POLITECHNIKA WROCŁAWSKA

Wydział Podstawowych Problemów Techniki

Praca inżynierska

Rozwój mobilnej aplikacji do rozpoznawania języka migowego z wykorzystaniem transformerów

Development of mobile application for sign language recognition using transformers

Autor: Maciej Grzesik

Opiekun: Prof. uczelni dr hab. Sebastian Kraszewski

Wrocław 2024

Moim ...

Spis treści

1 Wprowadzenie **7**
- 1.1 Motywacja . . . 7
- 1.2 Cel . . . 8
- 1.3 Zakres . . . 8

2 Teoria **9**
- 2.1 Architektura . . . 10
- 2.2 Zasada działania . . . 13
  - 2.2.1 Scaled dot-product . . . 13
  - 2.2.2 Multi-Head Attention . . . 13

3 Metodologia **15**
- 3.1 Implementacja modelu . . . 15
  - 3.1.1 Środowisko programistyczne . . . 15
  - 3.1.2 Przygotowanie danych . . . 16
  - 3.1.3 Analiza obrazów . . . 17
- 3.2 Implementacja aplikacji mobilnej . . . 17
  - 3.2.1 Środowisko programistyczne . . . 17
  - 3.2.2 Clean Architecture . . . 18
  - 3.2.3 BLoC . . . 19
  - 3.2.4 Struktura aplikacji mobilnej . . . 20

4 Wyniki **23**
- 4.1 Funkcjonalność aplikacji . . . 23
- 4.2 Testy aplikacji . . . 23
  - 4.2.1 Wyniki testów . . . 24
- 4.3 Interfejs użytkownika . . . 24
- 4.4 Problemy napotkane podczas implementacji . . . 24

5 Wnioski **25**
- 5.1 Analiza zastosowania transformerów . . . 25
- 5.2 Perspektywa rozwoju . . . 25

6 Podsumowanie **27**

Bibliografia **29**

Rozdział 1

Wprowadzenie

1.1 Motywacja

Wykluczenie społeczne osób głuchoniemych stanowi poważny problem w społeczeństwie, wynikający z braku mozliwości komunikacji werbalnej w sferze publicznej. Uszkodzenie słuchu, szczególnie w życiu codziennym, powoduje trudności w komunikacji z otoczeniem, które opiera się na konwencjonalnej mowie. Rodzi to problem z mozliwością partycypacji w prostych aktywnościach poczynając od uczestniczenia w zyciu społecznym do m.in. jakości otrzymywanych usług medycznych.

Nalezy zaznaczyć, iz znajomość języka migowego wsród ludzi zdrowych ogranicza się do sytuacji w których osoby im bliskie są dotknięte problemem uszkodzonego słuchu. Język ten, będący podstawową formą komunikacji dla osób głuchoniemych nie jest szeroko nauczany a jego znajomość często wymaga zaangazowania się w dodatkowe kursy tudziez szkolenia, nierzadko płatne.

Na podstawie badań Pauliny Malczewskiej wynika, że aż 90% ankietowanych głuchoniemych doświadcza alienacji oraz dyskryminacji ze strony słyszących [1]. Zjawiska te niezaprzeczalnie przyczyniają się do przewlekłego obnizenia nastroju, lęku przed byciem postrzeganym przez społeczeństwo oraz ogólnym spadkiem poczucia własnej wartości i samooceny. Te aspekty stawiają solidny grunt pod rozwój chorób psychicznych tj. depresja, epizody nastroju depresyjnego, zaburzenia adaptacyjne czy fobii społecznej co potwierdzają poszczególne wytyczne diagnostyczne zawarte w DSM-5 [2].

Istotnym jest zaadresowanie tego problemu za pomocą aplikacji wspierającej osoby głuchonieme. Nie tylko ułatwi to komunikację, ale także przyczyni się do zwiększenia inkluzywności społecznej, zapewniając równe szanse i redukując poziom dyskryminacji.

1.2 Cel

Głównym celem pracy jest stworzenie algorytmu opertego na transofrmerach czyli na architekturze głębokiego uczenia maszynowego, którego zadaniem będzie klasyfikacja znaków języka migowego. Istotnym jest zasięgnięcie do rozwiązań opartych na uczeniu maszynowym gdyz rozwiązania oparte na algorytmice nie sprawdzają się w przypadkach dotyczących widzenia komputerowego, które jest znaczącym elementem tej pracy. [elaborate on this] Umozliwi to tłumaczenie języka migowego na tekst w czasie rzeczywistym, co ułatwi komunikację osobom głuchoniemym.

Sam algorytm zaimplementowany zostanie w aplikacji mobilnej docelowo dedykowanej na smartfony z systemem oparacyjnym Android oraz iOS. Mozliwe jest to dzięki wykorzystaniu środowiska Flutter, które pozwala na tworzenie oprogramowania opartego o język Dart, a następnie na kompilowanie kodu przy wykorzystaniu natywnych narzędzi kompilacyjnych (Android Studio dla systemu Android; Xcode dla systemu iOS).

1.3 Zakres

W ramach pracy inżynierskiej zaprojektowany zostanie przyjazny interfejs użytkownika, który umożliwi proste korzystanie z aplikacji zarówno przez osoby głuche, jak i słyszące. Wykorzystanie środowiska Flutter pozwoli na implementacje interaktywnego interfejsu użytkownika przy wykorzystaniu wbudowanych funkcjonalności oraz zewnętrznych modułów utrzymywanych przez społeczność programistów. Dodatkowo zostanie wyeliminowana potrzeba pisania kodu w natywnym dla danego środowiska języku, co zapewnia aplikacji możliwość działania na różnych mobilnych systemach operacyjnych.

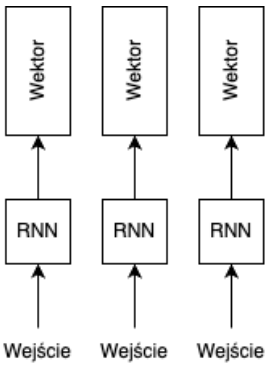
Zaimplementowany zostanie również model mający na celu klasyfikowanie odpowiednich filmów wideo do odpowiadającego im tesktu. W tym etapie projektu zostaną zastosowane odpowiednie algorytmy, które umożliwią skuteczne uczenie modelu na podstawie zebranych danych dotyczących gestów języka migowego. Następnie za pomocą odpowiednio napisanego potoku dane zostaną przesłane do modelu tym samym zaczynając proces uczenia. Proces ten będzie obejmował zarówno fazę wstępną, w której model będzie dostosowywany do charakterystyki danych, jak i fazę walidacji, w której oceni się jego dokładność i zdolność do generalizacji gestów języka migowego. Istotnym jest również przeprowadzenie testów funkcjonalności modelu w warunkach rzeczywistych aby potwierdzić jego poprawne działanie lub wprowadzanie ewentualnych poprawek. Tak skonstruowany i wyuczony model następnie zostanie zkonwertowany do odpowiedniego formatu który następnie zostanie zintegrowany z aplikacja mobilną.

Rozdział 2

Teoria

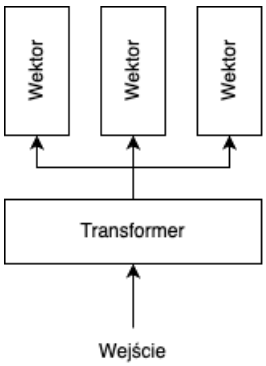
Kluczowym aspektem pracy, na której bazuje duża część funkcjonalności jest model uczenia maszynowego oparty na architekturze transformerów. Przy jego pomocy aplikacja będzie w stanie rozpoznawać znaki języka migowego w czasie rzeczywistym. Transformery stanowią szczególną część szeroko pojętej dziedziny uczenia maszynowego, znajdują one zastosowanie w przetwarzaniu języka naturalnego, wizji komputerowej, ale także w audio i przetwarzaniu multimedialnym. W odróżnieniu od poprzednio stosowanych modeli uczenia maszynowego opierających się na podejściu rekurencyjnym i mechanizmie uwagi, naukowcy pracujący dla firmy Google zaproponowali rozwiązanie wykorzystujące jedynie mechanizm uwagi. Jest to istotny element, który róznicuje transformery od rekurencyjnych modeli uczenia maszynowego. Wyzbycie się rekurencyjności w procesie uczenia przyczyniło się do zwiększenia równoległości, skutkiem czego jest znaczne przyspieszenie czasu uczenia modelu przy zachowaniu wysokiej precyzji [3].

W tradycyjnych modelach rekurencyjnych sieci neuronowych (RNN) dane przetwarzane są sekwencyjnie tzn., że są analizowane krok po kroku (patrz 2.1) [4] co skutkuje ograniczeniami m.in. w równoległości przetwarzaniu danych.



Rysunek 2.1: Przepływ informacji wejściowych w rekurencyjnych sieciach neuronowych

Rozwiązania inżynieryjne wykorzystane w transformerach pozwalają modelom na przetwarzanie wszystkich elementów sekwencji jednocześnie (patrz 2.2).



Rysunek 2.2: Przepływ informacji wejściowych w transformerach

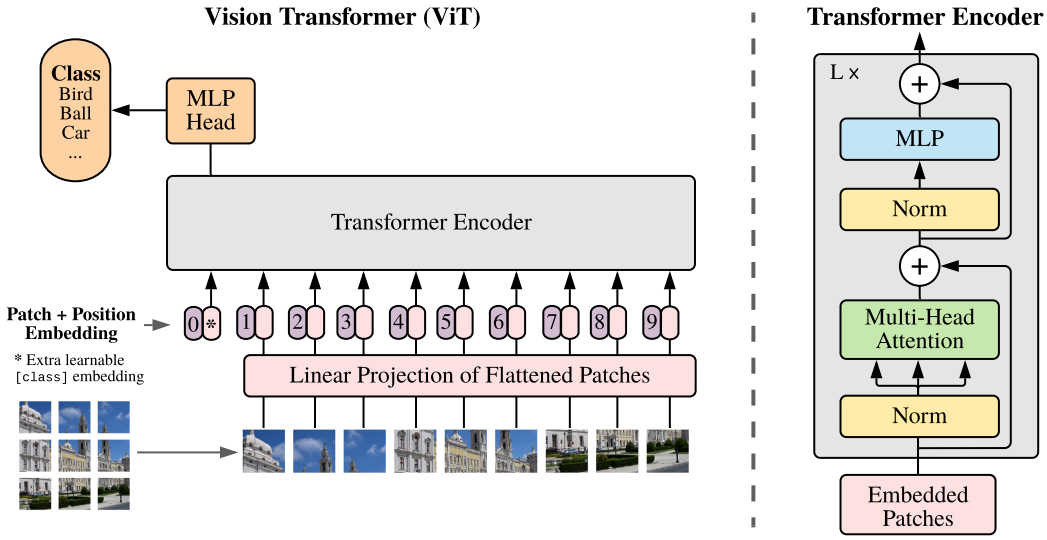
W praktyce oznacza to, że uczenie transformerów przebiega w określonej stałej ilości $O(1)$ sekwencyjnie wykonywanych operacji, modele RNN wymagają natomiast $O(n)$ sekwencyjnie wykonywanych operacji.

2.1 Architektura

Na architekturę transformera składają się dwa główne komponenty: koder oraz dekoder.

Koder składa się z warstwy osadzeń (eng. embedding layer) oraz kodowania pozycyjnego, po których następuje wiele warstw kodera. Warstwa osadzeń zamienia dane wejściowe (słowa, obrazy) na ich reprezentacje w postaci wektorów. Zazwyczaj reprezentacja jest wektorem o wartości rzeczywistej, który koduje znaczenie słowa w taki sposób, że oczekuje się, że słowa znajdujące się bliżej w przestrzeni wektorowej będą miały podobne znaczenie [5]. Osadzanie słów można uzyskać za pomocą modelowania języka i technik uczenia się cech, w których słowa lub frazy ze słownictwa są mapowane na wektory liczb rzeczywistych.

W przypadku modeli Vision Transformer (ViT) proces osadzania obrazów wygląda inaczej niż w przypadku osadzania słów. Obrazy są dzielone na małe, prostokątne fragmenty (eng. patches), które następnie przekształcane są na wektory. Zatem zamiast słow przetwarzane są fragmenty obrazów które dodatkowo mają przypisaną pozycje co pozwala modelowi rozróżnić położenie każdego fragmentu w konteście całego obrazu. W celu przeprowadzenia klasyfikacji jako parametr wejściowy sekwencji podaje się poza odpowiednio wydzielonymi fragmentami obrazu równiez „token klasyfikacyjny” (eng. classification token). Proces ten przedstawiono na schemacie 2.3 [6], a struktura sieci transformerowej jak podnoszą autorzy jest mozliwie najblizsza do struktury zaprezentowanej w artykule „Attention is All You Need”.



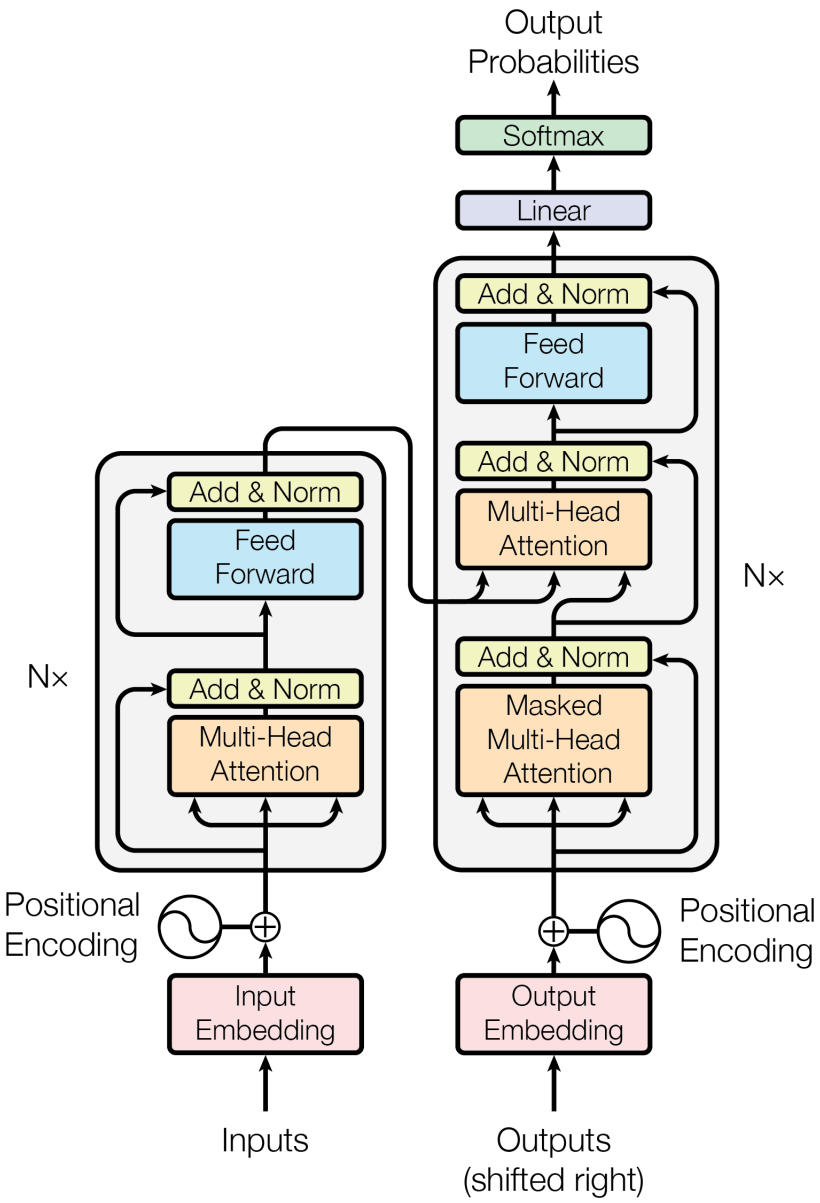
Rysunek 2.3: Schemat uczenia modelu Transformer Vision

Głównym zadaniem kodera jest przekształcanie wejściowych tokenów w kontekstowe reprezentacje. W przeciwieństwie do wcześniejszych modeli, które przetwarzały tokeny niezależnie, koder w transformerze przechwytuje kontekst każdego tokena w odniesieniu do całej sekwencji. Ponieważ transformery nie mają mechanizmu rekurencji, takiego jak RNN, używają kodowania pozycyjnego dodanego do warstwy osadzeń w celu dostarczenia informacji o pozycji każdego tokena w sekwencji. Każda z warstw kodera składa się z dwóch głownych komponentów: mechanizmu samo-uwagi i sieci neuronowej typu feed-forward. Koder pobiera dane wejściowe jako sekwencję wektorów wejściowych, stosuje mechanizm samo-uwagi aby utworzyć pośrednią sekwencję wektorów, a następnie stosuje warstwę feed-forward dla każdego wektora indywidualnie. Warstwy kodera są ułożone w stos, gdzie pierwsza warstwa kodera pobiera sekwencję wektorów wejściowych z warstwy osadzania, tworząc sekwencję wektorów. Ta sekwencja wektorów jest przetwarzana przez drugi koder i tak dalej. Dane wyjściowe z ostatniej warstwy kodera są następnie wykorzystywane przez dekoder.

Dekoder podobnie do kodera, składa się z warstwy embedding layer oraz kodowania pozycyjnego, po której następuje wiele warstw dekodera. Rola dekodera opiera się na tworzeniu sekwencji tekstowych. Każdy dekoder składa się z trzech głównych elementów: maskowanego mechanizmu samo-uwagi, mechanizmu uwagi krzyżowej i sieci neuronowej typu feed-forward. Dekoder działa w podobny sposób jak koder, ale wstawiony jest dodatkowy mechanizm uwagi, który zamiast tego czerpie istotne informacje z kodowań generowanych przez kodery. Mechanizm ten można również nazwać uwagą kodera-dekodera. Podobnie jak pierwszy koder, pierwszy dekoder pobiera informacje pozycyjne i osadzenia sekwencji wyjściowej jako dane wejściowe, a nie kodowania. Transformer nie może wykorzystywać bieżącego lub przyszłego wyjścia do przewidywania wyjścia, więc sekwencja wyjściowa

musi być częściowo zamaskowana, aby zapobiec temu odwrotnemu przepływowi informacji. W przypadku dekodowania, uwaga all-to-all jest nieodpowiednia, ponieważ token nie może zwracać uwagi na tokeny, które nie zostały jeszcze wygenerowane. W ten sposób moduł samo-uwagi w dekoderze jest przyczynowo maskowany. W przeciwieństwie do tego, mechanizm uwagi krzyżowej zwraca uwagę na wektory wyjściowe kodera, które są obliczane przed rozpoczęciem dekodowania przez dekoder. W związku z tym nie ma potrzeby maskowania w mechanizmie uwagi krzyżowej.

Poszczególne bloki architektury wraz z przepływem informacji w modelu zostały zaprezentowane na schemacie 2.4 [3].



Rysunek 2.4: Schemat blokowy architektury transformera

2.2 Zasada działania

Działanie transformera opiera się na mechanizmie tzw. self-attention oraz na omówionych warstwach kodujących i dekodujących. W ramach mechanizmu self-attention wykonywane są dwie główne operacje zwane scaled dot-product attention oraz multi-head attention.

2.2.1 Scaled dot-product

Scaled dot-product attention to działanie polegające na obliczeniu wag dla poszczególnych wartości, tym samym określa, jak bardzo dane fragmenty sekwencji są ze sobą powiązane. Sekwencja wejściowa jest dzielona odpowiednio na wektory K (klucze), Q (zapytania) o długości d_k i V (wartości) o długości d_v. Następnie wykonuje się iloczyn skalarny pomiędzy kluczami i zapytaniami, który dzieli się przez pierwiastek z d_k. Na koniec obliczana jest funkcja softmax w celu obliczenia wag. W praktyce macierz wyników obliczana jest na macierzach kluczy, wartosci i zapytań jak pokazano we wzorze 2.1.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Jak podkreślają autorzy, czynnik czynnik faktoryzujący $\frac{1}{d_k}$ przyczynia się do zwiększenia szybkości i jest bardziej efektywny pod względem wykorzystania pamięci, ze względu na możliwość wykorzystania zoptymalizowanych algorytmów wykonujących obliczenia na macierzach. Rozwiązanie to zostało zaproponowane w kontrze do istniejącej już funkcji uwagi zwanej additive attention.

2.2.2 Multi-Head Attention

Proces ten polega na liniowej projekcji zapytań, kluczy i wartości h razy. Każdy z poszczególnych elementów jest rzutowany na mniejsze przestrzenie za pomocą h różnych, uczonych projekcji liniowych, dzięki czemu uzyskuje się wymiar d_k dla kluczy i zapytań oraz d_v dla wartości. Następnie dla każdej z tych zredukowanych projekcji równolegle wykonywana jest funkcja uwagi co skutkuje uzyskaniem na wyjściu wektora o wymiarze d_v. Po zakończeniu obliczeń przez poszczególne głowy ich wyniki łączone są w jeden wektor co zostało pokazane we wzorze 2.2.

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1,...,\text{head}_h\right)W^O \tag{2.2}$$

Takie podejście sprawia, że model w procesie uczenia jest w stanie jednoczesnie analizować informacje z różnych podprzestrzeni reprezentacji przy zachowaniu podobnego kosztu obliczeniowego, jak w przypadku pojedycznej uwagi.

Rozdział 3

Metodologia

W poniższym rozdziale opisano proces implementacji aplikacji mobilnej oraz modelu rozpoznającego znaki języka migowego. Kluczowe elementy podczas tworzenia oprogramowania obejmowały zarówno odpowiednie przygotowanie danych wraz z etykietami do późniejszej klasyfikacji, stworzenie aplikacji mobilnej jak i integracja modelu z aplikacją.

Tworzenie kodu oparto na zasadach „czystego programowania" (ang. Clean Code). Takie podejście w programowaniu wymaga dbałości o czytelność, modułowość, łatwość rozbudowy oraz testowalność kodu. Wszystkie komponenty kodu zostały zaprojektowane w sposób zapewniający jak największą przejrzystość oraz możliwość wielokrotnego wykorzystania. Starano się implementować kod zgodnie z przyjętymi normami „czystej architektury" (ang. clean architecture) w programowaniu w określonym frameworku.

3.1 Implementacja modelu

3.1.1 Środowisko programistyczne

Wybór środowiska programistycznego w celu implementacji modelu klasyfikującego znaki języka migowego podyktowany jest istniejącymi rozwiązaniami w dziedzinie wizji komputerowej. Ze względu na zastosowanie wytrenowanego modelu oraz ze względu na trywialność w przetwarzaniu danych wybór padł na język Python.

Dodatkową zaletą użytkowania ww. języka programowania wśród innych popularnych języków stosowanych w uczeniu maszynowym tj. R, Java czy C++, jest Anaconda. Anaconda to darmowa, otwartoźródłowa dystrybucja języka Python (oraz R), zawiera ona wiele popularnych bibliotek używanych w uczeniu maszynowym, takich jak NumPy, pandas, scikit-learn, TensorFlow, Keras czy Jupyter Notebook. Pozwala również na zarządzanie wirtualnymi środowiskami,

a wraz z tym instalowaniem poszczególnych pakietów wymaganych do tworzenia oprogramowania.

Model na którym oparto prace pochodzi z badania proponującego przetwarzanie wideo jako sekwencji punktów orientacyjnych (ang. landmarks) dłoni, rąk oraz głowy wyekstraktowanych z poszczególnych klatek [1] [7]. Konkretnym zagadnieniem analizowanym w pracy jest stwierdzenie które punkty orientacyjne są znaczące w procesie klasyfikacji znaków języka migowego. Oprogramowanie dostarczone przez autorów badania stanowi odpowiedni fundament w pracy nad aplikacją mobilną.

Jako model klasyfikujący użyto wyszkolonego modelu „SPOTERnoPE" o dokładności 60.8%, który został wytrenowany na zbiorze stu klas pochodzących ze zbioru danych WLASL100 [2].

Wybór wyuczonego modelu podyktowany jest ograniczonymi zasobami obliczeniowymi, wymaganymi do wytrenowania modelu od podstaw. Proces ten jest wymagający pod kątem nie tylko samej nauki ale i również wstępnego przetwarzania danych. Dane które w kontekście tej pracy składają sie z filmów wymagają odpowiedniego przetworzenia na sekwencje klatek. Dla dużych zbiorów danych które są wymagane w procesie uczenia modelu opartego na architekturze transformerów takie operacje wymagają nie tylko odpowiedniej architektury karty graficznej jak CUDA ale i również znaczącej ilości pamięci RAM.

Istotnym czynnikiem przemawiającym za wyborem wyżej opisanego modelu jest sposób klasyfikacji oparty nie na sekwencji klatek a na sekwencji punktów orientacyjnych. Takie rozwiązanie pozwala modelowi skupić się w procesie uczenia jedynie na punktach orientacyjnych oraz zależnościami między nimi. Tym samym niwelowane są aspekty takie jak m.in. tło czy jakość obrazu, które występowałyby w momencie uczenia na surowych obrazach, a które mogłyby przyczynić się negatywnie do znajdowania relacji pomiędzy sekwencjami klatek w procesie uczenia.

3.1.2 Przygotowanie danych

Zaimplementowany model przetwarza dane wejściowe w postaci tensora składającego się z sekwencji punktów orientacyjnych wyekstraktowanych z klatek filmów wideo. Istotne jest zatem aby dane które mają zostać poddane klasyfikacji przechodziły przez ten sam proces wstępnego przetwarzania co dane użyte w procesie uczenia modelu.

Każdy film zatem dzielony jest odpowiednio na sekwencje klatek przy wykorzystaniu biblioteki OpenCV. Następnie klatki są przetwarzane tak aby uzyskać dane w postaci punktów orientacyjnych przy pomocy biblioteki MediaPipe.

[1] Repozytorium kodu użytego w ww. badaniu dostępne pod tym linkiem.
[2] Repozytorium kodu użytego do pozyskania zbioru danych dostępne pod tym linkiem.

3.1.3 Analiza obrazów

Analiza obrazów odbywa się na serwerze zewnętrznym opartym o framework Flask. Zastosowanie architektury klient-serwer pozwala na utrzymanie modułowości oraz na skalowalność aplikacji. Dodatkową zaletą zastosowania serwera Flask jest łatwe zarządzanie żądaniami HTTP.

Zastosowano stosunkowo prosty serwer zawierający jeden punkt końcowy (ang. endpoint) przyjmujący dane wideo zakodowane w formacie Base64. Przesłane filmy są przetwarzane w celu wyekstraktowania punktów orientacyjnych, które następnie są przekazywane do modelu klasyfikującego. Wyniki klasyfikacji są zwracane do aplikacji mobilnej, gdzie są prezentowane użytkownikowi.

Warto jednak zaznaczyć, iż istnieją rozwiązania pozwalające na implementacje wytrenowanych modeli wewnątrz kodu źródłowego aplikacji mobilnej. Natomiast ze względu na złożoność zarówno implementacyjną jak i obliczeniową zdecydowano się na pominięcie takiego rozwiązania na rzecz architektury klient-serwer.

3.2 Implementacja aplikacji mobilnej

3.2.1 Środowisko programistyczne

Jako środowisko programistyczne do napisania aplikacji mobilnej wybrano Flutter. Flutter jest otwartoźródłowym zestawem do tworzenia oprogramowania stworzonym przez Google, który umożliwia tworzenie aplikacji wieloplatformowych z jednej bazy kodu źródłowego w tym aplikacji mobilnych na platformy Android i iOS.

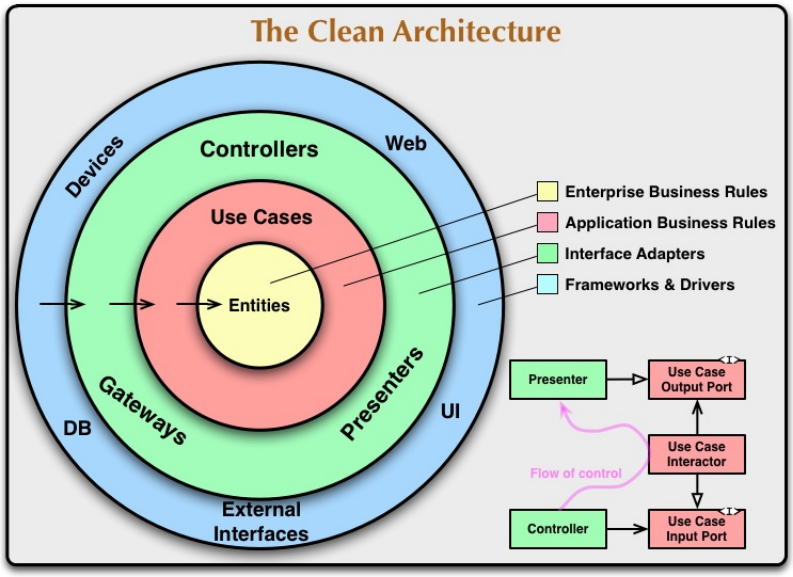
W odróżnieniu od innych frameworków UI, które polegają na platformie docelowej w celu zapewnienia silnika renderującego, Flutter dostarcza aplikacje z własnym silnikiem renderującym „Skia", co pozwala na identyczny wygląd aplikacji na wszystkich platformach docelowych. Silnik ten komunikuje się z natywnymi zestawami SDK (ang. Software Development Kit), co pozwala na dostęp do funkcji specyficznych dla platformy, a w rezultacie pozwala zbudować aplikacje, niezależnie od platformy docelowej.

Flutter oparty jest na języku programowania Dart, który jest językiem zorientowanym obiektowo, ze składnią podobną do języka C oraz posiada wbudowany Garbage Collector. Kod napisany w języku Dart jest kompilowany do kodu maszynowego, JavaScriptu lub WebAssembly [8]. Dodatkową zaletą korzystania z języka Dart jest system zarządzania zależnościami, który pozwala na łatwe dodawanie bibliotek zewnętrznych do projektu.

3.2.2 Clean Architecture

Podczas pisania oprogramowania aplikacji mobilnej zastosowano zasady „czystej architektury" (ang. clean architecture).

Czysta architektura to paradygmat projektowania oprogramowania wprowadzony przez Roberta C. Martina,zwanym w środowisku programistycznym jako „wujek Bob", którego celem jest tworzenie łatwego w utrzymaniu i skalowalnego oprogramowania poprzez organizowanie bazy kodu w odrębne warstwy z wyraźnymi zależnościami i obowiązkami. Filozofia ta opiera się na kilku zasadach, w tym Separation of Concerns (SoC), Dependency Injection i Single Responsibility Principle. Takie podejście pozwala na łatwe rozbudowywanie aplikacji oraz na testowanie poszczególnych komponentów niezależnie od siebie, ponieważ każdy komponent jest od siebie odizolowany [9]. Jak pokazano na rysunku 3.1, architektura zakłada istnienie czterech niezależnych warstw:



Rysunek 3.1: Diagram przedstawiający strukturę „czystej architektury".

1. Entities (Enterprise Business Rules) - jest najbardziej wewnętrzną wartswą. Zawiera obiekty, które reprezentują encje biznesowe, takie jak dane, logika biznesowa, reguły walidacji, itp.

2. Use Cases (Application Buisness Rules) - warstwa ta definuje tzw. przypadki użycia, czyli akcje które użytkownicy mogą wykonać w aplikacji. Logika tej warstwy wywołuje odpowiednie metody z warstwy Entities. Jest ona niezależna od interfejsu użytkownika, bazy danych ani innych zewnętrznych szczegołów.

3. Interface Adapters - warstwa odpowiedzialna jest za komunikacje pomiędzy warstwą przypadków użycia a zewnętrznymi interfejsami, takimi jak m.in. interfejs użytkownika.

4. Frameworks and Drivers - jest to najbardziej zewnętrzna warstwa, zawiera szczegóły implementacyjne takie jak frameworki czy interfejsy użytkownika.

Stosowanie „czystej architektury" pozwala na łatwe wymienianie zależności w przypadku gdy którakolwiek z zewnętrznych części systemu stanie się przestarzała np. zmiana frameworka UI czy bazy danych.

3.2.3 BLoC

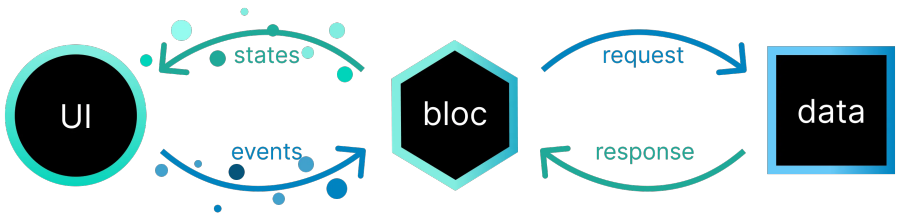
BLoC (Business Logic Component) pozwala na rozdzielenie warstwy prezentacji od logiki biznesowej. BLoC może być efektywnie wykorzystany jako część „czystej architektury", gdzie BLoC pełni rolę warstwy pośredniej między Use Cases a Frameworks and Drivers. Główną ideą BLoC jest zarządzanie stanem aplikacji poprzez przepływ danych oparty na zdarzeniach i reakcjach.

BLoC będąc pośrednikiem pomiędzy wymienionymi warstwami realizuje dwa kluczowe przepływy:

- Zdarzenia (events): Warstwa użytkownika wysyła zdarzenia do BLoC, które reprezentują akcje wykonane przez użytkownika, np. kliknięcie przycisku czy wprowadzenie tekstu w polu formularza.

- Stany (states): W odpowiedzi na zdarzenia, BLoC generuje nowe stany, które są przesyłane z powrotem do warstwy prezentacji. Stany te opisują, jak interfejs użytkownika powinien się zmienić, aby odzwierciedlić aktualny stan aplikacji.

Przepływ danych przedstawiono na rysunku 3.2.

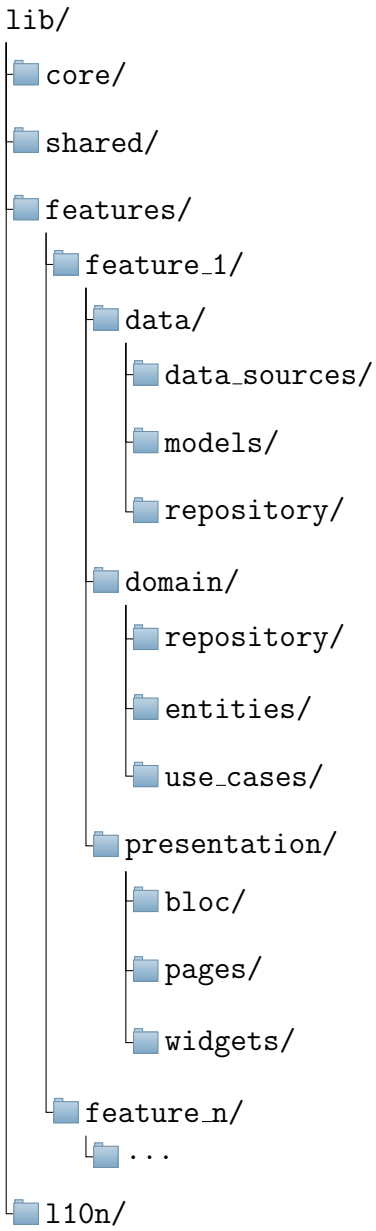


Rysunek 3.2: Schemat działania wzorca BLoC

Taki sposób ogranizacji kodu pozwala zachować jednokierunkowy przepływ danych, który minimalizuje błędy i ułatwia zarządzanie skomplikowanymi przepływami informacji w aplikacji.

3.2.4 Struktura aplikacji mobilnej

Biorąc pod uwagę powyższe założenia, struktura aplikacji mobilnej została zaprojektowana zgodnie z zasadami „czystej architektury" oraz z wykorzystaniem BLoC jak pokazano na rysunku 3.3.



Rysunek 3.3: Struktura katalogów aplikacji mobilnej

Przy tworzeniu odpowiedniej struktury kierowano sie zasadą „features first", co oznacza, że kod powinien być zorganizowany wokół funkcjonalności, a nie typów plików. Takie podejście pozwala na dodawanie nowych funkcjonalności do aplikacji bez konieczności zmiany istniejących plików. Dodatkowo tak struktura

pozwala na ewentualne usunięcie danej funkcjonalności bez negatywnego wpływu na funcjonowanie kodu w pozostałej części aplikacji.

Istotnym problemem podczas implementacji „czystej architektury" w aplikacji mobilnej o stosunkowo niedużej skali jest konieczność napisania kodu „boilerplate", czyli kodu, który jest konieczny do napisania, ale nie wnosi wartości dodanej. Sama struktura kodu początkowo staje się bardziej skomplikowana do implementacji, natomiast w dalszej perspektywie pozwala na zoorganizowane zarządzanie kodem aplikacji.

Rozdział 4

Wyniki

4.1 Funkcjonalność aplikacji

Aplikacja mobilna została oparta o nastepujące funkcjonalności:

- Rozpoznawanie znaków języka migowego: aplikacja mobilna pozwala użytkownikowi na nagranie znaku języka migowego, następnie wynik klasyfikacji przeprowadzony od strony serwera jest wyświetlany na interfejsie użytkownika.
- System rejestracji i logowania: aplikacja mobilna pozwala użytkownikowi na zarejestrowanie się oraz zalogowanie do systemu, zaimplementowaniu zewnętrznego serwisu FirebaseAuth.
- System zarządzania kontem: aplikacja mobilna pozwala użytkownikowi na zarządzanie swoim kontem, w tym zmianę hasła, wylogowanie oraz usunięcie konta. Dodatkowo użytkownik posiada możliwość zresetowania hasła w przypadku jego zapomnienia.
- Możliwość zmiany języka interfejsu: aplikacja mobilna pozwala użytkownikowi na zmianę języka interfejsu, dostępne są dwa języki: polski oraz angielski.
- Możliwość zmiany motywu interfejsu: aplikacja mobilna pozwala użytkownikowi na zmianę motywu interfejsu, dostępne są dwa motywy: jasny oraz ciemny.

4.2 Testy aplikacji

Istotnym elementem tworzenia oprogramowania jest pokrycie kodu testemi, które pozwala zweryfikować poprawność działania aplikacji oraz zapewnić jej niezadowne działanie w różnych warunkach. Aplikacja mobilna została przetestowana przy

pomocy testów jednostkowych. Kluczową role odegrało odpowiednie podejście to architektury systemu, które pozwoliło testować poszczególne komponenty niezależnie od siebie.

4.2.1 Wyniki testów

Wszystkie kluczowe komponenty aplikacji zostały pokryte testami jednostkowymi. Testy wykazały, że logika biznesowa zaimplementowana w BLoC działa poprawnie, a przepływ danych między warstwami w zastosowanej architekturze jest zgodny z oczekiwaniami.

4.3 Interfejs użytkownika

Podczas tworzenia oprogramowania starano się zapewnić odpowiednią strukturę interfejsu użytkownika, tak aby był on intuicyjny oraz łatwy w obsłudze. Kluczowym aspektem podczas prototypowania wyglądu interfejsu było zapewnienie odpowiedniej palety kolorów oraz zapewnienie odpowiedniego kontrastu między poszczególnymi elementami. Zapewnienie odowiedniej palety kolorów pozwala na zwiększenie czytelności interfejsu poprzez m.in. wyszczególnienie ważnych pod względem funkcjonalności elementów.

Zadbano również o to aby interfejs był responsywny oraz dostosowany do różnych rozdzielczości ekranów.

4.4 Problemy napotkane podczas implementacji

Największym problemem napotkanym podczas implementacji aplikacji był sposób w jaki odbywa się klasyfikacja znaków poprzez model, konkretnie, nie jest on w stanie bez odowiedniego przygotowania danych rozpoznawać więcej niż jednego znaku na raz. W tym celu zdecydowano się na ograniczenie nagrywanego znaku do jednego poprzez ograniczenie czasu nagrywania do 2 sekund.

Nie został rozwiązany problem dużej ilości klas, a co za tym idzie, dużą ilość możliwych znaków, przez co model nie jest w stanie rozpoznawać znaków w warunkach rzeczywistych podczas użytkownia aplikacji.

Rozdział 5

Wnioski

5.1 Analiza zastosowania transformerów

Ostatecznnie zastosowany model nie odniósł sukcesu w rozpoznawaniu znaków języka migowego. Ciężko jednoznacznie określić przyczynę takiego stanu rzeczy, jednakże można wskazać na kilka potencjalnych przyczyn. Pierwszą z nich jest zastosowanie modelu wytrenowanego na niewystarczająco dużym zbiorze danych. Wytrenowanie modelu na większym zbiorze danych mogłoby pozwolić na uzyskanie lepszych wyników.

5.2 Perspektywa rozwoju

Istotnym czynnikiem wymagającym dalszego rozwozju jest zastosowanie większego zbioru danych do wytrenowania modelu. W tym celu należałoby zebrać odpowiednie dane w postaci filmów wideo, a następnie odpowiednio je przetworzyć w celu uzyskania danych w postaci sekwencji punktów charakterystycznych dłoni, rąk oraz głowy. Obiecującym zbiorem danych jest zbiór How2Sign, który jest dostępny na licencji niekomercyjnej, jednak tym samym zawiera znaczną ilość danych.

Warto również zaznaczyć, iż zbieranie dodatkowych danych mogłoby być przeprowadzane z poziomu aplikacji mobilnej, gdzie użytkownik mógłby nagrywać swoje znaki języka migowego, a następnie przesyłać je do serwera w celu dalszej analizy. W takim przypadku zastosowanie bazy danych przechowującej nagrane znaki języka migowego byłoby konieczne. Takie rozwiązanie pozwoliłoby na zwiększenie ilości danych treningowych, nie tylko w angielskim języku migoweym ale i również w innych językach.

Takie rozwiązania wymagały by zaprzęgnięcia znacznej ilości zasobów obliczeniowych. Przydatnymi narzędziami w takim przypadku zdecydowanie były roz-

wiązania typu Cloud Computing, które nie tylko pozwoliłyby na przechowywanie danych, ale również na ich przetwarzanie.

Rozdział 6

Podsumowanie

W ramach pracy zaimplementowano aplikacje mobilną pozwalającą na rozpoznawanie znaków języka migowego w czasie rzeczywistym. Aplikacja została zaprojektowana w sposób umożliwiający użytkownikowi dostosowanie języka aplikacji (aktualne wsparcie dla języka polskiego i angielskiego) oraz motywu (jasny, ciemny). Aplikacja została zbudowana i przetestowana pod kątem funkcjonalności na urządzeniu fizycznym opartym o system iOS 18.1.1 oraz na emulatorze systemu operacyjnego Android [wpisac wersje]. Ze względu na specyfikacje pracy emulatora nie udało się sprawdzić poprawności przesyłu nagranego wideo na urzadzeniach opartych o system Android, ponieważ emulator nie wspiera możliwości symulowania obrazu z kamery. Dodatkowo w aplikacji mobilnej zaimplementowano system rejestracji oraz logowania przy użyciu zewnętrznego serwisu jakim jest Firebase. Aktualnie ta funkcjonalność nie niesie za sobą znaczącego wpływu na użytkowanie aplikacji, natomiast w ramach perspektywy rozwoju może pozwolić na wprowadzenie nowych funkcjonalności.

W ramach pracy również zaimplementowano serwer obsługujący klasyfikacje znaków języka migowego. Za pomocą wyuczonego modelu użytkownik jest w stanie z poziomu aplikacji wysłać zapytanie do serwera zawierające nagrany z kamery urządzenia mobilnego film przedstawiający osobę migającą, a następnie jako odpowiedź dostać proponowane słowo. Problemem tego rozwiązania jest relatywnie niska skuteczność zastosowanego modelu. W praktyce klafyikacja przeprowadzana jest z dokładnością wynoszącą 60.8% co często zwraca niepoprawnie sklasyfikowane słowa. Natomiast rozwój aplikacji jest obiecujący pod kątem wykorzystania większego zbioru danych do wytrenowania modelu. Zastosowanie zbioru danych How2Sign pozwoliłoby na nauczenie modelu o znaczącej dokładności, jednakowoż wymagana do tego moc obliczeniowa jest ogromna. Dodatkowo należy zaadresować fakt, iż klasyfikacja odbywa się na podstawie amerykańskiego języka migowego który znacząco różni sie od m.in. polskiego języka migowego. W celu rozszerzenia funkcjonalności o inne języki należałoby przygotować odpo-

wiednie zbiory danych zawierające znaczną ilość mignięć reprezentujących dane słowa (klasy).

Niemniej nawet dla małego zbioru danych zawierających maksymalnie kilka filmów reprezentujących dane słowo skuteczność modelu opartego na architekturze transformerów osiągneła zadziwiający wynik. Biorąc pod uwagę specyfikacje uczenia takich modeli, czyli fakt, iż zazwyczaj wymagana jest większa ilość danych treningowych od sieci konwolucyjnych, to wykorzystanie większego zbioru danych powinno przynieść obiecujące rezultaty warte zbadania.

Bibliografia

[1] P. Malczewska. Izolacja społeczna osób z uszkodzonym słuchem jako wspólny obszar badań pedagogiki i antropologii. *Pedagogika a etnologia i antropologia kulturowa. Wspólne obszary badań*, page 128, 2011. 7

[2] American Psychiatric Association. *Diagnostic and Statistical Manual of Mental Disorders (DSM-5®)*. American Psychiatric Publishing, 2013. 7

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. 9, 12

[4] M. Mamczur. Czym jest i jak działa transformer sieć neuronowa?, March 2020. Dostęp: 23 maj 2024. 9

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 2024. Online manuscript released August 20, 2024. 10

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. 10

[7] Cristina Luna-Jiménez, Manuel Gil-Martín, Ricardo Kleinlein, Rubén San-Segundo, and Fernando Fernández-Martínez. Interpreting sign language recognition using transformers and mediapipe landmarks. In *Proceedings of the 25th International Conference on Multimodal Interaction*, ICMI '23, page 373–377, New York, NY, USA, 2023. Association for Computing Machinery. 16

[8] Dart Team. Dart language - important concepts, 2024. Dostęp: 22 grudnia 2024. 17

[9] Robert C. Martin. The clean architecture, 2012. Dostęp: 22 grudnia 2024. 18