Raport w ramach projektu realizowanego przez: Snowdog sp. z o.o. pn. „Opracowanie innowacyjnej technologii oraz realizacja prototypu systemu analizującego obraz utrwalony techniką cyfrową przy pomocy smartfonu”

w ramach
Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020,
Oś Priorytetowa 1. Innowacyjna i konkurencyjna gospodarka,
Działanie 1.2. „Wzmocnienie potencjału innowacyjnego przedsiębiorstw Wielkopolski”.

Nr zadania	2
Poziom TRL	TRL in: 5 TRL out: 6
Numer projektu	RPWP.01.02.00-30-0175/17
Autorzy raportu	Radosław Zaworski, Bartłomiej Kwiatkowski
Czas pracy i zadania wykonywane przez pracowników na rzecz projektu	2.1. Projekt i implementacja prototypów (kwiecień - czerwiec 2019) 2.2. Dalsza rozbudowa i utrzymanie danych uczących (kwiecień - czerwiec 2019) 2.3. Testowanie i optymalizacja prototypów rozwiązania (czerwiec 2019)
Kamienie milowe:	Zakładany efekt końcowy (kamień milowy): 1. Działający prototyp systemu, tj. komponentów wymienionych w punkcie „Założony sposób rozwiązania”. 2. Implementacja zoptymalizowanego modułu analizy obrazu. 3. Uzupełniona baza danych uczących. 4. Wskaźnik rezultatu: Liczba architektur sieci o zwiększonej trafności rozpoznawania: 2 5. Wskaźnik rezultatu: Liczba architektur sieci o zwiększonej prędkości działania: 2
Zgodność z Planem w zakresie prac B+R (TAK/NIE)	TAK

1 Przedmiot badania

Badania na tym etapie skupiały się głównie na poszukiwaniu metod poprawienia jakości oraz szybkości działania sieci. W przypadku architektur A zaniechano dalszych badań nad VGG16, ze względu na

odstające wyniki od dwóch pozostałych modeli: InceptionV3 oraz ResNet50. W celu poprawienia jakości klasyfikacji przetestowano dwie techniki:

- Data Augmentation - powiększanie zbioru danych przez transformacje afiniczne, transformacje perspektywy, zmiany kontrastu, odcienia, nasycenia, przycinanie, rozmycia itd.
- Pogłębienie Fine Tuningu - dotychczas fine tuning został prowadzony do głębokości pierwszego bloku konwolucyjnego, podczas badań w tym etapie projektu testowano wpływ odmrożenia wag w kolejnych blokach.

Do przyspieszenia działania architektur A wykorzystano również dwie metody:

- Przycinanie filtrów (ang. filter pruning) - proces ten polega na usuwaniu najmniej istotnych filtrów z sieci w celu zmniejszenia ilości parametrów.
- Wykorzystanie biblioteki TensorFlow Lite na urządzeniach mobilnych.

Ponieważ jakość detekcji obiektów osiągnięta z użyciem architektury YOLO była bardzo satysfakcjonująca przeanalizowano ją pod względem możliwości przyspieszenia procesu, po czym wytypowano 3 potencjalne ścieżki działania:

- Zredukowanie ilości tzw. anchor boxes - anchor boxes są wykorzystywane w architekturach do detekcji obiektów w celu poprawienia jakości wyszukiwania wielu obiektów, w szczególności tych znajdujących się blisko siebie. Ponieważ w przypadku problemu rekomendacji produktu interesuje nas tylko jeden obiekt na zdjęciu, można pozwolić sobie na redukcję nawet do pojedynczego anchor box.
- Zmniejszenie rozmiarów obrazu wejściowego - zmniejszenie wymiarowości istotnie zmniejsza zapotrzebowanie na moc obliczeniową.
- Wykorzystanie starszej wersji architektury (v2) - najbardziej istotne zmiany w najnowszej wersji YOLO w stosunku do poprzednika wpływają na jakość detekcji, jednak odbywa się to pewnym kosztem mocy obliczeniowej.

Prowadzono również dalszą analizę zasadności stosowania architektur C służących do segmentacji obrazu. W tym celu powiększono zbiór danych o najnowszą wersje Open Images Dataset w której znalazły się dane treningowe do segmentacji interesujących nas klas obiektów.

Rozwinięta została również aplikacja mobilna oraz przygotowano prototyp ostatniego komponentu założonego sposobu rozwiązania - modułu rekomendującego produkty na podstawie danych pochodzących z modułu analizy obrazu.

2 Wynik działania

2.1 Badania architektur A

2.1.1 Poprawa jakości działania

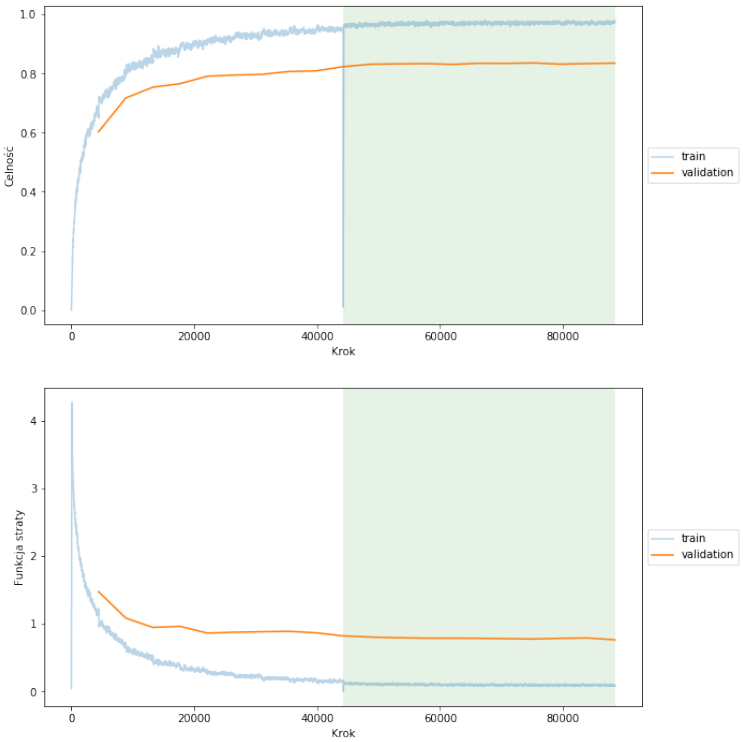
W pierwszej kolejności został zbadany wpływ powiększenia datasetu przez tzw. data augmentation. W tym celu wykorzystano bibliotekę imgaug która wspiera szeroki zakres technik manipulacji obrazu oraz w prosty sposób pozwala aplikować je w losowy sposób na oryginalnym zbiorze. Zastosowano następujące przekształcenia:

- losowe przycięcia 0% do 10% wymiarów obrazu,
- manipulacja kontrastem w zakresie 75% do 150%,
- manipulacja jasnością w zakresie 80% do120%, z tego 20% obrazów niezależnie każdy kanał co wpływa na barwę,
- skalowanie 80% do 120%,
- przesunięcie w osi pionowej i poziomej w zakresie -20% do 20% wymiarów obrazu,
- obrót -25°do 25°,
- pochylenie -8°do 8°.

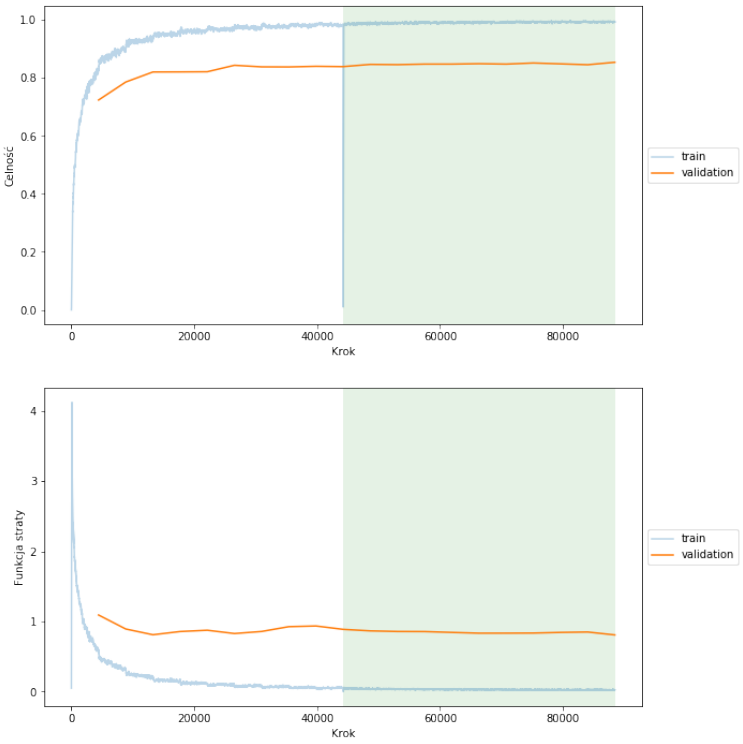
Każda z tych technik została zaaplikowana do zdjęć w sposób losowy dziesięciokrotnie, przez co ilość obrazów w zbiorze danych treningowych powiększyła się jedenastokrotnie. Na tym etapie istnieje ryzyko wycieku danych (ang. data leakage), tzn. przedostania się danych treningowych do zbioru walidacyjnego co zawyża wyniki modelu na tym drugim zbiorze. Jest to spowodowane tym, że mimo iż w wyniku wzbogacania powstaje nowy obraz to jest on nadal bardzo zbliżony do oryginalnego. Mając to na uwadze w pierwszej kolejności rozdzielono dane na odpowiednie podzbiory, a następnie przeprowadzano wzbogacenie tylko na zdjęciach treningowych.

Proces nauki analogiczny był do dostrajania z etapu pierwszego projektu i z tym będzie porównywany. Trening można prześledzić na wykresach: 1, 2. Porównanie wyników można zobaczyć w tabeli 1. Jasnozielony obszar wykresu oznacza drugi etap nauki - dostrajanie głębszych warst bloków konwolucyjnych. W obu przypadkach udało się zredukować błąd o 20% – 27% co jest zadowalającym wynikiem.

Następnym krokiem było pogłębienie fine-tuningu o kolejny blok konwolucyjny. Ten eksperyment jednak nie dawał zauważalnej poprawy wyników znacznie wydłużając proces nauki, więc został na tą chwilę zaniechany. W przyszłości można rozważyć badanie nauki całego modelu z parametrem learning rate ustawionym na bardzo małą wartość, jednak w przypadku zadania rekomendacji dalsze specjalizowanie modelu może przynieść odwrotny skutek do zamierzonego - tracąc umiejętność do generalizowania pogorszyć jakość proponowanych produktów.



Rysunek 1: Proces nauki InceptionV3 na wzbogaconym zbiorze danych.



Rysunek 2: Proces nauki ResNet50 na wzbogaconym zbiorze danych.

Tabela 1: Porównanie wyników

	Celność	
	InceptionV3	ResNet50
Bazowy zbiór	78.4%	79.8%
Powiększony zbiór	82.6%	85.4%

2.1.2 Poprawa szybkości działania

Do badania możliwości i wpływu usuwania filtrów wybrano architekturę ResNet. W literaturze dotychczas pojawiło się już wiele metod określania istotności filtrów, które dzielą się na te które wykorzystują dane (data-driven) oraz te które opierają się na właściwościach numerycznych wag [4][5]. Do badania użyto metody opartej na algorytmie k-średnich, która należy do tej drugiej grupy. Technika ta polega na analizie skupień filtrów - filtry które leżą najbliżej centroidy są pozostawiane, a te leżące dalej - usuwane [6]. Została zaprojektowana następująca strategia nauki:

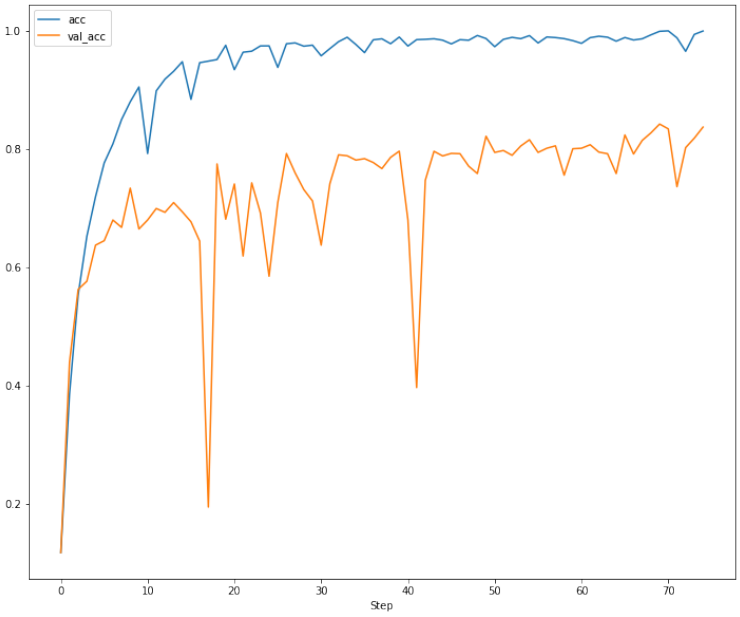
1. transfer learning - 10 epok;
2. przycinanie filtrów (ang. filter pruning) - maksymalnie 10 iteracji, 5 epok dostrajania;
3. dostrojenie w postaci powtórzenia kroku pierwszego - 20 epok.

Parametry dobrane zostały następująco:

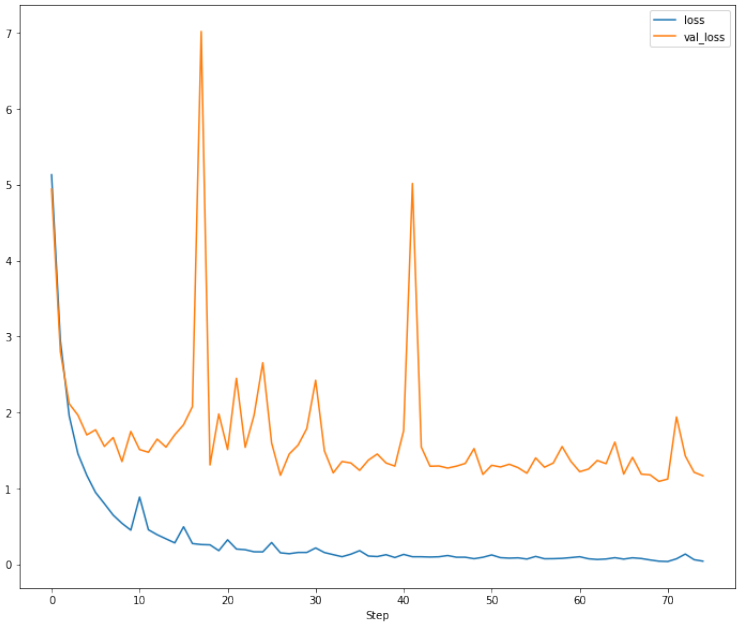
- learning rate: 0.001,
- rho: 0.99,
- współczynnik przycinania (ang. pruning factor): 0.9.

Współczynnik przycinania (ang. pruning factor) jest parametrem określającym tempo usuwania filtrów, mieści się w zakresie od 0 do 1, gdzie 0 oznacza usunięcie wszystkich filtrów w jednej

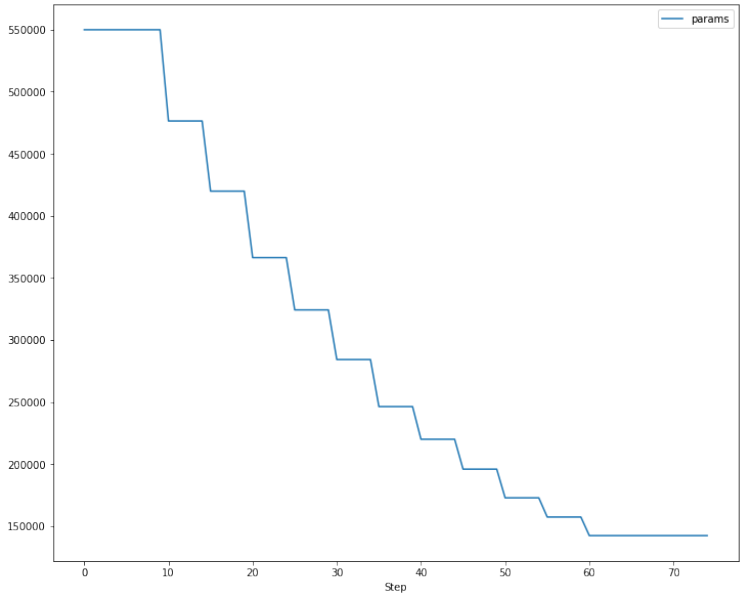
iteracji, a 1 nieusuwanie żadnych filtrów. Na wykresie 3 widoczny jest standardowy wykres nauki, z charakterystycznymi spadkami celności, wyraźnie widocznymi w szczególności na zbiorze treningowym. Są one spowodowane usunięciem części filtrów począwszy od 10 epoki, co każde kolejne 5 epok aż do epoki 60, po której następuje końcowy etap dostrajania. Na wykresie 5 widoczny jest spadek liczby parametrów z około 500000 do 150000. Mniejsza liczba filtrów w sieci oznacza mniejszą złożoność obliczeniową co powinno pozytywnie wpłynąć na czas klasyfikacji. Jak widać na wykresach 6, 7 średni czas klasyfikacji został zredukowany do 6.6ms czyli został zredukowany o połowę względem oryginalnego modelu.



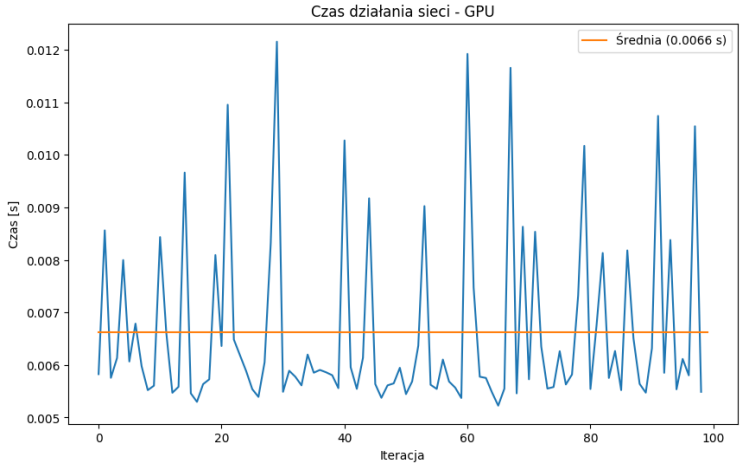
Rysunek 3: Nauka modelu ResNet z usuwaniem filtrów - celność.



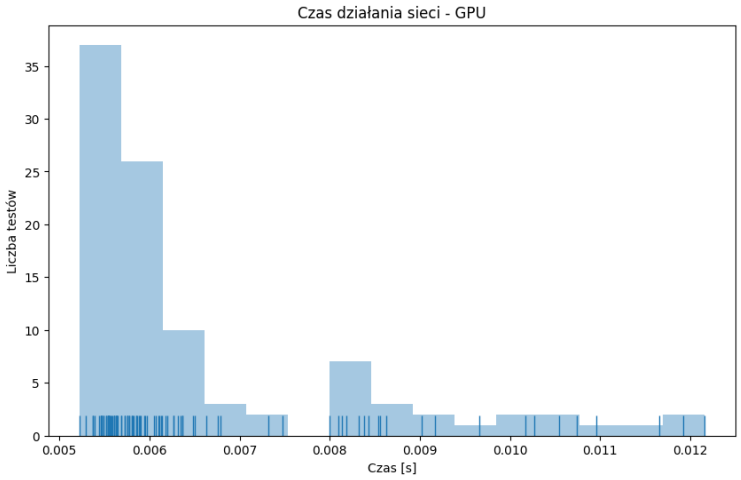
Rysunek 4: Nauka modelu ResNet z usuwaniem filtrów - funkcja straty.



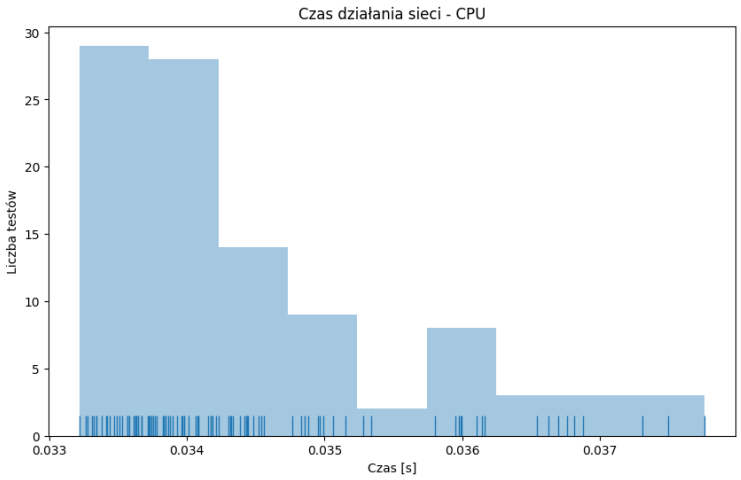
Rysunek 5: Redukcja liczby parametrów modelu ResNet podczas nauki.



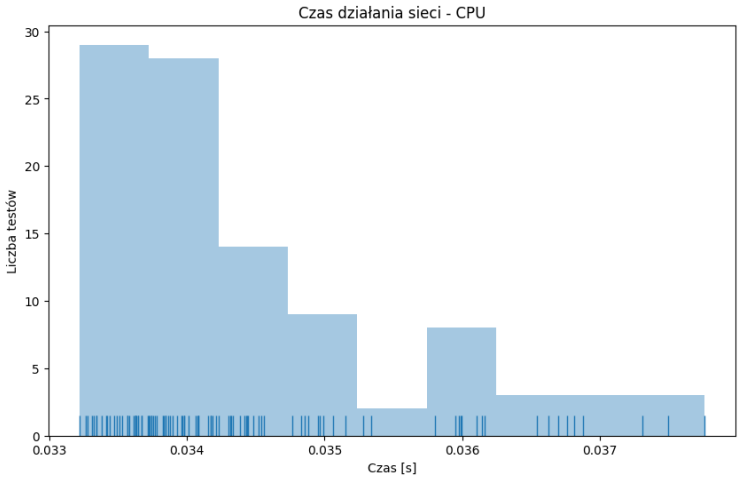
Rysunek 6: Czas klasyfikacji zoptymalizowanego modelu na procesorze graficznym.



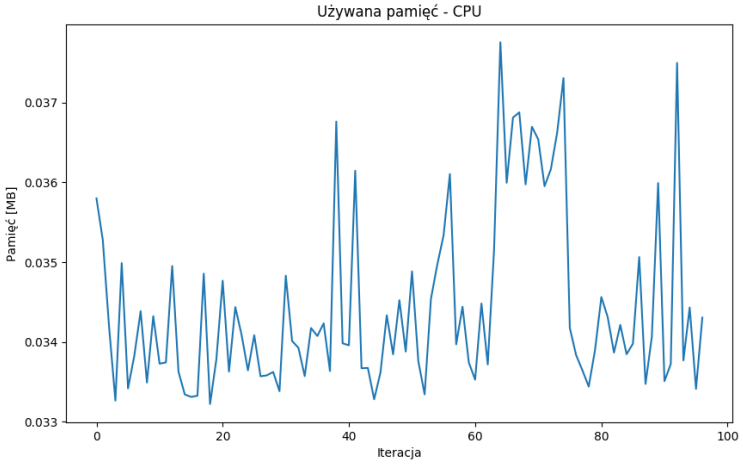
Rysunek 7: Czas klasyfikacji zoptymalizowanego modelu na procesorze graficznym- histogram.



Rysunek 8: Czas klasyfikacji zoptymalizowanego modelu na CPU.



Rysunek 9: Czas klasyfikacji zoptymalizowanego modelu na CPU - histogram.

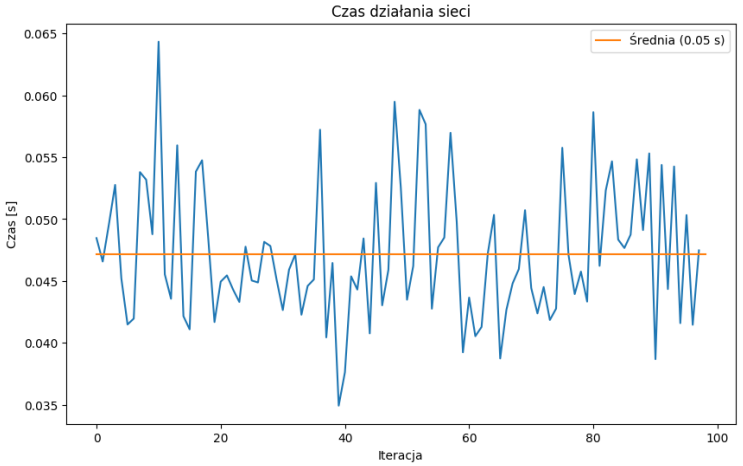


Rysunek 10: Zużycie pamięci przez model.

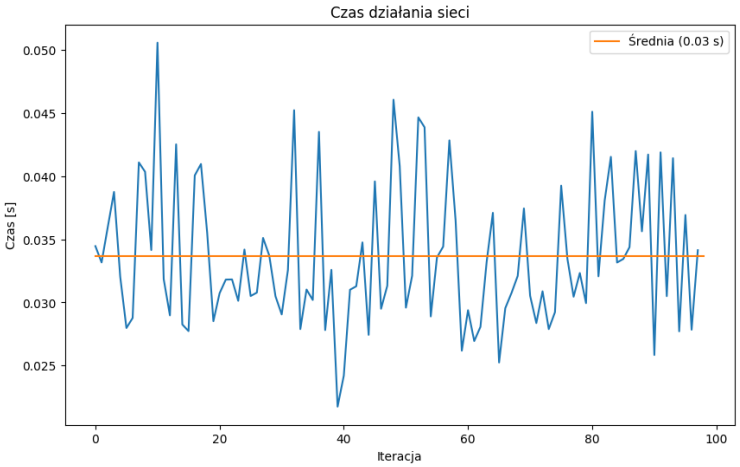
2.2 Badania architektury B

2.2.1 Poprawa szybkości działania

Przebadano odchudzoną wersję modelu YOLOv3 ze zmniejszoną rozdzielczością obrazu wejściowego z 320x320 na 256x256 pikseli oraz zredukowaną ilością anchor box z 9 do 6. Wykres 11 oraz 12 przedstawiają odpowiednio czas klasyfikacji na CPU i GPU. W przypadku CPU czas klasyfikacji został znacznie zredukowany z 0.32s do 0.05, w przypadku procesora graficznego różnica nie jest aż tak odczuwalna i została zmniejszona z 0.05s do 0.03s.



Rysunek 11: Czas klasyfikacji zoptymalizowanego modelu na CPU



Rysunek 12: Czas klasyfikacji zoptymalizowanego modelu na procesorze graficznym.

2.3 Aplikacja mobilna

Na potrzeby badań stworzono natywną aplikację działającą na systemie operacyjnym Android. Zamiast TensorFlow Mobile wykorzystano tym razem bibliotekę TensorFlow Lite by usprawnić działanie modelu na urządzeniu mobilnym. Ponieważ ta biblioteka jest zoptymalizowana pod urządzenia mobilne, jej format pliku z modelem różni się od formatu używanego w standardowej bibliotece Tensorflow. Model ResNet — w celu zastosowania go w aplikacji mobilnej — przekonwertowano do formatu zgodnego z TensorFlow Lite korzystając z narzędzia TensorFlow Lite converter [3]. Przy konwersji zwrócono uwagę na zachowanie takiego samego rozmiaru wejścia jak w oryginalnej sieci. Do obsługi kamery wykorzystano bibliotekę CameraX, która pozwoliła na wyświetlenie podglądu i dostęp do aktualnego obrazu w celach przeprowadzenia analizy. Ponieważ surowy obraz jest dostępny w formacie YUV, przed analizą wymagana była konwersja do formatu ARGB oraz dostosowanie rozmiaru odpowiednio do rozmiaru wejścia wykorzystywanej sieci. Aplikacja wyświetla posortowaną według prawdopodobieństwa listę zawierającą rozpoznane marki. Zrzut ekranu 13 prezentuje widok aplikacji.

Na wykresach 14, 15, 16, 17 przedstawiono wyniki testów szybkośći klasyfikacji. Zastosowanie TensorFlow Lite pozwoliło obniżyć czas klasyfikacji średnio o 13%. Trzeba jednak wziąć pod uwagę, że zmiana biblioteki istotnie wpłynęła na sposób pomiaru czasu. W przypadku TensorFlow Mobile możliwy jest dostęp do szczegółowych statystyk, dlatego ta biblioteka została wykorzystana podczas pierwszego etapu projektu do porównywania ze sobą różnych architektur. Z kolei do zmierzenia wydajności w TensorFlow Lite należy użyć specjalnego oprogramowania testującego wydajność modelu [1].

2.4 Moduł rekomendujący produkty

Przygotowano prototyp modułu rekomendującego w postaci prostego API oraz dodatkowo dla ułatwienia prezentacji demo w postaci strony internetowej przypominającej sklep online. Moduł rekomendacji wykorzystuje tensory wyjściowe z ostanich bloków konwolucyjnych modeli: VGG16, InceptionV3 lub ResNet. Te wyekstrahowane tensory intepretuje się jako opis cech obrazu i umieszczone zostają w strukturze drzewa k-wymiarowego. Taka struktura danych umożliwia wykonywanie zapytania gdzie wejściem jest pewien wektor, a wyjściem indeksy umieszczonych w niej wcześniej wektorów które znajdują się najmniejszej odległości euklidesowej od wektora wejściowego. Dzięki temu uzyskujemy identyfikatory produktów podobnych do produktu pochodzącego z zapytania. Na zrzucie ekranu 18 widać przykładową stronę produktu wraz ze zdjęciem, a poniżej 4 rekomendowane na podstawie wizualnego podobieństwa inne produkty. Pierwszy z nich jest tym samym zegarkiem co właśnie przeglądany, wynika to z tego, że najbliżej produktu z zapytania jest on sam. Przez odflitrowanie wyników do których odległość była równa 0 można pozbyć się tego zachowania jednak tutaj nie zastosowano go w celach deweloperskich, dzięki czemu widać czy zapytania działają prawidłowo.

2.5 Aplikacja do ręcznej klasyfikacji i nadawania atrybutów danym

W celu usprawnienia ręcznej klasyfikacji i nadawania atrybutów danym została zaimplementowana aplikacja webowa z wykorzystaniem frameworka Django w języku Python. Jako system zarządzania relacyjnymi bazami danych wykorzystywany jest w niej PostreSQL w wersji produkcyjnej oraz SQLite w wersji deweloperskiej. Aplikacja zbudowana jest z 2 modułów oraz panelu administracyjnego. Pierwszym z modułów jest komponent odpowiadający za logowanie użytkownika, pozwala to śledzić kto wprowadza zmiany do bazy danych. Drugim modułem jest klasyfikator umożliwiający przede wszystkim wygodną klasyfikację danych, nadawanie atrybutów oraz oznaczenie zdjęcia jako nieprawidłowe np. takiego na którym nie znajduje się produkt lub jest praktycznie niewidoczny. W tym module dodatkowo zaimplementowana jest strona główna z krótkim opisem działania systemu, historia wyborów użytkownika z możliwością dokonania korekty oraz stroną do wgrywania nowych danych, dostępną gdy zalogowany użytkownik ma prawa administratora. Panel administracyjny służy do zarządzania użytkownikami oraz dostępnym wyborem klas i atrybutów danych.
Dodatkowo narzędzie zostało przygotowane do potencjalnego wykorzystania w tzw. crowdsourcingu [2] i posiada ranking użytkowników, dzięki którego łatwo można śledzić ilość opisanych zdjęć przez poszczególnych użytkowników.

3 Wnioski z przeprowadzonego badania

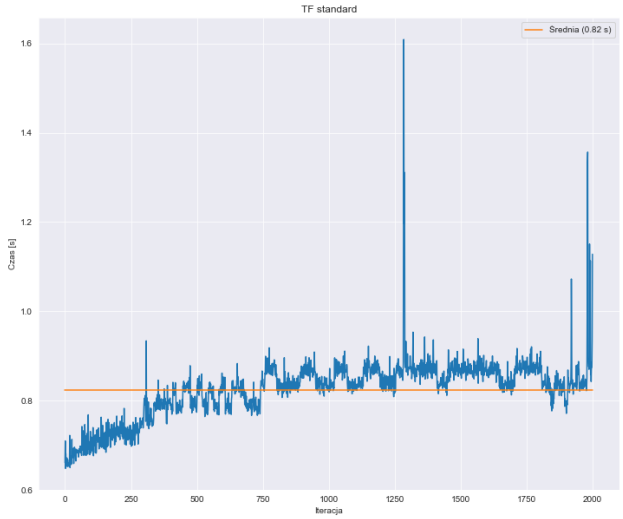
Podczas prac nad drugim zadaniem projektu zostały zaprojektowane oraz zaimplementowane pierwsze prototypy kompontentów systemu zgodnie z założonym sposobem rozwiązania. Aplikacja mobilna została rozwnięta w celu wykorzystania biblioteki TensorFlow Lite co pozwoliło poprawić wydajność jej działania. Moduł analizy obrazu został zoptymalizowany przez powiększenie zbioru danych przez tzw. data augmentation dzięki czemu została poprawiona trafność rozpoznawania architektur InceptionV3 oraz ResNet50. W przypadku architektury ResNet50 usprawniono również prędkość działania poprzez usunięcie redundantnych filtrów. Drugą architekturą która została zoptymalizowana pod względem prędkości działania jest YOLOv3, gdzie ze względu na bardzo dobre rezultaty zdecydowano się obniżyć rozdzielczość obrazu wejściowego i zredukowano ilość tzw. anchor box. Poza uzupełnioną bazą danych uczących przez wzbogacenie dodane zostały również dane segmentacyjne z najnowszej wersji Open Images Dataset. Dalszym krokiem w rozwijaniu tego modułu może być integracja architektur A i B oraz dostrojenie architektur A do zadania rekomendacji. Wykonanie prototypu modułu rekomendującego produkty pozowliło rozpocząć testy oraz zbieranie opinii, co ułatwi dalszy rozwój systemu. W 3 zadaniu będą prowadzone badania nad używaniem innych danych, takich jak dane historyczne użytkownika w celu poprawienia rekomendacji.

4 Zgodność realizacji działania z założeniami z Planu B+R

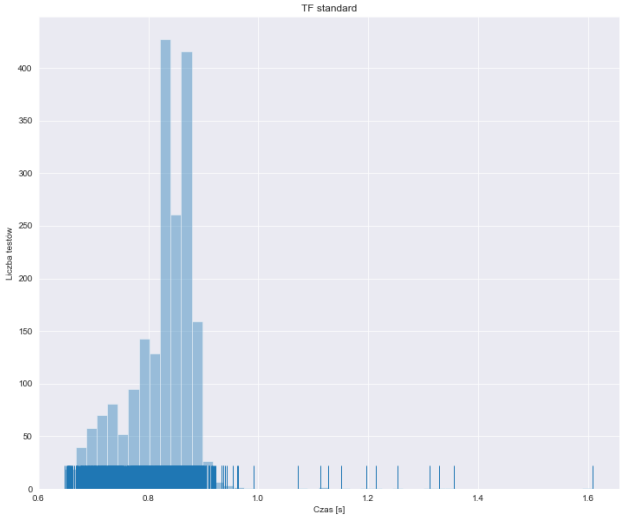
Powyższe badania oraz inne działania spełniły założenia Planu B+R, zrealizowano założenia kamieni milowych:

- Działający prototyp systemu, tj. komponentów wymienionych w punkcie 'Założony sposób rozwiązania'.
- Implementacja zoptymalizowanego modułu analizy obrazu.
- Uzupełniona baza danych uczących.
- Wskaźnik rezultatu: Liczba architektur sieci o zwiększonej trafności rozpoznawania: 2
- Wskaźnik rezultatu: Liczba architektur sieci o zwiększonej prędkości działania: 2

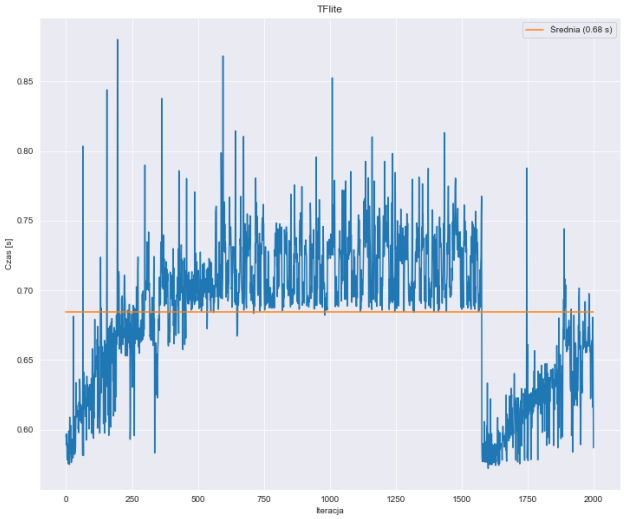
Rysunek 13: Zrzut ekranu aplikacji mobilnej.



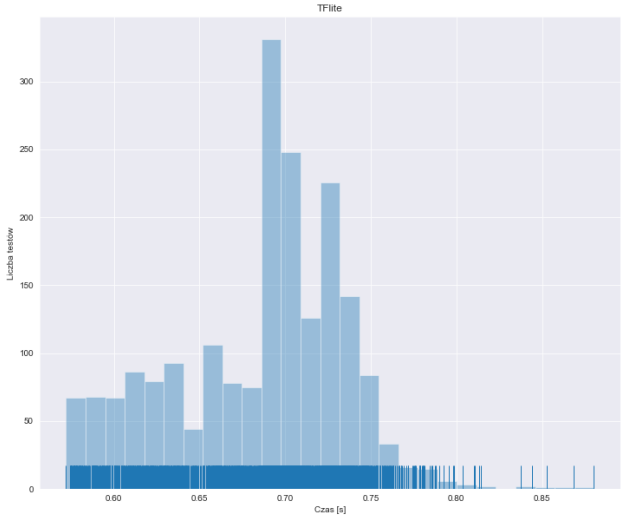
Rysunek 14: Czas klasyfikacji z wykorzystaniem TensorFlow Mobile.



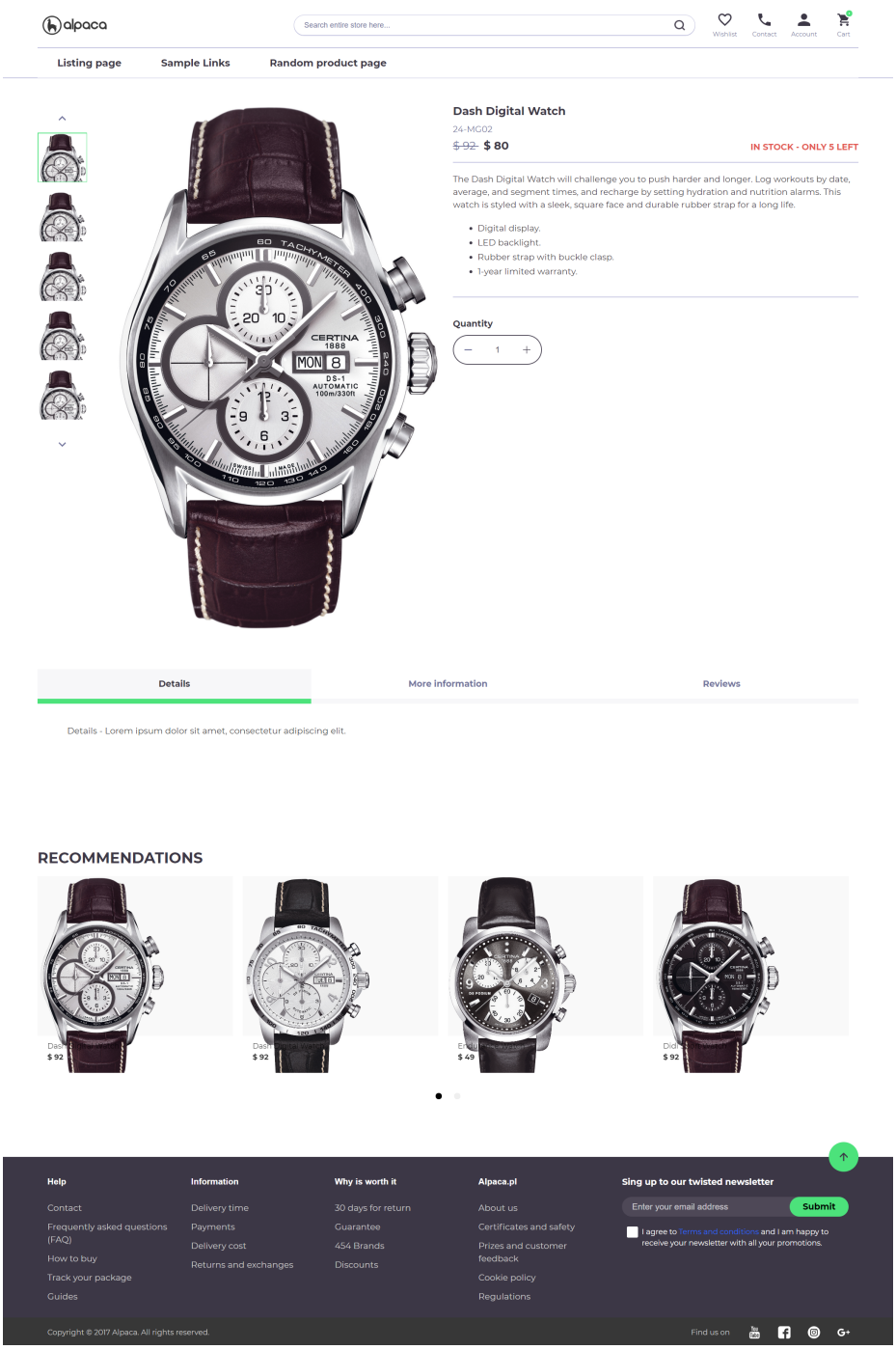
Rysunek 15: Histogram czasów klasyfikacji z wykorzystaniem TensorFlow Mobile.



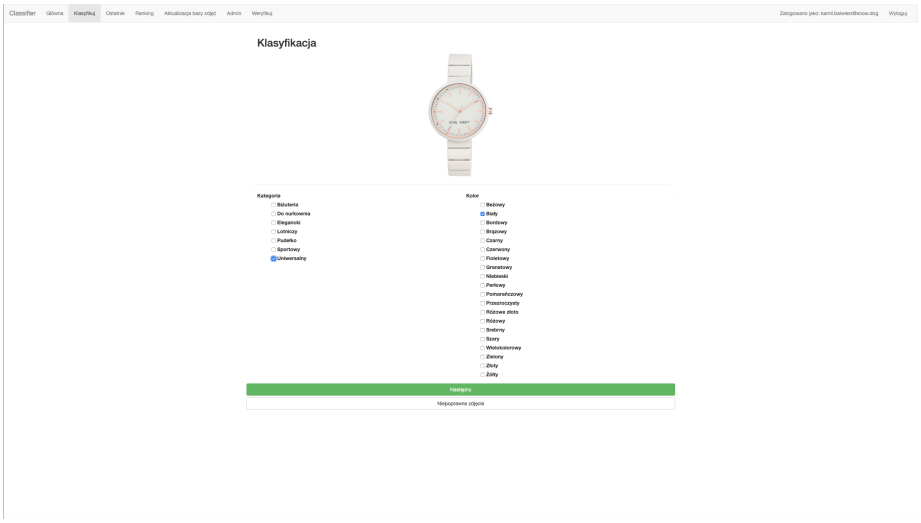
Rysunek 16: Czas klasyfikacji z wykorzystaniem TensorFlow Lite.



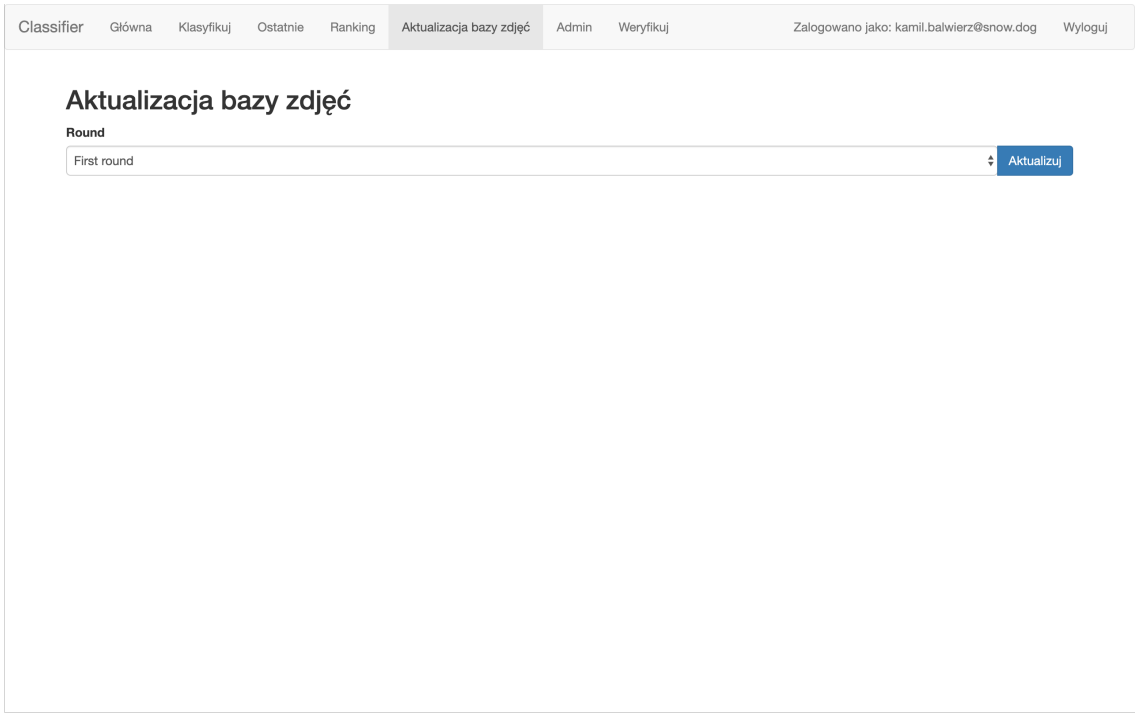
Rysunek 17: Histogram czasów klasyfikacji z wykorzystaniem TensorFlow Lite.



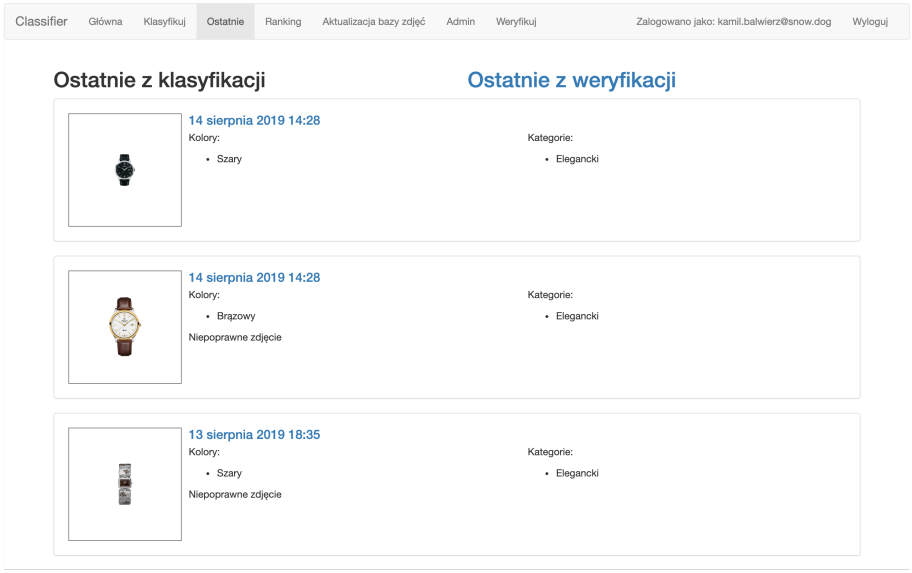
Rysunek 18: Zrzut ekranu demo modułu rekomendacyjnego.



Rysunek 19: Zrzut ekranu aplikacji do klasyfikacji i nadawania atrybutów danym - główny widok klasyfikacji



Rysunek 20: Zrzut ekranu aplikacji do klasyfikacji i nadawania atrybutów danym - widok aktualizacji



Rysunek 21: Zrzut ekranu aplikacji do klasyfikacji i nadawania atrybutów danym - widok historii

5 Bibliografia

Literatura

[1] Android performance benchmarks. `https://www.tensorflow.org/lite/performance/benchmarks`.

[2] Crowdsourcing. `https://pl.wikipedia.org/wiki/Crowdsourcing`.

[3] Tensorflow lite converter. `https://www.tensorflow.org/lite/convert`.

[4] Hanson, S.J., Pratt, L. Advances in neural information processing systems 1. pages 177–185, 1989.

[5] Hassibi, B., Stork, D.G. Second order derivatives for network pruning: Optimal brain surgeon. pages 164–171, 1993.

[6] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv e-prints*, page arXiv:1607.03250, Jul 2016.

Poznań, 28.06.2019r.

(miejscowość, data sporządzenia raportu, podpis autorów)

Poznań, 28.06.2019r.

(miejscowość, data sporządzenia raportu, podpis pracodawcy)