Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy informacyjno-decyzyjne

Wykrywanie gestów w strumieniu video za pomocą Raspberry Pi

Grzegorz Rogoziński
Numer albumu 293141

promotor
dr hab. inż. Mariusz Kamola

WARSZAWA 2024

Wykrywanie gestów w strumieniu video za pomocą Raspberry Pi

Streszczenie. Celem pracy był dobór i konfiguracja gotowego rozwiązania w zakresie wykrywania gestów w strumieniu wideo oraz uruchomienie go na jednopłytkowym komputerze Raspberry Pi. W pracy omówiono zastosowania dla narzędzi umożliwiających rozpoznawanie gestów, a w szczególności korzyści płynące z wykorzystania w tym celu Raspberry Pi. Następnie wskazane i opisane zostały różne systemy realizujące to zadanie. Wybrano spośród nich jeden, Temporal Shift Module, i przetestowano na zbiorze 4000 nagrań wideo pochodzących ze zbioru Kinetics 400 przy użyciu kilku metod optymalizacji jakości i szybkości działania klasyfikatora. Otrzymane wyniki zostały opracowane, poddane analizie i omówione pod kątem nakreślonych na początku zastosowań.

Słowa kluczowe: wykrywanie gestów, sieci neuronowe, Raspberry Pi, Kinetics 400, Temporal Shift Module

Gesture recognition in a video stream using Raspberry Pi

Abstract. The aim of the thesis was to select and configure an pre-existing solution for detecting gestures in a video stream and to run it on a single-board Raspberry Pi computer. The paper explores the applications of gesture recognition tools, emphasizing the advantages of employing Raspberry Pi for this purpose. Subsequently, several systems performing this task were identified and described. One of them, Temporal Shift Module, was chosen and evaluated on a dataset of 4000 video clips extracted from the Kinetics 400 dataset, using several methods to enhance the quality and speed of the classifier. The results obtained were processed, analyzed and discussed in light of the initially outlined applications.

Keywords: gesture recognition, neural networks, Raspberry Pi, Kinetics 400, Temporal Shift Module

Spis treści

1. Wstęp

1.1. Wprowadzenie

W dzisiejszych czasach niemal na każdym kroku otoczeni jesteśmy przez elektronikę. Coraz więcej urządzeń wyposażonych jest w zaawansowaną technologię, której celem jest zaoszczędzenie naszego czasu, zapewnienia komfortu, wygody i bezpieczeństwa. Czynności niegdyś manualne wykonują już powszechnie spotykane domowe sprzęty. Parzenie kawy powierzamy ekspresom, za pranie odpowiadają pralki, nad bezpieczeństwem czuwają kamery. Nawet niekiedy nasze światła i rolety obsługiwane są z poziomu telefonu. Z każdym z tych urządzeń w inny sposób wchodzimy w interakcję - czasami jest to klawiatura, panel z przyciskami, w innych przypadkach zasosowanie znajdują ekrany dotykowe, część zadań wykonuje się automatycznie zgodnie z ustalonym harmonogramem, a wybrane polecenia wydajemy przy pomocy głosu.

W takim świetle naturalnym krokiem naprzód wydaje się być mechanizm rozpoznawania gestów. Przetwarzanie obrazu jako metoda wprowadzania mogłoby z łatwością znaleźć miejsce wśród codziennych zastosowań. Nietrudno wyobrazić sobie, jak wygodnie byłoby gestem zastąpić wymagający wymiany baterii i często ulegający zgubieniu pilot od telewizora. Być może inteligenty odkurzacz sprzątający dom mógłby zostać wyproszony z pokoju machnięciem ręki. Oprócz tego można wymyślić szereg nowych zadań dla tej technologii, na przykład komputer z podłączoną kamerą mógłby czuwać nad nauką kroków tanecznych albo stylów pływania. Wreszcie pojawia się ogromny potencjał w kwestiach związanych z bezpieczeństwem, które wybiegają daleko poza domowe zastosowania. Kamera rozpoznająca gesty może alarmować o aktach wandalizmu albo przemocy, wykrywać upadek osoby starszej, przestrzegać przed niebezpieczną sytuacją na drodze.

Choć powyższe przykłady mogą brzmieć nowatorsko, próby zaaplikowania gestu jako HMI[1] nie są pomysłem nowym. Prace nad osiągnięciem tego celu przy użyciu przetwarzania obrazu trwały już w latach 90. XX wieku i stale postępują [1]. Dużo wcześniej natomiast realizowano wykrywanie gestów przy użyciu dodatkowych urzadzeń, wymagających określonych warunków i środowiska pracy. Z powodu tych ograniczeń to właśnie widzenie komputerowe[2] jest najpraktyczniejszą i najszerzej stosowaną metodą rozpoznawania gestów.

[1] ang. *Human-machine interaction*
[2] od ang. *computer vision*

1.2. Cel pracy

Mając na uwadze powyższe rozważania, celem niniejszej pracy jest zbadanie, w jakim stopniu dostępne obecnie rozwiązania są w stanie sprostać wspomnianym zastosowaniom. Kluczowymi wymogami będzie to, aby dobrany klasyfikator:

- był w stanie działać na urządzeniach o niewielkiej mocy obliczeniowej;
- czerpał informacje ze strumieni wideo.

Pierwszy punkt zdaje się być dość intuicyjny. Jeżeli myślimy o poszukiwanym rozwiązaniu w kontekście urządzeń smart home albo urządzeń mobilnych, to nie powinniśmy powierzyć obliczeń wydajnym, dedykowanym serwerom - oznaczałoby to albo konieczność posiadania takiej maszyny, albo usługę udostępniania jej zasobów za pośrednictwem internetu. W obu przypadkach przekłada się to na dodatkowe koszty utrzymania oraz na podniesienie złożoności konfiguracji systemu, a przez to ograniczenie jego dostępności. Z tego powodu jako platformę testową na potrzeby tej pracy wybrano Raspberry Pi 4B. Do licznych korzyści płynących z tego wyboru należą między innymi niski pobór energii elektrycznej, wydajność oddająca możliwości urządzeń mobilnych, stosunkowo niska cena, a także wysoka popularność.

Drugi ze wskazanych wymogów dotyczy faktu, że w pewnych sytuacjach czasowy charakter gestu nie jest możliwy do ujęcia w ramach pojedynczej klatki nagrania. O ile takie podejście mogłoby sprawdzić się na przykład przy wykrywaniu uśmiechu albo zaciśniętej pięści, nie pozwoliłoby na obsługę dynamicznych gestów, jak rozróżnienie przesunięcia dłoni w prawo od ruchu w lewą stronę. Jest to czynnik który dodatkowo komplikuje każdy etap zadania - w szczególności przygotowanie danych wejściowych i sam proces klasyfikacji. Należy zatem spodziewać się, że poszukiwane rozwiązanie może okazać się obliczeniowo lub czasowo wymagające i zaplanować w związku z tym zastosowanie kilku różnych metod optymalizacji.

W dalszej części pracy w pierwszej kolejności przedstawiony zostanie podział i specyfikacja zastosowań opisywanego systemu. Potem nastąpi przegląd i analiza dostępnych rozwiązań. Jedno z nich, najlepiej dopasowane do problemu, zostanie szczegółowo zaprezentowane. Poruszę kwestie jego architektury, przygotowania danych, sposobów optymalizacji jego pracy. Na końcu umieszczone będą wyniki i omówienie testów przeprowadzonych na Raspberry Pi 4B oraz płynące z nich wnioski końcowe.

2. Zastosowania i przegląd rozwiazań

2.1. Zastosowania

W pierwszym rozdziale wspomniane zostały niektóre z licznych zastosowań dla narzędzi umożliwiających rozpoznawanie gestów. Aby usprawnić dalsze rozważania dotyczące doboru technologii, a docelowo również omówienia wyników, warto podjąć próbę sprecyzowania grup zastosowań. Elementy każdej z grup będą posiadały podobne wymogi związane z pracą systemu, co ułatwi w przyszłości ocenę, do jakich zadań nalepiej sprawdzi się badane rozwiązanie.

Wśród najistotniejszych grup zastosowań wskazać należy:

- zastosowania użytkowe,
- zastosowania w edukacji,
- zastosowania w zakresie bezpieczeństwa.

2.1.1. Zastosowania użytkowe

Zastosowania użytkowe dotyczą przede wszystkim zadań określanych pojęciem smart home, powiązanych z automatyzacją pracy urządzeń domowych. Jako przykład wskazać tutaj można regulację temperatury, sterowanie roletami, oświetleniem, otwieraniem i zamykaniem okien. Najważniejszym kryterium działania systemu wykrywającego gesty jest w tym przypadku wysoka precyzja, zapobiegająca wykonaniu niepożądanych akcji. Mniej istotny jest zakres obsługiwanych gestów - każde z urządzeń prawdopodobnie rozróżniałoby kilka prostych ruchów służących do aktywacji poszczególnych funkcji. Przekłada się to jednocześnie na brak wysokich oczekiwań wobec czułości narzędzia - dany gest w razie potrzeby można z łatwością powtórzyć. Ponieważ wiele z procesów realizowanych w tym scenariuszu nie jest natychmiastowe (praca klimatyzacji, żaluzji), sam czas klasyfikacji gestu nie jest kluczowy, szczególnie jeżeli jego kosztem możemy uzyskać wyższą dokładność.

2.1.2. Zastosowania w edukacji

Kolejna kategoria - zastosowania w edukacji - gromadzi wszelkiego rodzaju interaktywne szkolenia, kursy, testy wykorzystujące obraz z kamery. Mogłyby one sprawdzić się podczas nauki i weryfikacji postępów w czynnościach takich jak taniec, pływanie, wykonywanie ćwiczeń fizycznych. W przeciwieństwie do poprzedniej sytuacji, tutaj dla wiarygodności wyniku istotna jest zarówno precyzja, jak i czułość. Czas klasyfikacji nadal pozostaje mało ważny, ponieważ nic nie stoi na przeszkodzie, aby wynik kursu przedstawić dopiero po jego zakończeniu. Nieco większy powinien być natomiast zakres rozpoznawanych gestów, które dodatkowo będą prawdopodobnie bardziej złożone.

2.1.3. Zastosowania w zakresie bezpieczeństwa

Trzecią grupę stanowią zastosowania dotyczące bezpieczeństwa. Może to być na przykład analiza obrazu z kamer w miejscach publicznych, lotniskach, szpitalach, celem wykrywania agresywnych zachowań lub niebezpiecznych wypadków. Ogromne znaczenie

ma dla tych zadań czułość narzędzia, aby wykryć jak największy procent zagrożeń, oraz czas klasyfikacji, aby jak najszybciej poinformować o nich obsługę, ochronę. Bardzo szeroki powinien być również zakres rozpoznawanych gestów czy akcji. Ponieważ system przeznaczony do tak krytycznych zadań nie powinien być pozostawiony bez nadzoru, na znaczeniu traci precyzja - lepiej, aby pracownik obsługi zweryfikował obraz budzący wątpliwość i samodzielnie dokonał oceny sytuacji. Ponadto niektóre przejawy przemocy mogą wynikać chociażby z interwencji policji i nie stanowią zagrożenia, co czyni pracę osoby nadzorującej system jeszcze istotniejszą.

2.2. Technologie rozwiązań

Istnieje wiele technologii, które w odpowiednich okolicznościach mogłby realizować przedstawione wcześniej zadania. Co więcej, nie wszystkie z nich realizują rozpoznawanie akcji za pośrednictwem strumieni wideo. Większość z nich posiada jednak cechy, które wykluczają lub utrudniają ich zastosowanie w tym przypadku. Dla uzyskania szerszego kontekstu, część z nich zostanie nakreślona z ich zaletami oraz wadami.

2.2.1. Rękawice i kontrolery rejestrujące ruch

Jednym z rozwiązań są rękawice rejestrujące ruch (ang. *data gloves*[1]). Występują w dwóch kategoriach: pasywne oraz aktywne. Te pierwsze charakteryzują się specyficznym wzorem lub kolorystyką, która umożliwia oszacowanie pozycji dłoni na poziomie oprogramowania przetwarzającego obraz z kamery. W wersji aktywnej wyposażone są w odpowiednie czujniki, na przykład akcelerometry lub czujniki magnetyczne. Chociaż zależnie od kategorii różnią się kosztem i dokładnością odczytu gestu, do wspólnych ich wad należy niski komfort stosowania i ograniczenie zakresu gestów wyłącznie do tych wykonywanych dłońmi. W związku z koniecznością noszenia i ewentualnego zasilania, trudno je dostosować do pracy w różnych środowiskach, szczególnie jeśli chodzi o zastosowania dotyczące bezpieczeństwa.

W podobny sposób działają innego rodzaju kontrolery. Mogą bazować na różnych zasadach - część trzymania w dłoniach (na przykład kontrolery stosowane w grach wideo), niektóre posiadania określonego pasywnego lub aktywnego przedmiotu takiego jak etykieta RFID[3], pozostałe wykorzystują jeszcze inne elementy, jak na przykład elektrody EMG[4][1]. Wiele z nich bardzo dobrze sprawdza się w ściśle określonym zakresie zadań, do których zostały stworzone, jednak dzielą te same problemy, które dotyczą rękawic - posiadają ograniczenia sprawiające, że trudno zastosować je w sposób uniwersalny.

2.2.2. Kamery

Alternatywnym, preferowanym dla tej pracy podejściem jest użycie kamer. Nie wymagają one od użytkownika obsługi dodatkowych urządzeń, a zamiast tego odpowiedzialność za oszacowanie pozycji obiektów przenoszą na oprogramowanie przetwarzające obraz.

[3] RFID - ang. *radio frequency identification* - systemy identyfikacji radiowej
[4] EMG, elektromiografia - badanie oparte o potencjały bioelektryczne mięśni i nerwów

Czasami stosuje się zabiegi dostarczające więcej informacji niż sam strumień wideo. Mogą to być próby wykrycia głębi poprzez porównanie obrazu z dwóch źródeł (wizja stereoskopowa), oświetlenie sceny w sposób umożliwiający obliczenie pozycji obiektu w przestrzeni albo emisję światła spoza zakresu widzialnego i pomiar czasu, jaki zajmuje jego powrót (ToF - ang. *Time of Flight*). Ponownie wracają tu jednak kwestie kosztów i uniwersalności stosowania, w szczególności przez konieczność wydzielenia stałego, zwykle niewielkiego obszaru, w jakim rozpoznaje się gesty.

W porównaniu do dotychczas wymienionych technologii, użycie pojedynczej kamery wiąże się z licznymi korzyściami. Wiele urządzeń posiada je wbudowane lub umożliwia ich podłączenie, są łatwo dostępne i przenośne. Wyzwaniem staje się natomiast interpretacja pozyskanych danych, a jej możliwości i perspektywy zastosowań zależą przede wszystkim od implementacji.

Interesującym rozwiązaniem są kamery z wbudowanymi algorytmami detekcji obiektów. Mogą to być rekurencyjne albo splotowe sieci neuronowe, takie jak model YOLO[5]. Wykrywają one pozycje oraz typy obiektów widocznych na obrazie, na przykład twarze ludzi albo tablice rejestracyjnych aut. Osiągnięcie ogromnej wydajności - w przypadku YOLO jest to od kilkudziesięciu do kilkuset obrazów na sekundę, zależnie od wersji sieci oraz urządzenia[2] - pozwoliło na pojawienie się również komercyjnych rozwiązań w postaci kamer monitoringu albo systemów ułatwiających nagrywanie i kadrowanie filmów. Ambicją tej pracy jest przegląd narzędzi pod kątem realizacji rozwiązań analogicznych, ale działających w oparciu o rozpoznawanie gestów.

2.3. Dostępne implementacje

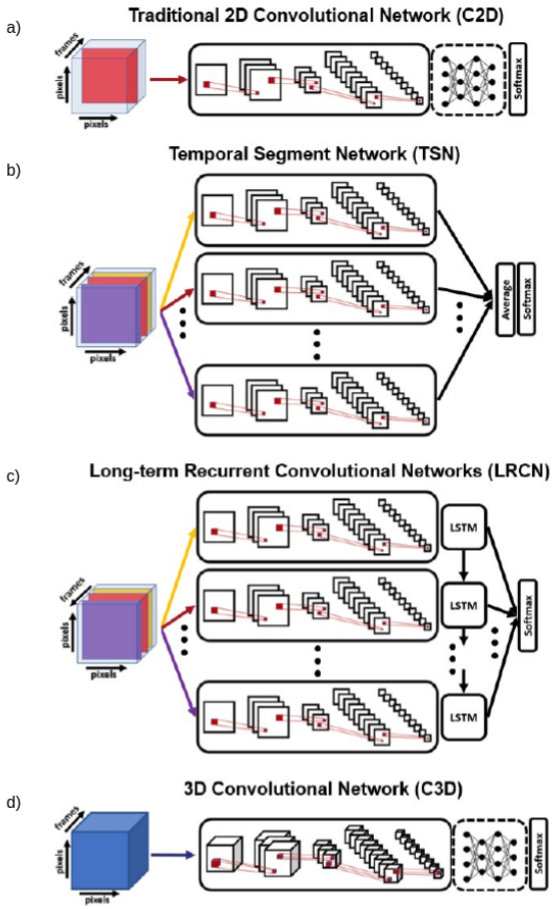
Podczas gdy metody detekcji obiektów w ostatnich latach osiągnęły bardzo wysoką skuteczność, rozpoznawanie akcji wymagających analizy strumieni wideo nadal pozostaje złożonym zagadnieniem. Najczęściej stosowaną techniką jest wykorzystanie splotowych sieci neuronowych (CNN[6]). Modele te można pogrupować na podejścia 2-wymiarowe oraz 3-wymiarowe, w zależności od tego, na jakich danych operują. Na rysunku 2.1 przedstawiono najpowszechniejsze typy sieci stosowane przy rozpoznawaniu gestów. Wśród podejść 2-wymiarowych wskazać należy 2D CNN, TSN[7] oraz LRCN[8]. Spośród sieci trójwymiarowych, stosuje się modele 3D CNN oraz I3D[9][3].

Modele 2D CNN wywodzą się bezpośrednio z metod detekcji obiektów - sieć zasilana jest pojedynczą klatką nagrania, jak widać na rysunku 2.1a. W przypadku sieci TSN, klip wideo segmentuje się w wymiarze czasu na określoną liczbę fragmentów, a z każdego fragmentu wybierana jest jedna klatka poddawana klasyfikacji w sposób zaprezentowany na rysunku 2.1b. Predykcję uzyskuje się następnie w wyniku uśrednienia pojedynczych wyników. Sieci LRCN działają podobnie, jednak w miejscu uśredniania stosują rekurencyjną sieć LSTM[10],

[5] YOLO - ang. *You Only Look Once*
[6] CNN - ang. *Convolutional Neural Network*
[7] TSN - ang. *Temporal Segment Network*
[8] LRCN - ang. *Long-term Recurrent Convolutional Neural Network*
[9] I3D - ang. *Inflated 3D*
[10] LSTM - ang. *Long-Short Term Memory*

umożliwiającą przechowanie informacji o akcjach wykrywanych w kolejnych fragmentach, co zobrazowano na rysunku 2.1c.

Sieci 3D CNN, widoczne na rysunku 2.1d, stanowią trójwymiarową analogię dla 2D CNN, jedynie dane wejściowe sieci zwiększone są o wymiar czasowy. Modele I3D bazują jeszcze mocniej na sieciach dwuwymiarowych - powstają dzięki przekształceniu wytrenowanego modelu 2D CNN przez powielenie jego filtrów, tworząc wymiar czasowy, a jednocześnie zachowując uzyskane na etapie trenowania wagi[4].



Rysunek 2.1. Najpowszechniejsze sposoby wykrywania gestów w klipach wideo[3].

Wybór jednego z modeli na potrzeby tej pracy nie jest na pierwszy rzut oka oczywisty. Zarówno podejścia 2-wymiarowe, jak i 3-wymiarowe mają swoje wady: te pierwsze wiążą się z utratą drobnych ilości wartościowych informacji traconych podczas ekstrakcji cech; drugie posiadają złożoność obliczeniową wpływającą negatywnie na wydajność działania, szczególnie jeżeli rozważamy zastosowania mobilne[3]. Rozważanym wyborem był początkowo model o nazwie Kinetics I3D. Ostatecznie jednak, z powodów opisanych w kolejnych rozdziałach, wybór ten uległ zmianie na rzecz sieci Temporal Shift Module.

3. Kinetics I3D

Model Kinetics I3D jest trójwymiarową siecią konwolucyjną opublikowaną w 2017 roku przez należące do Google przedsiębiorstwo DeepMind.

3.1. Architektura

Kinetics I3D bazuje na dwuwymiarowej sieci Inception-V1, która została przekształcona (ang. *inflated*) do trzech wymiarów poprzez powielenie jej filtrów wzdłuż wymiaru czasowego. W ten sposób zwykle kwadratowe filtry *N*x*N* stają się sześciennymi filtrami o wymiarach *N*x*N*x*N*. Dodatkowo sieć rozbudowana została o dodatkowy strumień przepływu danych wykorzystujący przepływ optyczny (ang. *optical flow*). Oba strumienie traktować można jako osobno wytrenowane sieci, jednak w domyślnej konfiguracji działają one równolegle, a uzyskane predykcje ulegają uśrednieniu[5].

Implementacja udostępniona przez DeepMind napisana została w języku Python z wykorzystaniem również rozwijanej przez DeepMind, opartej o TensorFlow, biblioteki Sonnet.

3.2. Dane wejściowe

Model dostarczony przez Deepmind trenowany był na zbiorze Kinetics 400. Jest to biblioteka złożona z ponad 300000 klipów o częstotliwości klatkowania wynoszącej 30 klatek na sekundę oraz długości nie przekraczającej 10 sekund, pochodzących z filmów opublikowanych w serwisie YouTube. Każde z nagrań zawiera akcję należącą do jednej z 400 klas czynności wykonywanych przez ludzi.

3.2.1. Strumień RGB

Dla strumienia RGB, dane wejściowe próbkuje się do częstotliwości klatkowania 25fps oraz skaluje w taki sposób, aby uzyskać krótszy bok klatki o długości 256 pikseli. Klasyfikacji podlega środkowy wycinek nagrania o wymiarach 224 na 224 piksele. Przekłada się to na wejście sieci o wymiarach *(1, num_frames, 224, 224, 3)*, gdzie kolejne wartości to: liczba nagrań w partii, liczba klatek nagrania, wysokość, szerokość, liczba kanałów (RGB). Otrzymaną próbkę danych zapisuje się w formacie .npy[6].

3.2.2. Optical Flow

Strumień *optical flow* wymaga wyznaczenia przepływu optycznego. Po przeprowadzeniu próbkowania do 25fps dokonuje się konwersji nagrania do odcieni szarości, a następnie stosuje się algorytm wyznaczenia przepływu optycznego TV-L1. W sposób taki sam jak dla RGB wybiera się wycinek klipu. Przyjmuje się dwa kanały wyjściowe *optical flow*, uzyskując wejście strumienia o wymiarach *(1, num_frames, 224, 224, 2)*[6].

Przepływ optyczny jest polem wektorowym opisującym przesunięcie każdego piksela między dwoma kolejnymi klatkami. Wykorzystuje się go w wielu dziedzinach: w widzeniu komputerowym umożliwia podążanie za ruchem obiektów, w medycynie pozwala śledzić

ruch narządów wewnętrznych, w samochodach autonomicznych - do obserwacji ruchu pieszych i pojazdów.

TV-L1 (ang. *Total Variation L1*) to jedna z metod iteracyjnego obliczania przepływu optycznego. W każdej kolejnej iteracji algorytm aktualizuje pole wektorowe w taki sposób, aby uzyskać gładkość przepływu sąsiadujących pikseli oraz aby zachować jak najmniejszą różnicę pomiędzy wartością piksela przed i po przesunięciu[7]. Ze względu na konieczność iteracyjnego rozwiązywania równań całkowych algorytm ten jest jednak wymagający obliczeniowo.

3.3. Próby testów i optymalizacji działania

Wstepne testy na niewielkim zbiorze nagrań pochodzących ze zbioru testowego Kinetics 400 nie przynosiły obiecujących rezultatów. Pierwszym wyzwaniem było samo uruchomienie klasyfikacji z zastosowaniem modelu Kinetics I3D, która wymagała bardzo specyficznych wersji bibliotek Sonnet oraz Tensorflow. Po znalezieniu odpowiednich zależności pojawiły się problemy powiązane z alokacją pamięci dla próbek złożonych z więcej niż 100 klatek podczas klasyfikacji przy użyciu Raspberry Pi. Dodatkowo algorytm wyznaczania przepływu optycznego zastosowany przez autorów okazał się obliczeniowo zbyt wymagający dla pozbawionego rozbudowanego GPU Raspberry Pi, dlatego używany był jedynie strumień RGB.

Powyższe zmiany przełożyły się na drastyczny spadek jakości klasyfikacji. Deklarowane przez twórców dokładności top1 74.2% oraz 91.3% nie były nawet zbliżone do otrzymanych - top1 47.3% i top5 69.08%. Jednocześnie czas klasyfikacji pozostawał względnie wysoki, przekraczając 40 sekund dla 10-sekundowych nagrań.

Nie powiodły się również próby optymalizacji pracy sieci przez zastosowanie kwantyzacji czy pruningu. Projekt od czasu udostępnienia nie był aktualizowany, a wymagane wersje zależności nie wspierały tego typu operacji. Konwersja modelu do nowszych wersji bibliotek, na przykład TensorFlow 2, nie tylko mijała się z celem niniejszej pracy, lecz także była skomplikowanym zadaniem, wymagającym na przykład reimplementacji wielu operacji pochodzących z biblioteki Sonnet.

Na tym etapie, nie mając perspektyw poprawy czasu ani dokładności klasyfikacji, podjęto decyzję o zmianie rozwiązania. Model Kinetics I3D, mimo innowacyjności i imponujących wyników przedstawionych kilka lat temu, obecnie cierpi z powodu pozostawionego długu technologicznego, utrutniającego lub uniemożliwiającego wykorzystanie z zachowaniem przyjętych dotychczas założeń.

3.4. Zmiana podejścia

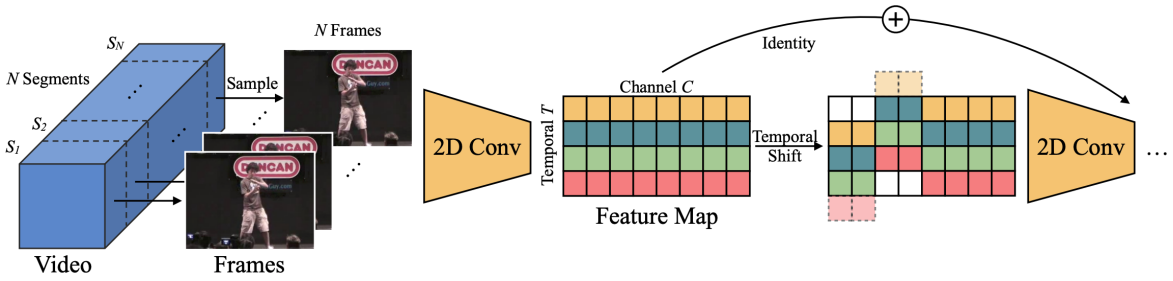
W wyniku napotkanych wydajnościowych oraz technologicznych, rozpoczęto poszukiwania rozwiązania wśród sieci dwuwymiarowych. Szczególnie obiecującym podejściem okazała się być opublikowana w 2019 roku sieć Temporal Shift Module. Stanowi ona modyfikację architektury TSN, która ma umożliwić połączenie wydajności sieci 2D z modelowaniem czasowym porównywalnym z sieciami 3D.

4. Temporal Shift Module

Idea stojąca za pojęciem Temporal Shift Module została przedstawiona w 2019 roku przez zespół MIT HAN Lab. W zaakceptowanym przez ICCV artykule zaproponowano generyczny i wydajny moduł umożliwiający sieciom 2D CNN zachowanie informacji o charakterze czasowym z wydajnością porównywalną z sieciami 3D CNN. Jego działanie opiera się o operację przesunięcia fragmentów map cech uzyskanych w poszczególnych warstwach wzdłuż wymiaru czasowego, pomiędzy sąsiednimi klatkami nagrania[4].

Autorzy udostępnili również funkcjonalną implementację rozwiązania wraz z modelami wytrenowanymi na wybranych zbiorach danych.

4.1. Architektura



Rysunek 4.1. Schemat działania sieci TSN z operacją przesunięcia TSM[8].

Udostępniona implementacja jest realizacją modelu TSN rozszerzonego o moduł TSM. Na rysunku 4.1 zobrazowano działanie zastosowanej operacji. W pierwszej kolejności dokonuje się segmentacji nagrania na N fragmentów. Z każdego fragmentu wybrana zostaje jedna klatka, dlatego w efekcie uzyskana macierz wejściowa jest względnie niewielka - jej wymiary to liczba fragmentów, liczba kanałów (RGB), wysokość i szerokość obrazka. Po przetworzeniu tak wyselekcjonowanych danych wejściowych przez pierwszą warstwę konwolucyjną, część informacji zawartych w uzyskanych mapach cech zostaje przesunięta na sąsiadujące klatki, oznaczone na schemacie odrębnymi kolorami. Operacja ta stosowana jest również po każdej kolejnej warstwie modelu.

Rozwiązanie bazuje na architekturze ResNet-50, zbudowanej dla potrzeb widzenia maszynowego. Jest to struktura 49 połączonych ze sobą warstw konwolucyjnych zakończona warstwą softmax. Warstwy konwolucyjne pogrupowane są w bloki rozbudowane o dodatkowe połączenie skrótowe, zapobiegające degradacji głębokich sieci[9].

Implementacja napisana została w języku Python przy użyciu biblioteki PyTorch. Samą operację przesunięcia TSM zawrzeć można w krótkim fragmencie kodu przedstawionym na listingu 1. Zmienna x jest w tym przypadku wyjściem warstwy o wymiarach: N (*batch size* - liczba nagrań w partii), T (liczba fragmentów), C (uzyskane mapy cech), H i W (wysokość i szerokość klatki). W linii 3 w wyniku dzielenia całkowitoliczbowego otrzymujemy liczbę kanałów, które ulegają przesunięciu w każdą ze stron. W kolejnych liniach następuje samo przesunięcie - najpierw w wyjściowej macierzy *out* zapisujemy części x z przesunięciem czasowym "w lewo"(w przeszłość), następnie kolejne kanały zapisywane

są z przesunięciem "w prawo"(w przyszłość), a wszystkie pozostałe przepisuje się bez przesunięcia.

Listing 1. Implementacja operacji TSM w języku Python[10].

```
# shape of x: [N, T, C, H, W]
out = torch.zeros_like(x)
fold = c // fold_div
out[:, :-1, :fold] = x[:, 1:, :fold]
out[:, 1:, fold: 2 * fold] = x[:, :-1, fold: 2 * fold]
out[:, :, 2 * fold:] = x[:, :, 2 * fold:]
return out
```

4.2. Dane wejściowe

Jako źródło dla danych testowych wybrano omawiany wcześniej zbiór nagrań Kinetics 400. Wybór ten jest bardzo korzystny w kontekście rozważanych zastosowań: nagrania nie są wykonane profesjonalnie, różnią się aspektami takimi jak jakość obrazu, oświetlenie, kadr, aktorzy. Zakres gestów zawiera wiele codziennych czynności dotyczących na przykład gotowania, praktykowania higieny osobistej, uprawiania sportów, gry na instrumentach. Akcje te mają bardziej lub mniej istotny charakter czasowy: gra na gitarze może sprowadzić się do wykrycia obiektu gitary, podczas gdy rozpoznanie stylu pływackiego wymaga porównania kolejnych klatek nagrania[11].

4.2.1. Zbiór testowy

Baza Kinetics 400 wydana została z myślą o uczeniu sieci neuronowych, dlatego też podzielona jest na zbiory: uczący, walidacyjny oraz testowy. Również jeden z udostępnionych przez twórców, wstępnie wytrenowanych modeli sieci TSM trenowany był właśnie przy użyciu tej bazy. Wykorzystano go w ramach tej pracy, jako dane testowe przyjmując 10% zbioru testowego Kinetics 400. Skutkuje to zestawem 4000 nagrań, po 10 na każdą klasę akcji. O ile może wydawać się to niewielką liczbą próbek, pod uwagę należy wziąć takie czynniki jak zajętość przestrzeni dyskowej i czas klasyfikacji prowadzonej na Raspberry Pi.

4.2.2. Przygotowanie danych

Nagrania wchodzące w skład zbioru Kinetics 400 dostępne są do pobrania za pośrednictwem organizacji Common Visual Data Foundation. Oprócz hostingu plików dostarcza ona także narzędzia automatyzujące proces pobierania[12].

Pobrane w formacie mp4 klipy należy przekonwertować do postaci zgodnej z wejściem modelu. W tym celu wykorzystano narzędzie ffmpeg, aby zachowując oryginalną częstotliwość klatkowania 30fps rozdzielić poszczególne klatki i zapisać je w postaci obrazków przeskalowanych do wysokości 331 pikseli. Służące do tego polecenie zawiera się w listingu 2.

Listing 2. Preprocessing danych wejściowych TSM

```
ffmpeg -i video_id.jpg -vf scale=-1:331 -q:v 0 video_id/img_%05d.jpg
```

Operacja przygotowania danych przeprowadzona została na odrębnym urządzeniu i nie obejmują jej testy wydajnościowe przedstawione na dalszym etapie. Wynika to przede wszystkim z faktu, że od urządzenia rozpoznającego gesty w czasie rzeczywistym nie oczekujemy konwersji plików wideo, a wystarczy bieżący zapis klatek w odpowiednim formacie. Przykład działania klasyfikacji online dostępny jest w repozytorium TSM[10].

4.3. Optymalizacja działania

Testy modelu przeprowadzone zostały z zastosowaniem kilku różnych operacji optymalizacyjnych. Należą do nich: modyfikacja liczby fragmentów, pruning oraz kwantyzacja sieci.

4.3.1. Modyfikacja liczby fragmentów

Segmentacja nagrania na fragmenty jest pierwszym etapem procesu klasyfikacji, widocznym na rysunku 4.1. Z każdego fragmentu wybierana jest tylko jedna klatka, dlatego zmniejszanie liczby fragmentów powinno negatywnie wpływać na dokładność klasyfikacji, szczególnie w przypadku gestów o mocno dynamicznym charakterze. Możemy również oczekiwać, że czas klasyfikacji będzie zmieniał się proporcjonalnie do liczby fragmentów.

Implementacja udostępniona przez twórców umożliwia podanie liczby fragmentów jako argument wywołania przekazywany w momencie uruchamiania testów. W ramach tej pracy nagrania dzielone były na 4, 6 oraz 8 fragmentów. Należy mieć na uwadze, że są to wartości względnie małe dla klipów o długości 10 sekund - w najgorszym przypadku wybrane klatki dzieli aż 2,5 sekundy, co może być powodem uzyskania błędnych wyników dla szybko wykonywanych gestów.

4.3.2. Pruning

Pruning jest operacją usuwania parametrów z istniejącej sieci. Ma na celu głównie zmniejszenie rozmiaru modelu, a niekiedy podniesienie szybkości jego działania, przy zachowaniu dotychczasowej dokładności.

Zastosowany w tych testach fragment kodu, widoczny na listingu 3, pozwolił na usunięcie 25% najmniejszych wag w warstwach konwolucyjnych. Realizuje on iterację po każdej warstwie modelu, wykonując na warstwach *Conv2d* polecenie *prune.l1_unstructured*, zerujące jednostki o najniższej normie L1 - w tym przypadku równej wartości bezwzględnej parametru. Wywołanie *prune.remove* usuwa wyzerowane wagi, zmniejszając wagę modelu.

Listing 3. Operacja pruningu przy użyciu biblioteki PyTorch[13].

```
1 for name, module in net.base_model.named_modules():
2   if isinstance(module, torch.nn.modules.conv.Conv2d):
3     prune.l1_unstructured(module, name='weight', amount=0.25)
4     prune.remove(module, 'weight')
```

4.3.3. Kwantyzacja

Kwantyzacja polega na dyskretyzacji wag sieci. Oczekiwanym wynikiem tej operacji jest podniesienie szybkości działania modelu i zmniejszenie jego rozmiaru, jednak może ona negatywnie wpłynąć na dokładność klasyfikacji.

Metody zapisane listingu 4 umożliwiły konwersję modelu w taki sposób, aby wykonywał działania posługując się 8-bitowymi liczbami całkowitymi zamiast, jak poprzednio, 32-bitowymi liczbami zmiennoprzecinkowymi. Służy do tego biblioteka *qnnpack*, dostarczająca implementacje odpowiednich operacji.

Po wywołaniu metody *quantize_base_model* należy przeprowadzić klasyfikację przynajmniej na części danych, aby umożliwić kalibrację - oszacowanie występujących minimalnych i maksymalnych wartości zmiennoprzecinkowych. Metoda *convert_base_model* następnie dokonuje konwersji modelu, aby stosował moduły obsługujące wartości całkowite.

Listing 4. Operacja kwantyzacji przy użyciu biblioteki PyTorch[14].

```
1 def quantize_base_model(self):
2   backend = "qnnpack"
3   self.base_model.qconfig = torch.quantization.get_default_qconfig(backend)
4   torch.backends.quantized.engine = backend
5   self.base_model = torch.quantization.prepare(self.base_model, inplace=True)
6
7 def convert_base_model(self):
8   self.base_model = torch.quantization.convert(self.base_model, inplace=True)
```

5. Testy modelu

W niniejszym rozdziale przedstawiono wyniki przeprowadzonych testów modelu TSM.

5.1. Platforma testowa

Testy odbywały się na komputerze Raspberry Pi 4B wyposażonym w 8GB pamięci RAM oraz 64-bitowy system operacyjny Debian Bullseye.

5.2. Testowane kryteria

Dla każdej kombinacji operacji optymalizacyjnych zbadane zostały następujące miary jakości klasyfikacji: czas, dokładność, czułość oraz precyzja. Dodatkowo poruszona została również kwestia zużycia energii.

5.2.1. Dokładność

Dokładność jest stosunkiem poprawnie zaklasyfikowanych próbek do wszystkich próbek poddawanych klasyfikacji. Podawana jest zazwyczaj w dwóch wariantach pomiaru: top1 oraz top5. Pierwszy z nich - top1 - wyliczany jest w oparciu o pierwszy, najbardziej prawdopodobny rezultat otrzymany na wyjściu sieci neuronowej. Wynik top5 bierze pod uwagę pierwsze 5 klas, które wykryto z największą pewnością.

W przypadku klasyfikacji gestów pomiar top5 jest szczególnie wartościowy, ponieważ niektóre klipy obrazują kilka akcji jednocześnie, jednak posiadają tylko jedną etykietę[11].

Deklarowana przez autorów modelu dokładność na zbiorze danych Kinetics 400 to 74.1% dla top1 oraz 91.2% dla top5[4].

5.2.2. Czułość i precyzja

Rezultat pracy klasyfikatora można przedstawić przy użyciu macierzy błędów. Jest to tabela, w której zestawione zostały ze sobą klasy predykowane z klasami rzeczywistymi. Tabela 5.1 przedstawia macierz błędów dla przypadku klasyfikacji binarnej.

Tabela 5.1. Macierz błędów klasyfikatora binarnego

	wartość rzeczywista: prawdziwa	wartość rzeczywista: fałszywa
predykcja: pozytywna	True Positive (TP)	False Positive (FP)
predykcja: negatywna	True Negative (TN)	False Negative (FN)

Jeżeli klasyfikator przeprowadzi pozytywną predykcję prawdziwej próbki (należącej do danej klasy), otrzymujemy wartość True Positive. Wartość False Negative uzyskujemy, kiedy fałszywa próbka poprawnie zostanie uznana za nienależącą do badanej klasy. Wynik True Negative to próbki z klasy zaklasyfikowane jako do niej nienależące, natomiast False Positive - próbki spoza klasy uznane za należące.

Definicja czułości (ang. *recall*) przy zastosowaniu powyższych pojęć to:

$$Recall = \frac{TP}{TP + FN}$$

W praktyce jest ona miarą tego, jak wiele obiektów należących do klasy zostało rozpoznanych poprawnie.

Definicja precyzji (ang. *precision*):

$$Precision = \frac{TP}{TP + FP}$$

Precyzja określa, jak wiele obiektów przypisanych do klasy rzeczywiście do niej należy.

Pojęcia te możemy rozszerzyć na problemy wieloklasowe. Otrzymane w nich macierze błędów posiadają odpowiednio więcej rzędów i kolumn, jednak jesteśmy w stanie dla każdej klasy z osobna zredukować je do postaci widocznej w tabeli 5.1. Wtedy kolumna zawierająca wartości fałszywe staje się sumą kolumn dla wszystkich klas innych niż aktualnie badana; rząd z predykcjami negatywnymi jest sumą rzędów wszystkich predykcji innych, niż dla obecnej klasy. Wartości czułości i precyzji wyliczamy finalnie poprzez uśrednienie tych wskaźników uzyskanych dla każdej klasy[15].

5.3. Wyniki

Wartości liczbowe każdego z badanych kryteriów dla kombinacji operacji optymalizacyjnych zawierają się w poniższych tabelach 5.2 oraz 5.3.

Tabela 5.2. Dokładność i czas klasyfikacji dla różnych kombinacji operacji optymalizacyjnych.

Zastosowane operacje			Wyniki		
liczba fragmentów	kwantyzacja	pruning	dokładność top1 [%]	dokładność top5 [%]	czas [s]
8	0	0	70.65	89.65	7.26
8	0	1	70.5	89.6	7.407
8	1	0	70.1	89.2	3.38
6	0	0	69.85	88.8	5.61
4	0	0	67.08	87.53	3.71
8	1	1	70.12	89.42	3.27
6	0	1	69.55	88.46	5.603
4	0	1	66.43	87.4	3.857
6	1	0	69.18	88.48	2.535
4	1	0	66.5	87.35	1.674
6	1	1	69.13	88.18	2.552
4	1	1	66.18	87.28	1.709

Pierwszym nasuwającym się spostrzeżeniem dotyczącym tabeli 5.2 jest fakt osiągnięcia czasu krótszego niż czas trwania pojedynczego klipu. Z pewnością możliwe byłoby zatem przeprowadzenie takiej klasyfikacji w czasie rzeczywistym. Zmniejszenie liczby fragmentów, kwantyzacja oraz pruning skutkują nawet 4-krotnym skróceniem czasu klasyfikacji, jednak odbywa się to kosztem dokładności.

Dokładność nawet w najlepszym przypadku nie osiąga spodziewanej wartości. Jest to prawdopodobnie skutek różnic w sposobie pomiaru: wyniki deklarowane przez autorów

uzyskane są poprzez podział nagrania na 10 odcinków równej długości i uśrednienie wyników uzyskanych poprzez klasyfikację każdego wycinka[4]. Powyższe wyniki są natomiast rezultatem jednokrotnej klasyfikacji każdej próbki danych.

Tabela 5.3. Czułość i precyzja klasyfikacji dla różnych kombinacji operacji optymalizacyjnych.

Zastosowane operacje			Wyniki	
liczba fragmentów	kwantyzacja	pruning	precyzja [%]	czułość [%]
8	0	0	72.096	70.7
8	0	1	71.916	70.65
8	1	0	71.799	70.412
6	0	0	71.127	69.85
4	0	0	68.523	67.075
8	1	1	71.71	70.375
6	0	1	71.045	69.55
4	0	1	67.704	66.425
6	1	0	70.8123	69.512
4	1	0	68.196	66.787
6	1	1	70.763	69.337
4	1	1	67.44	66.3

Otrzymane i zawarte w tabeli 5.3 wartości precyzji i czułości są do siebie bardzo zbliżone; są również bardzo bliskie uzyskanej dokładności top1. Żadna z operacji optymalizacyjnych nie wpływa na jeden z parametrów inaczej niż na drugi. Trudno będzie zatem wysnuć dodatkowe wnioski poza tym, że osiągnięta precyzja sieci jest nieco większa niż jej czułość.

5.4. Wnioski

5.4.1. Czas klasyfikacji

Osiągnięcie granicy czasu rzeczywistego umożliwia wykorzystanie rozwiązania w bardzo elastyczny sposób. W tej kwestii spełniamy warunki wszystkich omawianych zastosowań - użytkowych, w edukacji, w bezpieczeństwie.

Oszczędności czasowe wynikające z podjęcia różnego rodzaju optymalizacji sieci stwarzają dodatkowe możliwości. Może to być równoleglizacja klasyfikacji online; równoczesna analiza kilku strumieni wideo mogłaby ograniczyć wydatki na sprzęt i otworzyć rozwiązanie na nowe zastosowania. Możemy na przykład skonfigurować system, w którym jedno Raspberry Pi działające w domowej sieci obsługuje kilka urządzeń korzystających z mechanizmu rozpoznawania gestów. Innym pomysłem jest próba podniesienia dokładności poprzez równoległą analizę kilku wycinków tego samego nagrania.

5.4.2. Dokładność

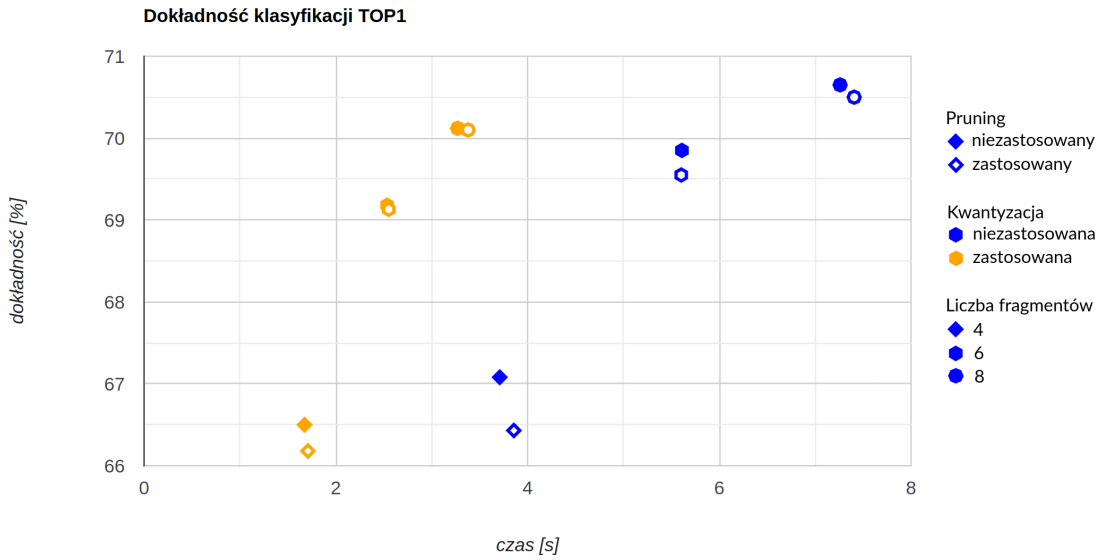
Trudno obiektywnie ocenić, czy uzyskana dokładność jest zadowalająca. Pozytywnym znakiem jest fakt, że otrzymane wartości są zbliżone do oczekiwanych. Sugerując się

testami tej samej sieci trenowanej na innych, mniejszych zbiorach nagrań[3], możemy jednak sądzić, że opracowanie systemu spełniającego konkretny cel i obsługującego niewielką liczbę gestów przyniesie o wiele wyższą dokładność.

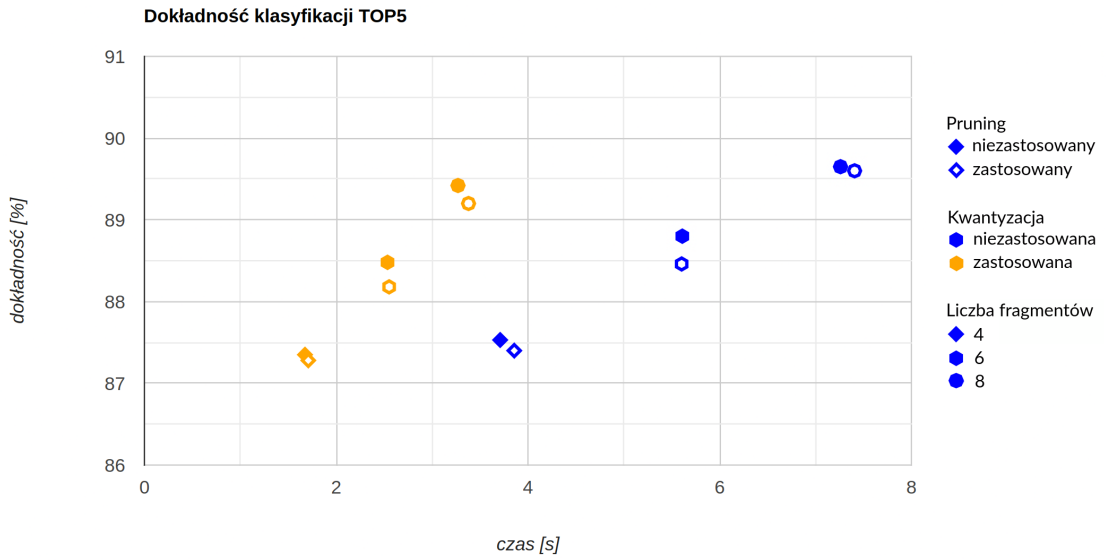
Jednocześnie osiągnięcie dokładności top1 na poziomie 70% nie przekreśla użyteczności rozwiązania. Możemy wyobrazić sobie zastosowanie TSM jako element większego systemu, który czerpie informacje z wielu źródeł. W takim systemie finalna ocena sytuacji zależy również od innych elementów - na przykład działającej równolegle sieci dokonującej detekcji obiektów czy mikrofonu umożliwiającego pomiar natężenia dźwięku. Gest oddania strzału z pistoletu będzie bardziej prawdopodobny, gdy wykryjemy towarzyszący mu hałas; krok taneczny zaznaczony być może odgłosem klaśnięcia w dłonie; detekcja jedynie zwierzęcia na obrazie wykluczy błędnie wykryty gest człowieka. Podczas gdy dane cząstkowe pochodzące z pojedynczych źródeł mogą być tutaj niewystarczające do podjęcia decyzji, odpowiednia ich kombinacja może zapewnić istotnie wyższą użyteczność.

5.4.3. Wpływ operacji optymalizacyjnych

Zebrane dane istotnie zyskują na czytelności po umieszczeniu ich na wykresie dokładności klasyfikacji względem czasu. Otrzymane w ten sposób wykresy 5.1 oraz 5.2 zamieszczono poniżej. Wygląd poszczególnych markerów obrazuje zastosowane podejścia optymalizacyjne - brak wypełnienia oznacza wykorzystanie pruningu, pomarańczowy kolor wskazuje na obecność kwantyzacji, natomiast liczba boków wielokąta to liczba fragmentów, na które TSM segmentuje nagranie przed klasyfikacją.



Rysunek 5.1. Dokładność top 1 klasyfikacji TSM



Rysunek 5.2. Dokładność top 5 klasyfikacji TSM

Charakterystyczną cechą uzyskanych wykresów 5.1 oraz 5.2 jest uporządkowanie punktów w pary. Odpowiedzialnym za to czynnikiem okazuje się być zastosowanie pruningu. Co więcej, w każdym przypadku powoduje on pogorszenie jednego lub obu kryteriów. Jest to w pewnym stopniu spodziewany wynik - zazwyczaj po przeprowadzeniu pruningu ponownie trenuje się sieć, aby dostroić ją do zbioru danych, czego zakres tej pracy nie obejmował. Dodatkowo potencjalne korzyści płynące z operacji usuwania wag zerowych mogą nie wystąpić, jeżeli od początku stosowano właściwą architekturę sieci.

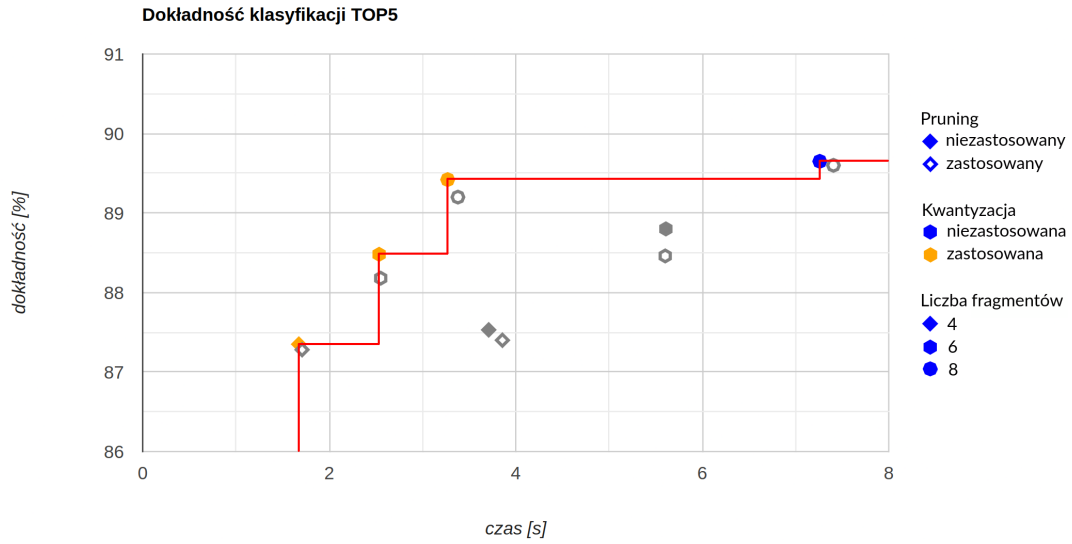
Nieporównywalnie lepsze rezultaty przyniosło wykorzystanie kwantyzacji. Kosztem nieznacznego spadku dokładności - w najgorszym przypadku o 0.67 punkta procentowego - ponad dwukrotnie skróciło ono czas klasyfikacji. Operacja ta powinna być bardzo opłacalna dla większości zastosowań.

Zmniejszenie liczby fragmentów przynosi korzyść czasową, jednak powoduje bardziej zauważalny spadek dokładności niż w przypadku kwantyzacji. Do decyzji potencjalnego użytkownika należy ocena, czy warto segmentować nagrania na mniejszą liczbę fragmentów, w zależności od preferencji i rozważanych zastosowań.

5.4.4. Rozwiązania optymalne w sensie Pareto

Warto zauważyć, że zbiór otrzymanych rozwiązań nie jest zdominowany - nie można wskazać jednej kombinacji zastosowanych operacji, która zapewniłaby najkrótszy czas przy najwyższej dokładności. Przestrzeń tych wyników należałoby więc rozważyć w kategoriach optymalizacji wielokryterialnej. Na wykresie 5.3 zaznaczono rozwiązania optymalne w sensie Pareto.

Funkcja celu w tej sytuacji próbuje maksymalizować dokładność i minimalizować czas. Idealnym rozwiązaniem byłaby 100-procentowa dokładność trwająca 0 sekund, co jest oczywiście niemożliwe. Pozostaje nam jedynie wybór kompromisowego rozwiązania ze zbioru rozwiązań efektywnych.



Rysunek 5.3. Front Pareto dla klasyfikacji top5

Istnieje wiele sposobów wyboru takiego rozwiązania. Możemy zdefiniować nową funkcję celu, na przykład poprzez przypisanie wag każdemu dotychczasowemu kryterium. Innym podejściem jest hierarchizacja kryteriów uszeregowanych od najważniejszego do najmniej ważnego.

5.4.5. Pobór energii

Kwestia poboru energii elektrycznej przez Raspberry Pi nie została zmierzona w ramach pracy. Nie ma jednak podstaw aby sądzić, że wykonywane testy wpłyną na zużycie energii w inny sposób, niż standardowa eksploatacja urządzenia. Istnieją badania[16][17] sprawdzające i modelujące pobór mocy RPi. Stwierdzają one, że najistotniejszymi czynnikami w tej dziedzinie pozostaje wykorzystanie CPU, interfejsów sieciowych oraz urządzeń peryferyjnych.

Mając na uwadze, że przez cały okres testowania procesor komputera był wykorzystany w pełni i nie korzystano w żadnym stopniu z interfejsów sieciowych ani dodatkowych komponentów, możemy założyć, że pobór energii pozostawał stały. Przyjmując maksymalne zmierzone wartości opublikowane w dokumentacji Raspberry Pi[18] otrzymujemy szacowaną moc 6,327W. Podłączenie modułu kamerki dla potrzeb klasyfikacji online może podnieść tę wartość do 7,65W.

6. Podsumowanie

W ramach niniejszej pracy dokonano przeglądu metod rozpoznawania gestów w strumieniach wideo. Jedno rozwiązanie, najlepiej dopasowane do postawionego problemu, zostało poddane szczegółowym testom. Uzyskane wyniki wskazują, że z całą pewnością możliwa jest realizacja wielofunkcyjnych, mobilnych systemów rozpoznawania gestów.

Jednym z największych wyzwań był dobór i uruchomienie rozwiązania na platformie Raspberry Pi. Na wczesnych etapach pracy rozważane były różne modele - niestety zależności części z nich nie były dostępne dla procesorów w architekturze ARM i wymagały ręcznego budowania, inne nie przynosiły zadowalających wyników wydajnościowych. Zmiany podejścia za każdym razem wiązały się z dużym nakładem czasu poświęconego na czytanie dokumentacji, artykułów, ale przede wszystkim na ponowne przeprowadzanie klasyfikacji.

6.1. Perspektywy rozwoju

Jedną ze ścieżek rozwoju projektu jest wykonanie funkcjonalnego systemu, przeznaczonego do rozpoznawania gestów w konkretnym celu. Wymagałoby to zaprojektowania oprogramowania wykorzystującego model w czasie rzeczywistym, skompletowania zbioru danych, wytrenowania sieci.

Bibliografia

[1] J. Galván-Ruiz, C. M. Travieso-González, A. Tejera-Fettmilch, A. Pinan-Roescher, L. Esteban-Hernández i L. Domínguez-Quintana, „Perspective and Evolution of Gesture Recognition for Sign Language: A Review", *Sensors*, t. 20, nr. 12, 2020, ISSN: 1424-8220. DOI: 10.3390/s20123571. adr.: https://www.mdpi.com/1424-8220/20/12/3571.

[2] J. Redmon, S. K. Divvala, R. B. Girshick i A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection", *CoRR*, t. abs/1506.02640, 2015. arXiv: 1506.02640. adr.: http://arxiv.org/abs/1506.02640.

[3] M. Hutchinson, S. Samsi, W. Arcand i in., „Accuracy and Performance Comparison of Video Action Recognition Approaches", wrz. 2020, s. 1–8. DOI: 10.1109/HPEC43674.2020.9286249.

[4] J. Lin, C. Gan i S. Han, „Temporal Shift Module for Efficient Video Understanding", *CoRR*, t. abs/1811.08383, 2018. arXiv: 1811.08383. adr.: http://arxiv.org/abs/1811.08383.

[5] J. Carreira i A. Zisserman, „Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", *CoRR*, t. abs/1705.07750, 2017. arXiv: 1705.07750. adr.: http://arxiv.org/abs/1705.07750.

[6] DeepMind, *Implementation of Kinetics I3D*, Dostęp zdalny (5.02.2024): https://github.com/google-deepmind/kinetics-i3d, 2017.

[7] C. Zach, T. Pock i H. Bischof, „A Duality Based Approach for Realtime TV-L1 Optical Flow", t. 4713, wrz. 2007, s. 214–223, ISBN: 978-3-540-74933-2. DOI: 10.1007/978-3-540-74936-3_22.

[8] HANLab, *TSM: Temporal Shift Module for Efficient Video Understanding*, Dostęp zdalny (5.02.2024): https://hanlab18.mit.edu/projects/tsm/.

[9] K. He, X. Zhang, S. Ren i J. Sun, „Deep Residual Learning for Image Recognition", *CoRR*, t. abs/1512.03385, 2015. arXiv: 1512.03385. adr.: http://arxiv.org/abs/1512.03385.

[10] HANLab, *Implementation of TSM: Temporal Shift Module for Efficient Video Understanding*, Dostęp zdalny (5.02.2024): https://github.com/mit-han-lab/temporal-shift-module, 2019.

[11] W. Kay, J. Carreira, K. Simonyan i in., „The Kinetics Human Action Video Dataset", *CoRR*, t. abs/1705.06950, 2017. arXiv: 1705.06950. adr.: http://arxiv.org/abs/1705.06950.

[12] CVDF, *Kinetics Datasets Downloader*, Dostęp zdalny (5.02.2024): https://github.com/cvdfoundation/kinetics-dataset.

[13] *Dokumentacja PyTorch: Pruning Tutorial*, Dostęp zdalny (5.02.2024): https://pytorch.org/tutorials/intermediate/pruning_tutorial.html.

[14] *Dokumentacja PyTorch: Quantization Recipe*, Dostęp zdalny (5.02.2024): https://pytorch.org/tutorials/recipes/quantization.html.

[15] M. Grandini, E. Bagli i G. Visani, *Metrics for Multi-Class Classification: an Overview*, 2020. arXiv: 2008.05756 [stat.ML].

[16] G. Bekaroo i A. Santokhee, „Power consumption of the Raspberry Pi: A comparative analysis", w *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, 2016, s. 361–366. DOI: 10.1109/EmergiTech.2016.7737367.

[17] N. P. Sheppard, „Power Consumption of the Raspberry Pi", Dostęp zdalny (5.02.2024): https://www.alittleresearch.com.au/sites/default/files/2023-10/PowerConsumptionRaspberryPi_rev_v2.pdf, 2020.

[18] RaspberryPi, *Dokumentacja Raspberry Pi: Power supply*, Dostęp zdalny (5.02.2024): https://github.com/raspberrypi/documentation/blob/develop/documentation/asciidoc/computers/raspberry-pi/power-supplies.adoc.

Wykaz symboli i skrótów

CNN – ang. *Convolutional neural network*
EMG – elektromiografia
HMI – ang. *Human-machine interaction*
ICCV – ang. *International Conference on Computer Vision*
LRCN – ang. *Long-term Recurrent Convolutional Neural Network*
LSTM – ang. *Long-Short Term Memory*
RFID – ang. *Radio frequency identification*
RPi – Raspberry Pi
ToF – ang. *Time of Flight*
TSM – ang. *Temporal Shift Module*
TSN – ang. *Temporal Segment Network*
YOLO – ang. *You Only Look Once*

Spis rysunków

Spis tabel