JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 36 (2025) 342–349
Received: 16 May 2025
Accepted: 1 July 2025

Comparison of chosen image classification methods on Android

Porównanie wybranych metod klasyfikacji obrazów na platformie Android

Mariusz Krzysztof Zapalski*, Patryk Konrad Żabczyński, Paweł Powroźnik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The authors compared three lightweight convolutional networks (MobileNet-V1, EfficientNet-Lite0 and ResNet-50) for image classification on Android smartphones using TensorFlow Lite and a multithreaded CPU. They measured inference time, CPU load and memory usage across various devices. EfficientNet-Lite0 proved the best compromise - providing high accuracy, short and consistent processing times and moderate resource demands. MobileNet-V1 was the fastest but less precise, while ResNet-50 achieved the highest accuracy at the expense of speed and memory. In practice, EfficientNet-Lite0 is recommended, and further research into optimizations such as quantization, pruning and adaptive frame sampling is advised.

Keywords: convolutional neural networks; image classification; tensorflow lite

Streszczenie

Autorzy zestawili trzy lekkie konwolucyjne sieci (MobileNet-V1, EfficientNet-Lite0 i ResNet-50) pod kątem klasyfikacji obrazów na smartfonach z Androidem, wykorzystując TensorFlow Lite i wielowątkowy CPU. Przeprowadzono pomiary czasu inferencji, obciążenia procesora i zużycia pamięci na różnych modelach urządzeń. EfficientNet-Lite0 okazał się najlepszym kompromisem - zapewniając wysoką dokładność, krótki i równy czas przetwarzania oraz umiarkowane zużycie zasobów. MobileNet-V1 działa najszybciej, lecz kosztem precyzji, a ResNet-50 osiąga topową trafność, ale wolniej i przy dużym zapotrzebowaniu na pamięć. W praktyce rekomenduje się EfficientNet-Lite0, a także dalsze badania nad optymalizacjami, takimi jak kwantyzacja, przycinanie sieci czy adaptacyjne próbkowanie klatek.

Słowa kluczowe: konwolucyjne sieci neuronowe; klasyfikacja obrazów; tensorflow lite

*Corresponding author
Email address: **mariusz.zapalski@pollub.edu.pl** (M. K. Zapalski)

1. Wstęp

W ostatnich latach, zastosowanie głębokich sieci konwolucyjnych (CNN) do przetwarzania obrazów bezpośrednio na urządzeniach mobilnych („on-device") stało się kluczowym obszarem badań w dziedzinie informatyki [1, 2]. Rosnące oczekiwania użytkowników dotyczące szybkości działania aplikacji oraz zwiększona potrzeba ochrony prywatności danych skłaniają do unikania przesyłania obrazów do przetwarzania w chmurze. W odpowiedzi na te trendy, badacze koncentrują się na projektowaniu lekkich, wydajnych modeli CNN, zdolnych do efektywnego działania przy ograniczonych zasobach obliczeniowych smartfonów.

Platforma TensorFlow Lite (TFLite) odgrywa istotną rolę w tym procesie, oferując narzędzia do optymalizacji (np. kwantyzacja po treningu) i uruchamiania modeli uczenia maszynowego na urządzeniach mobilnych, w tym na platformie Android [3, 4]. Umożliwia ona inferencję, czyli wykorzystanie wytrenowanego modelu do klasyfikacji nowych danych, bezpośrednio na urządzeniu, często z możliwością wykorzystania różnych akceleratorów sprzętowych, choć niniejsze badanie koncentruje się na podstawowej inferencji z wykorzystaniem CPU, aby wyeliminować zmienność związaną z różnymi implementacjami wsparcia sprzętowego [14].

Istnieje wiele architektur CNN zoptymalizowanych pod kątem zastosowań mobilnych, takich jak MobileNet-V1, EfficientNet-Lite0 czy ResNet-50. Dotychczasowe prace i benchmarki [2, 4, 5] wskazują, że modele te oferują różny, teoretyczny kompromis między dokładnością klasyfikacji, liczbą parametrów a złożonością obliczeniową (np. mierzoną w FLOPs). Naszą motywacją do podjęcia tych badań była jednak luka w dostępnej wiedzy: brakowało systematycznej, empirycznej analizy porównującej te popularne architektury w warunkach jak najbardziej zbliżonych do rzeczywistego zastosowania – tj. podczas przetwarzania ciągłego strumienia wideo „na żywo" (bez buforowania klatek), bezpośrednio na CPU smartfonów klasy średniej przy użyciu TFLite. Uważamy, że zrozumienie rzeczywistej wydajności (mierzonej czasem inferencji, stabilnością tego czasu oraz zużyciem zasobów CPU i pamięci) w takim dynamicznym scenariuszu jest kluczowe dla deweloperów aplikacji mobilnych, którzy muszą dokonać świadomego wyboru modelu, balansując między jakością predykcji a płynnością działania aplikacji w czasie rzeczywistym.

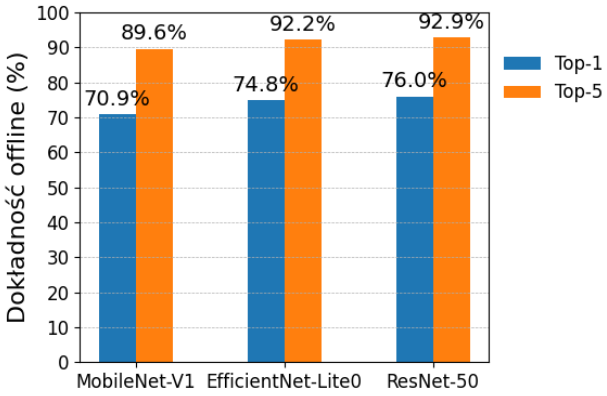
Głównym osiągnięciem niniejszej pracy jest dostarczenie właśnie takiej empirycznej analizy i ilościowych wyników porównawczych dla modeli MobileNet-V1, EfficientNet-Lite0 i ResNet-50, działających na platformie Android z TFLite i wielowątkową inferencją na CPU. Przeprowadzone eksperymenty na różnych urządzeniach pozwoliły na zebranie konkretnych danych dotyczących nie tylko średniego czasu przetwarzania klatki wideo, ale także stabilności tego czasu (wariancji opóźnień) oraz realnego obciążenia procesora i zużycia pamięci RAM. Na podstawie tych wyników, badanie wskazuje praktyczne rekomendacje, identyfikując architekturę EfficientNet-

Lite0 jako oferującą najbardziej zrównoważony kompromis między skutecznością klasyfikacji a wydajnością w badanym, realistycznym scenariuszu mobilnym. Wypełniamy tym samym zidentyfikowaną lukę, oferując deweloperom cenne wskazówki przy wyborze modelu do zastosowań wymagających klasyfikacji obrazów w czasie rzeczywistym na urządzeniach mobilnych.

W związku z powyższym, celem niniejszych badań jest empiryczne zweryfikowanie, który z porównywanych modeli konwolucyjnych sieci neuronowych – MobileNet-V1, EfficientNet-Lite0 czy ResNet-50 – oferuje najlepszy kompromis pomiędzy wysoką skutecznością klasyfikacji obrazów a niskim średnim czasem inferencji podczas przetwarzania strumienia wideo w aplikacji na platformie Android, wykorzystując bibliotekę TensorFlow Lite i wielowątkową inferencję na CPU. Postawiono przy tym następującą hipotezę badawczą: (H) spośród trzech analizowanych architektur sieci neuronowych, model EfficientNet-Lite0 zapewni najbardziej optymalny kompromis, łącząc wysoką dokładność klasyfikacji obrazów z najniższym średnim czasem przetwarzania pojedynczej klatki strumienia wideo pochodzącego z kamery w czasie rzeczywistym na platformie Android. W kolejnych rozdziałach przedstawiono przegląd literatury, opis zastosowanych technologii, metodykę przeprowadzonych badań, analizę wyników oraz wnioski końcowe.

2. Przegląd literatury

Rozwój mobilnego przetwarzania obrazu bazuje na fundamentach opracowanych w klasycznych pracach nad głębokimi sieciami konwolucyjnymi. W przełomowej publikacji zaprezentowano sieć AlexNet, ośmiowarstwowy model z 60 mln parametrów, wykorzystujący funkcję aktywacji ReLU (Rectified Linear Unit, liniowa jednostka prostująca) oraz technikę Dropout (losowe wyłączanie neuronów), co pozwoliło osiągnąć 62,5% dokładności Top-1 (dokładność jednej najlepszej prognozy) na zbiorze ImageNet dzięki przyspieszeniu treningu na GPU (Graphics Processing Unit, procesor graficzny) [1]. Dokument [6] definiuje kluczowe metryki oceny modeli: Precision (precyzja), Recall (czułość), mAP (mean Average Precision, średnia precyzja) oraz opisuje procesy trenowania i inferencji, stanowiąc teoretyczną podstawę dla wszystkich późniejszych eksperymentów. W odpowiedzi na konieczność działania „on-device" na urządzeniach o ograniczonych zasobach opracowano lekkie architektury. W kontekście mobilnego przetwarzania obrazu termin „lekki model" odnosi się do konwolucyjnej sieci neuronowej, której liczba parametrów, złożoność obliczeniowa oraz rozmiar binarny zostały zredukowane do poziomu umożliwiającego efektywną inferencję „on-device" na smartfonach. W praktyce za modele lekkie uznaje się te zawierające zwykle poniżej 5 mln parametrów, wymagające setek milionów operacji MAC/FLOPs na pojedyncze wejście oraz zajmujące kilkanaście-kilkadziesiąt megabajtów. W pracy poświęconej MobileNet-V1 autorzy pokazali, że dzięki głęboko separowalnym konwolucjom możliwe jest zredukowanie liczby parametrów do ~4,2 mln i kosztu obliczeniowego do ~569 M MAC (multiply–accumulate operations, operacje mnożenia i akumulacji) na obraz 224×224, przy zachowaniu ~70,9% Top-1 i ~89,6% Top-5 (dokładność pięciu najlepszych prognoz) (Rysunek 1) [7]. W mobilnych implementacjach wykazano ponadto, że post-training quantization do INT8 (8-bitowy format całkowitoliczbowy) obniża rozmiar modelu czterokrotnie przy marginalnym spadku trafności [4, 5].



Rysunek 1: Porównanie dokładności Top-1 i Top-5 modeli.

Obok modeli lekkich, badania koncentrowały się również na sieciach o wysokiej precyzji. ResNet-50, wyposażony w połączenia rezydualne i ~25,6 mln parametrów, osiąga około 76,0% Top-1 (92,9% Top-5) (Rysunek 1) na ImageNet, jednak kosztem ~3,8 G FLOPs (floating point operations, operacje zmiennoprzecinkowe) i rozmiaru ~98 MB przed kwantyzacją [2]. Benchmark AI Benchmark ujawnił, że inferencja na CPU (Central Processing Unit, procesor ogólnego przeznaczenia) Snapdragon 855 trwa 300–350 ms na obraz, co uniemożliwia płynną klasyfikację wideo w czasie rzeczywistym bez wsparcia GPU lub NNAPI (Neural Networks API) [2, 3]. Aby pogodzić wysoką dokładność z wydajnością, wprowadzono EfficientNet-Lite, w szczególności wariant Lite0, który rezygnuje z bloków SE (Squeeze-and-Excitation, moduły uwagi) i zastępuje aktywację Swish funkcją ReLU6. Model o ~4,7 mln parametrach i ~407 M MAdds (multiply-adds, operacje mnożenia i dodawania) osiąga ~74,8% Top-1 w FP32 (32-bitowy format zmiennoprzecinkowy) (Rysunek 1) oraz ~74,1% po kwantyzacji do INT8, jednocześnie zapewniając płynną inferencję „on-device" dzięki lepszemu wsparciu heterogenicznych akceleratorów [4]. W kontekście aplikacji asystujących dla osób niewidomych i słabowidzących autorzy [8] opisali In-Door Assistant, łączący detekcję przeszkód z CNN i kalibracją kamery do estymacji odległości (~90% skuteczności, błąd 5–8%). Praca [9] przedstawiła mobilne rozwiązanie do wizualnej klasyfikacji obiektów z wykorzystaniem TensorFlow Lite (TFLite) na smartfonach, osiągając ~85% trafności dla niestandardowych kategorii. Badanie [10] zaproponowało kompleksowy pipeline rozpoznawania i śledzenia (CNN + synteza mowy + interfejs webowy), potwierdzając ponad 83% dokładności dla 1000 klas oraz udostępnianie snapshotów opiekunom w czasie rzeczywistym. Prace [3, 11] wykazały skuteczność połączenia detekcji SSD (Single Shot

Detector) MobileNet z syntezą mowy (87–92% trafności dla 9 klas), podkreślając rosnące znaczenie integracji przetwarzania obrazu i interfejsu głosowego**.** Fragmentaryczność wyników wynikająca z różnorodności frameworków mobilnych stwarza wyzwania. Analizy porównawcze sześciu bibliotek (TensorFlow Lite, PyTorch Mobile, ncnn, MNN, Mace, SNPE – Snapdragon Neural Processing Engine) ujawniły, że czasy inferencji dla tego samego modelu mogą różnić się o 20–30% w zależności od silnika oraz wersji sterowników [3, 12]. Autorzy zwrócili też uwagę na zjawisko „zimnego startu" TFLite, kiedy pierwsze wywołanie inferencji jest wielokrotnie wolniejsze od kolejnych, negatywnie wpływając na percepcję responsywności aplikacji. W obszarze adaptacyjnych rozwiązań on-device badacze [13, 14] poruszyli temat continual learning (uczenie ciągłe), wskazując na potrzebę minimalizacji zakłóceń w inferencji i optymalnego zarządzania pamięcią podczas lokalnej aktualizacji wag. W detekcji obiektów autorzy [15] zaprezentowali ThunderNet – lekki detektor dwustopniowy z backbone'em SNet i modułami CEM (Context Enhancement Module, moduł wzmacniania kontekstu) oraz SAM (Spatial Attention Module, moduł uwagi przestrzennej), osiągający 19,2 AP (Average Precision, precyzja) na zbiorze COCO (Common Objects in Context) przy efektywnej inferencji na CPU ARM (~24 fps), co dowodzi, że dobrze zaprojektowane podejścia dwustopniowe mogą konkurować z rozwiązaniami one-stage. Kompleksowe porównanie YOLO (You Only Look Once)/Tiny-YOLO w środowiskach TFLite, OpenCV (biblioteka komputerowego przetwarzania obrazu) i Snapdragon SDK potwierdziło korzyści hybrydowego pipeline'u dla czasu przetwarzania i zużycia zasobów [12]. Podsumowując, literatura wskazuje, że efektywne przetwarzanie obrazu na urządzeniach mobilnych wymaga zharmonizowanego doboru architektury modelu, technik kwantyzacji, konfiguracji środowiska inferencji oraz wyboru odpowiedniego frameworka. To tło umożliwia rzetelną weryfikację tezy niniejszej pracy, że EfficientNet-Lite0 oferuje najlepszy kompromis między skutecznością a czasem inferencji w scenariuszu real-time classification (klasyfikacja w czasie rzeczywistym) na Androidzie z użyciem TensorFlow Lite i wielowątkowej inferencji na CPU.

3. Omówienie technologii

3.1. Środowisko Android

System Android (system operacyjny mobilny oparty na Linuxie), dostarcza deweloperom bogaty zestaw narzędzi i bibliotek do tworzenia aplikacji mobilnych [16]. Kluczowymi elementami, które wpływają na wydajność i stabilność aplikacji związanych z inferencją w czasie rzeczywistym, są:

- model aplikacji i cykl życia: Android wykorzystuje komponenty Activities (ekran aplikacji) i Fragments (fragment UI), nadzorowane przez systemowy menedżer cyklu życia. Zarządzanie zasobami (np. dostępem do kamery, obsługą pamięci) musi być ściśle skoordynowane ze zdarzeniami onStart, onResume, onPause i onStop (metody cyklu życia), by unikać wycieków pamięci i przeciążeń wątków.
- środowisko wykonawcze ART (Android Runtime – środowisko uruchomieniowe): aplikacje Kotlin/Java kompilowane są do kodu bajtowego, który podczas instalacji jest optymalizowany techniką Ahead-Of-Time (AOT – kompilacja z wyprzedzeniem). Dzięki temu uruchamianie metod inferencyjnych jest szybsze, a zarządzanie pamięcią – bardziej przewidywalne niż w środowisku interpretowanym.
- obsługa wielowątkowości: UI Thread (Main Thread – główny wątek interfejsu) musi pozostać responsywny, dlatego obciążające zadania, takie jak przetwarzanie obrazu czy inferencje, delegowane są do puli wątków (np. ThreadPoolExecutor – wykonawca wątków). Pozwala to na równoległe wykonywanie zapytań do modelu bez blokowania interfejsu użytkownika i zapewnia stałą częstotliwość odświeżania podglądu wideo.
- API kamery Camera2 (Application Programming Interface – interfejs programowania aplikacji): nowoczesne urządzenia oferują niskopoziomowy dostęp do strumienia wideo. Wykorzystanie formatu YUV_420_888 (format obrazu z 4:2:0 subsamplingiem kolorów) oraz mechanizmu zero-copy (bez kopiowania danych między warstwami) minimalizuje narzut związany z transferem klatek, co jest kluczowe przy przetwarzaniu w wysokiej rozdzielczości.

W zaproponowanym rozwiązaniu, aplikacji natywnej (Kotlin, min. API 24 – poziom interfejsu Android 7.0) te elementy zostały zintegrowane w modularnej architekturze: komponenty odpowiedzialne za obsługę kamery, kolejkę zadań i zarządzanie cyklem życia współpracują, by dostarczyć płynną i niezawodną ścieżkę od przechwycenia obrazu do prezentacji wyników klasyfikacji.

3.2. TensorFlow Lite

TensorFlow Lite (TFLite) to lekka wersja TensorFlow zaprojektowana z myślą o urządzeniach o ograniczonych zasobach - smartfonach, tabletach czy urządzeniach wbudowanych. Kluczowe założenia i mechanizmy, które umożliwiają efektywną inferencję na Androidzie to:

- interpreter zoptymalizowany pod kątem CPU i delegatów GPU/NNAPI: TFLite wykorzystuje Interpreter, który może być skonfigurowany do pracy wielowątkowej (setNumThreads) oraz wspierany przez niskopoziomowe biblioteki (XNNPack) lub hardware'owe (lub: sprzętowe) delegaty (np. NNAPI dla akceleracji GPU). Pozwala to elastycznie dobierać kompromis między zużyciem energii a szybkością działania.
- statyczne bufory i przetwarzanie obrazów: Aby uniknąć kosztownych alokacji pamięci w pętli inferencyjnej, wejściowe i wyjściowe bufory (TensorImage, TensorBuffer) tworzone są raz przy inicjalizacji modelu. Obrazy są wstępnie przetwarzane (zmiana rozmiaru, normalizacja), a następnie przekazywane bez kopiowania do interpretera.
- mechanizm delegowania i ponownej inicjalizacji: Zmiana parametrów inferencji (liczba wątków, delegat, model) wymaga unieważnienia bieżącej instancji interpretera i stworzenia nowej. Dzięki temu

eksperymenty porównawcze są powtarzalne, a konfiguracja inferencji - transparentna.
- pomiar i kontrola jakości inferencji: Klasyfikacja każdej klatki jest mierzona czasowo, co umożliwia statystyczną analizę wydajności. Rejestracja czasu przed i po inferencji pozwala ocenić opóźnienia i optymalizować pipeline.

Podsumowując, połączenie środowiska Android (ART, Camera2, Lifecycle) z mechanizmami TensorFlow Lite (statyczne bufory, delegaty, wielowątkowość) tworzy spójną platformę do przeprowadzania weryfikowalnych eksperymentów z klasyfikacją obrazów w czasie rzeczywistym.

3.3. Opis analizowanych modeli

Sieci konwolucyjne (CNN) to klasa głębokich sieci neuronowych wyspecjalizowanych w przetwarzaniu danych o strukturze siatki (przede wszystkim obrazów) poprzez wykorzystywanie hierarchii cech przestrzennych [17]. Podstawowym elementem CNN jest warstwa konwolucyjna, w której uczące się filtry (kernels) przesuwają się po obrazie, wydobywając lokalne wzorce, takie jak krawędzie czy tekstury. Bezpośrednio po konwolucji zwykle stosuje się nieliniowe funkcje aktywacji, najczęściej ReLU, co pozwala modelowi uchwycić złożone zależności niespełnialne przez operacje liniowe.

Kolejnym etapem jest warstwa poolingowa, która zmniejsza wymiary map cech i wprowadza odporność na przesunięcia obrazu, agregując informacje z lokalnych obszarów wejściowych. Po sekwencji konwolucji i poolingów otrzymane mapy cech są spłaszczane (flatten) do wektora, który następnie trafia do jednej lub kilku warstw w pełni połączonych (fully connected) w celu dokonania końcowej klasyfikacji lub regresji [18]. Proces uczenia CNN polega na propagacji wstecznej (backpropagation) i minimalizacji funkcji kosztu przy pomocy metod gradientowych, co pozwala na optymalizację wag filtrów w wielu warstwach jednocześnie. Dzięki tej hierarchicznej reprezentacji cech CNN potrafią wykrywać proste wzory w warstwach płytkich, a następnie łączyć je w coraz bardziej złożone struktury w warstwach głębszych.

Sieci konwolucyjne znalazły zastosowanie w zadaniach rozpoznawania obrazów i wideo, takich jak detekcja obiektów, segmentacja semantyczna czy analiza medyczna, osiągając w wielu przypadkach stan techniki [19-21].

3.3.1.MobileNet-V1

MobileNet-V1 to lekka sieć konwolucyjna, w której kluczową innowacją są separowalne konwolucje (ang. depthwise separable convolutions; operacja dzieląca tradycyjną konwolucję na filtrację każdego kanału obrazu osobno oraz łączenie wyników małą konwolucją 1×1). W efekcie model potrzebuje zaledwie kilku milionów parametrów (wag sieci; liczba liczb uczonych podczas treningu) i setek milionów operacji MAC (multiply–accumulate; mnożenie i dodawanie), co znacząco ogranicza zapotrzebowanie na moc obliczeniową. Trening na zbiorze ImageNet przeprowadza się z wykorzystaniem standardowych technik augmentacji (sztuczne zwiększanie różnorodności danych, np. przez obracanie czy przycinanie obrazów) oraz optymalizatorów takich jak SGD (stochastic gradient descent; metoda optymalizacji kierująca zmiany wag w kierunku minimalizacji błędu) lub Adam (adaptacyjny optymalizator wykorzystujący moment i skalowanie współczynnika uczenia) [23].

3.3.2.EfficientNet-Lite0

EfficientNet-Lite0 opiera się na koncepcji kompozytowego skalowania (ang. compound scaling; równoczesne, zrównoważone zwiększanie głębokości sieci, szerokości kanałów i rozdzielczości obrazu), co pozwala w pełni wykorzystać każdy dodatkowy parametr bez nadmiernego wzrostu obliczeń. Wariant Lite0, mający kilka milionów parametrów i setki milionów operacji MAdd (multiply–add; kroczące mnożenie i dodawanie), osiąga lepszą dokładność Top-1 (procent poprawnie sklasyfikowanych obrazów w pierwszej próbie) niż wcześniejsze sieci mobilne. Trening zwykle rozpoczyna się od fine-tuningu (dostrajenia) pretrenowanego modelu EfficientNet-B0, a następnie może być uzupełniony o kwantyzację (redukowanie precyzji wag do 8 bitów w celu zmniejszenia rozmiaru modelu i przyspieszenia inferencji) [24].

3.3.3.ResNet-50

ResNet-50 to sieć wprowadzająca połączenia resztkowe (ang. residual connections; dodawanie wejścia warstwy do jej wyjścia w celu ułatwienia przepływu gradientu), dzięki którym można trenować bardzo głębokie modele bez problemu zaniku gradientu (problem, w którym sygnał do aktualizacji wag zanika w miarę przekazywania przez wiele warstw). Architektura opiera się na blokach bottleneck (układ trzech konwolucji $1\times1 \rightarrow 3\times3 \rightarrow 1\times1$, gdzie pierwsza zmniejsza liczbę kanałów, a ostatnia ją przywraca), co optymalizuje liczbę operacji. ResNet-50 ma około 25 mln parametrów i wymaga kilku miliardów FLOPs (floating-point operations; operacji na liczbach zmiennoprzecinkowych) na pojedynczą prognozę. Standardowy trening obejmuje 90 epok (pełnych przebiegów przez zbiór danych) z SGD, harmonogramem zmian współczynnika uczenia (learning rate schedule) i klasyczną augmentacją [22,25].

4. Metodyka badań

4.1. Środowisko testowe

Badania przeprowadzono na trzech różnych smartfonach ustawionych na specjalnym stojaku, przed którym wyświetlano prezentację zawierającą 100 obrazów zmieniających się co 1 sekundę. Specyfikację urządzeń zestawiono w Tabeli 1.

4.2. Zbiory danych i metryki oceny

W badaniach wykorzystano gotowe, wytrenowane i zoptymalizowane modele TFLite pobrane z oficjalnych repozytoriów online [23-25]. Dokładność Top-1 każdego z modeli była znana z dokumentacji źródłowej i wynosiła: 70,9% dla MobileNet-V1, 75,1% dla EfficientNet-Lite0 oraz 76,0% dla ResNet-50 (Rysunek 1). Modele te zostały pierwotnie wytrenowane na szerokozakresowym

zbiorze danych ImageNet. W trakcie eksperymentów nie trenowano modeli ponownie, a pomiar dotyczył wyłącznie inferencji: na podstawie 100 pomiarów dla każdego modelu obliczono średni czas przetwarzania klatki (ms) oraz odchylenie standardowe.

Tabela 1: Specyfikacja testowanych urządzeń mobilnych

Model	Google Pixel 7 Pro	Samsung Galaxy S10+	Huawei P30 Pro
CPU	Tensor G2 Octa-core	Snapdragon 855 Octa-core	Kirin 980 Octa-core
-	8-core: 2×2.85 GHz Cortex-X1 + 2×2.35 GHz Cortex-A78 + 4×1.80 GHz Cortex-A55	8-core: 1×2.84 GHz Kryo 485 + 3×2.42 GHz Kryo 485 + 4×1.78 GHz Kryo 485	8-core: 2×2.6 GHz Cortex-A76 + 2×1.92 GHz Cortex-A76 + 4×1.8 GHz Cortex-A55
GPU	Mali-G710 MP7	Adreno 640	Mali-G76 MP10
RAM	12 GB (LPDDR5)	8 GB (LPDDR4X)	8 GB (LPDDR4X)
OS version	Android 15	Android 12	Android 12

4.3. Procedura eksperymentu

Przed rozpoczęciem pomiarów aplikację uruchamiano na każdym z trzech urządzeń, ładując kolejno modele MobileNet-V1, EfficientNet-Lite0 i ResNet-50 w formacie FP32. Następnie rozpoczęto emisję prezentacji 100 obrazów (po jednej klatce na sekundę) na ekranie skierowanym wprost na aparaty urządzeń. Obrazy te pochodziły ze zróżnicowanego zbioru reprezentującego typowe obiekty i sceny spotykane w codziennym otoczeniu, aby zasymulować rzeczywiste warunki użytkowania aplikacji do klasyfikacji obrazów. Całe badanie dla jednego modelu na jednym urządzeniu trwało 100s, podczas których dla każdej klatki rejestrowano:

- czas odpowiedzi modelu (od chwili dostarczenia bufora wejściowego do zakończenia inferencji) za pomocą wewnętrznych znaczników czasu w aplikacji.
- wskaźniki użycia CPU i pamięci RAM przy pomocy Android Studio Profiler.

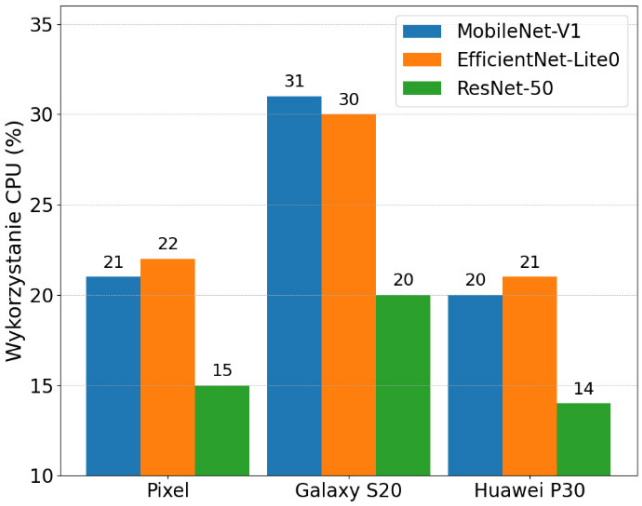
Po upływie 100 s aplikację restartowano, aby wyeliminować ewentualne efekty cache i wycieków zasobów. Restart aplikacji przed każdą nową sesją pomiarową miał również na celu zminimalizowanie wpływu tzw. „zimnego startu" biblioteki TensorFlow Lite, gdzie pierwsze wywołania inferencji mogą być wolniejsze.

5. Omówienie wyników

W niniejszym rozdziale przedstawiono analizę osiągów trzech badanych architektur (MobileNet-V1, EfficientNet-Lite0 i ResNet-50) na podstawie ośmiu wykresów (Rysunki 3-9). Omówienie obejmuje stabilność czasów inferencji, obciążenie procesora, wykorzystanie pamięci oraz kompromis między szybkością a jakością klasyfikacji.

5.1. Obciążenie procesora

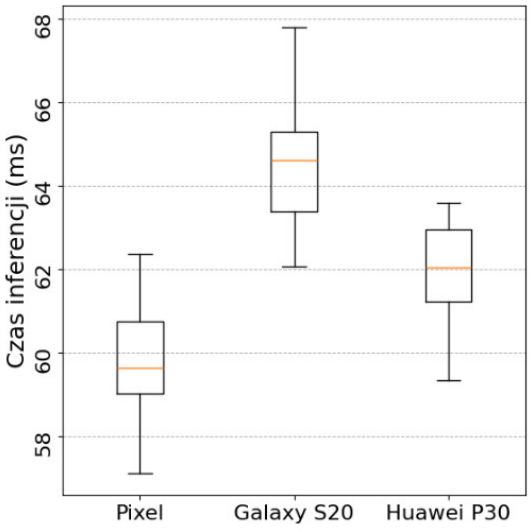
Rysunek 2 ilustruje średnie wykorzystanie CPU podczas inferencji dla trzech modeli. Lekkie sieci MobileNet-V1 i EfficientNet-Lite0 utrzymują umiarkowane obciążenie (około 20–30 %), przy czym EfficientNet-Lite0 generuje nieznacznie wyższe zapotrzebowanie na zasoby niż MobileNet-V1.ResNet-50, pomimo znacznie dłuższego czasu pojedynczej inferencji, obciąża procesor procentowo mniej, co wynika z mniejszej liczby wywołań na sekundę.



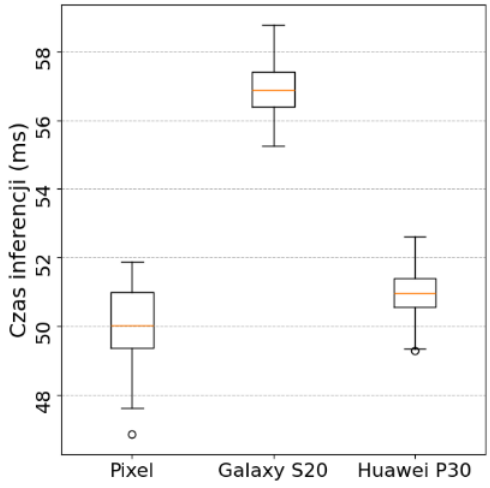
Rysunek 2: Wykorzystanie CPU dla różnych modeli na Urządzeniach.

5.2. Stabilność czasów inferencji

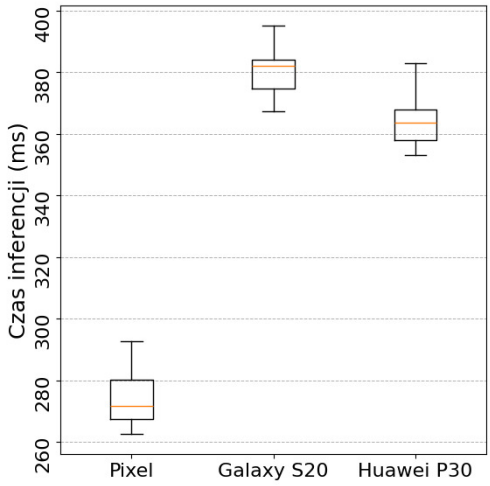
Analiza rozkładów czasów inferencji (Rysunki 3-5) wskazuje na istotne różnice w przewidywalności i spójności obliczeń. MobileNet-V1 (Rysunek 3) oraz EfficientNet-Lite0 (Rysunek 4) charakteryzują się wąskimi „pudełkami" i krótkimi „wąsami", co świadczy o niewielkiej wariancji latencji i wysokiej powtarzalności. Natomiast ResNet-50 (Rysunek 5) prezentuje szerszy rozkład, sugerując większe fluktuacje czasu przetwarzania i potencjalne problemy z utrzymaniem stałej responsywności procesora.



Rysunek 3: Czas inferencji MobileNet-V1 na różnych urządzeniach.



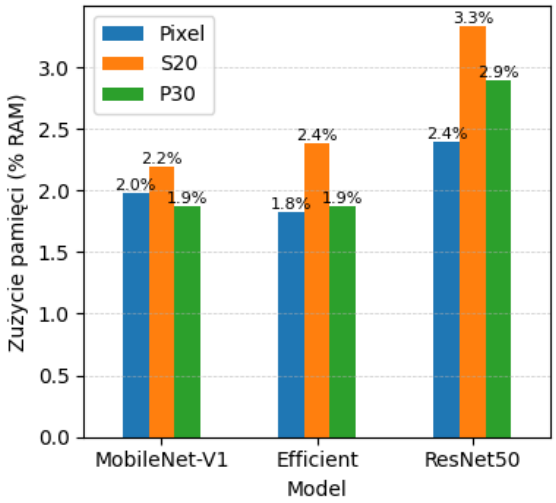
Rysunek 4: Czas inferencji EfficientNet-Lite0 na różnych urządzeniach.



Rysunek 5: Czas inferencji ResNet-50 na różnych urządzeniach.

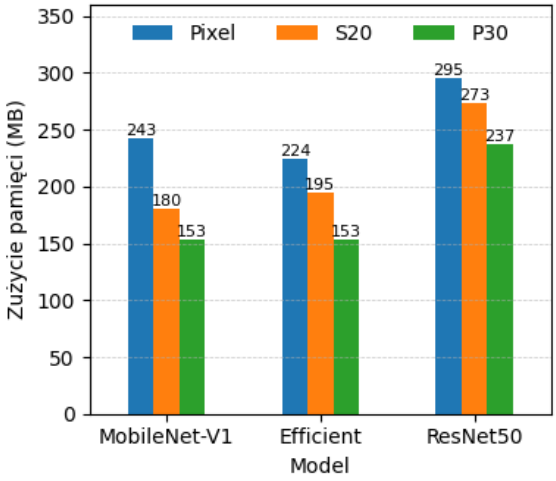
5.3. Wykorzystanie pamięci

Zużycie pamięci RAM przedstawiono na dwóch wykresach: Rysunek 6 pokazuje wartości w procentach całkowitej dostępnej pamięci, a Rysunek 7 - w megabajtach.



Rysunek 6: Procentowe zużycie pamięci RAM przez modele na różnych urządzeniach.

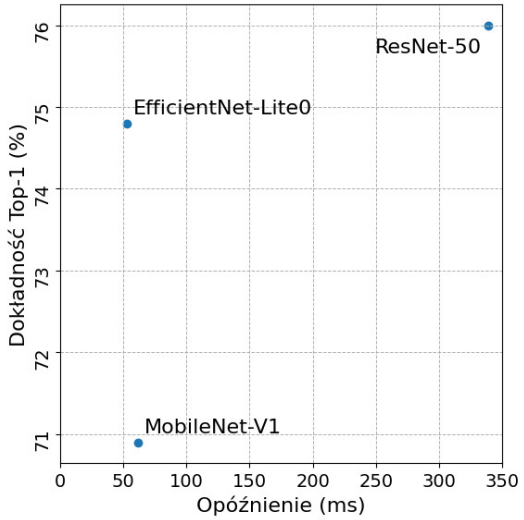
MobileNet-V1 i EfficientNet-Lite0 zajmują niewiele pamięci (poniżej 2,5 % RAM, ok. 180–225 MB), co czyni je optymalnymi dla urządzeń o ograniczonych zasobach. ResNet-50 znacząco przewyższa je pod tym względem (około 3 % RAM, ok. 240-295 MB), co może stanowić barierę na słabszych telefonach..



Rysunek 7: Zużycie pamięci RAM przez modele na różnych urządzeniach (w MB).

5.4. Kompromis szybkości i jakości

Na wykresie przedstawionym jako Rysunek 8 zestawiono znaną z dokumentacji Top-1 Accuracy oraz średni czas inferencji każdego modelu. MobileNet-V1 plasuje się w strefie niskiej latencji przy nieco niższej dokładności, EfficientNet-Lite0 reprezentuje kompromis między krótkim czasem przetwarzania a wysoką skutecznością, natomiast ResNet-50, pomimo najwyższej dokładności, wykazuje zbyt duże opóźnienie, by był praktycznym wyborem dla zastosowań w czasie rzeczywistym.

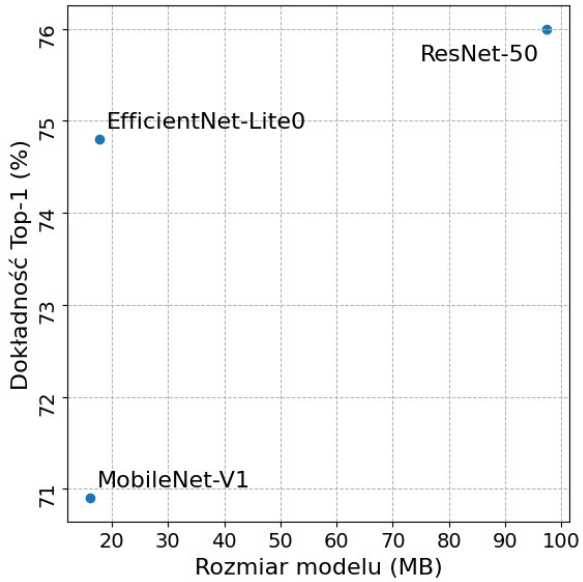


Rysunek 8: Zależność między opóźnieniem inferencji a dokładnością Top-1.

5.5. Rozmiar modelu a skuteczność

Rysunek 9 ukazuje zależność między rozmiarem pliku modelu a jego Top-1 Accuracy. MobileNet-V1 i EfficientNet-Lite0, choć bardzo kompaktowe (< 20 MB),

osiągają wyniki zbliżone do ResNet-50 (ok. 98 MB), co potwierdza ich efektywność w zastosowaniach mobilnych.



Rysunek 9: Zależność między rozmiarem modelu a dokładnością Top-1.

6. Wnioski

Na podstawie wyników przedstawionych w Rozdziale 5 można stwierdzić, że spośród trzech porównywanych architektur to EfficientNet-Lite0 oferuje najbardziej zrównoważone działanie w mobilnej klasyfikacji obrazów. Model ten łączy krótki czas inferencji z niemal najwyższą dokładnością Top-1, potwierdzając nasze założenie badawcze. MobileNet-V1, choć cechuje się równie stabilnymi czasami przetwarzania i niskim zużyciem pamięci, osiąga wyraźnie niższą trafność. Z kolei ResNet-50, mimo że zapewnia nieznacznie lepszą precyzję, wykazuje znaczne fluktuacje latencji i duże zapotrzebowanie na pamięć, co czyni go mało praktycznym w aplikacjach wymagających płynnej pracy w czasie rzeczywistym bez akceleratorów.

Przedstawione w Rozdziale 5 boxploty (Rysunki 3-5) podkreślają, jak istotna jest stabilność inferencji: EfficientNet-Lite0 i MobileNet-V1 generują przewidywalne czasy, natomiast ResNet-50 może zaskakiwać opóźnieniami. Z kolei analiza obciążenia CPU (Rysunek 2) i zużycia pamięci (Rysunki 6-7) uwidoczniła, że lekkie modele pozostają optymalne dla urządzeń z ograniczonymi zasobami, podczas gdy większy model wymaga znacznie więcej pamięci, co może prowadzić do problemów z działaniem na słabszych telefonach. Punktowy wykres opóźnienie vs dokładność (Rysunek 8) oraz zależność rozmiaru pliku od Top-1 Accuracy (Rysunek 9) jasno wskazują, że EfficientNet-Lite0 znajduje się w centrum obszaru kompromisu między wydajnością a jakością, co potwierdza hipotezę.

Kierunki dalszych badań obejmują szereg uzupełnień i rozszerzeń: warto przeprowadzić eksperymenty z kwantyzacją i prunningiem, by zbadać wpływ optymalizacji numerycznych na czas inferencji i dokładność; testy na różnorodnych urządzeniach z różnymi SoC pozwolą ocenić adaptacyjność modeli do specyfiki sprzętu; analiza delegacji do GPU, DSP czy NNAPI umożliwi ocenę korzyści wynikających z akceleracji; realne scenariusze użytkowe (np. w środowisku AR czy asystentów wizualnych) pozwolą uwzględnić nieprzewidywalne czynniki otoczenia oraz dynamiczne zarządzanie pamięcią; wreszcie badania nad adaptacyjnym próbkowaniem klatek mogłyby zoptymalizować proporcję między szybkością przetwarzania a zachowaniem odpowiedniej jakości klasyfikacji.

Podsumowując, potwierdziliśmy, że EfficientNet-Lite0 jest najbardziej efektywną architekturą dla mobilnej klasyfikacji obrazów w środowisku Android z użyciem TensorFlow Lite i wielowątkowego CPU. Zalecane rozszerzenia badań mogą przyczynić się do dalszego zwiększenia wydajności modeli i jeszcze lepszego dostosowania ich do zróżnicowanego ekosystemu urządzeń mobilnych.

Literatura

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems 25 (2012) 1097–1105.

[2] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, L. Van Gool, AI benchmark: running deep neural networks on Android smartphones, Proceedings of the European Conference on Computer Vision (ECCV) Workshops (2018) 288–314, https://doi.org/10.1007/978-3-030-11021-5_19.

[3] C. Luo, X. He, J. Zhan, L. Wang, W. Gao, J. Dai, Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices (2020), https://arxiv.org/abs/2005.05085.

[4] Zbiór danych na temat modeli, TensorFlow Blog, https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html, [01.05.2025].

[5] D. Baldota, S. Advani, S. Jaidhara, A. Hatekar, Object Recognition using TensorFlow and Voice Assistant, International Journal of Engineering Research & Technology 10(9) (2021) 359–362.

[6] S. A. Jakhete, P. Bagmar, A. Dorle, A. Rajurkar, P. Pimplikar, Object Recognition App for Visually Impaired, In 2019 IEEE Pune Section International Conference (PuneCon) (2019) 1-4, https://doi.org/10.1109/PuneCon46936.2019.9105670.

[7] O. Alsing, Mobile object detection using TensorFlow Lite and transfer learning, Master thesis, KTH Royal Institute of Technology, Stockholm, 2018.

[8] N. Khaled, S. Mohsen, K. El-Din, E. Akram, In-door assistant mobile application using CNN and TensorFlow, In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE) (2020) 1-6, https://doi.org/10.1109/ICECCE49384.2020.9179386.

[9] N. Parikh, I. Shah, S. Vahora, Android smartphone based visual object recognition for visually impaired using deep learning, In 2018 International Conference on Communication and Signal Processing (ICCSP) (2018) 420–425, https://doi.org/10.1109/ICCSP.2018.8524493.

[10] F. Ashiq, M. Asif, CNN-based object recognition and tracking system to assist visually impaired people, IEEE Access 10 (2022) 14819–14834, https://doi.org/10.1109/ACCESS.2022.3148036.

[11] G. Khekare, K. Solanki, Real time object detection with speech recognition using TensorFlow Lite, GIS Science Journal 9(3) (2022) 552-559.

[12] I. Martinez-Alpiste, G. Golcarenarenji, Q. Wang, J. M. Alcaraz-Calero, Smartphone-based real-time object recognition architecture for portable and constrained systems, Journal of Real-Time Image Processing 19 (2022) 103–115, https://doi.org/10.1007/s11554-021-01164-1.

[13] G. Demosthenous, V. Vassiliades, Continual Learning on the Edge with TensorFlow Lite (2021) 1-8, https://doi.org/10.48550/arXiv.2105.01946.

[14] L. Pellegrini, V. Lomonaco, G. Graffieti, D. Maltoni, Continual Learning at the Edge: Real-Time Training on Smartphone Devices, https://arxiv.org/abs/2105.13127.

[15] Z. Qin, Z. Li, Z. Zhang, Y. Bao, G. Yu, Y. Peng, J. Sun, ThunderNet: towards real-time generic object detection on mobile devices, Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2019) 6717–6726, https://doi.org/10.1109/ICCV.2019.00682.

[16] Dokumentacja Android, https://developer.android.com/reference/android/hardware/camera2/package-summary.html, [01.05.2025].

[17] Artykuł poświęcony konwolucyjnym sieciom neuronowym, IBM, https://www.ibm.com/think/topics/convolutional-neural-networks, [01.05.2025].

[18] Artykuł poświęcony konwolucyjnym sieciom neuronowym, GeeksforGeeks, https://www.geeksforgeeks.org/introduction-convolution-neural-network, [01.05.2025].

[19] P. Powroźnik, Polish emotional speech recognition using artifical neural network, Advances in Science and Technology, Research Journal 8 (2014) 24-27, https://doi.org/10.12913/22998624/562.

[20] P. Powroźnik, P. Wójcicki, S. W. Przyłucki, Scalogram as a representation of emotional speech, IEEE Access 9 (2021) 154044-154057, https://doi.org/10.1109/ACCESS.2021.3127581.

[21] M. Skublewska-Paszkowska, P. Powroźnik, E. Łukasik, Attention temporal graph convolutional network for tennis groundstrokes phases classification, In 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (2022) 1–8, https://doi.org/10.1109/FUZZ-IEEE55066.2022.9882822.

[22] K. Nowomiejska, P. Powroźnik, M. Skublewska-Paszkowska, K. Adamczyk, M. Concilio, L. Sereikaite, R. Zemaitiene, M. D. Toro, R. Rejdak, Residual Attention Network for distinction between visible optic disc drusen and healthy optic discs, Optics and Lasers in Engineering 176 (2024) 108056, https://doi.org/10.1016/j.optlaseng.2024.108056.

[23] Repozytorium modelu MobileNet-V1, https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md?utm_source, [01.05.2025].

[24] Repozytorium modelu EfficientNet-Lite0, https://github.com/tensorflow/tpu/blob/master/models/official/efficientnet/lite/README.md, [01.05.2025].

[25] Repozytorium modelu ResNet-50, https://huggingface.co/qualcomm/ResNet50, [01.05.2025].