Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Inżynieria Systemów (INS)**

PRACA DYPLOMOWA
INŻYNIERSKA

Moduł samouczący do systemu automatyki domowej

Przemysław Grzegorz Barcicki

Opiekun pracy
dr. inż. Patryk Schauer

Słowa kluczowe: automatyzacja domowa, uczenie maszynowe, smart-home, python, tensorflow

WROCŁAW 2023

STRESZCZENIE

Tutaj sobie na razie zostawie lorem ipsum, streszczenie napisze pod koniec/później w trakcie pisania.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

ABSTRACT

Same as above c:

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

SPIS TREŚCI

1. WSTĘP

1.1. WPROWADZENIE DO PROBLEMATYKI

Automatyzacja to określenie na metody mające na celu ograniczenie ludzkiej interakcji do minimum w różnych procesach. Stosuje się ją w wielu dziedzinach począwszy na przemyśle, kończąc na procesach informatycznych. Pewnym jednym konkretnym obszarem zyskującym w ostatnim czasie dużo zainteresowania jest pojęcie inteligentnego domu (smart-home). Automatyka domowa to konkretne określenie na zastosowanie automatyzacji wśród urządzeń smart-home skupiających się na obszarze zastosowań gospodarstwa domowego. Celem takiego zastosowania jest kontrola pracy urządzeń znajdujących się w domu do osiągnięcia konkretnego ich stanu w sposób minimalizujący ludzką interakcję. W praktyce takie systemy to bardzo dobre rozwiązanie które niesie ze sobą wiele zalet i uproszczeń w codziennym życiu [16].

Zasada czy reguła w automatyce to w dokładny sposób określenie jakie akcje muszą zostać wykonane pod warunkiem pewnego stanu systemu. Warunkiem początkowym może być każdy stan dowolnego urządzenia w systemie, zmiana stanu dowolnego urządzania lub całkowicie zewnętrznego bodźca. W przypadku takiej automatyzacji często też korzysta się z kombinacyjnego połączenia wielu źródeł i informacji w celu stworzenia stanu początkowego, który jeśli wystąpi, jest przesłanką do wysłania przez system zarządzający urządzeniami polecenia do wykonania pojedynczej lub ciągu akcji. Każde dostępne na rynku dedykowane oprogramowanie obsługujące automatykę domową dostarcza różne narzędzia do szybkiego i intuicyjnego sporządzania takich zasad ale także dostarcza sposoby na komunikację z tymi urządzeniami. Mamy zatem pełny system, który agreguje wiele systemów, protokołów i sposobów wymiany danych w celu monitorowania i sterowania urządzeniami w ramach (lokalnej) sieci.

1.2. OPIS PROBLEMU

Mimo tego, że analiza i projektowanie systemów inteligentnej automatyki domów zaczęła się kilka dekad temu, istnieje wiele nierozwiązanych problemów, które muszą zostać rozwiązane, aby taka forma stała się popularna. Problemami są nieprawidłowe algorytmy oraz niskie celności wynikowe [11].

Mimo wygody w używaniu i łatwości w sporządzaniu zasad automatyki domowych zastosowań pojawiają się pewne trudności. Jednym z nich jest problem ze zmiennością

ludzkich nawyków. Sporządzenie zasad sprawia, że dane czynności są wykonywane wtedy i tylko wtedy gdy zostanie spełniony pewien konkretny stan określony przez nas podczas sporządzania danej reguły. Z logicznego punktu widzenia jest to odpowiednia odpowiedź systemu, natomiast z tego praktycznego już nie, ponieważ dana reguła może zostać wykonana mimo tego, że nie chcieliśmy aby się wykonała. Dodaje to pewien wymóg dodatkowego komplikowania danej zasady poprzez dodatkowe warunki w systemie lub ciągłe jej edytowanie aby odpowiadała naszym zmiennym nawykom.

Bez względu na to jakie narzędzia dostarcza nam dany system, sporządzanie dużej ilości skomplikowanych reguł może być problematyczne. Same środowiska zarządzania takimi systemami mimo wspierania tworzenia na tyle skomplikowanych zasad mogą nie być do tego przystosowane. Często zdarza się, że system automatyki udostępnia dodatkowe, jako wtyczki lub dodatki, moduły do tworzenia reguł za pomocą interpretowanych języków programowania [1], [17], [3]. Tworzy to natomiast zestaw innych problemów, gdzie użytkownik chcący stworzyć takie reguły, musi w przynajmniej podstawowy sposób znać języki języki programowania oraz dodatkowo w jaki sposób sporządzać te skrypty aby mogły być wykonywane przez oprogramowanie nadzorujące.

1.3. CEL PRACY

Celem niniejszej pracy jest opracowanie modułu do systemu automatyki domowej (smart-home) wspierającego tworzenie reguł decyzyjnych w oparciu o mechanizmy uczące się zachowań użytkownika.

1.4. ZAKRES PRACY

Zakres pracy obejmuje przegląd literatury w zakresie działania i budowy systemów wspomagania podejmowania decyzji podejmowania decyzji opartych o uczenie maszynowe.

2. PRZEGLĄD ISTNIEJĄCYCH ROZWIĄZAŃ

W tym rozdziale omówiono literaturę i istniejące rozwiązania problemu rozwiązywanego w dalszej części tej pracy. Pod uwage brano prace, które korzystają z klasycznych metod uczenia maszynowego oraz metod zawierających elementy sieci neuronowych. Analizie zostaną poddane użyte algorytmy jak i podejścia do osiągnięcia celu wspomagania decyzji automatyki domowej.

2.1. SYSTEMY KORZYSTAJĄCE Z ALGORYTMÓW WYKRYWANIA EPIZODÓW

Episode Discovery (wykrywanie epizodów) to metoda data mining'u (kopania danych) wykorzystująca istniejący ciąg występujących po sobie wydarzeń do wykrywania w nich powtarzalnych znaczących epizodów. Wśród epizodów można wyróżnić, tak zwane, epizody znaczące, które według zależnych od algorytmu charakterystyk, określają dany epizod jako często występujący.

Czesto można spotkać się pewną interpretacją ciągu zdarzeń w systemie gdzie każda występująca po sobie w pewnym oknie czasowym akcja jest reprezentowana jako odpowiedni znak ze zbioru wybranych możliwych zdarzeń. W przypadku opisu stanu systemu za pomocą reprezentacji znakowej, korzysta się z metodyki, gdzie duża litera oznacza przejście stanu danego urządzenia w załączone, a mała litera oznacza wyłączenie. W przykładowych ciągach zdarzeń, `baDgb`, `abD`, `Dagb`. Można zauważyć, że zawsze po wystąpieniu wyłączenia urządzenia `a`, następuje wyłączenie urządzenia `b`. W rzeczywistości algorytmy są bardziej zaawansowane i wykrywają epizody znaczące które występują często, ale nie zawsze, ale także operują na dużo dłuższych łańcuchach zdarzeń.

Wykorzystywanie algorytmów wykrywania epizodów na zapisanych już strumieniach wydarzeń pozwala na znalezienie pewnych nawyków i zależności z codziennego korzystania z domowych urządzeń. Znalezione i wyodrębnione epizody mogą zostać użyte z innymi algorytmami w celu podniesienia ich celności. Tak przetworzone dane wejściowe z dodatkowym użyciem innego algorytmu dają zdecydowanie lepsze wyniki niż w wypadku użycia samych sieci neuronowych bądź samego wykrywania epizodów [5], [11].

Ważnym elementem wykorzystania technik wykrywania epizodów jest prawidłowy wybór algorytmu (SPADE, SPEED, WINEPI, ...) jak i jego hiperparametrów, ponieważ inne wartości parametrów wybierających epizod znaczący może mocno wpływać na końcowy wynik [11]. O ile w przypadku algorytmu, nieprawidłowy wybór może ograniczać się do

wydłużonego czasu poszukiwania epizodów, a co za tym idzie, większego zużycia energii przez system, tak w przypadku niewłaściwych parametrów, algorytm może proponować za dużo akcji i nawyków, które źle będą wpływały na końcowy wynik, co w końcu sprawi, że komfort korzystania z takiego systemu będzie mały.

Podejście wykrywania epizodów, jest metodą skupiającą się na pewnych ciągach zdarzeń, które nie biorą pod uwagę żadnych innych zewnetrznych parametrów. Sprawia to, że system uczy się ciągów wydarzeń bez względu na aktualny stan systemu, czyli na przykład porę dnia, temperaturę w pomieszczeniu, dzień tygodnia czy też pogodę. Połączenie klasycznych algorytmów episode discovery z dodatkowymi algorytmami które, biorą pod uwagę inne parametry systemu, rozwiązuje ten problem, ale dodaje dużo skomplikowania w implementacji.

Korzystanie z metody dużych i małych liter do oznaczania wyłączeń i włączeń urządzeń tworzy dodatkowo pewne ograniczenia w postaci braku lub bardzo skomplikowanej obsługi urządzeń o niebinarnych stanach. O ile w przypadku urządzeń gdzie stanów jest kilka, jak na przykład systemy HVAC, można każdy z trybów pracy zinterpretować jako inną czynność, tak w przypadku urządzeń gdzie istnieje teoretycznie nieskończenie wiele stanów pośrednich, tak jak na przykład w ściemniaczach żarówkowych, nie jest możliwe reprezentowanie każdego z nich.

2.2. SYSTEMY KORZYSTAJĄCE Z PROCESÓW MARKOWA

Podczas modelowania zmian stanu systemu automatyki domowej można skorzystać z podejścia gdzie próbujemy opisać ciąg zdarzeń za pomocą prawdopodobieństwa przejść między stanami. Bardzo pomocne w takim podejściu okazuje się korzystanie z modeli Markowa. Istnieje pewne rozszerzenie modeli, które okazuje się bardziej pomocne w wypadku modelowania ludzkiej interakcji i zachowań z powodu uwzględnienia niezależnych i nieznanych przez system stanów, nazywanym ukrytym modelem Markowa. Czyste podejście z prawdopodobieństwem odpowiada na pytanie, jaka jest szansa na wykonanie akcji `A` pod warunkiem tego, że poprzednio wykonana akcja to `B`, dodatkowe ukryte informację pomagają w dokładniejszym określeniu przewidywanego stanu systemu.

Podobnie jak w przypadku wykrywania epizodów (2.1) do wytworzenia reprezentacji zmian systemu wykorzystywany jest literowy zapis, co sprawia, że te podejścia mają takie same ograniczenia. Jedna z prac [19], próbuje rozwiązać problem związany z rzeczywistą (niebinarną) naturą pewnych stanów poprzez kwantyzację w konkretne stany. Konkretne wartości temperaturowe zamieniane są na arbitralny identyfikator wskazujący na daną temperaturę, co używane jest w modelu jako jeden z parametrów.

Podejście znalezione w [19], próbuje także rozwiązać za pomocą autorskiej metody IHMM (Improved Hidden Markov Models) problem powiązania pewnych konkretnych nawyków i ciągów wydarzeń z pewną temporalną zmienną, co sprawia, że system podpo-

wiada konkretne akcje dokładniej o konkretnych porach dnia, lecz dalej nie rozwiązuje problemu rozpoznania np. dni tygodnia czy pogody na dworze.

Inna implementacja [2], korzysta z innego, również autorskiego rozwinięcia modeli markowa TMM (Task-based Markov Model), w której skupia się na zidentyfikowaniu wysokopoziomowych zadań, które to dalej są wykorzystywane do stworzenia pomniejszych modeli reprezentujących ciąg zdarzeń w konkretnym zadaniu. Zadania są identyfikowane na podstawie przerwy pomiędzy kolejnymi wydarzeniami oraz na podstawie dodatkowej informacji o lokalizacji danego urządzenia. Zadania wraz z informacjami o ilości, długości i położenia w przestrzeni używane są przez algorytm k-średnich do pogrupowania zadań i stworzenia zbioru konkretnych ciągów.

2.3. SIECI NEURONOWE

Często spotyka się w parze z innym algorytmem sieci neuronowe, ponieważ są w stanie poprawić zdecydowanie celność predykcji niskim kosztem [5]. Istnieją pewne rozwiązania, gdzie sieci neuronowe są głównym sposobem na przeprowadzania predykcji. Ich szerokie zastosowanie i wiele istniejących rozwiązań sprawia, że są w stanie występować na prawie każdej platformie obliczeniowej. Dodatkowo w ciągu ostatniej dekady widać było zdecydowany rozrost zastosowań i nowych rozwiązań korzystających w pewien sposób z głębokich sieci neuronowych.

Różne istniejące zastosowania i modele głębokich sieci neuronowych daje duże pole do ich zastosowań. W przypadku pracy autorstwa S. Umamageswari [18], zastosowano jedno z popularnych w ostatnim czasie rozwiązań LSTM (long short-term memory). Rekurencyjne sieci neuronowe korzystające z warstw LSTM są w stanie zauważać występujące w dużej odległości danych uczących po sobie pewne zależności. Jest to dobre zastosowanie dla sieci wykorzystujących dużą ilość danych wejściowych zawierające pewne informacje temporalne. Takie modele matematyczne w procesie propagacji wstecznej błędu, są w stanie "nauczyć" się ciagu konkretnych zadań wykonywanych o konkretnych porach dnia, prezentując systemowi przykładowe historyczne wystąpienia tych zadań [12].

Ciekawym podejściem okazuje się wykorzystanie konwolucyjnych sieci neuronowych w przypadku wprowadzania dodatkowych zmiennych do systemu [9]. System w celu określenia następnej akcji którą ma podjąć bierze pod uwagę wyraz twarzy użytkownika. Obraz twarzy, przekształcany jest przez sieci konwolucyjne do zestawu informacji mówiącym o samopoczuciu użytkownika.

2.4. PODSUMOWANIE I WNIOSKI

Biorąc pod uwagę przeprowadzoną analizę, widać, że wstępne przetwarzanie danych i odpowiednie ich spreparowanie jest bardzo kluczowe. Prawidłowa reprezentacja stanu w

systemie jest kluczowa, ponieważ to na podstawie tego system będzie przeprowadzał swoje predykcje. W przypadku tradycyjnych algorytmów uczenia maszynowe widać ich zdecydowane ograniczenia, których rozwiązanie wymaga zdecydowanego komplikowania problemu poprzez wprowadzanie kwantyzowanych informacji czy zmiennych reprezentujących wartości temporalne. Metodą dającą największą elastyczność okazuje się wykorzystanie głębokich sieci neuronowych, ze względu na ich możliwość, przy wcześniejszym odpowiednim przekształceniu wejścia lub wyjścia, wprowadzania danych o różnych formatach, czy sterowania szerokim zakresem różnych urządzeń.

3. WYMAGANIA PROJEKTOWE

Celem tego projektu jest wytworzenie systemu zdolnego do wspomagania użytkownika w eksploatacji inteligentnych urządzeń IoT znajdujących się w jego domowym środowisku poprzez wykonywanie pewnych czynności za użytkownika.

Automatyzacja nie powinna wykonywać się wprost, tj. bez ludzkiej interakcji. Użytkownik korzystający z tego systemu będzie spełniał kluczową rolę, ponieważ wymaga się, aby system dokańczał powtarzalne ciągi zadań wykonywanych przez użytkownika.

Wymagane jest aby rozpoczęcie używania tego systemu wymagała od użytkownika minimum konfiguracji, system powinien pobierać dane z istniejącego już systemu automatyki domowej i używać ich w celu wytworzenia reguł. Dodatkowo, sam system powinien być niezależny od architektury komputera na którym się znajduje. Zaleca się, aby moduł współpracował z gotowymi systemy automatyki domowej. W wypadku utraty przez komputer obsługujący system prądu, system powinien wznowić swoją pracę bez ponownego procesu tworzenia reguł decyzyjnych.

Wymaga się, aby system nie wymagał dużej ilości prac w celu dodania nowych typów źródeł i typów ujść danych, tak aby ewentualne dodawanie obsługi urządzeń dla których nie istnieje gotowe wsparcie było najprostsze. System powinien móc wykorzystywać dodatkowe źródła danych w celach predykcji.

W wypadku innej odpowiedzi systemu niż użytkownik oczekuje, użytkownik powinien mieć łatwy sposób na cofnięcie wykonania błędnej akcji do poprzedniego stanu.

Biorąc powyższe pod uwagę, można określić listę wymagań funkcjonalnych i niefunkcjonalnych tego systemu. Wymagania funkcjonalne:

— wspomaganie codziennej eksploatacji urządzeń w domu,
— cofanie akcji,
— odczyt oraz zapis do pamięci stałej,
— wsparcie istniejącego systemu automatyki,
— dodawanie innych źródeł danych,

Wymagania niefunkcjonalne:

— minimum konfiguracji,
— działanie niezależne od architektury komputera,
— łatwość w tworzeniu rozszerzeń,

4. ARCHITEKTURA

4.1. ARCHITEKTURA ŚRODOWISKA I NARZĘDZIA

Na tym stadium pracy bardzo ważne jest dokładne zbadanie wymogów znajdujących się w rozdziale (3), w celu odpowiedniego przygotowania architektury systemu. Błędny wybór, może skutkować ograniczeniami co do zastosowań tego systemu.

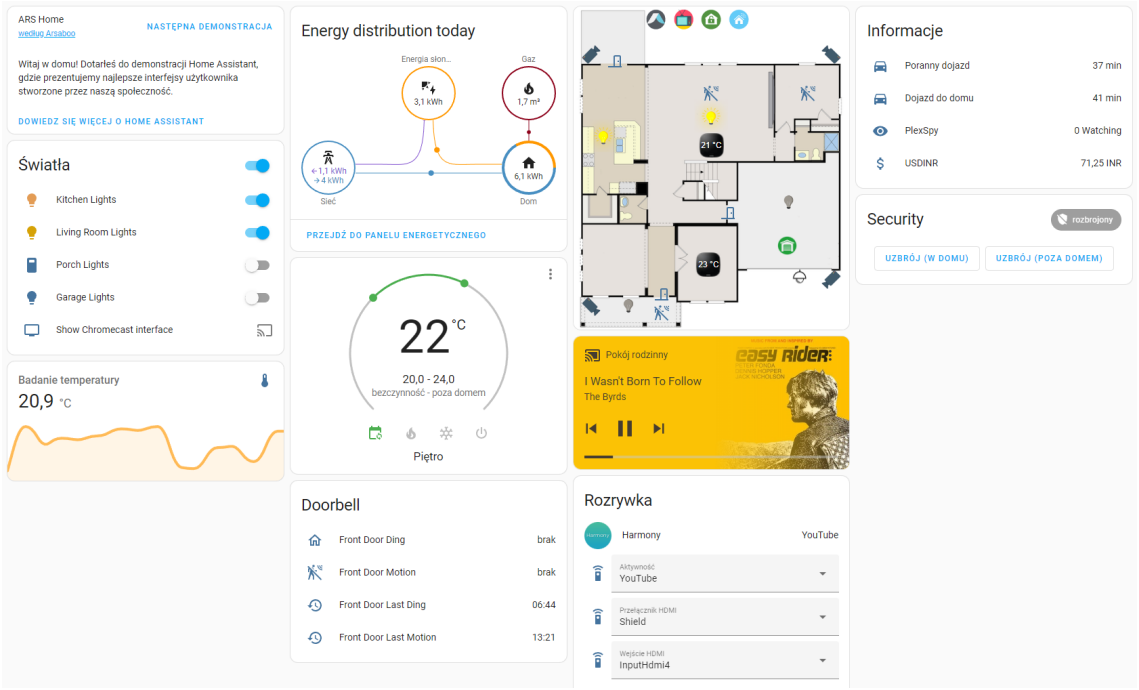
4.1.1. HomeAssistant – HA

Jako system automatyki domowej, kontrolujący wymianę informacji naszego modułu ze światem rzeczywistym wybrano platformę HomeAssistant. Jest to jeden z największych niekomercyjnych systemów tego typu. Jego przewagą w porównaniu do niektórych innych dostępnych na rynku systemów są rzesze fanów i majsterkowiczów, którzy oferują świetne wsparcie i pomagają w rozwiązywaniu problemów. HA skupia się także na prywatności. Programiści open-source jak i użytkownicy systemu zalecają utrzymywanie systemu na swojej własnej infrastrukturze w domu. Dzięki wiernym fanom i samej architekturze projektu system posiada bardzo bogatą bibliotekę integracji z różnymi systemami, od systemów typu Google Assistant w celu dodawania funkcjonalności sterowania głosem, przez własnościowe systemy zarządzania oświetleniem w domu aż po wsparcie dodatkowych bezprzewodowych protokołów komunikacyjnych przeznaczonych do zastosowań domowych. Bardzo ważnym atutem poza szerokim polem różnych integracji jest także system rozszerzeń, gdzie użytkownik może dodać pewną kompletnie nieistniejącą w systemie funkcjonalność do poprawy działania systemu, czy automatyzacji innych rzeczy niezwiązanych z domem.

Na obrazie (4.1) znajduje się dostępne na stronie internetowej demo interfejsu mające na celu pokazanie użytkownikom możliwości całego systemu, ale także sposoby komponowania przykładowego panelu zarządzania domem.

4.1.2. AppDaemon – AD

System pozwalający stworzenie modułu samouczącego powinien dawać maksimum możliwości i niezależności, zatem wybrano AppDaemon. AppDaemon to środowisko wykonawczego Pythona, w wielowątkowej architekturze piaskownicy, służące do pisania aplikacji automatyzacji dla projektów automatyki domowej (i nie tylko), których wymogiem jest solidna architektura sterowana zdarzeniami. AD jest od razu gotowe do współpracy z systemem HomeAssistant, co sprawia, że integracja systemów HA/AD jest bezproblemowa.



Rys. 4.1. Przykładowy panel sterowania domem dostarczony przez HomeAssistant

System pozwala natychmiastowo reagować na zmiany stanów urządzeń znajdujących się w domowym środowisku, poprzez korzystanie z asynchronicznej architektury callback.

4.1.3. Python i Tensorflow - tf

Wykorzystanie struktury HA/AD ogranicza nas do wyboru Pythona jako przewodniego języka programowania w tym przedsięwzięciu. Wybór tego języka nie jest jednak ograniczeniem w tym projekcie, ponieważ posiada on bardzo wielką rzeszę fanów tworzących i udoskonalających paczki kodu dodające nowe możliwości w celu powiększenia pola zastosowań tego języka.

Konkretnym jednym zastosowaniem, który w ostatnim czasie budzi wiele uwagi, jest zastosowanie języka Python do celów uczenia maszynowego. Jedną z bibliotek dostarczających rozwiązania związane z tworzeniem, "uczeniem" i eksploatacją modeli różnego rodzaju sieci neuronowych jest paczka Tensorflow. Tf implementuje wiele gotowych modeli i algorytmów przyspieszających tworzenie modeli sieci neuronowych. Dodatkowo w celu przyspieszenia procesu liczenia błedu propagacji, biblioteka korzysta z różnego rodzaju akceleratorów obliczeń, w tym kart graficznych.

4.1.4. Docker

Ze względu na wymóg niezależności od platformy na jakiej będzie znajdować się ten system, zdecydowano o wyborze jako jednej z głównych technologii, systemu Docker w celu zapewnienia aplikacjom pracującym pod nim odpowiednich warunków niezależnie od systemu operacyjnego na jakim się znajdują. Dodatkowym powodem, który sprawił że

wybrano tą technologię jest gotowe wsparcie systemów HA i AD do pracy w konteneryzowanym środowisku bez dużej ilości konfiguracji. Gotowe obrazy aplikacji znajdują się w sieci, a ich instalacja pod warunkiem posiadania środowiska Docker jest bardzo prosta i szybka.

Dodatkowo narzędzia które dostarcza Docker, pozwalają na tworzenie własnych kontenerów i udostępnianie ich innym co sprawi, że nie będzie to rozwiązanie przygotowywane specjalnie pod konkretne środowisko automatyki.

4.1.5. Pozostałe narzędzia i biblioteki

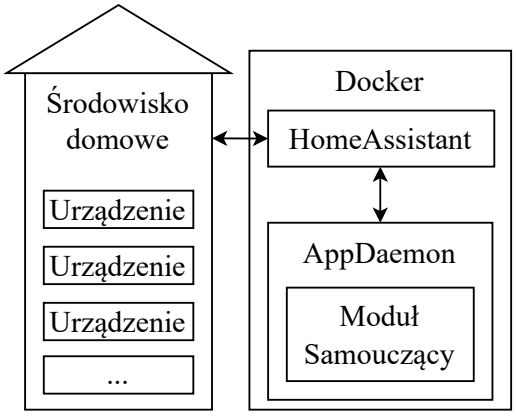
W celu zarządzania środowiskiem programistycznym, wykorzystano platformę Pipenv, która to ułatwia zarządzanie wirtualnymi środowiskami języka Python oraz ułatwia instalację wszystkich bibliotek potrzebnych w danej bazie kodu. Aby wytworzyć gotową paczkę kodu która dalej, która zostanie zainstalowana w AD, wykorzystano narzędzie setuptools. Do wszelakich numerycznych obliczeń oraz interfejsu z Tensorflow została wykorzystana biblioteka numpy. Automatyzowanie środowiska programistycznego zostało wykonane za pośrednictwem Makefile.

W celu poprawy jakości możliwego w przyszłości rozszerzenia projektu, zdecydowano o skorzystaniu z kilku programistycznych udogodnień. Cała praca podczas tworzenia będzie zapisywana w systemie kontroli wersji git, gdzie będzie można cofać lub wprowadzać zmiany w kodzie na różnych gałęziach niezależnie od siebie. Dodatkowo skorzystano z narzędzia pre-commit, dzięki któremu przed każdym zapisem konkretnej wersji modułu do systemu kontroli wersji, na kodzie będzie przeprowadzana statyczna analiza oraz pewne porządki mające na celu poprawę jakości pracy nad kodem.

4.1.6. Podsumowanie

Na obrazie (4.2) znajduje się zaproponowana architektura wysokopoziomowa środowiska. Urządzenia automatyki znajdujące się w domowym środowisku będą komunikowały się z HomeAssistant w celu aktualizowania swojego stanu, ten będzie przesyłany dalej do AppDaemon gdzie będzie znajdował się nasz moduł obsługujący wybrane zdarzenia. Korzystając z informacji o zdarzeniu, system będzie przewidywał następną akcję i wykonywał pozostałą przewidzianą w danym epizodzie zmianę stanu. Informację o poleceniu zmiany stanu będzie obsługiwał AD, który to dalej bedzie przesyłał tą informację do HA, który zdecyduje jak z danym urządzeniem się skomunikować i jaką wiadomość wysłać.

Wewnątrz bloku z modułem samouczącym znajdzie się dopiero kod napisany w języku Python, który będzie reagował na zmiany stanu i w odpowiedni sposób go preparował do przekazania modelowi sieci neuronowych a następnie przekształcany do formy umożliwiającej sterowanie urządzeniami w domu.



Rys. 4.2. Proponowane środowisko rozwiązania problemu.

4.2. ARCHITEKTURA MODUŁU

Ze względu na wymóg obsługi różnych typów urządzeń oraz źródeł danych, bardzo ważnym aspektem do zaprojektowania jest struktura systemu tłumaczącego dane na takie które, zrozumie biblioteka Tensorflow, zaprojektowanie samych modeli głębokich sieci neuronowych, ale także tłumaczenie wygenerowanej przez model odpowiedzi do takiej, którą zinterpretuje reszta modułu.

4.2.1. Dane wejściowe i wyjściowe

System zarządzania automatyką domową podczas swojej pracy zbiera informacje na temat zmian stanów urządzeń którymi steruje i zapisuje je w lokalnej bazie danych. Nie jest ona jednak dostępna w łatwy sposób dla programisty. AppDaemon w swoim interfejsie programistycznym udostępnia możliwość ściągnięcia historii zmian wybranego urządzenia z ostatniego czasu. Dane przekazane z zapytania do aplikacji są w postaci listy słowników języka Python o strukturze zawartej w listingu (4.1). Słownik to pewna struktura danych w której informacja jest zawarta w wartościach dla konkretnego klucza [8].

Warto zauważyć, że informacja zawarta w polu `state`, ma inne znaczenie w zależności od typu urządzenia czy źródła. W przypadku urządzenia typu przerzutnik stabilny o binarnych pozycjach, dane w kluczach `state` oraz `last_changed` wskazuje na czas kiedy zaszła zmiana do jakiego stanu, a w przepadu zwykłego przycisku, wystarcza samo pole ze stanem, ponieważ znajduje się tam czas ostatniej zmiany. Wartości pola stanu nie ograniczają się do prostych wartości, system HomeAssistant dodaje kilka dodatkowych wirtualnych urządzeń, które zmieniają swój stan raz dziennie, a ich wartość stanu przy każdej zmianie przyjmuje czas, np. następnego wschodu czy zachodu słońca.

W celu zaspokojenia wymagania integracji z wieloma urządzeniami system powinien na podstawie dostarczonej mu informacji o typie urządzenia tłumaczyć ciąg historycznych zmian na dane treningowe do modelu sieci. Aby rozszerzenie tego rozwiązania było możliwe przez użytkownika, proponuje się, aby system korzystał z klas abstrakcyjnych. Pozwoli

```
[
    {
        "entity_id": "input_boolean.test_switch_1",
        "state": "off",
        "attributes": {
            "icon": "mdi:lightbulb-on-outline",
            "friendly_name": "Testowy przelacznik 1"
        },
        "last_changed": "2023-09-11T13:07:52.856406+00:00",
        "last_updated": "2023-09-11T13:07:52.856406+00:00"
    },
    ...
]
```

Listing 4.1: Historyczne informacje na temat stanu urządzenia pochodzące z systemu AppDaemon.

to na stworzenie bazowego obiektu i funkcji jakie powinno obsługiwać dane urządzenie w celu współpracy z modułem [10]. Dodatkowo wymusi to na użytkowniku tworzącym dodatkowe elementy, stworzenie wszystkich wymaganych funkcji, a nie jej pewnej części, co pomaga w redukcji błędów programistycznych.

W celu odpowiedniego przygotowania danych wyjściowych pochodzących z sieci neuronowej, można skorzystać z zaproponowanego wyżej kodu obsługującego dane wejściowe poprzez stworzenie dodatkowych funkcji obsługujących przemiane danych.

Proponowana struktura danych na której moduł będzie się opierał to zestaw dwóch macierzy (4.1). Pierwsza zawierająca informacje o aktualnym stanie systemu składająca się z u informacji wejściowych nazywana macierzą stanu, druga składająca się z w skalarów opisująca zmiany w stanach nazywana macierzą przejścia.

$$\begin{aligned} \mathbf{S}_t &= [s_1, s_2, \ldots, s_u] \\ \mathbf{Z}_t &= [z_1, z_2, \ldots, z_w] \end{aligned} \tag{4.1}$$

Pierwsza z nich opisuje stan środowiska domowego w danym momencie t, druga z nich wskazuje na zmianę stanu urządzeń do następnego momentu w czasie $t + 1$. Jesteśmy w stanie określić pewną funkcję G w zależności od typów urządzeń i źródeł, która na podstawie aktualnego stanu oraz zmiany w danym momencie, generuje nam następną macierz $\mathbf{S}_{t+1}$ (4.2).

Tabela 4.1. Tabela zawierająca listowanie warstw w pojedynczej sieci neuronowej

Warstwa	Aktywacja	Ilość neuronów
0	`linear`	Zależna od ilości urządzeń
1	`ReLU`	128
2	`ReLU`	64
3	`tanh`	1

$$G\left(\mathbf{S}_t, \mathbf{Z}_t\right) = \mathbf{S}_{t+1} \tag{4.2}$$

$$\mathbf{Z}_t = P(\mathbf{S}_t) \tag{4.3}$$

Biorąc powyższe pod uwagę, proponowane rozwiązanie powinno na podstawie macierzy stanu $\mathbf{S}$ w momencie k przewidywać następny zestaw ruchów użytkownika, czyli macierz przejścia $\mathbf{Z}$ dla tego samego momentu w czasie (4.3).

4.2.2. Architektura głębokich sieci neuronowych

Sercem pracy całego modułu są modele matematyczne sieci neuronowych. Ze względu na wymóg minimalnej konfiguracji, ich logiczna konfiguracja, powinna być wybrana tak, aby były w stanie nauczyć się epizodów akcji użytkownika bez zbędnej wielkości i skomplikowania. Dodatkowym powodem ograniczenia skomplikowania takich sieci jest zdecydowane wydłużenie procesu uczenia sieci neuronowych dla tych które, zawierają dużo wag [15].

Model sieci neuronowej zawarty w takim module będzie pracował z różnymi typami urządzeń wejściowych gdzie dane będą sformatowane w różny sposób, dodatkowo dane wyjściowe będą musiały być interpretowane przez moduł inaczej, w zależności od celu. Tworzenie dużych i skomplikowanych sieci nie zawsze sprawia, że jest ona w stanie lepiej nauczyć się przekazywanych jej danych ze względu na klątwę wymiarowości [7].

Biorąc powyższe pod uwagę, proponuje się, aby moduł podczas tworzenia początkowych modeli nie tworzył jednej monolitycznej sieci odpowiedzialnej za wszystkie wejścia i wyjścia, a tworzył tyle sieci, ile jest urządzeń do sterowania. Umożliwi to sieciom lepsze dostosowanie sie do urządzeń i natury ich używania kosztem lekko dłuższego czasu uczenia.

Proponowana struktura takich sieci znajduje się w tabeli (4.1). Pierwsza warstwa w tym modelu będzie zależna od ilości urządzeń i źródeł dodatkowych danych, a ostatnia warstwa będzie zawierała jedno wyjście. Taka struktura, będzie agregowała wszystkie wyjścia do jednej formy dostarczanej reszcie komponentów.

4.2.3. Eksport i import sieci

Aby móc zapisać modele sieci neuronowych, wszystkie części systemu mające z nimi styczność muszą wspierać takie działanie. W szczególności dwoma krytycznymi elementami w przypadku tego modułu samouczącego jest biblioteka Tensorflow oraz tworzony przez nas system konwersji danych. Oba te elementy powinny współpracować ze sobą na tyle dynamicznie, aby różne konfiguracje modeli oraz danych wejściowych były poprawnie obsługiwane.

Biblioteka Tensorflow dostarcza swoje metody na zapis i odczyt modeli do pamięci stałej, ale są one ograniczone z kilku powodów. Jednym z nich jest brak możliwości zapisywania struktury i wag do własnego bufora w pamięci czy swojego deskryptora pliku. Sprawia to, że każda osobna zapisana sieć będzie tworzyła jeden osobny plik na dysku, dodatkowo bez możliwości dołączenia własnych danych. Operowanie na dużej ilości urządzeń obsługiwanych przez moduł będzie generowało jeszcze więcej plików, które mogą zostać uporządkowane do postaci jednego, czytanego raz w fazie rozruchu systemu.

W celu eksportu i importu sieci zostanie zaproponowany teoretycznie autorski sposób serializowania i deserializowania danych [13] w którym wszystkie definicje oraz – równie ważne – wagi wyuczonych sieci neuronowych zostaną zapisane w odpowiednim formacie do pliku archiwum `.zip`. Zapisanie wszystkich potrzebnych informacji do odtworzenia całej struktury w jednym pliku ułatwi przenoszenie użytkownikowi systemu między komputerami, ale także odstraszy go od ewentualnego ręcznego zmieniania wartości wag w systemie, poprzez zaciemnienie widocznej struktury.

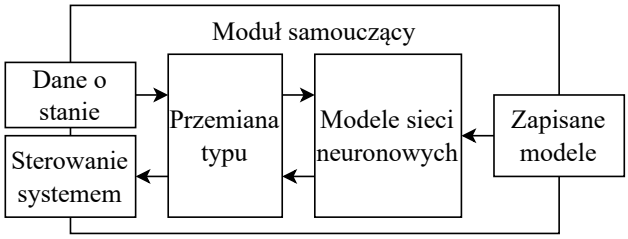
Proponowana architektura plików wewnątrz archiwum dla każdej z sieci osobno będzie składała się z:

1. Surowej deklaracji modelu, w której znajdzie się dokładny opis kształtu i wymiarów każdej z warstw potrzebnej do dokładnego jej odtworzenia.
2. Pliku z zapisanymi wagami każdej warstwy w sieci.
3. Pliku z informacjami pośrednimi, które nie są krytyczne do działania systemu.

Dodatkowo w celu określenia czy nie doszło do zmiany dostępnych dla systemu urządzeń oraz w celu zapisania innych informacji ogólnych o wszystkich dostępnych w modelu sieciach będzie potrzebny jeden dodatkowy plik znajdujący się katalogu głównym archiwum.

4.2.4. Podsumowanie

Na rysunku (4.3) znajduje się proponowana struktura całego modułu samouczącego. Dane o stanie i zmianach będą przechodziły strumieniem wgłąb systemu. Dalej po obliczonej na podstawie stanu systemu predykcji, system będzie sterował urządzeniami w środowisku domowym.



Rys. 4.3. Proponowana struktura działania modułu.

5. IMPLEMENTACJA

5.1. DANE WEJŚCIOWE

Krok przemiany danych wejściowych do wykorzystania w module został podzielony na dwa osobne etapy. Jeden z tych etapów, nazywany krokiem interpretacji różni się w zależności od źródła skąd pochodzą dane. W zależności od tego, czy system korzysta z danych historycznych w procesie uczenia, czy korzysta z danych aktualnych w celu predykcji inaczej przechodzą proces interpretacji. Drugim etapem jest etap transformacji i wygląda on tak samo dla każdego źródła niezależnie od typu.

5.1.1. Interpretacja historyczna

W przypadku źródła danych historycznych, system musi rozpoznawać epizody działań domowników w celu ich nauki. W tym celu, aby wygenerować zbiór danych uczących, przegląda się posortowaną listę wydarzeń w czasie, celem określenia epizodów akcji. Począwszy od pierwszego wydarzenia w dostępnej dla systemu historii, przegląda się ją w poszukiwaniu ciągu akcji o łącznym czasie nie większym niż pewien z góry określony parametr nazwany `EPISODE_DELTA` od momentu wystąpienia pierwszej akcji w epizodzie. Ta wartość została wprowadzona w celu sprawdzenia czy zmiana stanu w epizodzie się nie przedawniła. W przypadku gdy jedno urządzenie zmienia swój stan kilkukrotnie w przeciągu jednego epizodu, brana jest tylko pod uwagę policzona zmiana i stan względem poprzedniego, przed liczonym epizodem. W tabeli (5.1) został zaznaczony znaleziony dla przykładowej historii epizod. Kolorem został zaznaczony znaleziony przez algorytm epizod akcji.

Warto zauważyć, że wybrana długość tego parametru będzie mocno wpływała na liczność i wielkość epizodów. Wybranie zbyt krótkiego czasu epizodu będzie rozdzielać powiązane ze sobą czynności ale będzie rozróżniała zmiany stanu tego samego urządzenia w czasie, a wybranie zbyt długiego czasu będzie łączyło kilka niezależnych akcji ze sobą, możliwie je ze sobą niwelując. Sprawia to, że wybranie optymalnej długości epizodu jest bardzo ważne aby system mógł dostosować się do użytkownika.

Do określenia działania, dla każdego innego typu urządzeń z osobna, skorzystano z abstrakcyjnej klasy `DeviceHistoryGeneric`, opisujących strukturę funkcji jakie powinna dana klasa implementować w celu poprawnego działania. Abstrakcyjna funkcja na podstawie stanu urządzenia w momencie t, stanu urządzenia w momencie k oraz czasu do którego dany epizod powinien się skończyć, zwraca w postaci rzeczywistej liczby, stan oraz przejście

Tabela 5.1. Tabela przedstawiająca działanie algorytmu szukania epizodów.

Urządzenie	Stan	Czas
...	...	...
światło kuchnia	on	16:52:48
światło kuchnia	off	17:32:12
klimatyzacja	17	17:33:00
światło salon	on	17:33:10
telewizor salon	on	17:33:15
światło balkon	on	19:32:42
...	...	...

stanu dla danego epizodu w czasie. Pseudokod takiej funkcji został zawarty w listingu (5.1). Warto zauważyć, że gdy nie ma poprzedniego stanu, tj. historia nie sięga na tyle wstecz, to kod musi obsługiwać wykrywanie obu wartości. W przypadku interpretacji prostych urządzeń, nie jest to problemem, ponieważ jesteśmy w stanie wydedukować przejście stanu i aktualny stan na podstawie informacji pochodzącej z systemu, tak w przypadku gdy tej możliwości nie ma, pojedynczy błędny wynik powinien być zdecydowaną mniejszością po interpretacji reszty historii.

```
def get_past_state(aktualny, poprzedni, do_momentu):
    wartość_stanu = 1.0 jeśli aktualny = "on" wpw 0.0
    wartość_przejścia = 0.0
    # Czy mamy historię na temat poprzedniego?
    jeśli poprzedni nie istnieje:
        # zakładamy, że zmiana stanu odbyła się dawno w historii
        return wartość_stanu, wartość_przejścia

    # Czy zmiana się nie przedawniła?
    jeśli poprzedni.czas_zmiany < do_momentu:
        wartość_przejścia = 1.0 jeśli aktualny = "on" wpw -1.0

    return wartość_stanu, wartość_przejścia
```

Listing 5.1: Pseudokod funkcji interpretującej stany ze źródła historycznego dla typu urządzenia przełącznika astabilnego on/off.

5.1.2. Interpretacja bieżąca

Bieżąca interpretacja akcji użytkownika wygląda bardzo podobnie do analizy historycznej. Informacja o bieżącym stanie całego systemu przechowywana jest przez moduł w pamięci. Podczas pracy systemu, mamy pewność, że system AppDaemon zgłosi zmianę stanu tylko jednego urządzenia za każdym uruchomieniem naszej asynchronicznej funkcji. Wiemy zatem, że musimy tylko obsłużyć zmianę stanu jednego urządzenia i zapisać jego

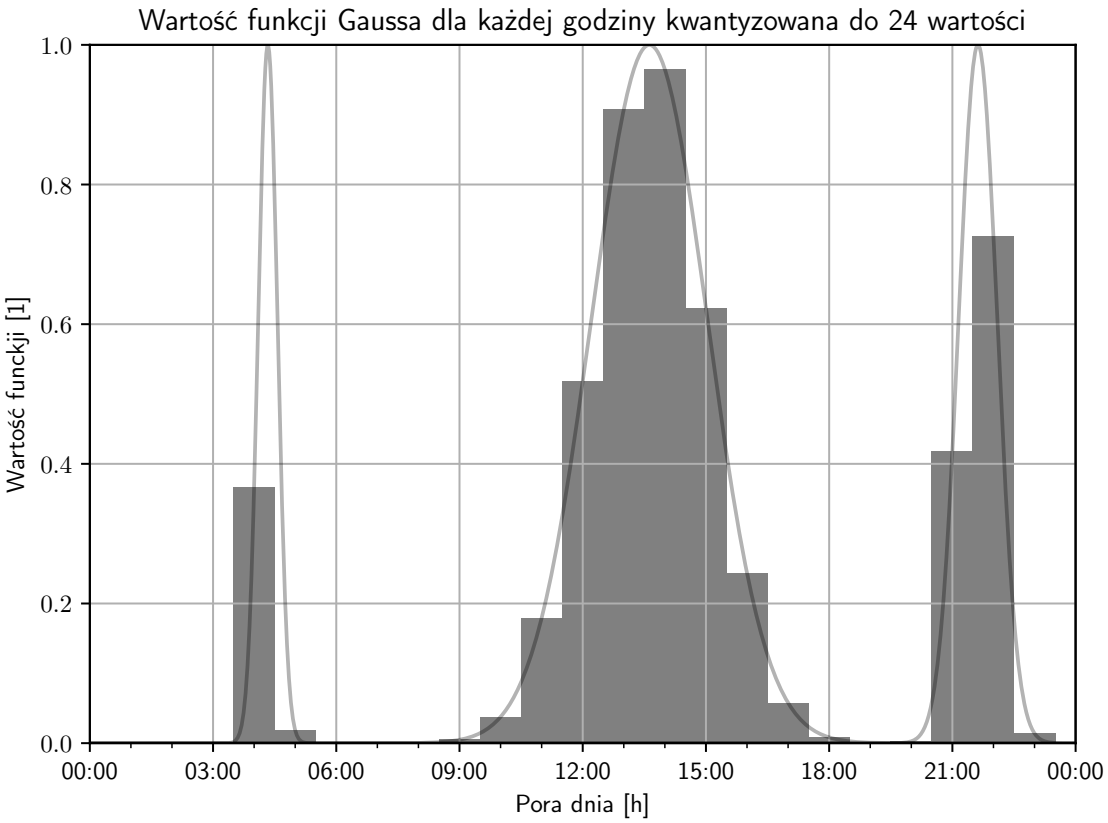
stan do pamięci. Informacja o tym, jakie urządzenie zmieniło się do jakiego stanu będzie potrzebna w procesie predykcji, które będzie opisane w późniejszym etapie. Korzystając z tej samej klasy dla typu urządzenia co w przypadku interpretacji historycznej w podobny sposób określamy stan i przejście stanu urządzenia. W tym wypadku ze względu na pewność, że dane urządzenie zmieniło swój stan dokładnie w momencie uruchomienia danej funkcji, obliczanie stanu i przejścia jest zdecydowanie prostsze ponieważ, nie trzeba rozważać przedawnienia się zmiany.

5.1.3. Transformacja

Po prawidłowej ekstrakcji zmian stanu, przed przekazaniem ich do sieci neuronowych dane są dodatkowo przekształcane. W przypadku prostych urządzeń typu przełączniki astabilne, ten proces nie wprowadza do danych żadnej zmiany. W przypadku bardziej skomplikowanych typów danych, takich jak zmienne temporalne wskazujące na np. dzień tygodnia, czy porę dnia, przeprowadzane są pewne dodatkowe operacje rozkładające jedną konkretną informację na kilka różnych wartości, które lepiej będzie sieciom neuronowym powiązać ze akcjami. W celu możliwości wprowadzenia rozszerzalności tutaj ponownie wykorzystano podejście obiektowe i skorzystano z klasy abstrakcyjnej. Zaproponowana klasa abstrakcyjna `Convertable` opisuje dwie funkcje, które dany konwerter danych musi zaimplementować. Jedna z nich opisuje przejście danych do wersji akceptowalnej przez sieci neuronowe, druga tłumaczy dane wygenerowane przez sieć neuronową do formy obsługiwanej przez resztę systemu. Jednym specjalnym przypadkiem takiego konwertowania danych gdzie potrzebna jest zaimplementowana jedna z obu funkcji, jest przekazywanie zmiennych temporalnych do sieci.

W celu przekazania do sieci jak największej ilości rzeczywiście użytecznych informacji o porze dnia, tak aby była ona w stanie w swojej strukturze zapisać nawyki użytkownika, informacja o czasie jest podzielona. Podział jednej zmiennej czasowej odbywa się poprzez podzielenie jej na 24 różne informacje, gdzie każda z nich wskazuje na pewną wartość zależną od odległości indeksu danej zmiennej od godziny wydarzenia epizodu. Dokładniej, jest to wartość funkcji Gaussa dla wartości zależnej od indeksu, z parametrem μ ustawionym na moment w ciągu dnia podczas którego wydarzył się ten epizod.

Ważnym parametrem w przypadku funkcji Gaussa poza parametrem μ, jest parametr opisujący szerokość dzwona. W zastosowaniach statystycznych nazywany odchyleniem standardowym σ. W przypadku takiego zastosowania obliczania wartości funkcji na podstawie czasu, szerokość mówi nam o tym, jak bardzo akcje występujące o konkretnej porze w ciągu dnia mogą być proponowane o innych podobnych godzinach. Ustawienie tego parametru zbyt szeroko, spowoduje że system będzie rozpoznawał wykonywanie konkretnych akcji o bardzo szczegółowych godzinach w ciągu dnia, ustawienie tego za szeroko, będzie proponowało wykonywanie akcji nieadekwatnych do konkretnej pory dnia. Na obrazie (5.1) znajduje się przykładowy zarys wartości funkcji dla kilku konkretnych



Rys. 5.1. Policzone wartości funkcji Gaussa wskazującej czas dla godziny 4:20:02, 13:37:21, 21:37:21.

pór dnia, razem z zaznaczonymi dokładnymi przebiegami krzywizny dzwonowej. Każda z zaznaczonych godzin posiada inny parametr σ w celu pokazania wpływu tego parametru na policzone wartości.

5.2. SIECI NEURONOWE

Ze względu na wymóg osobnego przetwarzania wielu źródeł różnego typu w systemie, gdzie część z nich może kolejno zostać jeszcze podzielona na dalsze drobniejsze informacje, całość systemu została umieszczona w obiekt będący menedżerem. Głównym zadaniem menedżera jest agregowanie do jednego obiektu wielu konwerterów i agentów. Agent w tym systemie, to obiekt z pewnymi z góry określonymi funkcjami opakowujący modele matematyczne dostarczone przez bibliotekę Tensorflow wykorzystywany do schowania za warstwą abstrakcji, detali implementacji [14] [10]. Takie wykorzystania menedżerów i obiektów agregujących przydaje się w przypadku tworzenia systemów dynamicznych polegających na wczytywaniu i zapisywaniu obiektów do pamięci stałej. W przypadku tego modułu samouczącego, daje to osobie rozszerzającej, możliwości wykorzystania innych struktur niż wcześniej określono (4.1) lub nawet wykorzystanie kompletnie innych metod predykcji w celu osiągnięcia danego celu.

5.2.1. Dynamiczne tworzenie

Jedną bardzo wygodną możliwością udostępnianą przez bibliotekę Tensorflow dla modeli sieci neuronowych są nazwane wyjścia oraz wejścia dla sieci. Pozwala to na dynamiczne tworzenie i eksploatację modeli bez żadnego bardzo skomplikowanego systemu przetwarzającego nazwę wejść na globalną pozycję w macierzy wejściowej. System w celu stworzenia modelu sieci neuronowej wykorzystuje obiekt `Model` udostępniany przez bibliotekę wykonawczą Tensorflow – Keras. Do stworzenia modelu sieci neuronowej, korzystamy z informacji o nazwach wejść oraz nazwach wyjść. System podczas tworzenia nowej sieci, tworzy listę warstw typu `Input` i każdej z nich przypisuje nazwę kolejnego wejścia do sieci, następnie każda z tych warstw, o kształcie `(wielkość próbki, 1)`, jest do siebie dodawana wzdłuż drugiego wymiaru za pomocą warstwy typu `Concatenate`. Powstała warstwa ma kształt `(wielkość próbki, ilość wejść)`. W tym momencie kolejne warstwy dodaje się tak samo jak w przypadku zwykłego modelu funkcyjnego biblioteki Tensorflow i następuje zgodnie z listą warstw podaną we wcześniejszym rozdziale (4.1). Ostatnia warstwa, ta wyjściowa, tak jak w przypadku warstw wejściowych generowana jest z listy nazw wyjść. Dynamiczne tworzenie wyjść, razem z modelem modularnej transformacji, opisanej wcześniej, daje bardzo potężne narzędzie do sterowania domem.

5.2.2. Serializowanie i deserializowanie

Zapis oraz import danych w systemach komputerowych z formy istniejącej w pamięci do formatu, którą można zapisać na dysk jest bardzo dużym zagadnieniem samym w sobie. W przypadku języka Python jest bardzo dużo możliwości godnych tego zadania. Istnieje wbudowana biblioteka taka jak Pickle, która zapisuje wprost bit po bicie zawartość pamięci programu do pliku na dysku, ale zapisuje bardzo dużo redundantnych informacji nad którymi nie mamy kontroli. Istnieją rozwiązania które są czytelne dla ludzi, ale osiągają mniejszą efektywność zapisu danych, takie jak XML [4] czy JSON [6].

W wypadku zapisu i odczytu modeli matematycznych potrzebna jest obsługa trzech różnych informacji, które zostały wymienione wcześniej (4.2.2). Opis logicznej struktury sieci, składa się z informacji o każdej z warstw. Każda z warstw zawiera informacje o jej wielkości, typie, rodzaju aktywacji, nazwach oraz informacji z jakimi warstwami sąsiaduje. Dostarczana przez Tensorflow metoda dla obiektu modelu zwraca wszystkie te informacje w postaci słownika, zawierającego tekst, listy oraz kolejne warstwy słowników. Klasa odpowiedzialna za model implementuje konstruktor [10], który korzystając z informacji zawartych w opisanym słowniku odtwarza dokładną strukturę modelu. Sytuacja wygląda bardzo podobnie dla wag w modelu. Każda warstwa udostępnia funkcję zwracającą listę wszystkich ważnych dla danego typu warstwy parametrów, a druga w obiekcie modelu, przyjmuje listę tych parametrów w celu odtworzenia wag.

W celu implementacji zapisywania i odczytu danych z dysku skorzystano z wbudowanej w Python biblioteki obsługującej pliki `.zip` oraz z metody serializacji JSON. Do obsługi procesu zapisu i odczytu stworzono klasę `ModelSerializer`, implementującą metody potrzebne do skorzystania z danego obiektu jako menedżera kontekstu języka Python [8]. Taki sposób implementacji zapewni, że wszystkie wymagane operacje na pliku i buforach w pamięci zostaną wykonane bez ingerencji użytkownika, co pozwoli na uniknięcie błędów. Stworzony obiekt dostarcza dwie główne metody. Jedna z nich służąca do odczytu i odtworzenia całej struktury menedżerów do takiej wykorzystanej przez moduł oraz drugiej wykorzystywanej do zapisu całego menedżera do pamięci stałej, które działają analogicznie w odwrotnym kierunku.

Pewną emergentną zaletą takiego połączenia serializowania za pomocą JSON wewnątrz kompresowalnego archiwum plików jest zmniejszenie miejsca które zajmują wszystkie modele na dysku, o nawet 57%. Opis wytworzonego w ten sposób archiwum znajduje się w listingu (5.2).

```
Zip file size: 698977 bytes, number of entries: 19
?rw-------  2.0 unx     8626 b- 23-Nov-22 10:47 kuchnia/declaration.json
?rw-------  2.0 unx   264887 b- 23-Nov-22 10:47 kuchnia/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 kuchnia/meta/info.json
?rw-------  2.0 unx     8640 b- 23-Nov-22 10:47 ekspres/declaration.json
?rw-------  2.0 unx   265319 b- 23-Nov-22 10:47 ekspres/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 ekspres/meta/info.json
?rw-------  2.0 unx     8646 b- 23-Nov-22 10:47 salon/declaration.json
?rw-------  2.0 unx   265341 b- 23-Nov-22 10:47 salon/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 salon/meta/info.json
?rw-------  2.0 unx     8648 b- 23-Nov-22 10:47 telewizor/declaration.json
?rw-------  2.0 unx   265556 b- 23-Nov-22 10:47 telewizor/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 telewizor/meta/info.json
?rw-------  2.0 unx     8656 b- 23-Nov-22 10:47 balkon/declaration.json
?rw-------  2.0 unx   265197 b- 23-Nov-22 10:47 balkon/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 balkon/meta/info.json
?rw-------  2.0 unx     8658 b- 23-Nov-22 10:47 sypialnia/declaration.json
?rw-------  2.0 unx   265469 b- 23-Nov-22 10:47 sypialnia/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 sypialnia/meta/info.json
?rw-------  2.0 unx      273 b- 23-Nov-22 10:47 info.json
19 files, 1644270 bytes uncompressed, 696557 bytes compressed:  57.6%
```

Listing 5.2: Listowanie plików wewnątrz archiwum zawierajacego przykładowe modele sieci pochodzącego z programu zipinfo.

5.3. STEROWANIE SYSTEMEM

5.4. FORMATOWANIE DANYCH UCZĄCYCH

5.5. STRUKTURA WEWNĄTRZ DOCKER'A

PODSUMOWANIE

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

BIBLIOGRAFIA

[1] Cockburn, A., *Dokumentacja appdaemon*.

[2] Cook, D.J., Youngblood, M., Heierman, E.O., Gopalratnam, K., Rao, S., Litvin, A., Khawaja, F., *Mavhome: An agent-based smart home*, w: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).* (IEEE, 2003), str. 521–524.

[3] openHAB Community openHAB Foundation, *Dokumentacja pisania skryptów*.

[4] Goldfarb, C.F., Prescod, P., *Charles F. goldfarb's XML handbook* (Prentice Hall PTR, Harlow, England, 2002).

[5] Heierman, E., Cook, D., *Improving home automation by discovering regularly occurring device usage patterns*, w: *Third IEEE International Conference on Data Mining* (2003), str. 537–540.

[6] I Code Academy, *Json for beginners* (Whiteflowerpublsihing, 2020).

[7] Keogh, E., Mueen, A., *Curse of Dimensionality* (Springer US, Boston, MA, 2010), str. 257–258.

[8] Lutz, M., *Learning Python* (O'REILLY', 2013).

[9] Majeed, R., Abdullah, N.A., Ashraf, I., Zikria, Y.B., Mushtaq, M.F., Umer, M., *An intelligent, secure, and smart home automation system*, Scientific Programming. 2020, tom 2020, str. 4579291.

[10] Martin, R.C., *Czysty Kod* (Helion, 2014).

[11] Marufuzzaman, M., Tumbraegel, T., Rahman, L.F., Sidek, L.M., *A machine learning approach to predict the activity of smart home inhabitant*, J. Ambient Intell. Smart Environ. 2021, tom 13, 4, str. 271–283.

[12] Pang, B., Nijkamp, E., Wu, Y.N., *Deep learning with tensorflow: A review*, Journal of Educational and Behavioral Statistics. 2020, tom 45, 2, str. 227–248.

[13] Reis, J., *Fundamentals of data engineering* (O'Reilly Media, Sebastopol, CA, 2022).

[14] Roberts, E.S., *Programming Abstractions in C++* (Pearson, Upper Saddle River, NJ, 2013).

[15] Shah, B., Bhavsar, H., *Time complexity in deep learning models*, Procedia Computer Science. 2022, tom 215, str. 202–210. 4th International Conference on Innovative Data Communication Technology and Application.

[16] Szabłowski, S., *Projektowanie dydaktycznych systemów automatyki domowej*, Dydaktyka informatyki. 2019, , 14, str. 137–146.

[17] Team, D., *Dokumentacja pisania skryptów domoticz*.

[18] Umamageswari, S.S.M., Kannan, M.K.J., *Tensorflow-based smart home using semi-supervised deep learning with context-awareness*, Journal of Mathematical and Computational Science. 2022.

[19] Wu, E., Zhang, P., Lu, T., Gu, H., Gu, N., *Behavior prediction using an improved hidden markov model to support people with disabilities in smart homes*, w: *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2016), str. 560–565.

SPIS RYSUNKÓW

SPIS LISTINGÓW

SPIS TABEL