**Wydział Podstawowych Problemów Techniki**

kierunek studiów: **inżynieria biomedyczna**

specjalność: **informatyka biomedyczna**

PRACA DYPLOMOWA

INŻYNIERSKA

**Rozwój mobilnej aplikacji do rozpoznawania języka migowego z wykorzystywaniem transformerów**

Maciej Grzesik

Opiekun pracy

**dr hab. inż. Sebastian Kraszewski**

Wrocław 2024

Spis treści

1. Wprowadzenie 2

2. Teoria 3

2.1. Transformery 3

2.1.1. Zasada działania 3

2.1.2. Architektura 5

3. Metodologia 6

3.1. Projekt interfejsu użytkownika 6

3.2. Implementacja funkcjonalności 6

3.3. Testowanie i optymalizacja 6

3.3.1. Algorytm 6

3.3.2. Aplikacja mobilna 7

4. Implementacja 8

4.1. Środowisko programistyczne 8

4.1.1. Algorytm 8

4.1.2. Aplikacja mobilna 8

4.2. Kontrola wersji 8

5. Dyskusja 9

6. Bibliografia 10

Wprowadzenie

Osoby głuche zmagają się z wykluczeniem społecznym, ze względu na brak możliwości komunikacji za pomocą artykułowanej dźwiękowo mowy. Na podstawie badań Pauliny   
Malczewskiej wynika, że aż 90% ankietowanych głuchych doświadcza alienacji oraz  
dyskryminacji ze strony osób słyszących [1].

Celem pracy jest stworzenie rozwiązania w postaci aplikacji mobilnej, której zadaniem będzie polepszenie zadowolenia z życia wśród osób głuchych. Aplikacja powinna być  
przystępna dla wszystkich osób ją użytkujących, a jej zadaniem będzie poszerzenie  
możliwości nauki języka migowego oraz ułatwienie komunikacji osób głuchych w życiu  
codziennym. Tym samym, zadaniem aplikacji jest zmniejszenie poczucia wykluczenia wśród osób głuchych w społeczeństwie poprzez wykorzystanie rozwiązań inżynierskich. Stworzenie aplikacji wspierającej osoby głuche nie tylko ułatwi komunikację, ale także przyczynie się do  
zwiększenia inkluzywności społecznej, zapewniając równe szanse i redukując poziom  
dyskryminacji.

Aplikacja będzie umożliwiała tłumaczenie tekstu pisanego oraz mówionego na filmy prezentujące poszczególne znaki migowe oraz wtórne tłumaczenie znaków prezentowanych do kamery telefonu na tekst pisany lub mówiony. W tym celu do zaimplementowania aplikacji mobilnej wykorzystane zostanie środowisko Flutter umożliwiające tworzenie wieloplatformowych aplikacji mobilnych, tym samym niwelując potrzebę organizacji środowiska  
projektowego w zależności od systemów operacyjnych użytkowników końcowych.

Główna funkcjonalność aplikacji będzie się opierać na transformerach, czyli modelach głębokiego uczenia maszynowego, gdzie głównym aspektem będzie stworzenie modelu  
przetwarzającego wideo w czasie rzeczywistym na tekst pisany oraz na mowę oraz wtórne  
tworzenie filmów na podstawie tekstu. Transformery są szeroko stosowane w rozwiązaniach  
wymagających przetwarzanie języka naturalnego. Są to między innymi zadania polegające na tłumaczeniu zdań czy rozumieniu wideo. Istotnym jest również fakt, że transformery z dużym sukcesem wykorzystywane są w wizji komputerowej. Wyżej wymienione aspekty wybór transformerów w celu rozwiązania problemu inżynierskiego.

Ze względu na ograniczone zasoby zbiorów danych tj. brak ogólnodostępnych filmów prezentujących znaki polskiego języka migowego za pomocą których można byłoby nauczyć model, wykorzystane zostaną filmy generowane za pomocą modeli przetwarzających tekst na wideo, takich jak Dall-E 3 udostępnionych przez firmę OpenAI czy Imagen Video firmy   
Google’s Brain. Biorąc pod uwagę ograniczenia projektowe tej pracy stworzona aplikacja ma za zadanie funkcjonować jako Minimal Viable Product (MVP).

Teoria

• Transformery

Transformery to zaawansowane modele uczenia maszynowego, szeroko stosowane  
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.  
Skutkiem tego podejścia jest znaczne przyspieszenie czasu uczenia modelu przy zachowaniu wysokiej precyzji [2].

• Zasada działania

Działanie transformera opiera się na warstwie kodującej oraz na warstwie dekodującej. Zadaniem warstwy kodującej jest zakodowanie argumentów wejściowych (np. tekstu, obrazów) do postaci numerycznej. Warstwa dekodująca, wykorzystuje przekazane z warstwy  
kodującej zakodowane informacje do przetworzenia ich na wartości wyjściowe (np. do tekstu czy wideo).

Istotnym elementem, który różnicuje transformery od rekurencyjnych modeli uczenia maszynowego jest mechanizm uwagi. W tradycyjnych modelach rekurencyjnych sieci  
neuronowych (RNN) dane przetwarzane są sekwencyjnie tzn., że są analizowane krok po kroku (patrz Rysunek 1) [3] co skutkuje ograniczeniami m.in. w równoległości przetwarzaniu danych. Uczenie transformerów przebiega w określonej stałej ilości sekwencyjnie  
wykonywanych operacji, modele RNN wymagają natomiast sekwencyjnie  
wykonywanych operacji.

Rysunek 1 Przepływ informacji w rekurencyjnych  
sieciach neuronowych

Rozwiązania inżynieryjne wykorzystane w transformerach pozwalają modelom na przetwarzanie wszystkich elementów sekwencji jednocześnie (patrz Rysunek 2) [3].

Rysunek 2 Przepływ informacji wejściowych  
w transformerach

Mechanizm uwagi pozwala na bezpośrednie ważenie i sumowanie informacji ze wszystkich pozycji w sekwencji. Obliczana jest ona za pomocą wzoru

gdzie Q oznacza macierz zapytania (query), K oznacza macierz klucza (key), V oznacza  
macierz wartości (Value) a dk jest liczbą wymiarów kluczy [2].

Wykorzystanie mechanizmu self-attention pozwala na zmniejszenie odległości  
pomiędzy ścieżkami przechodzącymi przez sieć neuronową. Pozwala to na łatwiejsze uczenie zależności dla dużych zbiorów danych [4].

• Architektura

Rysunek 3 Schemat blokowy architektury transformera

Zanim dane przetwarzane przez transformer zostaną przekazane do kodera muszą  
zostać wstępnie przetworzone. Proces ten można podzielić na dwa kroki:

• Input Embedding polega na zamienieniu danych wejściowych na przystępne dla  
obliczeń komputerowych wektory.

• Positional Encoding składa się z wektorów, których zadaniem jest nadanie kontekstu na podstawie pozycji danych w sekwencji. Jest to istotne, gdyż z punktu widzenia maszyny nie jest wstanie odróżnić ona informacji użytych w różnym znaczeniu.

Sam koder składa się z dwóch elementów:

• Multi-Head Attention polega na obliczeniu wektora uwagi dla poszczególnych  
informacji przekazanych do tej warstwy. W tym etapie obliczany jest tzw. wektor uwagi za pomocą którego określa się wagę poszczególnych informacji. W przypadku przetwarzania wideo identyfikowane są kluczowe momenty takie jak m.in. gesty  
poprzez nadanie wag poszczególnym klatkom wideo w zależności od ich znaczenia. Ignorowane są mniej istotne informacje lub szum.

• Feed Forward, w tym etapie obliczone wektory przekazywane są do sieci MLP (Multi Layer Perceptron), która używana jest dla każdego wektora uwagi.

Tak przygotowane dane zostają przekazane do dekodera, którego zadaniem jest  
przewidywanie kolejnych słów czy obrazów. W tym procesie udział biorą poszczególne bloki:

• Embedding proces ten przebiega dokładnie tak samo jak w przypadku warstwy  
kodującej. Dane zamieniane są na wartości numeryczne w postaci macierzy.

• Positional Encoding również odbywa się w taki sam sposób jak w przypadku warstwy kodującej. Obliczane są wektory nadające kontekst informacjom.

• Masked Multi-Head Attention polega na zamaskowaniu, czyli przemnożeniu przez macierz zer.

• Multi-Head Attention with encoder łączy wektor wyjściowy z warstwy enkodującej  
z wektorem z poprzedniego kroku ze sobą. Podczas tego etapu sprawdzane jest  
w jakim stopniu każdy wektor jest ze sobą powiązany.

• Feed Forward jest to sieć, której zadaniem jest uproszczenie tłumaczenia wektora, aby łatwiej można było przerobić transformerowi wyniki parowań. Następnie w etapie   
linear layer przekształcane są wyniki mające na ten sam wymiar co dane wejściowe. Następnie przy użyciu funkcji softmax zmieniane są wyniki prawdopodobieństwa.

Poszczególne bloki architektury wraz z przepływem informacji w modelu zostały  
zaprezentowane na schemacie (patrz Rysunek 3) [2].

Metodologia

• Projekt interfejsu użytkownika

• Implementacja funkcjonalności

• Testowanie i optymalizacja

• Algorytm

W celu przetestowania zdolności przewidywania znaczenia znaków języka migowego wybrano osiem losowych słów ze zbioru testowego. Wyniki dopasowania wraz z prawdopodobieństwem pewności algorytmu zestawiono w tabeli poniżej (patrz Tabela 1).

Tabela 1 Zestawienie wyników testów dokładności algorytmu  
dla konwersji wideo do tekstu

L.p.

Znak  
migowy

Wynik  
dopasowania

Prawdopodobieństwo  
dopasowania [%]

1

Owoc

Owoc

84.91

2

Księżyc

Księżyc

85.36

3

Lato

Lato

85.18

4

Uśmiech

Uśmiech

85.02

5

Eksplozja

Eksplozja

85.73

6

Ptak

Ptak

84.65

7

Klawiatura

Klawiatura

85.13

8

Myszka

Myszka

85.01

Wyniki testów wskazują, że algorytm skutecznie rozpoznaje i dopasowuje polskie znaki  
migowe do odpowiednich słów. Ze względu na ograniczoną ilość danych oraz ograniczony czas uczenia średnia dokładność testowanych danych wyniosła około 85% (patrz Tabela 1).

Istotne jest również przetestowanie zdolności generowania filmów wideo przez  
algorytm na podstawie tekstu wpisanego przez użytkownika. Wyniki przedstawione w tabeli pokazują jednoznacznie, że algorytm jest w stanie skutecznie przetworzyć tekst na  
odpowiednie sekwencje wideo z wysoką dokładnością (wyższa niż w przypadku konwersji wideo do tekstu). Przedstawiono osiem przykładowych fraz wraz z wynikami dopasowania  
i prawdopodobieństwem pewności algorytmu (patrz Tabela 2).

Tabela 2 Zestawienie wyników testów dokładności algorytmu  
dla konwersji tekst do wideo

L.p.

Fraza  
tekstowa

Wynik  
dopasowania

Prawdopodobieństwo  
dopasowania [%]

1

Jabłko

Jabłko

88.91

2

Noc

Noc

89.36

3

Wakacje

Wakacje

88.18

4

Radość

Radość

87.32

5

Smutek

Smutek

88.73

6

Ptak

Ptak

91.65

8

Przycisk

Przycisk

93.61

9

Komputer

Komputer

89.13

• Aplikacja mobilna

W celu testowania poszczególnych funkcjonalności fragmentów kodu przeprowadzone zostały testy jednostkowe. Testy jednostkowe mają za zadanie sprawdzić funkcjonalność  
pojedynczej klasy, metody czy funkcji a ich celem jest sprawdzenie poprawności logiki,  
obliczeń oraz sytuacji brzegowych testowanych elementów [5].

Dodatkowo przeprowadzone zostały testy widżetów interfejsu graficznego. Testy te mają na celu weryfikacje poprawności zachowań poszczególnych elementów takich jak  
otrzymywanie informacji o wydarzeniu, odpowiadanie na wydarzenie czy sprawdzenie  
poprawności ułożenia na układzie urządzenia [5].

Ze względu na dużą złożoność i czas wykonywania nie zostały przeprowadzone testy integracyjne. Uwzględniają one m.in. testy obciążeniowe i wydajnościowe mające na celu sprawdzenie czy aplikacja funkcjonuje zgodnie z oczekiwaniami pod różnymi warunkami.  
Należy jednak zaznaczyć, iż takie testy są kluczowym etapem w cyklu wdrażania oprogramowania, polegającym na łączeniu poszczególnych modułów aplikacji w całość i testowaniu ich współdziałania [6].

Implementacja

• Środowisko programistyczne

• Algorytm

W celu napisania oraz testowania modelu uczenia maszynowego skorzystano z zintegrowanego środowiska programistycznego PyCharm firmy JetBrains, które jest przeznaczone do tworzenia aplikacji w języku Python. PyCharm oferuje szereg zaawansowanych funkcji, które znacząco ułatwiają pracę nad kodem. Przede wszystkim zapewnia on wygodne  
i przejrzyste środowisko do pisania kodu, z funkcjami takimi jak automatyczne uzupełnianie, linting (analiza kodu pod kątem błędów), refaktoryzacja oraz integracja z systemami kontroli wersji. Ponadto, posiada również wbudowany debugger, który umożliwia dokładną analizę  
i testowanie poprawności funkcjonowania kodu w czasie rzeczywistym.

Jako interpreter wybrano środowisko Conda. Pozwala to na zastosowanie wbudowanych modułów przeznaczonych m.in. do wizualizacji danych oraz szkolenia modeli uczenia maszynowego. Dodatkowo pozwala to na zarządzanie instalowanymi paczkami z zewnętrznych źródeł. Jest to istotny aspekt ze względu na wykorzystanie w projekcie struktury MonAI opartej na strukturze PyTorch.

PyTorch jest otwarto źródłową biblioteką stworzoną w języku Python do uczenia  
maszynowego z wsparciem obliczeń z użyciem procesora graficznego, co jest kluczowe  
w przypadku stosowania obszernych zbiorów danych takich jak przetwarzanie wideo. Na  
podstawie tej struktury powstała struktura MonAi specjalizująca się w zagadnieniach uczenia  
maszynowego z aplikacjami medycznymi, z bezpośrednim naciskiem na obrazowanie  
medyczne [7].

• Aplikacja mobilna

Do stworzenia aplikacji mobilnej wykorzystano zintegrowane środowisko programistyczne Visual Studio Code firmy Microsoft ze względu na dużą możliwość dopasowania funkcjonalności za pomocą wbudowanych w program wtyczek. Skorzystano z tej funkcji  
w celu zainstalowania struktury Flutter [8].

Wybór wyżej wymienionego środowiska jest podyktowany brakiem istniejącego  
rozwiązania od firmy JetBrains.

• Kontrola wersji

W inżynierii oprogramowania istotne jest zarządzanie projektem, pozwala to między innymi na zorganizowane przechowywanie plików, śledzenie zmian czy powrót do  
poprzednich wersji kodu. Systemem kontroli wersji wykorzystanym w projekcie jest Git, który jest jednym z najpopularniejszych i najczęściej stosowanych narzędzi tego typu.

Git pozwala na efektywne zarządzanie kodem źródłowym poprzez tworzenie niezależnych gałęzi (branch), które można swobodnie modyfikować i łączyć (merge) z główną wersją projektu. Dzięki temu osoby zaangażowane w projekt mogą pracować równocześnie nad  
różnymi funkcjonalnościami bez ryzyka konfliktów i utraty danych. System ten zapewnia  
również dokładne śledzenie historii zmian wprowadzanych do projektu, co umożliwia łatwe identyfikowanie i naprawianie błędów [9].

Do zarządzania repozytorium Gita wykorzystano platformę GitHub, która oferuje  
dodatkowe funkcjonalności takie jak zdalne przechowywanie kodu, współpracę zespołową oraz integrację z innymi narzędziami do ciągłej integracji (CI) i ciągłego dostarczania (CD).   
GitHub umożliwia tworzenie żądań ściągnięcia (pull request), które są kluczowym elementem procesu przeglądu kodu (code review). Dzięki nim każdy wprowadzany do głównego  
repozytorium kod może być dokładnie sprawdzony przez innych członków zespołu, co  
zwiększa jakość i bezpieczeństwo projektu.

Dyskusja

Pełny kod źródłowy aplikacji mobilnej, algorytmu opartego na transformerach oraz zbioru danych wykorzystanego do uczenia modelu opartego na transformerze dostępny jest pod   
adresem https://github.com/Maciej-Grzesik

Bibliografia

[1]

P. Malczewska, „Izolacja społeczna osób z uszkodzonym słuchem jako wspólny obszar badań pedagogiki i antropologii,” w *Pedagogika a etnologia i antropologia kulturowa. Wspólne obszary badań*, Cieszyn–Katowice, Wydawnictwo Uniwersytet Śląski w Katowicach, 2011, p. 128.

[2]

A. Vasawani, „Attention is All You Need,” w *Neural Information Processing Systems*, 2017.

[3]

M. Mamczur, „www.miroslawmamczur.pl,” 29 03 2020. [Online]. Available: https://miroslawmamczur.pl/czym-jest-i-jak-dziala-transformer-siec-neuronowa/. [Data uzyskania dostępu: 23 05 2024].

[4]

Y. B. P. F. i. J. S. Sepp Hochreiter, „Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,” 2001.

[5]

Flutter, „docs.flutter.dev,” [Online]. Available: https://docs.flutter.dev/testing/overview. [Data uzyskania dostępu: 26 05 2024].

[6]

M. A. Ould, Testing in Software Development, 1986.

[7]

MONAI, „https://monai.io/,” [Online]. Available: https://monai.io/. [Data uzyskania dostępu: 25 05 2024].

[8]

Microsoft, „code.visualstudio.com,” [Online]. Available: code.visualstudio.com. [Data uzyskania dostępu: 25 05 2024].

[9]

S. Chacon, „https://git-scm.com/,” Software Freedom Conservancy, 2014. [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control. [Data uzyskania dostępu: 25 05 2024].

[10]

S. Jaiswal, „www.datacamp.com,” [Online]. Available: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning. [Data uzyskania dostępu: 25 05 2024].