**Nazwa Projektu:**

Wyrachowana Betty

**Wykonawcy projektu:**

Artur Krepski

Robert Górski

Piotr Zieliński

Wojciech Łosowski

**Zakres Projektu:**

Program służy do wykonywania obliczeń matematycznych, w oparciu wyłącznie o komendy głosowe użytkownika. System ten przeprowadza działania matematyczne objęte zakresem pracy typowego kalkulatora kieszonkowego z wyświetlaczem 8-mio cyfrowym: dodawanie, odejmowanie, mnożenie, dzielenie, pierwiastkowanie stopnia drugiego. Dane do obliczeń podawane są w postaci komend głosowych przez użytkownika. Wynik dokonanych obliczeń, zostaje przekazany użytkkownikowi rónież w formie głosowej.

Program do poprawnego działania wymaga stałego dostępu do internetu.

**Logo projektu:**



**Technologie użyte w projekcie:**

Program zostanie napisany w języku **Python 3+**, ze względu na duży zbiór bibliotek, które są dostępne jako oprogramowanie open source i łatwe w implementacji.

Użyte komponenty oprogramowania:

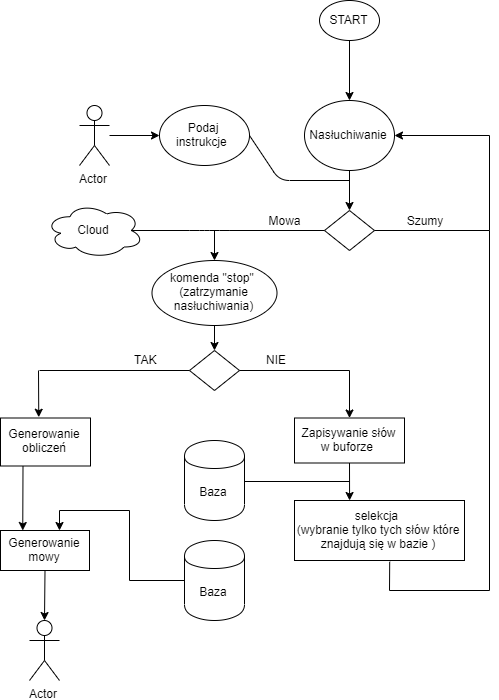
* SpeechRecognition 3.8.1
* pyAudio
* Google - speech  API
* Pyttsx3
* Tkinter

**Założenia funkcjonalne programu:**

1. Program po uruchomieniu, oczekuje na podanie komend przez użytkownika, poprzez prowadzone w pętli pobieranie do bufora 20 ms długości nagrań  sygnałów akustycznych, dokonywanych przez zewnętrzny mikrofon. Mikrofon obsługiwany jest przez aplikację pyAudio. Pobrany sygnał jest analizowany na obecność wzrostu amplitudy, świadczącej o rozpoczęciu podawania komendy, czyli wyznaczeniu tzw onset-u. Niewykrycie onset-u powoduje opróżnienie bufora i wejście programu ponownie do pętli wyznaczania onset-u.
2. Przerwanie pętli wskutek oznaczenia onset-u, rozpoczyna fazę przechwycenia komendy. Od momentu oznaczonego , jako onset, pobrany za pośrednictwem mikrofonu sygnał jest przekazywany poprzez aplikację google - speech API do serwera obliczeniowego Google, w celu wyizolowania słowa kluczowego, “stop”, kończącego podawanie komendy przez użytkownika. Niewykrycie słowa kończącego komendę w ciągu 15s, lub przerwa pomiędzy wypowiadanymi słowami powyżej 3s, powoduje powrót programu do pracy w pętli wyznaczania onsetu.
3. Pobranie słowa kończącego komendę, jest momentem rozpoczęcia konwersji komendy głosowej na symbole kodu ASCII, przez serwer google - speech API. Rozpoznana sekwencja znaków jest pobierana przez program Wyrachowana Betty i poddana analizie w kierunku obecności sekwencji znaków w porządku: liczba, typ działania matematycznego, liczba.
4. Brak takiej sekwencji, powoduje powrót programu do pracy w pętli wyznaczania onsetu.
5. Wraz z momentem oznaczenia wzorcowej sekwencji znaków, program przeprowadza żądane działania matematyczne, a wynik w postaci znaków kodu ASCII poddaje obróbce przez składowe biblioteki Pyttsx3, zamieniające go na sygnał audio, emitowany jako wynik operacji, przez głośnik komputera.

**Schemat systemu:**

Na diagramie poniżej, została przedstawiona ogólna budowa programu, jego komponenty i zależności



**Interface aplikacji:**

W programie przewidziano minimalny panel sterowania z poziomu użytkownika, posiadający jedynie podstawowe opcje pracy (Start, Zakończ), oraz ustawień (wielkość okna programu, opcja pracy w tle). Wynika to z podstawowego założenia funkcjonalnego aplikacji, bazującego na sterowaniu pracą programu poprzez wydawanie komend głosowych.

**Opis oprogramowania**

Kody źródłowe opisanych aplikacji, wraz ze zbiorami próbek audio, znajdują się w zdalnym repozytorium Github:

<https://github.com/msgarski/Wyrachowana_Betty>

**Opis głównego programu:**

Folder Main\_program, zawiera plik: monitorofeverything.py, oraz weight\_best\_cnn.hdf5

**- monitorofeverything.py – autor: Robert Górski;**

Po uruchomieniu, otwierane są 3 wątki:

- w pierwszym wątku, w czasie ciągłym pobiera bloki sygnału dźwiękowego o rozmiarze domyślnym 1024 i przekazuje je do kolejki FIFO

- w drugim wątku, funkcja word\_creating(), analizuje pobrane z kolejki bloki próbek, pod kątem energii sugnału. Po rozpoznaniu początku i końca słowa, wstawia je do drugiej kolejki FIFO.

- w trzecim wątku, funkcja evaluate() odbiera z tej drugiej kolejki FIFO w każdej iteracji po jednym słowie w postaci numpy array i po stworzeniu z niego spektrogramu, przekazuje do funkcji predict(), porównującej słowo z wytrenowanym modelem.

Efektem działania funkcji predict() jest indeks tablicy klas próbek, pod którym zapisana jest klasa, która w tym porównaniu uzyskała najwyższy wynik prawdopodobieństwa. Według podanego indeksu, rozpoznawana jest etykietatej klasy, która przekazana jest do osbnej listy, z której program pobiera ją i traktuje jako składnik działania matematycznego, którego wykonanie jest celem działania programu, jako całości. Wynik działania, obliczony wg normalnych reguł matematyki, jest przekazywany do kolejnego modułu programu WyrachowanaBetty, którym jest moduł: Synteza

Program ulega zamknięciu po rozpoznaniu w wypowiedzi użytkownika słowa: „koniec”, które powoduje wyjście z pętli w funkcji evaluate() i ustawienie flagi zamknięcia wątków. Po ich zamknięciu, zlikwidowaniu ulega również obiekt pyAudio, a cały program powinien przestać działać.

Dostrajanie programu:

Trudność stanowi dobranie następujących współczynników:

- gap\_size = 0.06s - dopuszczalny czas w sek., obniżenia energii wewnątrz słowa

- noise\_multiplier = 1.5 - sztuczny mnożnik wyznaczanego na początku pracy programu poziomu szumu - poziom szumu jest brany jako średnia z wyliczonych wartości rms dla każdej z próbek pobranych w czasie pierwszych 500 ms trwania programu. Średnia nie wydaje się tu odpowiednia, dlatego sztucznie podwyższam ją nieco.

- shortest\_word = 0.15s - założony czas trwania najkrótszego użytecznego słowa - chodzi o wykluczenie z nagrania mmożliwie jak najwięcej przypadkowych trzasków i stuknięć

Przyjęte przeze mnie wstępne wartości dla tych zmiennych, są tak dobrane, aby z jednej strony były dopasowane do rozmiaru bloku(1024, ew 512) i słowo nie było dzielone, a z drugiej strony, aby po wypowiedzeniu jednego słowa nie trzeba było robić nienaturalnych przerw przed następnym.

**- moduł: Synteza – autor: Artur Krepski;**

rolą tego składnika aplikacji jest przekonwertowanie otrzymanej wartości liczbowej wyniku, na sygnał dźwiękowy, a konkretnie, głos ludzki, wypowiadający podany wynik działania.

Zasada tworzenia wypowiedzi, polega na konkatenacyjnej syntezie mowy z gotowych słów, które odtwarzane w odpowiedniej kolejności, mają odwzorować podaną liczbę.

**- moduł Snow\_boy – autor: Piotr Zieliński;**

opis modułu:

Snowboy to działający w czasie rzeczywistym, zawsze nasłuchujący oraz działający w trybie offline wysoce konfigurowalny silnik wykrywania słów kluczowych , który działa na systemach Linux, OS X oraz Raspberry Pi.

Zalety systemu Snowboy:

• wysoce konfigurowalny definiować własnych słów kluczowych

• program nie korzysta z Internetu i nigdzie nie przesyła głosu

• lekki (działa na Raspberry Pi i zużywa mniej niż 10% procesora)

Snowboy jest to szybka alternatywa dla programu służącego do nasłuchiwania oraz wyłapywania i przetwarzania słów. Z punktu programistycznego ma to oczywiście spore minusy gdyż może się okazać, że nie mamy zbyt wielkiego pola manewru jeśli chodzi o algorytmy i funkcje działające nad wychwytywaniem słów kluczowych. Nie mamy również bezpośredniego wpływu na ingerencję w odbierany sygnał. Dzięki temu jednak, system ten jest bardzo szybki i prosty w implementacji.

Jednakże, projekt Snowboy był powierzchownie przez nas zbadany. Możliwe że istnieją możliwości na ustawienie poszczególnych parametrów dotyczących nasłuchiwania, sygnału, szumu, ciszy itp. Na pewno jest to rzecz warta uwagi oraz poświęcenia chwili czasu na zapoznanie się z tym tematem.

Przygotowywanie próbek dźwięku

Przygotowanie próbek odbywanie się poprzez nagranie głosu lub dźwięku poprzez stronę internetową która po zebraniu trzech próbek generuje model w postaci pliku z rozszerzeniem .pmdl

Gotowy model możemy pobrać z witryny oraz bezpośrednio podczepić pod nasz projekt importując go w kodzie do funkcji nasłuchującej.

Możemu również skorzystać z gotowych próbek głosu innych użytkowników dostępnych na portalu.

**Programy narzędziowe i wspomagające pracę nad główna częścią aplikacji:**

**- audiomonitor\_3 threads – autor: Robert Górski;**

moduł, umożliwiający monitorowanie aktywności dźwiękowej użytkownika i wyodrebniający z odbieranych dźwięków, pojedyncze słowa, wypowiedziane przez człowieka. Moduł ten, bazujący na wzajemnej współpracy 3 wątków, stał się podstawą do stworzenia głownej części tytułowego programu. Dokładniejszy opis zasady pracy, został umieszczony przy opisie części głównej tytułowej aplikacji.

**- sample\_recording – autor: Robert Górski;**

program mający za zadanie ułatwienie i przyspieszenie procesu pozyskiwania próbek audio do Deep Learningu. Po uruchomieniu, pobiera zapisane w pliku tekstowym na dysku w bieżącym folderze, listę słów, których wypowiedzenie powinno być zarejestrowane. Pliki, będące zapisem pojedynczego słowa, są następnie zapisywane na dysku w formacie .wav, uzyskując etykiety, składające się z nazwy klasy danej próbki i identyfikatora osoby je wypowiadającej. Założony czas trwania rejestracji jednej próbki to 3 sekundy, częstotliwość próbkowania – 44100, 16bitów, mono. Pliki nie są przez ten skrypt modyfikowane, są zapisywane w stanie surowym.

**- sample\_tuning – autor: Robert Górski;**

w tym katalogu umieszczony jest program trimmer\_audio, którego zadaniem jest opracowanie surowych próbek dźwiękowych w celu ujednolicenia ich długości, co jest konieczne do pracy z nimi przy tworzeniu modelu Keras. Program umożliwia w pierwszej kolejności pozbawienie surowej próbki wszelkiego balastu dźwiękowego, nie będącego zapisem interesujacego nas słowa. W obecnym stanie funkcjonalność ta pozwala na skuteczne wyizolowanie próbkowanych słów w ok 40% próbek. Algorytm identyfikacji początku i końca wypowiadanego słowa wymaga dalszego udoskonalenia, z uwagi na zaburzające jego pracę nieoczekiwane dźwięki otoczenia, będące mylnie interpretowane jako część wypowiadanego słowa. Pozostała część próbek musi być niestety oczyszczana w sposób ręczny, co w moim przypadku zastało wykonane w programie Audacity.

Druga funkcjomnalność tego skryptu polega na uzupełnieniu wyizolowanej pó©bki słowa o fragment ciszy, tak, aby czas trwania całego nagrania wyniósł dokładnie 2 sekundy.

- Folder: **calculator\_deep\_learning** – folder ten zawiera następujące skrypty:

**- interpreter\_of\_predictions.py – autor: Robert Górski;**

skrypt ten nie jest, w odróżnieniu od pozostałych, samodzielnie działającym programem. Jest to implementacja metody, użytej w głównej części tytułowej aplikacji, która służy do przeprowadzenia działania matematycznego, na podstawie dostarczonych, sklasyfikowanych przez model Keras, danych.

Argument, który przyjmuje funkcja, jest indeksem tablicy przewidywań modelu. Wewnątrz funkcji indeks ten jest „konwertowany na korespondującą z nim etykietę klasy i stosownie do rozpoznanego rodzaju działania matematycznego, lub liczby, program przeprowadza to działanie i przekazuje wynikk w postaci argumentu do innej części programu, mającej za zadanie przekonwertowanie tego wyniku na postać audio do wyemitowania użytkownikowi.

**- finding\_class\_name\_in\_model.py – autor: Robert Górski;**

programik powstał w wyniku potrzeby dopasowania wyniku przewidywania, otrzymanego z porównania wyuczonego modelu z nowymi danymi, do nazw klas, korespondujących z tymi wynikami. Nie znajdując innej metody pozyskania nazw klas z zapisanego modelu, przygotowałem program, który pobierając jeszcze raz znane i skategoryzowane dane, wyświetla swoje przypuszczenia na ekranie, uzupełnione o nazwą klasy badanej próbki. Umożliwiło to stworzenie listy etykiet klas i wykorzystanie ich w pracy głównej aplikacji.

**- weights\_best\_cnn.hdf5** – plik jest zapisem modelu, powstałego w wyniku uczenia go na materiale badanym, w sieci konwolucyjnej z trzea warstwami i 120-oma epokami.Wyniki uzyskane na zbiorze testowym ( tu takim samym jak walidacyjny), doszły do 67% skuteczności.

**- learning\_data – autor: Robert Górski;**

folder zawierający cały zasób skategoryzowanych próbek, obejmujących 50 klas, przygotowanych do pobrania przez pracujący generator modelu Keras.

**- deepLearning.py – autor: Robert Górski;**

aplikacja, w której na pierwszym etapie, z zadeklarowanego folderu, pobierane są pojedynczo próbki, z każdej przygotowywany jest MFCC ( mel-spektrogram), a format wszystkich zebranych w postaci struktury numpy ndarray próbek, dostosowywany jest do pracy sieci konwolucyjnej (format danych ma kształt: ilość, wysokość, szerokość, ilość kanałów – co jest odpowiednikiem typowym dla obrazów graficznych, w skali szarości).

Z tak przygotowanych dacych, wtaz z przypisaną do każdej próbki etykietą, wydzielany jest podzbiór testowy, liczący ok 10% całej ilości materiału badawczego, na podstawie którego, wytrenowany w każdej iteracji (epoce) model, sprawdza skuteczność swojej nauki.

Bardzo dobrą praktyką jest, aby wydzielić jeszcze trzeci podzbiór danych, mający na celu poddanie modelu swoistemu egzaminowi po skończonym cyklu nauki. Niestety, z powodu zbyt małej ilości materiału badawczego, te rolę pełni w moim przypadku zbiór testowy. Jest to oczywista niedoskonałość mojego modelu, ponieważ nie powinno dochodzić do sytuacji, gdy sprawdzamy skuteczność wyuczenia modelu na materiale, który on może znać. W dalszym etapie pracy nad tym programem, zamierzam zastosować inną metodę doboru materiału do nauki przez model, tzn: metodę walidacji krzyżowej, która umożliwia dokładniejsze weryfikowanie umiejętności modelu, opartego na pracy z małym zbiorem danych.

Innym pomysłem jest oparcie tworzenia modelu, nie na sieci konwolucyjnej, ale na warstwach gęstych. Poczatkowy wybór, został pokierowany moją małą wiedzą na temat pracy CNN i przekonaniem, że sieci konwolucyjne, jako lepiej radzące sobie w przypadku obrazów, w moim przypadku również pracować będą na swego rodzaju obrazach. Analogia ta nie wydaje mi się jednak do końca słuszna i zamierzam wypróbować drugą wspomnianą opcję.