstwie widoku mogą zostać zmienione dane przechowywane w modelu.

Jednakże, niewskazane jest używanie tego wzorca przy implementowaniu aplikacji na system Android [7]. Popularnym problemem jest posiadanie kontrolera, który zawiera całą logikę. Może się nawet zdarzyć, że kontroler będzie odpowiedzialny za elementy wyświetlane na ekranie, co może skutkować dużą ilością kodu nawet dla prostego ekranu. Dodatkowo, w systemie Android występują komponenty o nazwie `Activity` zawierające kod definiujący zachowanie kontrolek i pól widocznych na danym ekranie. Aby stworzyć dowolny widok należy stworzyć klasę rozszerzającą ten komponent oraz plik z rozszerzeniem xml, który definiuje wygląd oraz położenie komponentów. Taka klasa jest odpowiedzialna za interakcję z użytkownikiem, tworzenie okna naszej aplikacji i uruchamianie innych elementów aplikacji. Często więc można napotkać w niej logikę biznesową, jako że mylona jest z kontrolerem, co jest błędem i może skutkować stworzeniem kodu, który jest trudny do utrzymania, rozszerzenia o kolejne funkcje oraz do testowania.

Dobrą alternatywą dla tego wzorca jest MVP (z ang. *Model View Presenter*). Zapewnia on modularność, czysty i łatwy do utrzymania kod oraz ułatwia przeprowadzanie testów. W MVC dopuszczalne było aby widok zawierał informacje o modelu. Tutaj występuje całkowite ich odizolowanie poprzez tworzenie prezentera, który pośredniczy pomiędzy nimi. Model w tym przypadku pozostaje bez zmian, a widok nadal jest oparty na klasach `Activity` oraz plikach xml. Prezenter zawiera całą logikę biznesową oraz jest świadomy jedynie nazw metod i ich argumentów, które pozwalają na oddziaływanie na widok. Za to widok implementuje te metody, pozwalając na interakcję z użytkownikiem oraz zmianę właściwości kontrolek. Czyli obowiązki kontrolera z poprzedniego wzorca zostały podzielone. Dobrą praktyką, wykorzystywaną w projekcie, jest także tworzenie osobnego prezentera dla każdego widoku, czyli dla każdej klasy dziedziczącej po klasie `Activity`.

4.3.2. Serwer

W przypadku serwera należało przede wszystkim określić jakiego typu będzie to serwer. Głównym wymaganiem była łatwa komunikacja z klientem oraz możliwość użycia go przy innych aplikacjach. Wobec tego, użyta technologia powinna być obsługiwana zarówno przez aplikacje mobilne i desktopowe, ale również przez przeglądarki. Rozwiązaniami jakie pasowałyby do wymagań jest między innymi serwer oparty o Sockety lub o architekturę REST API. W pracy użyto serwera typu REST [8], ze względu na bezstanowość, łatwą

komunikację, domyślne wielowątkowe przetwarzanie zapytań oraz większą uniwersalność. Jest to rozwiązanie oparte na głównych zasadach wzorca REST, które najczęściej implementowane jest na bazie protokołu HTTP. Polega na udostępnianiu klientowi interfejsów, pod którymi znajdują się odpowiednie funkcje zaimplementowane przez programistę. Główną zaletą korzystania w nim z protokołu HTTP jest z góry ustalony i najbardziej powszechny standard komunikacji oraz niezależność implementacji serwera od aplikacji klienckiej i na odwrót. Wobec tego, każdy program może wykorzystywać dowolny język programowania i technologie – jedynym wymaganiem jest korzystanie z ustalonego protokołu i odwoływanie się do konkretnych interfejsów. Przy projektowaniu takich aplikacji należy kierować się kilkoma wytycznymi:

- interfejsy są oparte na zasobach – jako zasób rozumiana jest jakakolwiek usługa, która jest dostępna dla klienta,
- zasoby są dostępne pod odpowiednimi adresami URL (tzw. endpointami), które je identyfikują,
- wysyłane żądania są niezależne i bezstanowe,
- komunikacja klient-serwer odbywa się na podstawie wymiany danych o odpowiednich, wcześniej zdefiniowanych, strukturach,
- interfejsy są jednolite – oznacza to, że powinny udostępniać używanie standardowych metod protokołu komunikacji, który jest wykorzystywany.

Sposób działania:

1. Klient przygotowuje zapytanie w postaci odpowiedniego adresu wraz z uzupełnionymi argumentami.
2. Wysyła zapytanie do interfejsu wystawionego przez serwer.
3. Serwer otrzymuje zapytanie i je przetwarza.
4. Serwer zwraca odpowiedź do klienta poprzez poprzedni interfejs.

Do komunikacji można użyć znanych metod HTTP takich jak `GET`, `POST`, `PUT` i `DELETE`. W aplikacji zostały zdefiniowane trzy adresy, przy użyciu których klient może porozumiewać się z serwerem. Zostaną one opisane w kolejnych rozdziałach.

5. Stos technologiczny

Zarówno w aplikacji mobilnej jak i na serwerze użyto biblioteki OpenCV w wersji 4.1.1. Jest to jedno z najpopularniejszych narzędzi służących do przetwarzania obrazu. Zawiera wiele modułów [2], takich jak: przetwarzanie obrazu, analizę wideo, detekcję obiektów, uczenie maszynowe czy stabilizację wideo. Biblioteka może być podstawą do tworzenia bardzo zaawansowanych systemów przetwarzania obrazu. Jej dużym atutem jest łatwa dostępność i istnienie wielu implementacji na różne systemy i języki programowania.

Cała aplikacja mobilna została stworzona przy użyciu środowiska programistycznego Android Studio. Jest to środowisko oferowane przez Google przy współpracy z firmą JetBrains. Istnieje możliwość programowania w języku Java, jednak zalecane jest korzystanie z języka Kotlin, który stale zyskuje na popularności. Jest to statycznie typowany język programowania działający na maszynie wirtualnej Javy [9]. Został on zaprojektowany z myślą o łatwym testowaniu, tworzeniu zwięzłego kodu, interpolacyjności z innymi językami działającymi na JVM, działaniu na wielu platformach oraz bezpieczeństwie. Istotnym usprawnieniem jest stworzenie rozwiązań zabezpieczającymi przed bardzo popularnym błędem „NullPointerException", który jest znany z programów napisanych w języku Java. W programie użyto także biblioteki RetroFit. Umożliwia ona w łatwy sposób tworzenie RESTowego klienta. Pozwala na pobieranie i przesyłanie danych w formie obiektów lub schemacie Json za pośrednictwem HTTP. Dzięki niej programista może wykonywać stworzone zapytania w sposób synchroniczny oraz asynchroniczny [10]. Dodatkowo, wykorzystano bibliotekę PhotoView, która udostępnia moduł służący do tworzenia dodatkowego okna ze zdjęciem, w którym może być przybliżane i oddalane przez użytkownika [11].

Serwer został stworzony przy użyciu środowiska programistycznego o nazwie IntelliJ IDEA, skierowanego do tworzenia aplikacji w języku Java. Użyto także frameworka Spring Boot [12], do utworzenia serwera typu REST. Pozwala on bardzo szybko stworzyć szkielet całej aplikacji przygotowany do dalszych modyfikacji. Dzięki niemu programista otrzymuje gotowy do uruchomienia program oparty o bibliotekę Spring. Zawiera on wiele gotowych klas, rozwiązań i adnotacji, które znacząco przyspieszają implementację programów. Dodatkowo, dostępne są elementy przygotowane szczególnie dla aplikacji typu REST, na przykład adnotacja `@RestController` umieszczona nad klasą informuje aplikację, że w klasie będą się znajdować metody, które odpowiadać będą za wystawione adresy, czy adnotacja `@PostMapping` umieszczona nad metodą informuje o tym, że metoda implementuje endpoint typu `POST`. Framework zawiera także narzędzia do automatyzacji budowy aplikacji. Do wyboru są dostępne dwa warianty: Maven oraz Gradle. W przypadku tego projektu wykorzystano pierwszy z nich. Do serializacji i deserializacji pomiędzy strukturą Json a klasami Javowymi, wykorzystano bibliotekę Gson.

6. Opis działania systemu

Użytkownikowi umożliwia się zrobienie zdjęcia i wykorzystanie do przetwarzania, ale również wybranie jednego lub większej ilości zdjęć lub filmów z galerii systemowej. Aktualnie umożliwiane jest wybranie tylko jednej z tych trzech opcji w danym momencie. Z przeprowadzonego przeglądu wynika, że wszystkie operacje z dziedziny przetwarzania obrazów są dostępne w wersji OpenCV przeznaczonej na system mobilny. Jednakże, operowanie na plikach wideo – to znaczy dzielenie na klatki, ponowne ich łączenie i zapisywanie do nowego pliku wideo – jest na nim ograniczone. Możliwe jest zapisanie tylko do pliku o rozszerzeniu avi, które nie jest obsługiwane przez system Android [13]. Dlatego więc opcja, dotycząca przetwarzania wideo, dostępna jest tylko przy użyciu zewnętrznego serwera, w którym istnieje możliwość zapisu do pliku o rozszerzeniu mp4.

Przetwarzanie odbywa się na telefonie w sposób asynchroniczny – każde zdjęcie lub film przetwarzany jest w osobnym wątku, co przyspiesza otrzymanie wyników. Zaś serwery typu REST z założenia obsługują żądania wielowątkowo.

Przetwarzanie obrazu, zarówno w aplikacji jak i serwerze, przeprowadzane jest w następujący sposób:

1. Z pliku wczytywana zostaje macierz obrazu.
2. Czytana zostaje struktura mówiąca o operacjach do przeprowadzenia.
3. Uruchamiana zostaje pętla o długości ilości operacji, która po kolei aplikuje operacje używając do tego wczytanej macierzy. Po każdej iteracji macierz zostaje nadpisana macierzą z wynikiem operacji, wobec tego każda kolejna modyfikacja operuje na najnowszej postaci obrazu.
4. Macierz zostaje przekształcona do struktury umożliwiającej wysłanie poprzez sieć lub umieszczenie do komponentu wyświetlającego zdjęcie na urządzeniu mobilnym.

W przypadku przetwarzania pliku wideo, należy początkowo podzielić plik na klatki, odczytać ich ilość wyświetlaną na sekundę oraz rozdzielczość. Po przetworzeniu wszystkich klatek, zostają one połączone i zapisane w nowym pliku o formacie mp4 o właściwościach, które wcześniej odczytaliśmy.

Na zdjęcia lub filmy można nakładać dowolną ilość operacji, które zostały dodane do systemu. A są nimi:

- wyrównanie histogramu
- obliczenie histogramu
- ekstrakcja linii poziomych lub pionowych
- znajdowanie krawędzi
- zastosowanie filtru bilateralnego
- rozmycie Gaussa
- przekształcenia morfologiczne – erozja, dylatacja, otwarcie, zamknięcie, gradient, Top Hat oraz Black Hat

- podstawowe progowanie
- zastosowanie operatora Sobel'a oraz operatora Laplace'a
- zastosowanie otoczki wypukłej
- znajdowanie krawędzi metodą Canny

Nie są to wszystkie dostępne funkcje dotyczące przetwarzania obrazu zawarte w bibliotece OpenCV, jednak celem było przedstawienie pewnej ich ilości, która pozwoli na ukazanie możliwości tych algorytmów oraz przeprowadzenie testów wydajnościowych.

6.1. Komunikacja

Jako, że stworzony serwer jest typu REST komunikacja z nim odbywa się, poprzez odnoszenie się do odpowiednich endpointów i wysyłanie określonych żądań. Jednym z wymogów aplikacji była możliwość aplikowania na jeden obraz wielu przekształceń. Wobec tego, nieoptymalne byłoby stworzenie osobnego endpointu dla każdej z operacji, ponieważ za każdym razem wymagałoby to oczekiwania na odesłanie wyniku poprzedniego przetwarzania, a następnie ponowne jego wysłanie. Odbywałoby się to tyle razy ile zaznaczylibyśmy operacji. Dlatego, pomimo że wydaje się być to rozwiązanie uniwersalne i pozwalające na bezproblemowe rozszerzenie w przyszłości, zdecydowano się na inne wyjście.

Zamiast tego, stworzone zostały tylko dwa główne adresy: `/process_image` oraz `/process_video`. Pierwszy służący do przetwarzania obrazów, a drugi do wideo. Oba posiadają identyczną strukturę:

- metoda typu `POST`,
- pierwszym argumentem jest plik zawierający zdjęcie lub wideo,
- drugim argumentem jest tekst typu Json o narzuconej strukturze zawierający informacje o operacjach jakie mają zostać przeprowadzone,
- jako wynik zwracana zostaje tablica bajtów zawierająca przetworzony obiekt.

Konieczność podzielenia wynika z innej obsługi wideo i pojedynczych zdjęć. Wideo należy przed przetwarzaniem podzielić na klatki, a na końcu je ze sobą połączyć. Zaś obraz wystarczy tylko raz odczytać.
Plik przesyłany jako argument jest typu `MultipartFile`. Zaś struktura Json, zawierająca informacje o operacjach, wygląda następująco:

```
{
  "methods" : [
    { "arguments": [ "5" ],
      "id":"GAUSSIAN_BLUR" },
    { "arguments": [ "4", "0" ],
      "id":"EROSION" }
  ]
}
```

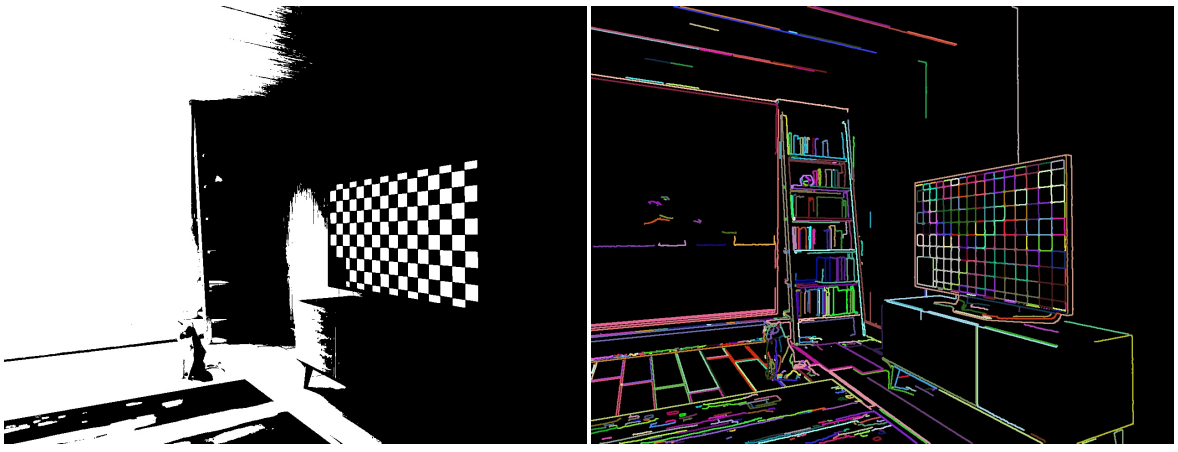
Jak widać, struktura zawiera listę metod, a każda z nich zawiera identyfikator operacji oraz jej argumenty. Takie rozwiązanie wymagało stworzenia takich samych identyfikatorów w kodzie serwera i aplikacji oraz pamiętanie o kolejności przekazywanych argumentów wraz z ich znaczeniem.

Dodatkowo, stworzony został trzeci – `/test_connection` – który korzysta z metody `GET` i służy do sprawdzania jakości połączenia. Dzięki niemu aplikacja mobilna może sprawdzić czy jest w stanie połączyć się z serwerem oraz z jaką szybkością są przesyłane dane.

6.2. Przykładowe wyniki operacji

Poniżej przedstawione zostały wyniki trzech przykładowych operacji wraz z opisem ich działania, których użyto do przetworzenia tego samego obrazu.

Rysunek 4. Obraz oryginalny oraz przykład erozji



Rysunek 5. Przykład podstawowego progowania oraz znajdowania konturów

Erozja – przykład podstawowego przekształcenia morfologicznego [14]. Przed jej przeprowadzeniem należy wybrać element strukturalny, który pozwoli na detekcję kształtów obiektu oraz analizę jego wnętrza. Cała operacja polega na przemieszczeniu elementu strukturalnego po wszystkich elementach obrazu i przeprowadzanie dla każdego z nich wspólnego iloczynu logicznego, który wskaże kolor jaki powinien przyjąć piksel.

Podstawowe progowanie – przedstawienie obrazu w formie czarno białej, gdzie użytkownik wyznacza granicę, która dzieli piksele na czarne i białe, zależnie od poziomu ich jasności.

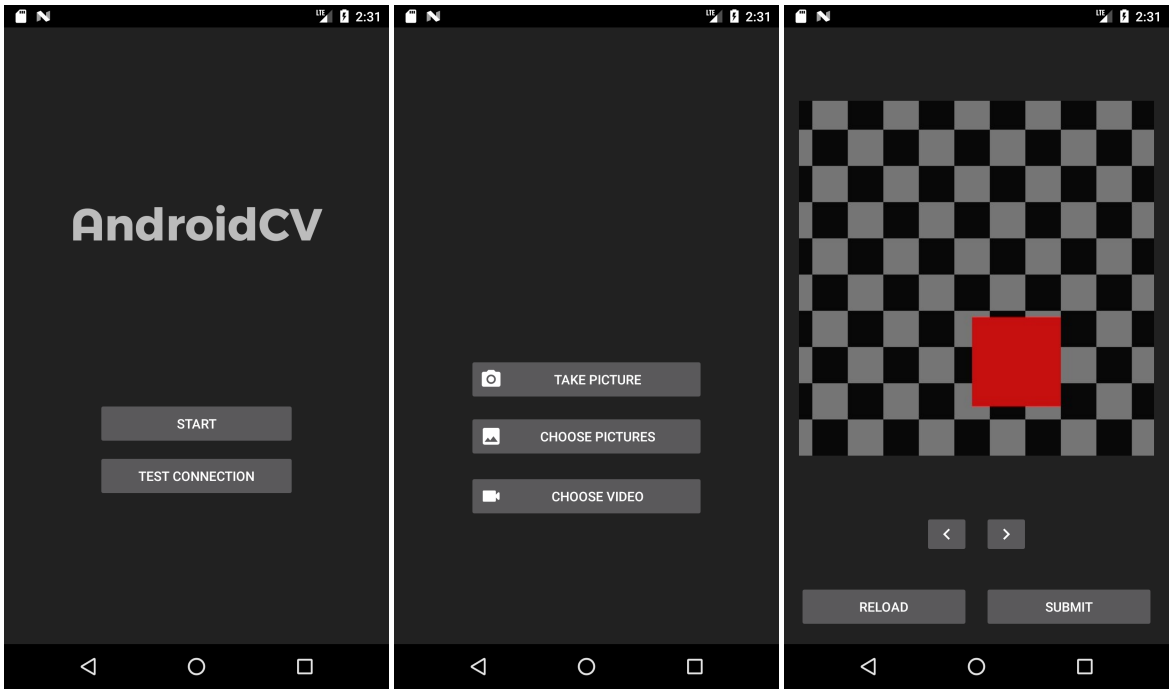
Znajdowanie konturów – jako kontur rozumiana jest linia łącząca wszystkie punkty wzdłuż granicy obrazu o tej samej intensywności. Na rysunku różnymi kolorami zaznaczone są osobne krawędzie.

7. Interfejs użytkownika

Celem interfejsu użytkownika jakiejkolwiek aplikacji jest niewątpliwie przejrzystość, intuicyjność i łatwość w użytkowaniu. Użytkownik powinien bez trudu odnaleźć się w funkcjach aplikacji, bez instrukcji czy pomocy.

Po uruchomieniu aplikacji, użytkownik widzi ekran z jej nazwą – AndroidCV – oraz przyciski `START` oraz `TEST CONNECTION`. Po kliknięciu drugiego z nich, sprawdzany jest stan połączenia, a rezultat wyświetlany jest w dodatkowym dialogu. W przypadku naciśnięcia pierwszego z nich, użytkownik przekierowywany jest do ekranu wyboru zdjęcia lub pliku wideo. Do wyboru ma trzy opcje:

- zrobienie nowego zdjęcia, używając do tego przycisku `TAKE PICTURE` – po jego wyborze użytkownik przekierowywany jest do aparatu systemowego. Po zrobieniu zdjęcia użytkownik wraca do ekranu aplikacji;
- wybór jednego lub większej ilości zdjęć, używając do tego przycisku `CHOOSE PICTURES` – po jego wyborze użytkownik przekierowywany jest do widoku galerii, skąd może wybrać wyświetlone zdjęcia;
- wybór jednego lub większej ilości plików wideo, używając do tego przycisku `CHOOSE VIDEO` – po jego wyborze użytkownik przekierowywany jest do widoku galerii, skąd może wybrać wyświetlone pliki wideo.

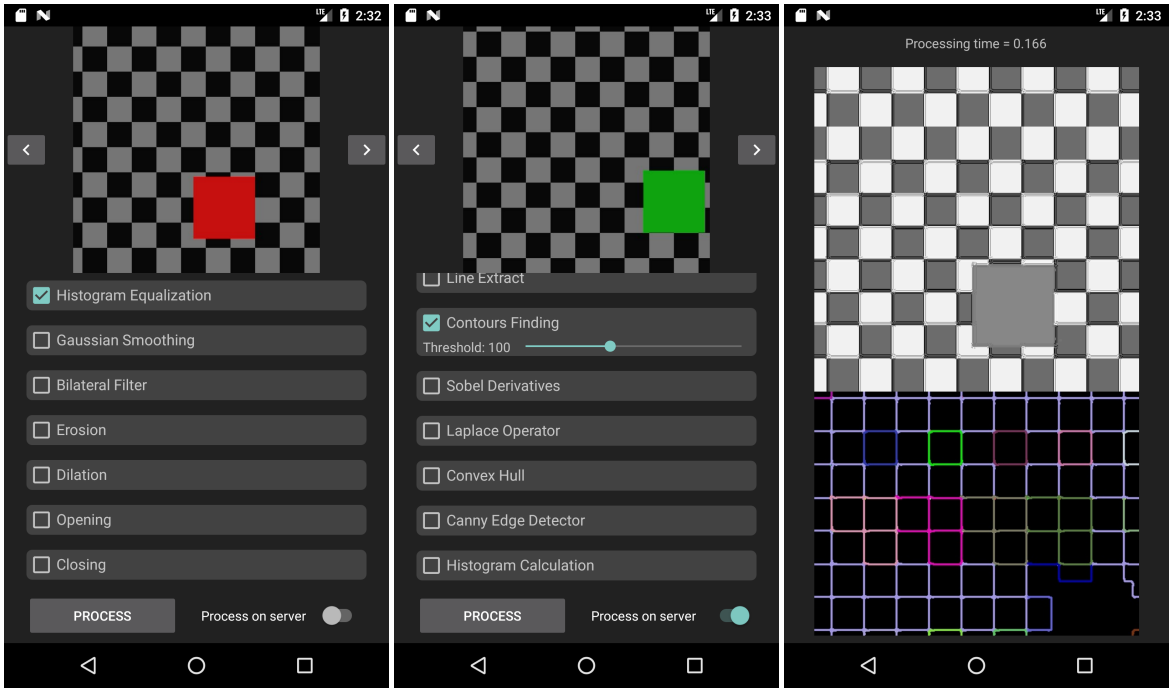


Rysunek 6. Ekran startowy oraz ekran wyboru plików do przetworzenia

Po wyborze, użytkownikowi wyświetlany jest zaznaczony plik. W przypadku wideo jest ono automatycznie odtwarzane. Jeśli użytkownik wybrał większą ilość plików, wyświetlone zostają dodatkowe strzałki, umożliwiające zmianę na kolejną lub poprzednią pozycję.

Została także przewidziana możliwość zmiany wyboru – służy do tego przycisk `RELOAD`. Wtedy wszystkie wybrane elementy są resetowane, a użytkownikowi na nowo pokazany jest ekran z trzema przyciskami, który zostały opisany na początku. W przypadku chęci zatwierdzenia, należy nacisnąć przycisk `SUBMIT`.

W tym momencie użytkownik zostaje przekierowany do głównego ekranu aplikacji – menu operacji. Na górze ekranu wyświetlane jest okno z podglądem na zdjęcie lub wideo, które aktualnie brane jest pod uwagę w ramach przetwarzania. Tak jak na poprzednim ekranie, w przypadku większej ilości elementów wyświetlane są strzałki służące do zmiany plików. Poniżej umieszczone są w przewijanej liście wszystkie dostępne operacje jakie można zaaplikować. Wyboru dokonuje się poprzez zaznaczenie pola wyboru przy nazwie operacji. Jeśli operacja zawiera dodatkowe argumenty, które należy sprecyzować, wyświetlane zostają także odpowiednie pola, dzięki którym można je ustawić. Na każdy plik można nałożyć dowolną ilość operacji. Ważnym aspektem jest również to, że każdy modyfikowany jest osobno, wobec tego dla każdego z nich należy wybrać operacje jakie mają zostać wykonane, a także miejsce gdzie ma być on przetworzony. Służy do tego przełącznik w prawym dolnym rogu ekranu `Process on server`. W przypadku jego zaznaczenia, plik zostanie wysłany na serwer. Jeśli użytkownik modyfikuje plik wideo, przełącznik jest zablokowany w stanie przetwarzania na zewnętrznym serwerze, ze względu na brak wsparcia tej funkcji przez OpenCV na urządzeniach mobilnych. Zatwierdzenie wyborów odbywa się poprzez naciśnięcie przycisku `SUBMIT` w lewym dolnym rogu ekranu.



Rysunek 7. Ekran z menu operacji oraz ekran przedstawiający wyniki przetwarzania

Użytkownik zostanie wtedy przekierowany do ekranu gdzie pokazywane są wyniki przetwarzania. Początkowo, może się wyświetlić jedynie ekran ładowania, ze względu na to, że wyniki są pokazywane dopiero po przetworzeniu wszystkich plików. Na samej górze pokazywany jest czas w jakim udało się je przetworzyć. Dalsza część ekranu różni się zależnie od tego czy przetwarzane są zdjęcia czy pliki wideo. W pierwszym przypadku wszystkie obrazy są wyświetlane jeden pod drugim w przewijanej liście. Jeśli użytkownik chce dokładniej się przyjrzeć wynikowi, ma on możliwość naciśnięcia określonego obrazu, po czym wyświetli mu się dodatkowe okno, w którym może go przesuwać, przybliżać i oddalać. W przypadku przetwarzania wideo, wyświetlany jest widok znany z pozostałych ekranów – okno z przetworzonym wideo wraz ze strzałkami jeśli było ich więcej niż jedno. Dodatkowo umieszczony został przycisk służący do odtworzenia wyświetlanego filmu na nowo.

Przewidziane zostały także sytuacje niestandardowe, które również należało obsłużyć:

- Użytkownik nie znając możliwości swojego urządzenia mobilnego lub wybierając takie operacje, które wymagają dużego czasu przetwarzania nawet na bardzo wydajnych maszynach, może długo oczekiwać na jakiekolwiek wyniki na ostatnim ekranie aplikacji. Wobec tego, przy chęci anulowania i przyciśnięcia przycisku wstecz prowadzącego do poprzedniego ekranu, uruchomione wcześniej procesy zostaną automatycznie zamknięte, nie powodując niepotrzebnego obciążenia urządzenia.
- W przypadku przekroczenia długości dozwolonego czasu przetwarzania, który jest ustawiony w kodzie aplikacji, zostaje użytkownikowi wyświetlone powiadomienie o jego przekroczeniu. Zdjęcie lub wideo, które naruszyło to ograniczenie, nie zostanie wyświetlone w ostatecznych wynikach.
- Jeśli serwer jest nieosiągalny ze względu na brak połączenia internetowego lub brak uruchomionej instancji pod wyznaczonym adresem, wyświetlane jest użytkownikowi powiadomienie o brak możliwości przetwarzania na serwerze. Pliki przetwarzane na urządzeniu mobilnym, nadal zostaną przetworzone, a wynik wyświetlony.

8. Implementacja

Zarówno kod aplikacji jak i serwera został stworzony przy użyciu języków obiektowych. Wobec tego podstawową strukturą są klasy, które określają obiekty, którymi programista może operować. W przypadku aplikacji mobilnej dominują klasy związane z `Activity`, które określają zachowanie, widok oraz logikę programu. W przypadku serwera, klas jest znacznie mniej ze względu na to, że jego działanie zapewnia biblioteka Spring Boot. Cały szkielet znajduje się w gotowym, wewnętrznym kodzie, w który nie należy ingerować. Programiście zostaje tylko stworzenie kontrolera, który zawiera definicje wystawianych adresów. W tabeli 3 i 4 przedstawione zostały wszystkie klasy znajdujące się w kodzie aplikacji mobilnej oraz serwera.

Tabela 3. Klasy aplikacji mobilnej

Klasa	Opis
`MainActivity`, `MenuActivity`, `PictureSelectionActivity`, `ViewPictureActivity`, `ViewVideoActivity`	Klasy definiujące, dla określonego widoku, sposób interakcji z użytkownikiem, tworzenie okna oraz zachowanie umieszczonych na nim kontrolek
`MainActivityPresenter`, `PictureSelectionActivityPresenter`, `MenuActivityPresenter`, `ViewPictureActivityPresenter`, `ViewVideoActivityPresenter`	Klasy prezenterów dla każdej z poprzednio wymienionych klas dziedziczących po `Activity`. Każda z nich zawiera także osobny interfejs `View`
`ImageProcessingImpl`	Klasa z metodami statycznymi, które implementują operacje przetwarzania obrazu
`ImageProcessingMethod`, `ImageProcessingMethodWrapper`	Klasy zawierające model operacji do wykonania
`Methods`	Klasa typu `enum` zawierająca identyfikatory operacji
`ServerProvider`	Interfejs zawierający metodę mającą na celu tworzenie połączenia z konkretnym serwerem
`SingleServerProvider`	Klasa implementująca interfejs `ServerProvider`, która łączy za każdym razem z tą samą instancją serwera
`NetworkClientProvider`	Klasa statyczna zwracająca obiekt typu `ServerProvider`
`UploadAPIs`	Interfejs z dostępnymi endpointami serwera
`PhotoClientProcessing`	Klasa dziedzicząca po `AsyncTask`, realizująca przetwarzanie zdjęcia na urządzeniu mobilnym

`PhotoServerProcessing`	Klasa dziedzicząca po `AsyncTask`, realizująca przetwarzanie zdjęcia po stronie serwera
`ProcessingExceptions`	Klasa typu `enum` zawierająca identyfikatory wyjątków występujących podczas przetwarzania przy użyciu zewnętrznego serwera
`ProcessingTask`	Klasa zawierająca model jednego przetwarzania do uruchomienia
`VideoServerProcessing`	Klasa dziedzicząca po `AsyncTask`, realizująca przetwarzanie pliku wideo po stronie serwera
`LargePictureHandler`	Klasa umożliwiająca stworzenie mniejszej wersji wejściowego zdjęcia tzw. *thumbnail*

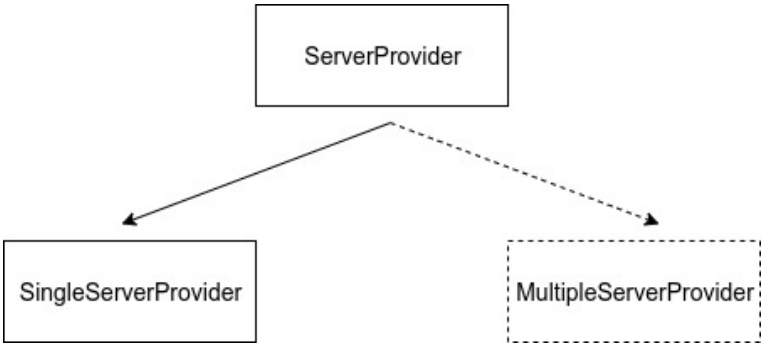
Dodatkowo dla każdej klasy z `Activity` istnieje plik xml, który definiuje wygląd i elementy danego widoku.

Tabela 4. Klasy serwera

Klasa	Opis
`ServerApplication`	Klasa uruchamiająca serwer i ładująca bibliotekę OpenCV
`ImageProcessingController`	Klasa reprezentująca kontroler serwera, deklarująca i definiująca udostępniane endpointy
`ImageProcessingMethodImpl`	Klasa z metodami statycznymi, które implementują operacje przetwarzania obrazu
`ImageProcessingMethod`, `ImageProcessingMethodWrapper`	Klasy zawierające model operacji do wykonania
`Methods`	Klasa typu `enum` zawierająca identyfikatory operacji
`ImageProcessor`	Interfejs zawierający metodę, której celem jest przetworzenie wejściowego obrazu
`StandardImageProcessor`	Klasa implementująca interfejs `ImageProcessor`, która przeprowadza operacje przetwarzające obraz na CPU serwera

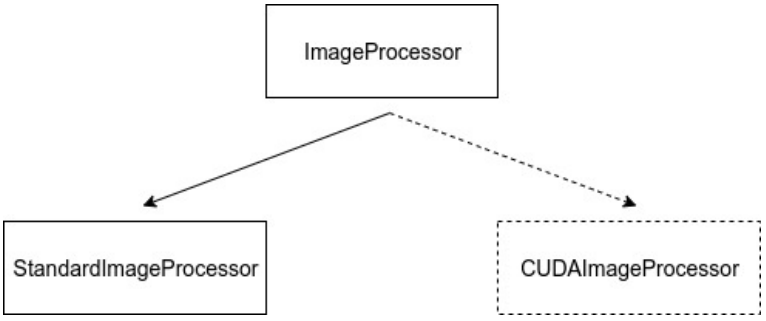
8.1. Dostępne interfejsy

Aby system był łatwy do rozszerzenia, dodano interfejsy, które łatwo pozwolą na zmianę działania aplikacji. Zostały one przedstawione na rysunkach 8 i 9, na których również zaznaczono istniejące klasy, które je implementują wraz z klasami, których istnienie przewiduje się w przyszłości. Te oznaczone zostały przerywaną linią.



Rysunek 8. Interfejs ServerProvider

`ServerProvider` to interfejs znajdujący się w kodzie aplikacji mobilnej, który posiada metodę `getRetrofitClient`, która zwraca obiekt umożliwiający nawiązanie połączenia z konkretnym serwerem. Instancję obiektu implementującą go, przechowuje klasa `NetworkClientProvider`. Aktualnie zwracany jest obiekt klasy `SingleServerProvider`, który komunikuje się za każdym razem z tą samą instancją serwera. Jeśli program miałby komunikować się z wieloma instancjami, należałoby stworzyć osobną klasę implementującą interfejs, odpowiednio zaimplementować wymaganą metodę i podmienić obiekt w klasie `NetworkClientProvider`.



Rysunek 9. Interfejs ImageProcessor

W przypadku serwera stworzono interfejs `ImageProcessor` zawierający metodę `process`. Przyjmuje ona macierz zdjęcia oraz strukturę zawierającą informacje o operacjach do przeprowadzenia. Jej celem jest przetworzenie zdjęcia i zwrócenie wyniku. Aktualnie implementuje go klasa `StandardImageProcessor` przeprowadzająca operacje na CPU, jednak w przypadku chęci dodania możliwości przetwarzania np. na procesorze graficznym, należałoby stworzyć nową klasę implementującą interfejs oraz podmienić obiekt typu `ImageProcessor`, który znajduje się w klasie `ImageProcessingController`.

8.2. Kluczowe fragmenty kodu

8.2.1. Powiązanie Widoku i Prezentera

W związku z wykorzystaniem wzorca MVP, w aplikacji mobilnej należało zastosować specjalne interfejsy i struktury. Tworząc dowolny widok należało stworzyć:

- klasę dziedziczącą po odpowiedniej klasie z grupy `Activity`, na przykład podstawowej `AppCompatAcitivity`,
- plik xml definiujący znajdujące się na widoku kontrolki i ich właściwości,
- klasę prezentera dla tworzonego widoku.

Przykładowa implementacja:

```
class MainActivityPresenter(private val view: View) {

    fun handleButtonClick() {
        val text = getData()
        view.changeTextView(text)
    }

    private fun getData() : String { //odniesienie do modelu }

    interface View {
        fun changeTextView(text: String)
    }
}

class MainActivity : AppCompatActivity(), MainActivityPresenter.View {

    private lateinit var presenter: MainActivityPresenter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main) //plik activity_main.xml

        presenter = MainActivityPresenter(this)

        button.setOnClickListener {
            presenter.handleButtonClick()
        }
    }

    override fun changeTextView(text: String) {
        textView.text = text
    }
}
```

Przy tworzeniu prezentera należy stworzyć interfejs, w tym przypadku nazwany został „View", który posiada metody, pozwalające na interakcję z widokiem. Dodatkowo, potrzebny jest konstruktor, który przyjmuje jako argument obiekt implementujący metody stworzonego interfejsu. W przypadku widoku, klasa `MainActivity` musi także implementować interfejs `MainActivityPresenter.View`. Dodatkowo, należy stworzyć zmienną typu `MainActivityPresenter`, która inicjowana jest podczas tworzenia widoku w funkcji `onCreate`. Widok ten zawiera dwa elementy zadeklarowane w pliku xml – przycisk o identyfikatorze „button" oraz wyświetlany tekst o identyfikatorze „textView". W kodzie odnosimy się do nich po nadanych nazwach jak do zmiennych. Aby umożliwić interakcje użytkownika z aplikacją, dodawany jest listener do przycisku, który wywołuje funkcję zaimplementowaną w klasie prezentera. Oprócz tego, należy zaimplementować funkcję z interfejsu – w tym przypadku zmianę wyświetlanego tekstu. W funkcji `handleButtonClick` prezenter może odwołać się do modelu pozyskując potrzebne dane, a następnie używając metod z interfejsu, przekazać je do widoku. Jak widać, klasy z widoku zawierają jedynie logikę związaną z obsługą wyświetlanych kontrolek, za to prezenter łączy widok z modelem oraz wywołuje odpowiednie akcje, o których implementacji nie ma świadomości, a mają wpływ na wyświetlany ekran.

8.2.2. Komunikacja klient-serwer

Aplikacja posiada trzy punkty dostępu – jeden służący do testu połączenia oraz dwa pozwalające na przetworzenie zdjęcia lub wideo. W przykładzie posłużono się kodem służącym do obróbki pojedynczego zdjęcia.

```
@PostMapping("process_image")
public @ResponseBody byte[] processImage(
            @RequestParam("file") MultipartFile file,
            @RequestParam("json") String json) throws IOException {

    ImageProcessingMethodWrapper options =
        new Gson().fromJson(json, ImageProcessingMethodWrapper.class);

    byte[] image = file.getBytes();

    Mat src = Imgcodecs.imdecode(new MatOfByte(image), Imgcodecs.IMREAD_COLOR);

    src = processor.process(src, options);

    (..) //transformacja macierzy na tablice bajtow

    return result;
}
```

Powyższy kod prezentuje definicję endpointu po stronie serwera. Adnotacja `@PostMapping` informuje bibliotekę Spring Boot, o tym że będzie on typu `POST`. Następnie, zdefiniowana jest nazwa, do której klient będzie mógł się odnieść w przy-

padku chęci jego użycia. Typ po adnotacji `@ResponseBody` mówi jakiego typu będzie zwracany wynik. Jako argumenty przyjmowane są dwa obiekty nazwane „file" oraz „json", odpowiadające odpowiednio plikowi typu `MultipartFile` oraz strukturze Json przekazywanej jako zwykły `String`. W implementacji metody zawarta jest deserializacja Json'a do obiektu typu `ImageProcessingMethodWrapper`. W następnej kolejności pobierana jest tablica bajtów z otrzymanego pliku, która jest później transformowana do macierzy, którą obsługuje biblioteka. W funkcji `process` odbywa się pętla, która aplikuje operacje i zwraca wynik.

W przypadku klienta użyto biblioteki RetroFit, która znacząco ułatwiła implementację komunikacji z serwerem. Przede wszystkim należy stworzyć klasę, która pozwoli na uzyskanie obiektu typu `Retrofit`, pozwalający na nawiązanie połączenia.

```
class NetworkClient {
    companion object { //companion object oznacza statycza czesc kodu
        private val BASE_URL = "http://10.0.2.2:8080/"
        var retrofit: Retrofit? = null

        fun getRetrofitClient(): Retrofit? {
            if(retrofit == null) {
                val okHttpClient = OkHttpClient.Builder()
                    .connectTimeout(10, TimeUnit.SECONDS)
                    .readTimeout(2, TimeUnit.MINUTES)
                    .writeTimeout(1, TimeUnit.MINUTES)
                    .build()
                retrofit = Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create())
                    .client(okHttpClient)
                    .build()
            }
            return retrofit
        }
    }
}
```

Dodatkowo należy stworzyć interfejs z deklaracjami endpointów, do których aplikacja ma wysyłać żądania.

```
interface UploadAPIs {
    @Multipart
    @POST("/process_image")
    fun processImage(@Part file: MultipartBody.Part, @Part("json") json: RequestBody)
        : Call<ResponseBody>
}
```

Ostatecznie, komunikacja wygląda następująco:

```
val retrofit = NetworkClient.getRetrofitClient()
val uploadAPIs = retrofit!!.create(UploadAPIs::class.java)
val file = File(filePath)

val fileReqBody = RequestBody.create(MediaType.parse("image/*"), file)
val filePart = MultipartBody.Part.createFormData("file", file.name, fileReqBody)

val json = Gson().toJson(methodsWrapper)
val jsonReqBody = RequestBody.create(MediaType.parse("multipart/form-data"), json)

val call = uploadAPIs.processImage(filePart, jsonReqBody)
val fileByteArray = call.execute().body()!!.bytes()
```

W powyższym kodzie osobno tworzone są ciała przesyłanych obiektów dla każdego elementu. W przypadku pliku wykorzystano `MultipartBody`, w którym należy umieścić plik o typie `image/*`. Zaś strukturę Json stworzono używając biblioteki Gson, która serializuje obiekt `methodsWrapper` do obiektu typu `String`, który następnie umieszczono jako `multipart/form-data`. Na końcu, kod odwołuje się do odpowiedniego adresu, do którego umieszczane są wcześniej stworzone obiekty i przy użyciu funkcji `execute` zapytanie zostaje wykonane w sposób synchroniczny. Cały ten kod wykonywany jest we wcześniej stworzonym, osobnym wątku na aplikacji mobilnej, jako że system Android nie pozwala na zapytania synchroniczne na głównym wątku. Biblioteka RetroFit udostępnia także komunikację w sposób asynchroniczny przy użyciu funkcji `enqueue`.

8.2.3. Implementacja przetwarzania obrazu

W przypadku przetwarzania obrazu implementacja jest prosta ze względu na gotowe funkcje zawarte w bibliotece OpenCV oraz istniejące poradniki pokazujące jak ich użyć w kodzie.

```
public static Mat gaussianBlur(Mat src, int kernelSize) {
    if (src.channels() == 1)
        cvtColor(src, src, COLOR_GRAY2BGR);

    Mat dst = new Mat();
    int kernel = (kernelSize * 2) + 1;
    Imgproc.GaussianBlur(src, dst, new Size(kernel, kernel), 0, 0);
    return dst;
}
```

Powyższy kod pozwala na zastosowanie rozmycia Gaussa. Funkcja przyjmuje jako argument macierz z obrazem oraz odpowiednie argumenty dotyczące poziomu rozmycia. Istotne jest aby operować na odpowiedniej macierzy, która posiada odpowiednią ilość kanałów. Ze względu na to, że niektóre funkcje operują na obrazach czarno białych, zdjęcia konwertuje się do postaci z jednym kanałem. W przypadku kolorowych zdjęć, według standardu RGB, jest ich trzy. Przed zastosowaniem jakiejkolwiek operacji przetwarzania

obrazu trzeba wcześniej upewnić się na jakiej macierzy powinna operować i w przypadku gdy posiadamy nieodpowiednią, należy ją przekonwertować na wymaganą. Jeśli taka czynność nie zostanie wykonana, biblioteka wyrzuci wyjątek. Konwersja wykonywana jest za pomocą funkcji `cvtColor` – parametr `COLOR_GRAY2BGR` powoduje konwersję do trójkanałowego obrazu, zaś `COLOR_BGR2GRAY` do jednokanałowego.

8.3. Test jednostkowe

W przypadku obu aplikacji, stworzone zostały testy, których celem było zapewnienie łatwego sposobu sprawdzenia poprawności działania systemu w przypadku modyfikacji kodu. Wykorzystano bibliotekę JUnit, która znacznie ułatwia ten proces. Przede wszystkim, skupiono się na testach sprawdzających:

- możliwość uruchomienia biblioteki OpenCV,
- możliwość przeprowadzenia operacji z biblioteki OpenCV,
- działanie konwersji zdjęć posiadających trzy kanały do jednokanałowych i na odwrót,
- poprawną serializację i deserializację pomiędzy strukturą Json, a klasą `ImageProcessingMethodWrapper`,
- poprawność wystawionych endpointów przez serwer.

Szczególnie ostatnie wymagały dodatkowej wiedzy o możliwościach testowania aplikacji napisanych przy pomocy frameworka Spring Boot, dlatego poniżej zaprezentowano przykład wraz z jego wyjaśnieniem.

```
@RunWith(SpringRunner.class)
@WebMvcTest
public class ServerApplicationTests extends OpenCVTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void photoProcessingEndpointTest() throws Exception {
        byte[] bytes = ... //tablica bajtow dowolnego zdjecia
        MockMultipartFile file =
            new MockMultipartFile("file", null, "image/*", bytes);

        String json = ... //dowolny json z metodami i argumentami

        this.mockMvc.perform(
            MockMvcRequestBuilders.multipart("/process_image")
                .file(file)
                .param("json", json))
            .andExpect(status().is(200));
    }
}
```

Przede wszystkim należało użyć adnotacji `@RunWith(SpringRunner.class)` oraz `@WebMvcTest`. Pozwalają one na uruchomienie podstawowego kontekstu serwera, dzięki któremu można odwoływać się do wystawionych adresów. Jednak aby móc to zrobić, dodatkowo należy należy stworzyć zmienną typu `MockMvc` i wstrzyknąć do niej zależność używając adnotacji `@Autowired`. W teście należało stworzyć obiekty, których wymaga testowany endpoint. Oprócz zwykłego obiektu typu `String`, użyto klasy `MockMultipartFile`, która podczas tworzenia przyjmuje argumenty dotyczące, między innymi, nazwy przesyłanego obiektu (w tym przypadku „file") oraz typu przesyłanej zawartości zawartej w tablicy bajtów (`image/*`). Następnie, przy użyciu zmiennej `mockMvc` i jej metody `perform`, można wysłać żądanie do docelowego adresu, uzupełniając przy tym odpowiednie parametry. Ostatecznie, funkcja oczekuje zwrócenia statusu HTTP o wartości równej 200, który informuje o poprawnym przetworzeniu żądania. W przypadku innej wartości test zakończyłby się niepowodzeniem.

8.4. Możliwości rozbudowania

Powstały system posiada jedynie zalążek możliwości jakie oferuje biblioteka OpenCV, jednak powstała pewna baza, która umożliwia łatwe jej rozszerzenie. Przede wszystkim można dodać pozostałe operacje przetwarzania obrazu zawarte w bibliotece. Do tego celu należy zaimplementować ich obsługę na serwerze i aplikacji mobilnej oraz dodać odpowiednie pole wyboru w widoku wyboru operacji. Dostępne funkcjonalności, takie jak poprawa jakości rozmazanego zdjęcia zawierającego element w ruchu, czy usuwanie szumów z niewyraźnego obrazu, nie wymagałyby bardzo dużej ilości czasu, a na pewno zwiększyłyby walory i możliwości systemu. Nie ograniczając się tylko do modułu zawierającego metody przetwarzające obraz, można sprawdzić dodatkowe możliwości z innych modułów biblioteki OpenCV i dodać na przykład funkcje z dziedziny sztucznej inteligencji czy detekcji obiektów.

Aktualna postać serwera jest dość prosta i ograniczona. Wspomnianym już pomysłem jest umożliwienie przetwarzania dużej ilości zdjęć lub filmów na kilku jego instancjach. W ten sposób system jeszcze szybciej przeprowadzałby przetwarzanie. W przypadku aplikacji mobilnej, dobrym pomysłem byłoby dodanie możliwości zmiany kolejności przeprowadzania operacji na danym zdjęciu, w przypadku gdy zaznaczyliśmy ich więcej niż jedną. Aktualnie nie ma takiej możliwości – im niżej operacja przedstawiona w liście na ekranie wyboru, tym później zostanie wykonana. Różna kolejność uruchomienia operacji na pewno może mieć wpływ na końcowy wynik.

Częstym problemem, który można napotkać podczas używania programu, jest decyzja dotycząca gdzie dany obraz przetworzyć. Użytkownikowi często nie jest znana wielkość zdjęcia oraz jego wymiary, co utrudnia wybór. Wszakże stworzony został prowizoryczny wskaźnik szybkości łącza, który powinien w pewien sposób to ułatwić, jednak nadal nie jest to kompletne rozwiązanie. Jako dodatkową funkcję, można by było stworzyć kod, który

na podstawie właściwości zdjęcia i jakości połączenia z serwerem, decydowałby za nas czy przetworzyć obraz na telefonie, czy je wysyłać i odebrać tylko wynik. Jeszcze ciekawszym rozwiązaniem byłoby umieszczenie sztucznej inteligencji, np. sieci neuronowej, która dzięki posiadaniu danych o poprzednich wynikach przetwarzania – jaka operacja jak długi czas była przeprowadzana na danej platformie – mogłaby jeszcze dokładniej i trafniej podejmować decyzję. Stworzenie takiej funkcji nie powinno być trudne ze względu na dużą ilość bibliotek oferujących gotowe moduły. Jedynym problemem mogłoby być pozyskanie dostatecznej ilości danych z informacjami potrzebnymi do wytrenowania takiej sieci, aby jej skuteczność była zadowalająca.

9. Testy

Testy mają na celu zbadanie wydajności systemu, potwierdzenie jego działania i słuszności wprowadzenia zewnętrznego serwera jako rozwiązania problemu dotyczącego niskiej wydajności urządzeń mobilnych oraz zaprezentowanie wyników jakie są osiągalne przy użyciu powstałej implementacji.

9.1. Opis środowiska i narzędzi

Do testów użyto takich samych zdjęć w formacie JPG, w trzech różnych rozmiarach:

- 1280 x 720 – 720p – 378,3 KB
- 2560 x 1440 – 2K – 1,35 MB
- 3840 x 2160 – 4K – 2,75 MB

Użyto także plików wideo o parametrach:

- 1280 x 720 – 5 MB – 739 klatek – 29 sekund
- 1280 x 720 – 1 MB – 132 klatek – 5 sekund

Sprzęt:

- Klient – Samsung Galaxy S4 I9505 – Snapdragon 600 1.9 GHz x 4 – 2 GB RAM
- Serwer – Asus UX410UA – Intel® Core™ i7-7500U CPU 3.5 GHz × 2 – 16 GB RAM

Telefon działał na systemie Android w wersji 7.1.2, zaś serwer na Ubuntu w wersji 19.10.

W przypadku przeprowadzania przetwarzania obrazu na serwerze, obiekty były przesyłane z prędkością około 9 MB/s – wyniki testów zawierają czasy związane z przesłaniem na serwer i z powrotem. Wszystkie czasy przetwarzania zawarte w poniższych tabelach są wyrażone w sekundach.

9.2. Wyniki i wnioski

Tabela 5. Wyniki testów – przetwarzanie obrazu na telefonie

Zdjęcie	Wyrównanie histogramu	Top Hat	Znajdowanie konturów
720p	0.314	2.228	0.876
2K	0.751	7.322	8.752
4K	1.471	16.336	25.521

Tabela 6. Wyniki testów – przetwarzanie obrazu na serwerze

Zdjęcie	Wyrównanie histogramu	Top Hat	Znajdowanie konturów
720p	0.629	0.706	0.58
2K	1.171	1.797	1.164
4K	1.724	3.707	2.692

Operacje użyte w testach, których wyniki przedstawione są w tabeli 5 i 6, posiadały argumenty: Top Hat – kernel 21 oraz elipsa jako element strukturalny; znajdowanie konturów – próg 255.

Patrząc na powyższe rezultaty widać, że w przypadku pierwszej operacji, która jest mało wymagająca, telefon radzi sobie lepiej niż serwer. Powodem może być dodatkowy narzut czasu związany wysłaniem i odesłaniem zdjęcia poprzez sieć. Przy pozostałych dwóch operacjach, serwer poradził sobie znacznie lepiej. Wszakże, przy małych zdjęciach różnica jest niewielka, jednak w przypadku znajdowania konturów, przy użyciu zdjęcia o rozmiarze 4K, wynik był lepszy aż o 22,8 sekundy.

Tabela 7. Wyniki testów – równoległe przetwarzanie zdjęć

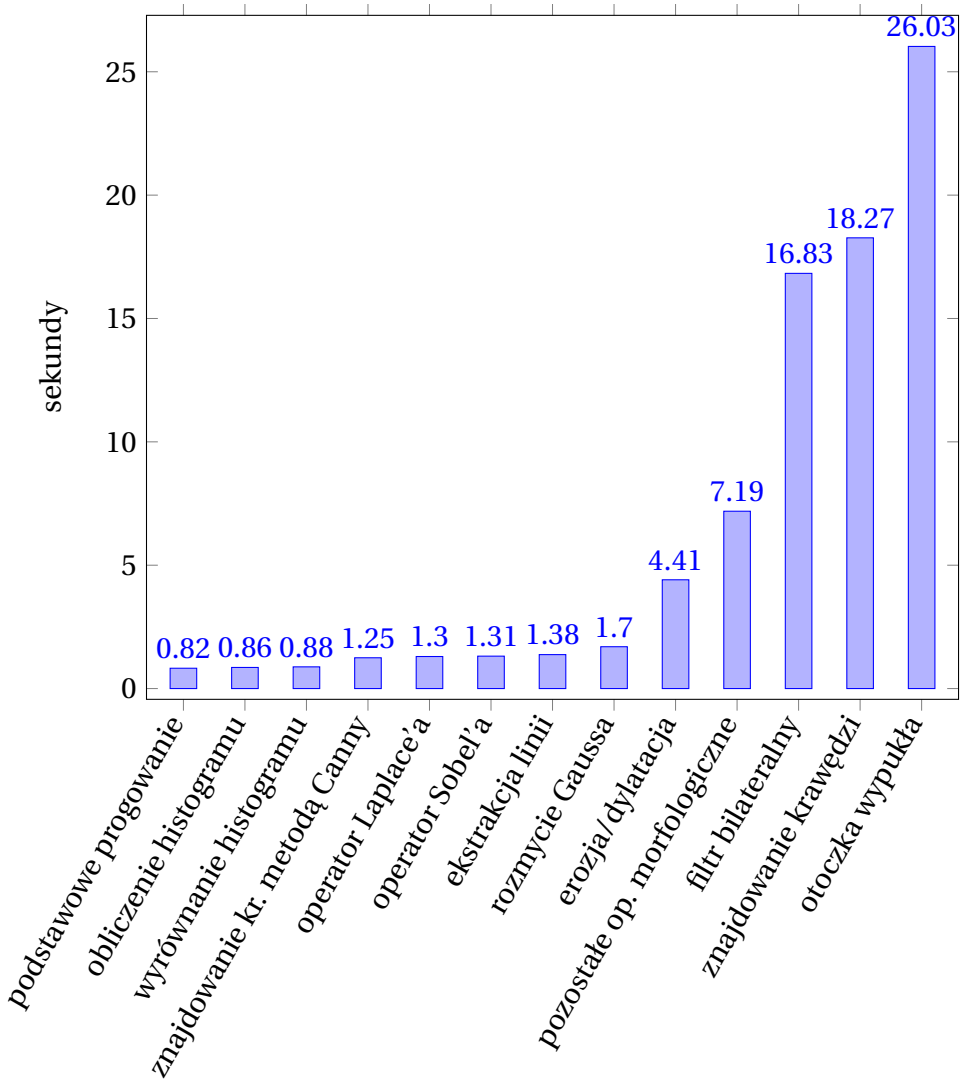
Zdjęcia przetworzone na telefonie	Zdjęcia przetworzone na serwerze	Wynik
720p, 2K, 4K	-	25,672
720p, 2K	4K	8.45
720p	2K, 4K	2.241
-	720p, 2K, 4K	2.278

Powyższe wyniki dotyczą testów związanych z równoległym przetwarzaniem trzech zdjęć o różnych rozmiarach, używając do tego operacji znajdowania konturów (próg 255). Jak widać, początkowo ostateczny czas przetwarzania był długi ze względu na uruchomienie go na telefonie. Już w kolejnym teście, gdzie zdjęcie o rozmiarze 4K wysłane zostało na serwer, najdłuższa operacja jest związana z przetworzeniem zdjęcia o rozmiarze 2K. W trzecim i czwartym przypadku, długość przetwarzania jest narzucona przez serwer, gdzie wynik związany jest ponownie z największym zdjęciem.

Rysunek 10 wskazuje czas wykonania poszczególnych operacji. W przypadku gdy dana opcja umożliwiała wybór argumentów, wybierane były takie, aby czas był jak najdłuższy. Jedynie w przypadku znajdowania krawędzi oraz otoczki wypukłej zastosowano granicę równą tylko 220 (im bliżej 0 tym dłuższe wykonanie), ze względu na to, że już przy tej wartości widać ogromny skok czasu wykonania porównując go do pozostałych przypadków. W teście wykorzystano zdjęcie 2K i przetwarzano je na urządzeniu mobilnym. Celem tego testu było pokazanie, które operacje powinny być przeprowadzane na serwerze, aby czas wykonania był jak najmniejszy. Z poniższych wyników widać, że bez wątpienia zalicza się do nich zastosowanie otoczki wypukłej, filtr bilateralny, znajdowanie krawędzi oraz operacje morfologiczne. Oczywiście uwzględniając to, że zdjęcie jest dostatecznie duże oraz argumenty operacji odpowiednio ustawione.

Pod pojęciem „pozostałe operacje morfologiczne” są rozumiane operacje: otwarcie, zamknięcie, gradient, Top Hat oraz Black Hat. Umieszczone zostały razem ze względu na bardzo podobny czas przetwarzania.

Rysunek 10. Czasy przetwarzania obrazu na telefonie z podziałem na dostępne operacje



Ciekawe wyniki można zauważyć także w tabeli 8. Wykonano tam dwa testy związane z przeprowadzeniem kilku operacji na zdjęciu o rozmiarze 2K. Wartości wskazują, że złożenie kilku operacji podczas jednego przetwarzania zajmuje mniej czasu, niż gdyby miały zostać przeprowadzone osobno. Najłatwiej to dostrzec patrząc na drugi test gdzie operacje, zgodnie z rysunkiem 10, osobno trwałyby przez około 22 sekundy, gdy razem zajęły 20 sekund. Spowodowane jest to brakiem konieczności ponownego wczytywania zdjęcia, przekształcania go do macierzy, którą obsługuje biblioteka OpenCV, oraz końcowego zapisu do bitmapy. Czynności te wykonywane są tylko jeden raz.

Tabela 8. Wyniki testów – przetwarzanie obrazu przy użyciu kilku operacji

Operacje	**Wynik**
wyrównanie histogramu + Top Hat	7.579
filtr bilateralny + erozja + podstawowe progowanie	19.97

Aby w pełni przedstawić funkcjonalność systemu, należało także przeprowadzić testy związane z przetwarzaniem plików wideo. Przeprowadzono je na dwóch filmach o różnej długości, ale o takiej samej rozdzielczości, używając przy tym wyrównania histogramu oraz operację morfologiczną – Top Hat.

Tabela 9. Wyniki testów – przetwarzanie wideo

Plik wideo	**Wyrównanie histogramu**	**Top Hat**
5 sekund	2.735	18.023
29 sekund	10.82	95.911

Jak widać przetwarzanie plików wideo jest znacznie dłuższe niż pojedynczego zdjęcia. Jest to związane z koniecznością modyfikacji każdej klatki – w 5 sekundowym filmiku było ich aż 132, co pokazuje jak długie musiałoby być przetwarzanie dłuższego pliku. Dodatkowo, ich rozdzielczość wynosiła 1280x720. W dobie filmów 4K, a nawet 8K, są to wymiary bardzo małe, dlatego w przypadku chęci obsługi tak dużych obrazów, należałoby zaimplementować przetwarzanie plików wideo obsługujące przeprowadzanie operacji na karcie graficznej, co z pewnością znacząco przyspieszyłoby otrzymanie wyniku.

10. Rozwój systemu

Jednym z głównych założeń systemu było stworzenie aplikacji mobilnej oraz serwera, które będą łatwe do rozszerzeń o nowe funkcjonalności. Powstały kod i architektura powstała w taki sposób, aby umożliwiała modyfikacje oraz rozbudowę systemu, bez konieczności gruntownej przebudowy. Dlatego postawiono na szczegółową modularyzację kodu, która sprawia że łatwo się go czyta i utrzymuje.

Aktualna postać systemu nie pozwala na uruchomienie jej na maszynie, do której dostęp miałoby wiele osób. Jest to przede wszystkim spowodowane bezpieczeństwem. Serwer aktualnie przyjmuje jakiekolwiek żądanie przychodzące z zewnątrz. Jest to niedopuszczalne ze względu na łatwe przeprowadzenie ataku, który miałby na celu wysłanie tak dużej ilości wymagających żądań, aby zająć wszystkie wolne zasoby maszyny, uniemożliwiając tym poprawne działanie. Rozwiązaniem tego problemu byłoby stworzenie logowania, dzięki któremu dostęp do funkcji aplikacji miałyby tylko uprawnione do tego osoby.

W przypadku chęci jeszcze szybszego przeprowadzania przetwarzania, można użyć modułu z biblioteki OpenCV, która umożliwia przeprowadzanie operacji na jednostce graficznej maszyny w środowisku CUDA stworzonym przez firmę Nvidia. Współczesne karty graficzne posiadają potężną moc obliczeniową, która używana jest między innymi przy przetwarzaniu dużej ilości danych, wielu programach naukowych, a co najważniejsze, stworzona architektura idealnie nadaje się do przetwarzania obrazów. Dzięki gotowemu modułowi, programista nie musi na nowo implementować skomplikowanych algorytmów na jednostce graficznej, które wymagają dodatkowej wiedzy o architekturze i działaniu tych urządzeń. Testy pokazują, że przetwarzanie obrazu może przyspieszyć nawet trzydziestokrotnie [15], co jest niewyobrażalnym skokiem wydajności w przypadku porównania do procesorów wykorzystywanych w komputerach. Ów skok byłby tym bardziej zauważalny porównując wydajność do urządzeń mobilnych.

Podczas przetwarzania obrazu na serwerze, pliki zapisywane są tymczasowo na dysku, a po ukończeniu potrzebnych działań są one usuwane. Nazwa pliku to aktualny czas systemowy w milisekundach. Jeśli system miałby działać na większą skalę, taki sposób nazewnictwa byłby niedopuszczalny, ze względu na szansę stworzenia dwóch plików o takiej samej nazwie. Rozwiązaniem jest stworzenie odpowiednio zaawansowanej funkcji hashującej, która tworzyłaby unikalną nazwę.

Dużym utrudnieniem jest uruchamianie serwera w nowym środowisku. Niestety, biblioteka OpenCV wymaga skomplikowanej instalacji i konfiguracji, które często mogą długo trwać. Dlatego, pomocne może być skorzystanie z technologii Docker [16]. Pozwala ona na tworzenie wirtualnej maszyny w kontenerze, w którym uruchamiane są tylko potrzebne procesy związane z umieszczaną na nim aplikacją. Wobec tego, jest to rozwiązanie o wiele wydajniejsze od wirtualizacji całego systemu. Środowisko jest tworzone na podstawie wcześniej przygotowanego pliku, w którym zdefiniowane są kroki instalacji

i konfiguracji potrzebnych bibliotek oraz zależności, które wykonywane są przy starcie kontenera. Jako wynik, otrzymujemy w pełni zautomatyzowany sposób na uruchamianie nowych instancji serwera na kolejnych maszynach, który jest bardzo szybki, wydajny i wygodny dla programistów.

Cała praca była głównie skierowana do użytkowników systemu mobilnego Android. Jednakże, powstały serwer może posłużyć znacznie większemu gronu odbiorców. Jakakolwiek przeglądarka czy aplikacja desktopowa, nie posiadając jakiejkolwiek biblioteki, jest w stanie dzięki niemu przetworzyć obraz wysyłając go poprzez sieć i otrzymać przetworzony wynik. Pozwala to zaoszczędzić bardzo dużą ilość duplikowanego kodu, ale przede wszystkim umożliwia korzystanie z zaawansowanych funkcji platformom, na których nie byłyby one dostępne. Wraz z dodawaniem dodatkowych funkcji, serwer mógłby oferować rozwiązania dla wielu programów i aplikacji. Jedynym wymogiem byłaby możliwość komunikacji przy pomocy protokołu HTTP i połączenie do internetu, jeśli serwer nie znajduje się w tej samej sieci. Niestety, serwer typu REST nie jest odpowiedni do wykorzystywania go w przypadku systemów czasu rzeczywistego związanych z przetwarzaniem obrazu. Narzut czasu, związany z wysłaniem i odesłaniem pliku, jest na tyle duży, że opóźnienie z tym związane nie byłoby akceptowalne w takich rozwiązaniach.

11. Podsumowanie

Stworzony został system, skierowany dla użytkowników urządzeń mobilnych, umożliwiający im przetwarzanie obrazu. Zapewnia on intuicyjny interfejs, wydajne działanie i pełne wsparcie operacji zawartych w bibliotece OpenCV. Problem wydajnościowy również został rozwiązany poprzez stworzenie zewnętrznego serwera, który ma możliwość przeprowadzenia zleconych operacji i odesłanie wyników, a przeprowadzone testy potwierdziły korzyści z jego zastosowania.

Ostateczna postać systemu nie jest kompletna ze względu na niewielką ilość zaimplementowanych operacji, jednak struktura kodu jak i architektura została zaprojektowana w taki sposób, aby była łatwa do dalszego rozwoju i modyfikacji. Bowiem celem tej pracy było stworzenie projektu, który może posłużyć do dalszych prac, badań i innych programów komputerowych. Użyto w tym celu popularnych narzędzi i koncepcji, aby tworzone moduły były jak najbardziej uniwersalne i generyczne. Stworzona została baza, która może służyć jako przykład tworzenia takich aplikacji, wraz z rozwiązaniami problemów jakie można po drodze napotkać. Z pewnością, nie muszą być to rozwiązania najlepsze, jednak starano się zaproponować wyjścia, które zapewnią dobrą jakość kodu, przejrzystość, ale również odpowiednią wydajność.

Bibliografia

[1] M. Hudelist, C. Cobârzan i K. Schoeffmann, „OpenCV Performance Measurements on Mobile Devices”, kw. 2014. DOI: 10.1145/2578726.2578798.

[2] *Dokumentacja biblioteki OpenCV*, Dostęp zdalny (12.01.2020): https://docs.opencv.org/master/.

[3] R. E. W. Rafael C. Gonzalez, *Digital Image Processing*. Upper Saddle River, NJ 07458: Pearson Prentice Hall, 2008, s. 1–25.

[4] W. K. Chen, *The Electrical Engineering Handbook*. Elsevier Academic Press, 2005, s. 891–911.

[5] *Dokumentacja dla programistów aplikacji na system Android*, Dostęp zdalny (13.01.2020): https://developer.android.com/docs.

[6] S. T. P. Glenn E. Krasner, „A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System”, 1988.

[7] J. Kim, *Getting Started with MVP (Model View Presenter) on Android*, Dostęp zdalny (12.01.2020): https://www.raywenderlich.com/7026-getting-started-with-mvp-model-view-presenter-on-android, 2018.

[8] Microsoft, *Projekt interfejsu API sieci Web*, Dostęp zdalny (12.01.2020): https://docs.microsoft.com/pl-pl/azure/architecture/best-practices/api-design, 2018.

[9] I. W. Marcin Moskala, *Android Development with Kotlin*. Packt Publishing, 2017, s. 7–16.

[10] *Dokumentacja biblioteki RetroFit*, Dostęp zdalny (12.01.2020): https://square.github.io/retrofit/.

[11] C. Banes, *Photo View*, Dostęp zdalny (12.01.2020): https://github.com/chrisbanes/PhotoView, 2019.

[12] *Dokumentacja frameworka Sping Boot*, Dostęp zdalny (19.01.2020): https://spring.io/projects/spring-boot.

[13] *Obsługiwane formaty wideo przez system Android*, Dostęp zdalny (13.01.2020): https://developer.android.com/guide/topics/media/media-formats.

[14] G. Szczurek, „Zastosowanie metod morfologii matematycznej do detekcji i dekompozycji obrazów”, 2003.

[15] *OpenCV - CUDA*, Dostęp zdalny (13.01.2020): https://opencv.org/cuda/.

[16] C. Anderson, „Docker”, *IEEE Software*, t. 32, nr. 3, s. 102–c3, 2015, ISSN: 1937-4194. DOI: 10.1109/MS.2015.62.

Spis rysunków

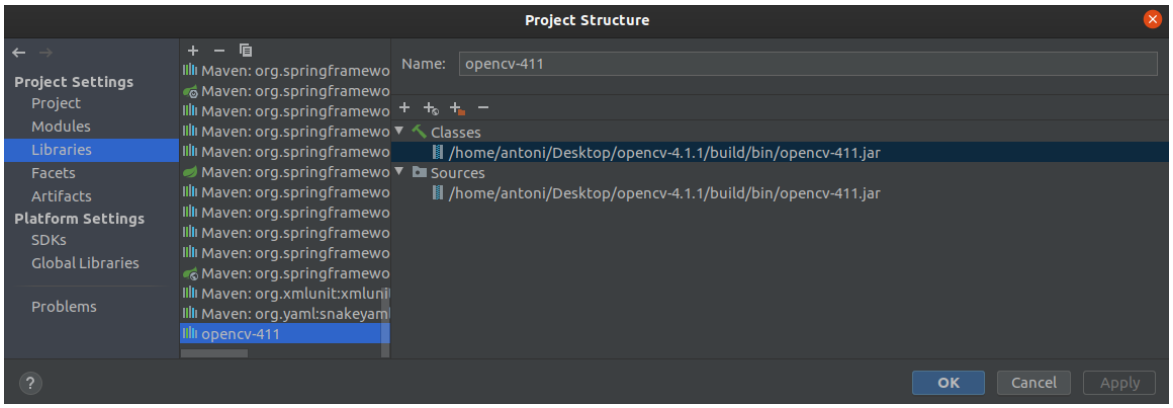
Spis tabel

Dodatek A

Często, bardzo pracochłonną czynnością, pojawiającą się w trakcie tworzenia programów komputerowych, jest konfiguracja środowiska programistycznego. Niestety, również w tym projekcie można tego doświadczyć ze względu na korzystanie z biblioteki OpenCV dla języka Java. Dlatego, w ramach pracy, stworzony został dodatek, którego celem jest ułatwienie kolejnym osobom uruchomienie powstałego systemu. Cała instrukcja skierowana jest dla użytkowników systemu Ubuntu w wersji co najmniej 18.04 oraz przedstawiane są czynności konfiguracji dla IDE o nazwie Android Studio 3.5.3 oraz Intellij IDEA 2019.3.1. Dodatkowo, wymagane jest posiadanie zainstalowanej co najmniej Javy 8. Pliki biblioteki OpenCV należy pobrać ze strony „sourceforge.net/projects/opencvlibrary/files/4.1.1/" – plik „OpenCV 4.1.1.zip" oraz „opencv-4.1.1-android-sdk.zip".

Kolejne kroki konfiguracji środowiska dla programu serwera:

- Zainstaluj bibliotekę ant używając komendy: `sudo apt-get install ant`
- Rozpakuj plik „OpenCV 4.1.1.zip"
- W rozpakowanym folderze, stwórz nowy o nazwie np. `build`
- W nowym folderze uruchom komendę: `cmake -D BUILD_SHARED_LIBS=OFF ..` Wyświetlenie na końcu `ant: NO` lub `JNI: NO` informuje o niepowodzeniu operacji ze względu na niepoprawną instalację biblioteki ant lub języka Java
- W przypadku powodzenia poprzedniej operacji, uruchom komendę: `make -j8`
- Zaimportuj projekt znajdujący się w folderze `REST-Server` przy użyciu programu Intellij IDEA
- Przejdź do zakładki `File -> Project Structure -> Libraries`, gdzie należy dodać bibliotekę nazywając ją `opencv-411` oraz wskazując na ścieżkę `../{stworzony folder}/bin/opencv-411.jar`. W przypadku istnienia pola o tej nazwie, należy tylko zmienić widniejącą ścieżkę.



Aplikacja mobilna powinna działać od razu po zaimportowaniu programu znajdującego się w folderze `AndroidCV`, jednak ze względu na częste komplikacje, przedstawiony został schemat dodawania biblioteki do projektu:

- Po zaimportowaniu projektu, rozpakuj plik „opencv-4.1.1-android-sdk.zip".
- Zaimportuj moduł OpenCV w aplikacji AndroidStudio przechodząc do zakładki `File -> New -> Import Module...`, gdzie należy wybrać ścieżkę wskazującą na folder `../opencv-4.1.1-android-sdk/sdk/java`
- W zakładce `File -> Project Structure -> Dependencies` dla pola app, dodaj dodany moduł jako `Module dependency`
- Jeśli pod ścieżką `../AndroidCV/app/src/main` nie znajduje się folder o nazwie `jniLibs`, należy go dodać i umieścić w nim pliki spod ścieżki `../opencv-4.1.1-android-sdk/OpenCV-android-sdk/sdk/native/libs`

W przypadku tworzenia nowego projektu z biblioteką OpenCV, należy pamiętać, że przed użyciem jakiejkolwiek operacji z biblioteki, trzeba ją najpierw załadować. Odbywa się to poprzez uruchomienie kodu `System.loadLibrary(Core.NATIVE_LIBRARY_NAME)`. Wystarczy to zrobić raz, przy starcie aplikacji.