

Sprawozdanie do projektu  
**Klasyfikacja produktów spożywczych przy użyciu Computer Vision.**  
**Porównanie modelu MobileNet w wersji podstawowej i**  
**skwantyzowanej. Aplikacja mobilna klasyfikująca produkty w**  
**czasie rzeczywistym**

Vitalii Shapovalov 196633  
Ruslan Rabadanov 196634  
Danylo Zakharchenko 196739

## Spis treści

<b>Wstęp.....</b>	<b>1</b>
Cel projektu.....	1
MobileNet.....	1
Architektura MobileNet V1.....	2
Dataset.....	2
<b>Fine-tuning.....</b>	<b>3</b>
<b>Konwertacja i kwantyzacja w format tensorflow lite.....</b>	<b>4</b>
<b>Aplikacja mobilna.....</b>	<b>5</b>
Delegaty w TF Lite.....	6
<b>Problemy z kompatybilnością i nasze zalecenia.....</b>	<b>7</b>
<b>Podsumowanie.....</b>	<b>7</b>

## Wstęp

### Cel projektu

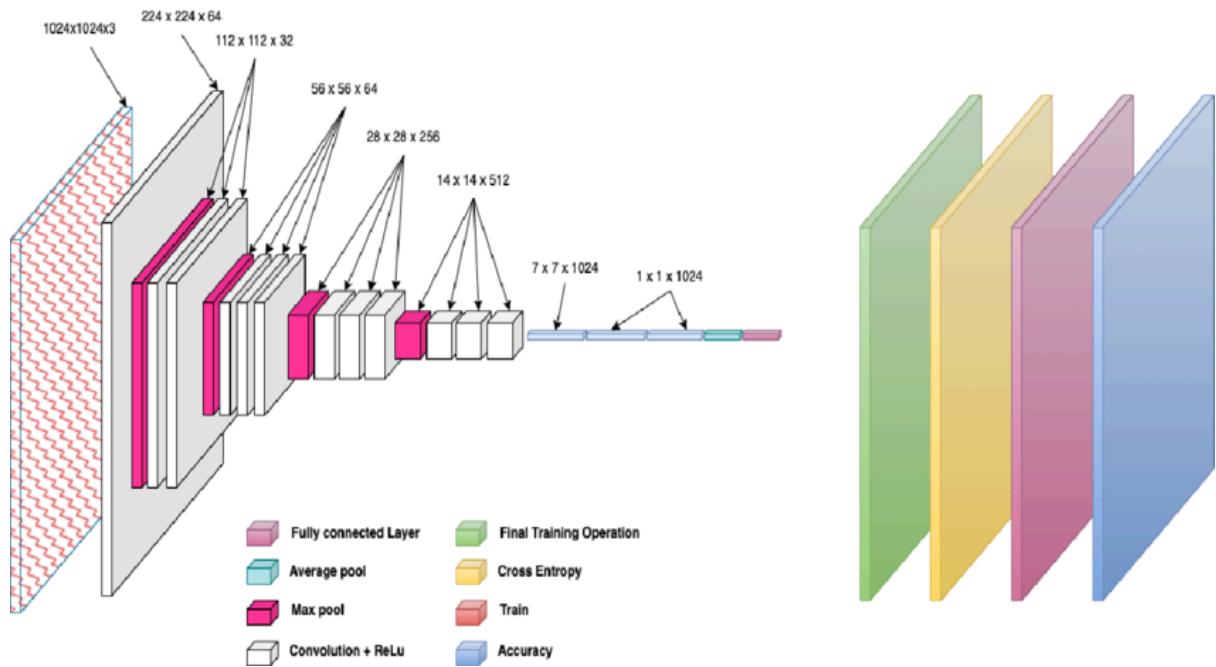
Celem naszego projektu jest stworzenie mobilnej aplikacji do klasyfikacji produktów spożywczych w czasie rzeczywistym oraz porównanie skuteczności modeli *MobileNet* w wersji podstawowej i skwantyzowanej.

### MobileNet

*MobileNet*, opracowany przez Google, jest rodzajem konwolucyjnej sieci neuronowej zaprojektowanej specjalnie do zastosowań na urządzeniach

mobilnych i wbudowanych systemach o ograniczonych zasobach obliczeniowych.

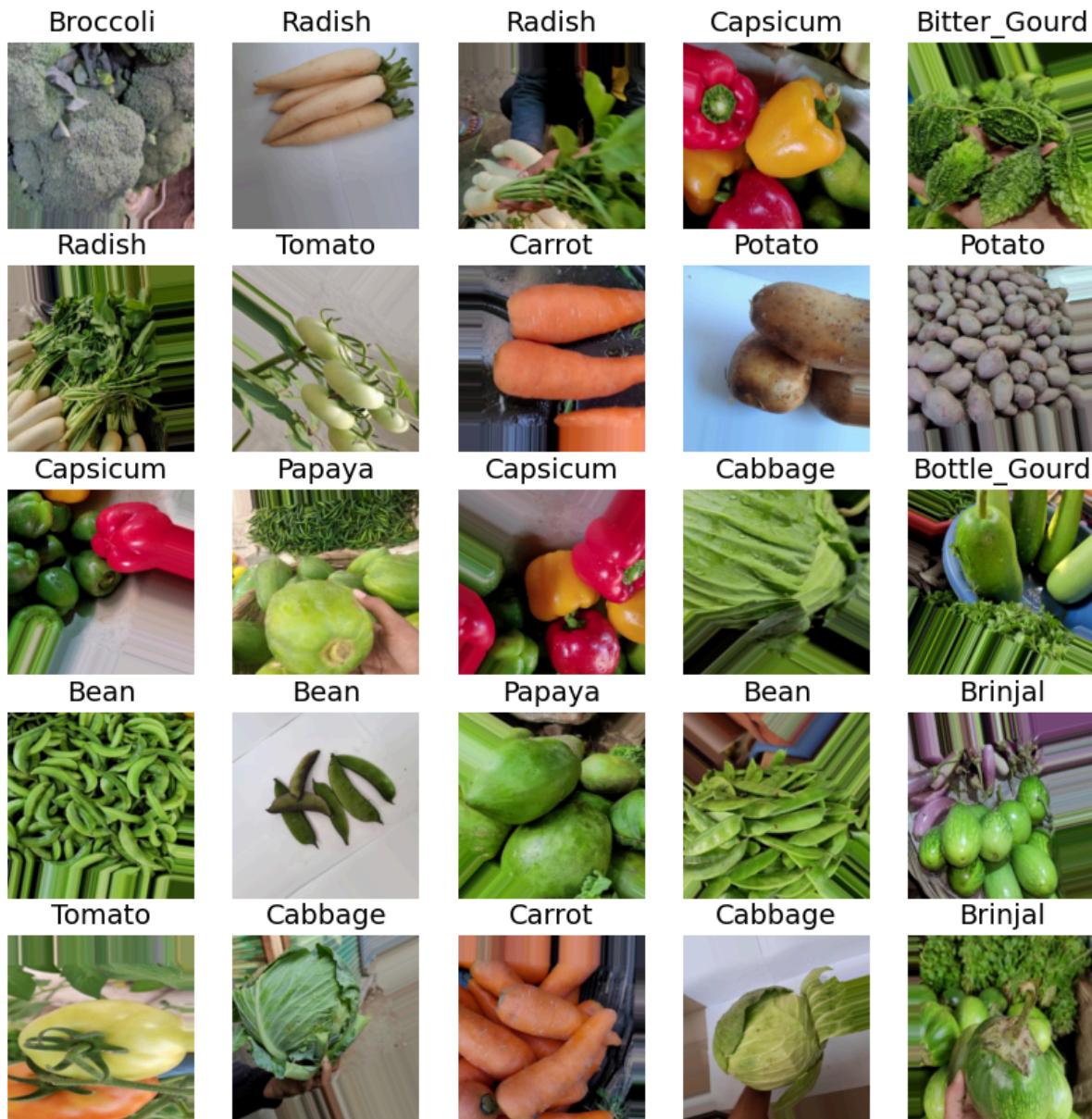
## Architektura *MobileNet V1*



## Dataset

Do treningu modeli zostały użyte dane z platformy [Kaggle](#). W tym zbiorze danych znajduje się 21000 obrazów z 15 klas, gdzie każda klasa zawiera łącznie 1400 obrazów. Każda klasa ma równą proporcję, a rozdzielcość obrazów wynosi  $224 \times 224$  pikseli i są one w formacie \*.jpg. Zbiór danych jest podzielony na trzy części: około 70% na trening, około 15% na testowanie i pozostałe 15% na walidację.

Przykładowe dane z datasetu:



## Fine-tuning

Model *MobileNetV1* został poddany fine-tuningowi, co pozwoliło na dostosowanie go do specyficznych danych z naszego zbioru. Proces fine-tuningu obejmował trenowanie modelu na nowych danych przy zachowaniu wcześniej wyuczonych wag, co umożliwia osiągnięcie

lepszych wyników przy mniejszej ilości danych treningowych.

W rezultacie ten model pozwala na klasyfikowanie produktów z 15 różnych kategorii z dokładnością 97%.

#### Jak to działa:

Inicjowany jest model **MobileNet** z wagami pobranymi z ImageNet, bez górnej warstwy (`include_top=False`) i z określonym kształtem wejściowym (`input_shape=(224, 224, 3)`).

Do wyjścia `base_model` dodajemy trzy dodatkowe warstwy:

- **GlobalAveragePooling2D**, aby zmniejszyć liczbę wymiarów tensora wyjściowego.
- **Dense** warstwę z 1024 jednostkami i funkcją aktywacji `relu`, służącą do uczenia bardziej złożonych cech.
- **Dense** warstwę końcową z 15 jednostkami i funkcją aktywacji `softmax`, aby uzyskać prawdopodobieństwo dla każdej z 15 klas.

Podczas procesu *fine tuning* wszystkie warstwy bazowe modelu *MobileNet* zostały ustawione jako nieuczące się (zamrożone), co oznacza, że wagi tych warstw nie są aktualizowane podczas procesu uczenia.

Model jest komplikowany z optymalizatorem *Adam*, funkcją straty `categorical_crossentropy` oraz metryką dokładności (`accuracy`). Trening trwał przez 3 epoki.

## Konwertacja i kwantyzacja w format tensorflow lite

Po zakończeniu treningu, model został przekonwertowany do formatu TensorFlow Lite w celu umożliwienia jego użycia na urządzeniach mobilnych. W ramach tego procesu przeprowadziliśmy dynamiczną kwantyzację post-treningową, co znacząco zmniejszyło rozmiar modelu i zwiększyło jego wydajność na urządzeniach o ograniczonej mocy obliczeniowej.

## **Porównanie modelu zwykłego i skwantyzowanego**

Model	Rozmiar (MB)	Dokładność (%)	Szybkość (ms)
Zwykły	26	97	67.24
Skwantyzowany	4.5	96.8	22.39

**Rozmiar:** Skwantyzowany model (\*.tflite) jest znacznie mniejszy (4,5 MB) w porównaniu do modelu (\*.keras) po fine-tuningu (26 MB), co czyni go bardziej odpowiednim do zastosowań na urządzeniach mobilnych z ograniczoną pamięcią.

**Dokładność:** Różnica w dokładności między modelem po fine-tuningu (97%) a skwantyzowanym modelem (96,8%) jest minimalna, co oznacza, że proces kwantyzacji nie wpłynął znacząco na skuteczność modelu.

**Szybkość:** Skwantyzowany model jest znacznie szybszy (22,39 ms) w porównaniu do modelu po fine-tuningu (67,24 ms), co jest istotne dla aplikacji w czasie rzeczywistym.

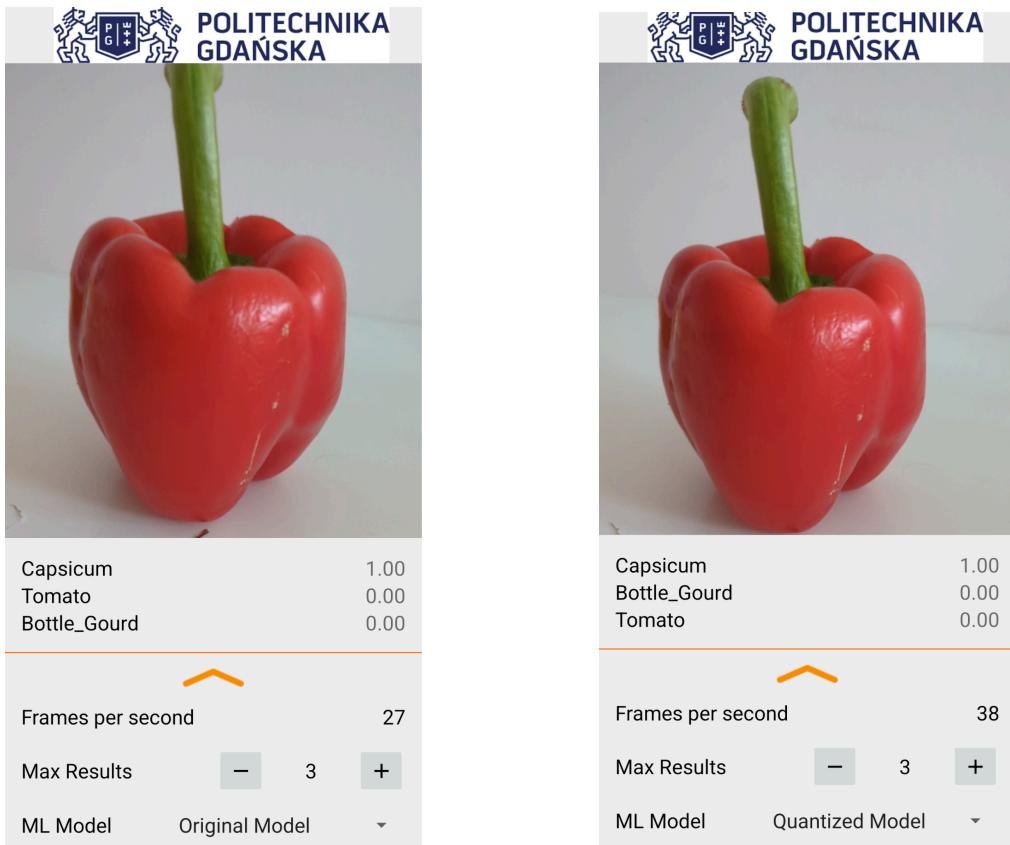
Podsumowując, skwantyzowany model oferuje znaczne korzyści pod względem rozmiaru i szybkości przy minimalnym spadku dokładności, co czyni go bardziej efektywnym rozwiązaniem dla zastosowań mobilnych.

## **Aplikacja mobilna**

Do demonstracji działania modelu było przewidziane wytworzenie aplikacji pod system operacyjny Android. Do realizacji aplikacji użyliśmy aplikacji przykładowej udostępnionej przez bibliotekę TensorFlow Lite, kod na którym bazowaliśmy jest dostępny na [GitHub](#).

Problem na który napotkaliśmy polegał na tym, że nie mogliśmy dodać metadanych do modeli, które są potrzebne żeby używać *TaskAPI*. Źródłem problemu jest biblioteka *tflite-support*, która musi umożliwiać dodanie takich metadanych, ale z jakiegoś powodu nie instaluje

się poprawnie przy użyciu *pip*, nawet jak zainstalowaliśmy wprost przez *.whl* plik, biblioteka napotkała się na jakieś błędy podczas działania. Żeby rozwiązać ten problem byliśmy zmuszeni do użycia mniej wygodnego *InterpreterAPI* i ręcznego napisania funkcji do preprocessingu obrazu z kamery w format przyjmowany przez model. Po tych krokach otrzymaliśmy działającą aplikację.



Aplikacja pozwala na wybór modelu (oryginalny/skwantyzowany) oraz pokazuję bieżące FPS (liczbę klatek na sekundę), które model jest w stanie przetworzyć. Generalnie FPS'y na modelu skwantyzowanym są wyższe o połowę w porównaniu do oryginalnego modelu. Chociaż warto zauważać, że liczba FPS w przypadku modelu skwantyzowanego jest mniej stabilna (zmienia się ciągle o 10 w górę i dół) wtedy jak oryginalny model trzyma liczbę FPS mniej więcej stałą.

## Delegaty w TF Lite

Co do tego jakich delegatów TF Lite używa można zapoznać się pod [tym linkiem](#). Co dotyczy GPU to trudno powiedzieć czy działa to dobrze na większości urządzeń, ponieważ testowaliśmy to na dwóch różnych urządzeniach i na żadnym tf lite nie było w stanie użyć GPU. W

przypadku CPU tensorflow używa zestawu rozkazów ARM Neon co jest “odpowiednikiem” SIMD z architektury x64. Także nowym rozwiązaniem w przypadku Androida jest NNAPI, ponieważ teraz niektóre urządzenia posiadają osobny układ scalony (NPU) do obliczenia sieci neuronowych.

## Problemy z kompatybilnością i nasze zalecenia

Jeżeli jest potrzeba używania biblioteki *tflite* to bardzo zalecamy zmienić wersję tensorflow z 2.16 (która jest najnowsza stanem na czerwiec 2024) na wersję 2.15, ponieważ na wersji 2.16 konwersja modelu z formatu keras/tensorflow w naszym przypadku nie zadziałała. Jednym z rozwiązań które można znaleźć jest używanie dev wersji 2.17. Spróbowaliśmy tą opcję i konwersja przebiegła pomyślnie, ale w tym przypadku już nie dało się użyć tego modelu w aplikacji mobilnej, ponieważ ta dev wersja jeszcze nie była dostępna na Android, a starsza wersja biblioteki nie działała z nowszym modelem.

Również jeżeli ktoś będzie chciał użyć modelu *tflite*, polecamy sprawdzić czy naprawiona została biblioteka *tflite-support*, która umożliwia dodawanie metadanych do modelu. Ponieważ w takim przypadku w TensorFlow Lite są dużo klas np. *ImageClassifier*, które znacznie ułatwiają pracę z modelem.

## Podsumowanie

Podsumowując, nasze badanie pokazuje, że kwantyzacja modeli jest skuteczną strategią do redukcji rozmiaru i poprawy wydajności aplikacji mobilnych przy minimalnym wpływie na dokładność. Chociaż dla potrzeb naszej aplikacji wykorzystanie zwykłej modeli też jest wystarczająco wydajne (skwantyzowany model działa o ~1.5 razy szybciej).