Webowy system wykrywania choroby Alzheimera z wykorzystaniem głębokich sieci neuronowych

Piotr Kuboń 252871

Spis treści

[1. Wstęp 2](#_Toc119853794)

[1.1. Cel 2](#_Toc119853795)

[1.2. Opis choroby Altzheimera 2](#_Toc119853796)

[1.3. Opis systemu 2](#_Toc119853797)

[1.4. Przebieg pracy nad systemem 3](#_Toc119853798)

[2. Projekt systemu 3](#_Toc119853799)

[2.1. Opis wykożystanych narzędzi 3](#_Toc119853800)

[2.2. Wykożystane wzorce projektowe 3](#_Toc119853801)

[2.2.1. MVC (Model-View-Controller) 3](#_Toc119853802)

[2.3. Modele danych 4](#_Toc119853803)

[2.4. Zbiór uczący i treningowy 6](#_Toc119853804)

[2.4.1. Przegląd zdjęć 6](#_Toc119853805)

[3. Implementacja systemu 7](#_Toc119853806)

[3.1. Przeprowadzone badania nad modelem sieci neuronowek: 7](#_Toc119853807)

[3.1.1. Badanie augumentacji na model – określenie, czy w wyniku zwiększenia zbioru danych uczących nastąpiła poprawa dokładności klasyfikacji 7](#_Toc119853808)

[3.1.2. Badanie wpływu funkcji aktywacji 7](#_Toc119853809)

[3.1.3. Badanie wpływu funkcji straty 7](#_Toc119853810)

[3.1.4. Badanie ilości warstw gęstych i konwolucyjnych 7](#_Toc119853811)

[3.1.5. Badanie ilości warstw gęstych i konwolucyjnych po augumentacji 7](#_Toc119853812)

[3.1.6. Badanie istotności pikseli – jakie obszary są wiążące – tak jak w książce dla mnista 7](#_Toc119853813)

[3.2. UI 8](#_Toc119853814)

[3.2.1. Panel logowania 8](#_Toc119853815)

[3.2.2. Strona główna 8](#_Toc119853816)

[3.2.3. Lista pacjętów 8](#_Toc119853817)

[3.2.4. Dodawanie pacjęta 8](#_Toc119853818)

[3.2.5. Edycja danych pacjęta 8](#_Toc119853819)

[3.2.6. Usunięcie pacjęta z serwisu 8](#_Toc119853820)

[4. Podsumowanie 8](#_Toc119853821)

[4.2. Podsumowanie wyników przeprowadzonych badań 8](#_Toc119853822)

[4.3. Reklama systemu / uwypulkenie zalet i przeznaczenia aplikacji 8](#_Toc119853823)

[4.4. Dalsze możliwości rozwoju aplikacji 8](#_Toc119853824)

[4.4.1. Sprawdzenie efektywności inych sieci 8](#_Toc119853825)

[4.4.2. Sprawdzenie efektywności głosowania większościowego różnych podsieci 8](#_Toc119853826)

[4.4.3. Wydzielenie w serwisie endpointów do REST API 8](#_Toc119853827)

[4.4.4. Poprawa inferfejsu UI – wykożystanie dedykowanego frameworka 8](#_Toc119853828)

[4.4.5. Wykożystanie bazy danych do przechowywania danych pracowników i pacjętów 8](#_Toc119853829)

[4.4.6. Szyfrowanie danych pracowników i pacjętów 8](#_Toc119853830)

[4.4.7. Wydzielenie serwisu webowego do chmury np. Azure 8](#_Toc119853831)

[1. Argumentacja wyboru oprogramowania: 11](#_Toc119853832)

[2. Kawałek o licencjach 12](#_Toc119853833)

# Wstęp

## Cel

Celem pracy jest stworzenie webowego systemu wykrywania i klasyfikacji choroby Alzheimera na podstawie przesłanych zdjęć rentgenowskich mózgu. W tym celu stworzono serwis internetowy z wykorzystaniem frameworku Flask oraz model sieci neuronowej z wykorzystaniem biblioteki TensorFlow. Przeprowadzono badania nad wpływem parametrów sieci na dokładność klasyfikacji zdjęć, oraz wybrano i zaimportowano model, który osiągnął najlepszy wynik.

## Opis choroby Alzheimera

Choroba Alzheimera dotyka z roku na rok coraz większą liczbę osób, szczególnie narażone są osoby starsze, z ograniczoną aktywnością fizyczną i pamięciową. Chorobę Alzheimera wywołują odkładające się w mózgu białka o patologicznej strukturze, stopniowo psujące neurony, powodując ich obumieranie, w wyniku czego dochodzi do coraz to większych zaników pamięci.

Wyróżnia się trzy stadia rozwoju choroby, z czego każde kolejne stadium charakteryzuje się coraz większym wpływem na organizm chorego, stopniowo uniemożliwiając mu samodzielne funkcjonowanie.

Obecnie nie jesteśmy w stanie w pełni wyleczyć choroby Alzheimera, a jedynie leczyć jej skutki i objawy oraz opóźniać jej rozwój przez odpowiednie leki i ćwiczenia. Dlatego tak ważne jest wykrycie choroby na jej wczesnym stadium rozwoju.

## Opis systemu

W celu usprawnienia procesu diagnozowania choroby Alzheimera wykonany został internetowy system wspomagający pracę lekarzy w ocenie stopnia zaawansowania choroby bądź jej braku.

Po wgraniu zdjęcia rentgenowskiego mózgu pacjenta, system dokonuje analizy, a następnie informuje lekarza o stopniu rozwoju choroby.

Postawienie diagnozy jest możliwe dzięki zastosowaniu modelu sieci neuronowej, wyuczonej na zbiorze tysięcy zdjęć rentgenowskich mózgu, zarówno osób chorych jak i zdrowych. Zdjęcia należące do zbioru uczącego odpowiednio modyfikowano poprzez przesunięcie, powiększenie oraz rotację w celu powiększenia zbioru danych uczących. Uzyskano w ten sposób znacznie większy zbiór danych, co przełożyło się na zwiększenie dokładności modelu. Dokonano również badania i porównania różnych architektur sieci neuronowej, badaniu podlegał również dobór parametrów sieci jak i proces wstępnej obróbki przekazywanego zdjęcia.

Wybrany model, który osiągną największą dokładność w klasyfikacji stopnia rozwoju choroby, wydzielono i wdrożono w system serwisu internetowego. Dodano również do serwisu logikę obróbki przesyłanego zdjęcia przed przekazaniem go do modelu klasyfikującego w celu zachowania spójności z formatem danych, którymi model uczono i sprawdzano.

W systemie można przeprowadzić takie czynności jak dodawanie, edytowanie i usuwanie danych pacjenta oraz dostęp do historii wcześniej przeprowadzonych badań.

System zapewni również jednoznaczną identyfikację pacjenta dzięki automatycznie inkrementowanemu numerowi id. Głównym zadaniem systemu jest natomiast klasyfikacja i identyfikacja choroby Alzheimera na podstawie przesłanego zdjęcia rentgenowskiego mózgu pacjenta.

Ze względów bezpieczeństwa, system działa w wewnętrznej sieci obiektu, ma to na celu wykluczenie możliwości połączenia się z nim przez osoby z zewnątrz. Dostęp do aplikacji odbywa się za pomocą przeglądarki internetowej. Dzięki takiemu podejściu, nie wymuszamy na lekarzu posiadania konkretnego systemu operacyjnego, zyskujemy również spójność systemu dla każdego użytkownika oraz bezpieczeństwo przechowywania i składowania danych.

## Przebieg pracy nad systemem

Projekt systemu zakładał następujące etapy:

* Zainstalowanie wymaganego oprogramowania na maszynie wirtualnej
* Stworzenie systemu webowego do zarządzania pajetami
* Przeprowadzenie badań nad modelem sieci w celu osiągnięcia jak największej dokładności
* Wydzielenie modelu i zastosowanie go w aplikacji

+ Ewentualnie - Dynamiczne tworzenie systemu za pomocą Jenkinsa i Dockera

# Projekt systemu

## Opis wykorzystanych narzędzi

Obecnie mamy wiele języków w których możemy napisać serwis internetowy, należą do nich między innymi PHP, JavaScript, Java, C# czy Python. Biorąc pod uwagę wszechstronność języka, oraz wsparcie dla korzystania z modeli sieci neuronowych, wybrany został język Python.

Po wybraniu języka, kolejną ważną decyzją, jest wybór frameworku, przy pomocy którego zostanie zbudowany serwis. Framework jest to zbiór modułów pomagających programiście w pisaniu serwisu internetowego. W szczególności, automatyzuje on działanie podstawowych funkcjonalności w serwisie, dzięki czemu programista nie musi skupiać się na pisaniu powtarzalnych fragmentów kodu. Aktualnie mamy na rynku wiele frameworków wspierających pisanie kodu w języku Python, należą do nich między innymi Django, Flask, Hug czy CherryPy. Najpopularniejsze z nich to Django, który jest wybierany głównie przy tworzeniu dużych rozwiązań webowych, oraz Flask, wybierany częściej do tworzenia mało skomplikowanych serwisów w których mamy dużą swobodę wyboru modułów rozszerzających jego funkcjonalności. Kierując się swobodą wyboru oraz wsparciem do tworzenia szablonów Jinja2 wybrany został framework Flask.

Aktualnie jednym z najpopularniejszych rozwiązań w tworzeniu sztucznych sieci neuronowych jest korzystanie z biblioteki TensorFlow, jest ona wybierana głównie ze względu na domyślną konfigurację wielu aspektów uczenia maszynowego. Popularną biblioteką wykorzystywaną do tworzenia i uczenia sieci jest również PyTorch, jednak ze względu na wcześniejsze doświadczenie oraz możliwość wdrożenia wytrenowanego modelu w serwisie internetowym, zdecydowano się na wykorzystaniu biblioteki TensorFlow.

W celu wytrenowania modelu oraz przeprowadzenia badań zdecydowano się na skorzystanie z usługi Google Colaboratory (w skrócie Colab). Oferuje ona między innymi darmowe środowisko wykonawcze, posiadające wydajne procesory graficzne, doskonale sprawdzające się w zastosowaniach uczenia sieci neuronowych. Usługa Google Colab udostępnia nam zasoby za pośrednictwem dynamicznie tworzonych maszyn wirtualnych, które zostają usunięte po upływie określonego czasu pracy lub wyczerpaniu zasobów. W tym czasie możemy przeprowadzić badania, wyuczyć sieć oraz wydzielić i pobrać model który okaże się najlepszy.

## Wykorzystane wzorce projektowe

### MVC (Model-View-Controller)

Tak jak sugeruje to nazwa wzorca, zależy nam na rozdzieleniu logiki aplikacji na trzy niezależna, ale powiązane ze sobą bloki. Blok modelu obejmuje zdefiniowanie modeli reprezentacji danych, które następnie będziemy przekazywać oraz którymi będziemy operować. W tym bloku zawiera się również logika połączenia z bazą danych oraz operacje na danych takie jak zapisz, pobranie, modyfikacja oraz usunięcie danych z bazy. Dzięki wydzieleniu logiki modelu jesteśmy w stanie bez ingerencji w pozostałe bloki zmieniać i dostosowywać logikę oraz kod programu, na przykład poprzez zmianę silnika bazy danych na inny. Istotna jest również możliwość rozbudowy już istniejących modeli o nowe parametry dzięki dziedziczeniu już istniejących modeli, zachowujemy w ten sposób wsteczną kompatybilność z logiką już występującą w programie, dzięki czemu nie musimy przepisywać całego kodu po wprowadzeniu rozszerzenia.

W serwisie wykorzystano następujące modele danych:

* Model użytkownika
* Model pacjenta
* Model badania

Blok widoku reprezentuje interfejs osoby korzystającej ze strony, zawiera warstwę wizualną oraz zajmuje się przedstawieniem danych przekazanych przez kontroler za pośrednictwem modelu. Dzięki zastosowaniu szablonów Jinja2, jesteśmy w stanie dynamicznie zmieniać zawartość strony w zależności od przekazanych danych. Oprócz zmiany poszczególnych wartości na stronie możliwe jest również dziedziczenie szablonów oraz importowanie ich, dzięki czemu jesteśmy w stanie wydzielić widok niektórych elementów do osobnych plików oraz importować ich zawartość wszędzie tam, gdzie jest to konieczne. Zyskujemy w ten sposób możliwość zarówno dynamicznych zmian widoku strony, jak również jedno miejsce gdzie należy wprowadzić poprawki w kodzie, aby były widoczne we wszystkich widokach na stronie.

W celu wprowadzenia danych i prezencji zwracanych danych użytkownikowi utworzono następujące widoki:

* widok logowania się użytkownika
* widok strony głównej serwisu
* widok listy pacjentów wraz z możliwością dodania nowego pacjenta do serwisu oraz usunięcia już istniejącego
* widok edycji danych pacjenta
* widok historii przeprowadzonych badań na pacjencie
* widok panelu do przeprowadzenia badania

Dodatkowo wprowadzono powidoki w celu wydzielenia powtarzających się elementów, w tym:

* menu
* blok pacjenta w liście

Ostatnim i zarazem najważniejszym blokiem modelu MVC jest kontroler. Blok ten odpowiedzialny jest za przetwarzanie danych modelu i przekazywanie ich do odpowiedniego widoku. Blok ten zajmuje się również przetworzeniem danych otrzymanych od użytkownika aplikacji za pośrednictwem widoku. W zależności od stopnia rozbudowania serwisu możliwe jest korzystanie z wielu kontrolerów, odpowiadających na żądania użytkownika wysłane na konkretny adres URL.

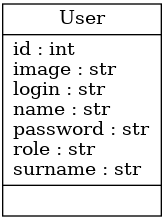
W skład bloku kontrolera w naszej aplikacji wchodzą kontrolery odpowiedzialne za:

* sprawdzenie danych logowania i blokowanie serwisu przed osobami niezalogowanymi
* dedykowany kontroler dla każdej z operacji dodania, pobrania, edycji i usunięcia pacjenta
* prezentację listy pacjentów
* prezentację listy przeprowadzonych badań na pacjencie
* badanie stopnia zaawansowania choroby Alzheimera na podstawie przesłanego zdjęcia skanu mózgu
* wylogowanie się użytkownika z serwisu

## Modele danych

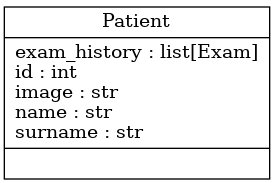
W projekcie do komunikacji pomiędzy bazą danych, kontrolerem i widokiem wykorzystano model pracownika, pacjenta oraz badania.

Model użytkownika posiadała następujące pola:



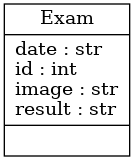
Pole id jest typu liczby całkowitej. Każdemu pracownikowi zostaje przydzielony unikalny numer id, w celu jednoznacznej identyfikacji modelu w bazie. Pozostałe pola są typu ciągu znaków (string). Pole name reprezentuje imię pracownika, surname jego nazwisko, natomiast image stanowi ścieżkę do zdjęcia zapisanego na serwerze. Pola login oraz password są danymi pracownika którymi loguje się do serwisu, natomiast role zawiera informację o funkcji którą pracownik pełni w organizacji.

Model pacjenta posiadała następujące pola:



Pole id jest typu liczby całkowitej. Każdemu pacjentowi zostaje przydzielony unikalny numer id, w celu jednoznacznej identyfikacji modelu w bazie. Pole exam\_history jest listą modeli typu Exam. Pozostałe pola są typu ciągu znaków (string). Pole name reprezentuje imię pacjenta, surname jego nazwisko, natomiast image stanowi ścieżkę do zdjęcia zapisanego na serwerze.

Model badania posiadała następujące pola:



Pole id jest typu liczby całkowitej. Każde badanie posiada unikalny numer id, w celu jednoznacznej identyfikacji modelu w bazie. Pozostałe pola są typu ciągu znaków (string). Pole result reprezentuje wynik przeprowadzonego badania, pole date zawiera informacje o dacie przeprowadzonego badania, natomiast image stanowi ścieżkę do zdjęcia zapisanego na serwerze.

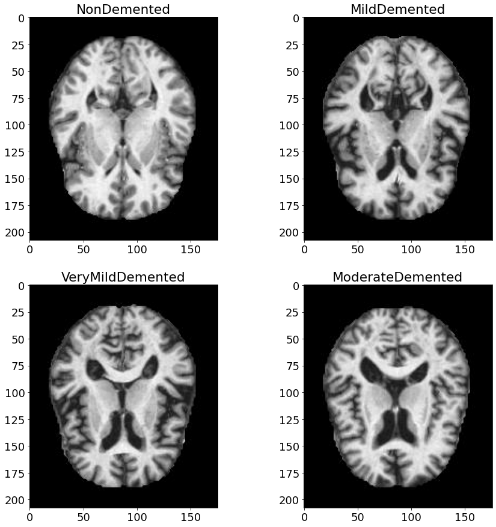
## Zbiór uczący i treningowy

W celu wytrenowania modelu wykorzystano zdjęcia ze zbioru Alzheimer's Dataset (4 class of Images). Zbiór ten zawiera zdjęcia mózgu wykonane metodą rezonansu magnetycznego wraz z poprawnie przydzielonymi etykietami. Zdjęcia w zbiorze mają format 176 pikseli szerokości i 208 pikseli wysokości, oraz zawierają trzy składowe kolorów.

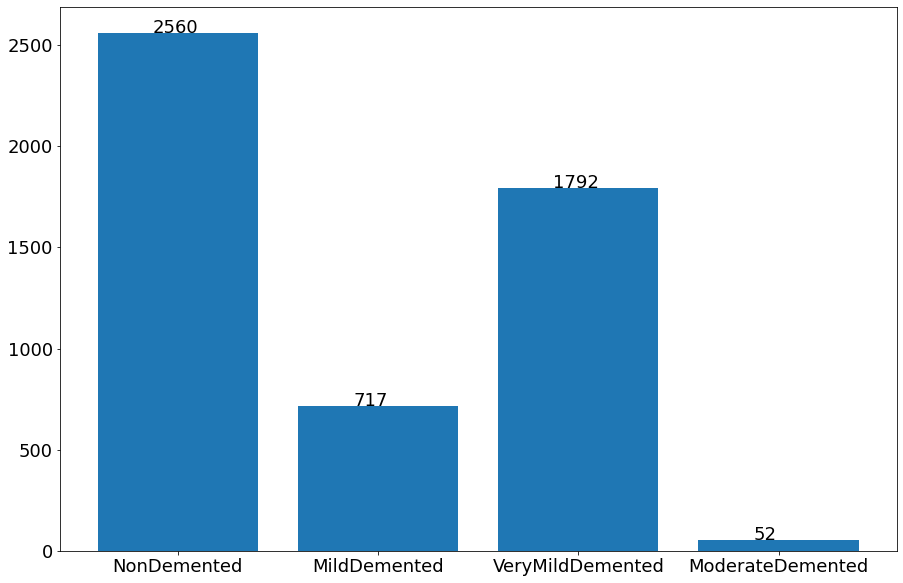
### Przegląd zdjęć

W skład zbioru wchodzą zdjęcia należące do czterech kategorii:

* Brak demencji
* Bardzo łagodna demencja
* Łagodna demencja
* Umiarkowana demencja



Poniższy wykres ilustruje liczbę zdjęć w każdej z kategorii:



Można zauważyć, że wśród zdjęć przeważają zdjęcia oznakowane jako brak demencji, natomiast zdjęć oznakowanych jako umiarkowana demencja jest najmniej.

# 

# Implementacja systemu

Projekt systemu został rozpoczęty od stworzenia maszyny wirtualnej za pomocą oprogramowania Oracle Virtual Box. W tym celu utworzono nową maszynę wirtualną z systemem operacyjnym Linux Ubuntu, narzucono maksymalne zużycie pamięci RAM i przydzielono ilość miejsca na dysku. Następnie zwiększono ilość procesorów do dwóch oraz przydzielono obraz systemu operacyjnego Linux Ubuntu 22.04. Wybrano minimalną instalację systemu oraz domyślne ustawienia, następnie przystąpiono do instalacji wymaganych modułów i oprogramowania.

Aplikacja powstała z wykorzystaniem frameworku Flask.

Aplikacja zawiera model sieci neuronowej, wyuczony przy wykorzystaniu usługi Google Colab. Pobrany model możemy następnie zaimportować do naszej aplikacji .

Opis tworzenia serwsu strona po stronie!

## Implementacja serwisu internetowego

### Panel logowania

Inputy , buton, validacja danych logowania, uderzenie do kontrolera, przekierowanie na sukcej, zwrócenie danych on fail

### Strona główna

Strona wykorzystuje szablony jinja2

Sprawdzić czy już je opisywałem, jak nie, to śmiało opisać

3 guziki – lista pac, str g, wyloguj

Informacje o pracowniku

Menu

### Lista pacjętów

Wyświetla listę , komponent pacjęta – powtarzalność dzięki jinja2

Opcje dodania, pacjęta, usuniecia pacjeta z listy

Przejście do edycji danych pacjęta, historii badań, strony z przeprowadzeniem badania

### Dodawanie pacjęta

### Panel logowania

### Strona główna

### Lista pacjentów

### Dodawanie pacjenta

### Edycja danych pacjenta

### Usunięcie pacjenta z serwisu

Ewentualny opis Jenkinsa + Dockera

+ Ewentualnie – Python ...

# Sztuczne sieci neuronowe

Sztuczne sieci neuronowe są obecnie jednym z najpopularniejszych algorytmów uczenia maszynowego. W odróżnieniu od klasycznego podejścia do budowania algorytmów, sieć neuronowa sama poprawia wartości wag połączeń, zgodnie z metodą wstecznej propagacji błędu. Zadaniem programisty jest dobór odpowiednich parametrów oraz architektury sieci, jak również dostarczenie przykładów uczących.

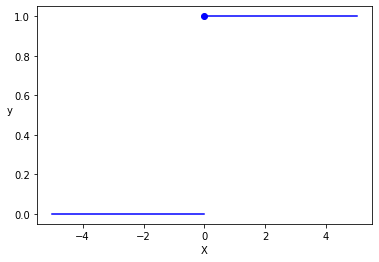
## 4.1. Początki sztucznych sieci neuronowych

Sztuczne sieci neuronowe były początkowo inspirowane sieciami neuronów biologicznych. Ich wynalezienie przypisuje się Warrenowi McCullochowi i Walterowi Pittsowi, którzy w 1943 roku w swoim artykule pokazali uproszczony model działania zespołów neuronów oraz opisali architekturę pierwszej sztucznej sieci neuronowej. Udowodnili oni, że za pomocą sieci neuronów, które posiadają co najmniej jedno wejście binarne oraz jedno wyjście binarne, możliwe jest rozwiązanie dowolnie trudnego zadania logicznego.

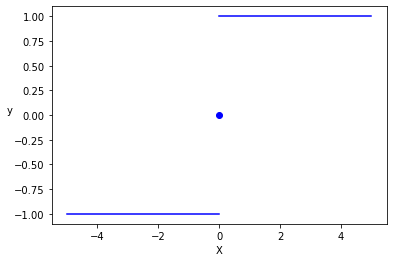
W 1957 roku Frank Rosenblatt zaproponował model pierwszego perceptronu. Model ten przyjmuje na wejście zbiór liczb oraz odpowiednio przyporządkowanych wag. Na wyjściu perceptron zwraca wynik potraktowania funkcją skokową sumy ważonej liczb podanych na wejście wraz z wagami.

Najpopularniejsze funkcje skokowe to:

* Funkcja skokowa Heaviside’a:



* Funkcja signum:



Pojedynczy perceptron pełni funkcję klasyfikatora liniowego, oblicza liniową kombinację danych wejściowych i jeżeli uzyskany wynik przekracza wartość progu, zwracany jest wynik klasyfikacji do klasy pierwszej, w przeciwnym przypadku, zwracany jest wynik klasyfikacji do klasy drugiej.

Niestety pojedynczy perceptron nie jest w stanie rozwiązać wielu zadań, w szczególności zadania klasyfikacji XOR, co wykazali w 1969 roku Minsky i Papert.

Ograniczenia pojedynczego perceptronu są jednak skutecznie niwelowane, przez łączenie wielu perceptronów w warstwy. Stworzoną w ten sposób architekturę nazywamy najczęściej perceptronem wielowarstwowym. Perceptron ten posiada jedną warstwę wejściową, co najmniej jednej warstwy ukrytej oraz jednej warstwy wyjściowej. W przypadku posiadania wielu warstw ukrytych, taką sieć nazywamy głęboką siecią neuronową. Dzięki zastosowaniu dwóch warstw w pełni połączonych jesteśmy w stanie rozwiązać problem klasyfikacji XOR.

Proces uczenia sieci głębokich jest zadaniem bardzo trudnym. Dopiero w 1986 roku Rumelhart, Hinton oraz Williams opublikowali algorytm wstecznej propagacji błędów, dzięki czemu sieć neuronowa jest w stanie sama modyfikować wartości wag poszczególnych połączeń, na podstawie różnicy wartości obliczonej i prawidłowej. W celu poprawnego działania algorytmu wprowadzono dodatkowe nieliniowe funkcje aktywacji, między innymi:

* Funkcja sigmoidalna
* Funkcja tangensa hiperbolicznego
* Funkcja Relu
* Funkcja softmax

Oprócz zadań klasyfikacji, sieci głębokie dobrze sprawdzają się również w zadaniach regresji, czyli w zadaniach predykcji jakiejś wartości na podstawie danych wejściowych. W przypadku predykcji pojedynczej wartości, wystarczy wydzielić jeden neuron wyjściowy, na wyjściu, którego otrzymamy przewidzianą wartość.

Istotnym parametrem, który definiujemy przy pracy z głębokimi sieciami neuronowymi jest funkcja straty. Funkcja ta określa miarę rozbieżności wartości przewidzianej z wartością poprawną. Najczęściej używa się funkcji:

* Błędu średniokwadratowego
* Błędu bezwzględnego
* Funkcji entropi krzyżowej

Kolejnym przełomem była publikacja sieci konwolucyjnych w analizie obrazów przez LeCuna w 1998 roku. Zaprezentował on sieć LeNet-5, która służyła do klasyfikacji ręcznie napisanych cyfr, co zostało później wykorzystane przez kilka banków, do rozpoznawania numerów na czekach.

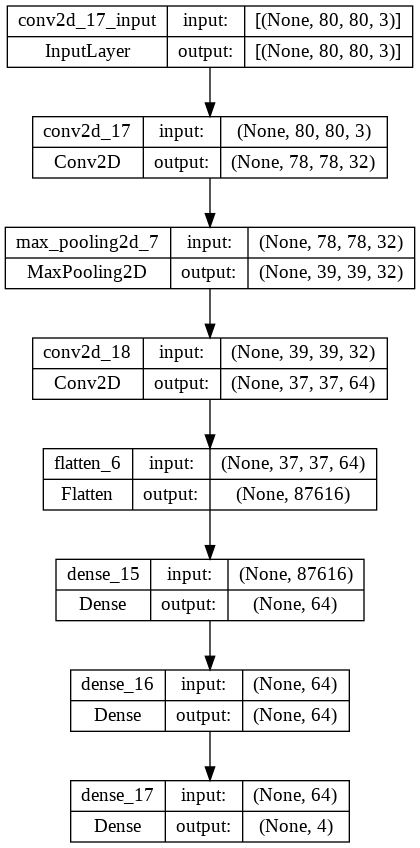
Sieci konwolucyjne – opis –

# Praca badawcza

### Model referencyjny

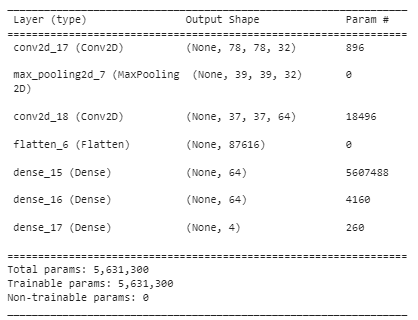
W celu sprawdzenia wpływu różnych parametrów na dokładność klasyfikacji przyjęto model referencyjny składający się z 8 warstw, w tym z:

* Warstwy wejściowej, przyjmującej obraz o wymiarach 80x80
* Warstwy splotowej składającej się z 32 filtrów z oknem splotu o wymiarach 3x3 i funkcji aktywacji relu
* Warstwy łączącej, zwracającej maksymalną wartość w oknie (max-pooling), z oknem o wymiarach 2x2
* Warstwy splotowej składającej się z 64 filtrów z oknem splotu o wymiarach 3x3 i funkcji aktywacji relu
* Warstwy spłaszczającej dane wejściowe
* Warstwy gęstej, w pełni połączonej, składającej się z 64 neuronów oraz funkcji aktywacji relu
* Warstwy gęstej, w pełni połączonej, składającej się z 64 neuronów oraz funkcji aktywacji relu
* Warstwy gęstej, w pełni połączonej, składającej się z 4 neuronów, będącej również warstwą wyjściową modelu

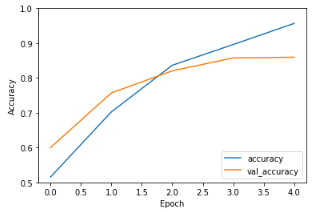


Ponadto przyjęto:

* Funkcję aktywacji Relu
* Za funkcję straty przyjęto funkcję entropi krzyżowej (ang. Sparse Categorical Crossentropy)
* Minimalizowano ją przy pomocy optymalizatora Adam (ang. Adaptive moment estimation)
* Parametr reprezentujący szybkość uczenia się (ang. learning\_rate) pozostawiono na domyślnej wartości wynoszącej 0.001
* Przyjęto 5 epok uczących
* Za metrykę przyjęto dokładność klasyfikacji (ang. accuracy)
* Wielkość partii (ang. batch size) pozostawiono na domyślnej wartości wynoszącej 32



Model referencyjny zawiera 5 631 300 parametrów oraz osiąga dokładność klasyfikacji na poziomie 86% dla danych testowy.



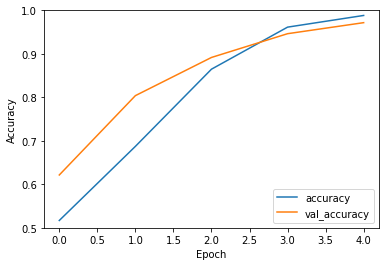
### Badanie wpływu normalizacji min-max na dokładność klasyfikacji

Celem normalizacja min-max jest przekształcenie zbioru danych, aby mieściły się w przedziale [0, 1]. Normalizacja odbywa się zgodnie ze wzorem:

W przypadku operowania na danych będących obrazami, maksymalna wartość X jaką jesteśmy uzyskać, to wartość 255, natomiast najmniejsza wartość to 0.

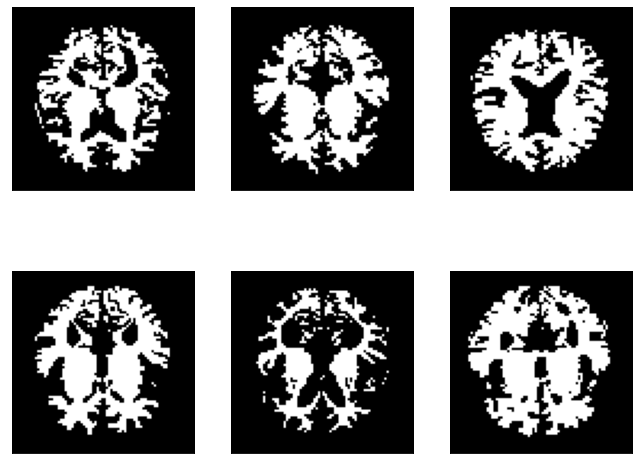
Oznacza to, że w celu przeprowadzenia procesu normalizacji, wystarczy podzielić wszystkie wartości pikseli przez 255.

W wyniku normalizacji danych metodą min-max oraz wyuczeniu tak zmodyfikowanymi danymi model referencyjny, byliśmy w stanie uzyskać 97% dokładności klasyfikacji.



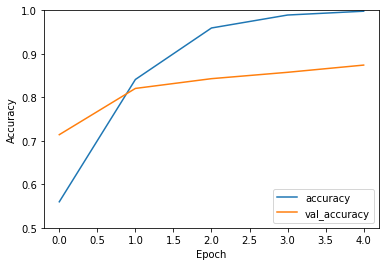
### Badanie wpływu progowanych danych

Celem progowanie jest zmiana wartości obrazu, tak, aby zawierały wartość albo maksymalną, albo minimalną. Piksele o wartości większej niż wartość progu przyjmują wartość 255, natomiast piksele o wartości mniejszej niż wartość progu przyjmują wartość 0.

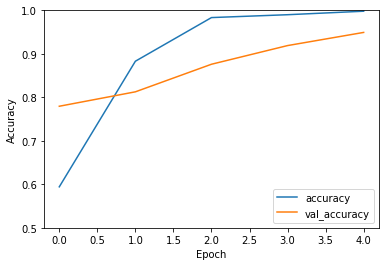


Na podstawie analizy rezultatów progowania przyjęto wartość progu równą 120.

W rezultacie udało się osiągnąć 87% dokładności klasyfikacji



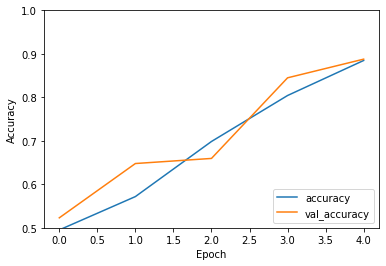
Natomiast wraz z normalizacją danych udało się uzyskać wynik 95% dokładności klasyfikacji.



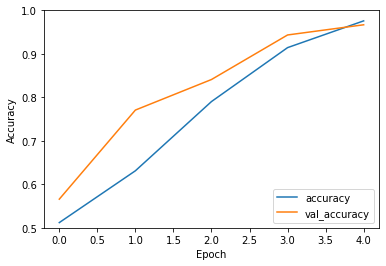
Uzyskane wyniki nie są lepsze od wartości bazowych. Dokładność klasyfikacji rośnie słabiej niż w przypadku modelu trenowanego danymi na których nie przeprowadzono operacji progowania, co może wskazywać na przeuczenie modelu spowodowane jest zbyt dużą utratą informacji w wyniku progowania.

### Badanie wpływu RGB a skali szarości

Dokładność dla modelu w skali szarości: 89%



Po normalizacji: 97%



Należy zwrócić uwagę iż uzyskujemy większa dokładność klasyfikacji na danych testowych, niż na danych uczących

Należało by pociągnąć proces uczenia jeszcze kilka epok w celu uzyskania jeszcze lepszych rezultatów

### Badanie wpływu funkcji aktywacji

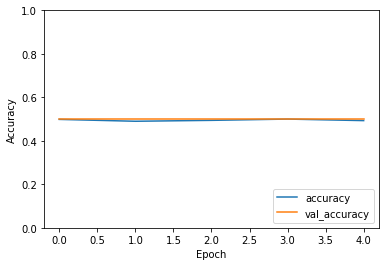
Funkcja aktywacji jest to funkcja której zadaniem jest zwrócenie danych wyjściowych na podstawie zbioru danych wejściowych. Określa ona, czy neuron powinien być aktywowany, czyli czy jego sygnał jest istotny z punktu widzenia sieci i czy powinien być przekazany dalej.

Badania przeprowadzono dla następujących funkcji aktywacji:

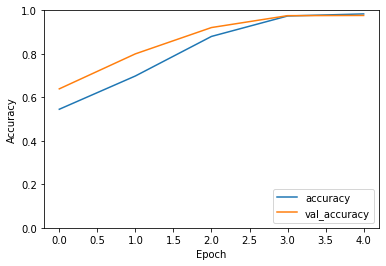
* Sigmoid
* Softmax
* Tanh
* Relu
* Leaky ReLU

#### Sigmoid

Osiągnięto dokładność klasyfikacji na poziomie 50%

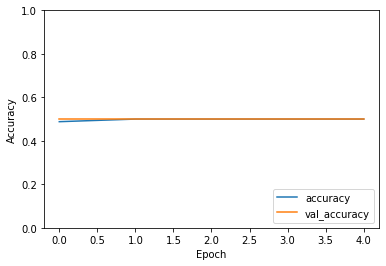


Po normalizacji: 98%

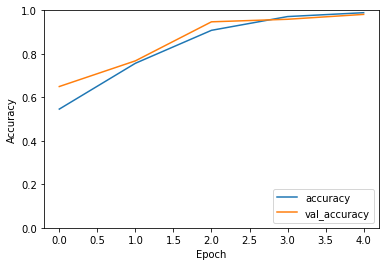


#### Softmax

Osiągnięto dokładność klasyfikacji na poziomie 50%

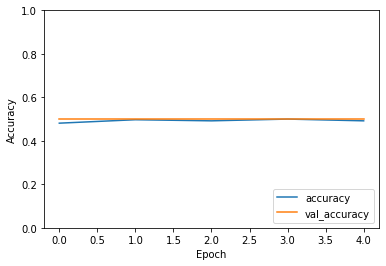


Po normalizacji: 98%

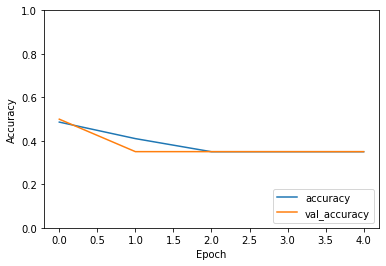


#### Tanh

Osiągnięto dokładność klasyfikacji na poziomie 50%

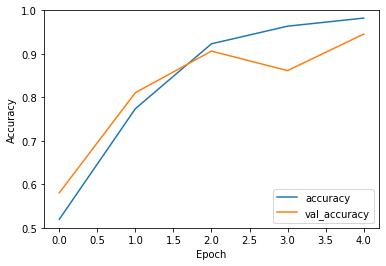


Po normalizacji: 35%

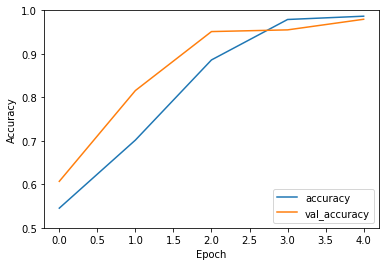


#### Relu

Osiągnięto dokładność klasyfikacji na poziomie 95%



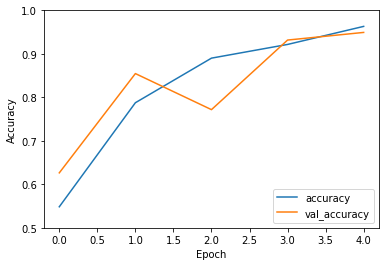
Po normalizacji: 98%



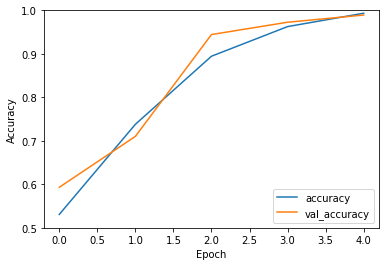
#### Leaky ReLU

Jako parametr alpha przyjęto sugerowaną wartość 0.3 Sprawdzić co dokładnie oznacza ten parametr

Osiągnięto dokładność klasyfikacji na poziomie 95%



Po normalizacji: 99%



Na podstawie otrzymanych rezultatów wynika, że do zadań klasyfikacji, jako funkcja straty najlepiej sprawdza się funkcja Relu oraz podobna do niej funkcja Leaky ReLU

### Badanie wpływu funkcji straty

### Badanie optymalizatorów

### Badanie ilości warstw gęstych i konwolucyjnych

### Badanie augmentacji na model – określenie, czy w wyniku zwiększenia zbioru danych uczących nastąpiła poprawa dokładności klasyfikacji

### Badanie ilości warstw gęstych i konwolucyjnych po augmentacji

### Badanie istotności pikseli – jakie obszary są wiążące – tak jak w książce dla mnista

# Podsumowanie

## Podsumowanie wyników przeprowadzonych badań

## Reklama systemu / uwypuklenie zalet i przeznaczenia aplikacji

## Dalsze możliwości rozwoju aplikacji

### Sprawdzenie efektywności innych sieci

### Sprawdzenie efektywności głosowania większościowego różnych podsieci

### Wydzielenie w serwisie endpointów do REST API

### Poprawa interfejsu UI – wykorzystanie dedykowanego frameworka

### Wykorzystanie bazy danych do przechowywania danych pracowników i pacjentów

### Szyfrowanie danych pracowników i pacjentów

### Wydzielenie serwisu webowego do chmury np. Azure

- linux

- flask

- sieci neuronowe

- trenowanie modelu – colab

- wdrożenie modelu do serwisu flask

- testy funkcjonowania systemu

- wydzielenie stawiania systemu do dockera

- utworzenie pipelinu w jenkinsie – spięcie z repo na githabie – on push – build and test + wdrożenie w kontenerze dockera

----

- jenkins

- docker

Choroba Altzheimera

* + Rosnąca liczba zachorowań
  + Grupa szczególnie narażona
  + Przyczyny
  + Trudna identyfikacja i jednoznaczne stwierdzenie / wykrycie horoby
  + Malejąca liczba specjalistów
  + Potrzeba stworzenia systemu wspomagającego lekarzy który wykrywałby, czy na zdjęciu widoczne są objawy choroby
  + Ostateczna decyzja należy do lekarza (może przeprowadzić wywiad, zna więcej informacji niż samo zdjęcie)
* Sztuczne sieci neuronowe
  + Historia
  + Podobieństwo początkowych neuronów do neuronów w mózgu
  + Perceptron
  + Propagacja wsteczne
  + Wiele warstw perceptronów – głęboka sieć neuronowa
  + Warstwy splotowe
  + Augumentacja
  + Parametry i hiperparametry
    - Funkcja straty
    - Funkcja aktywacji
  + Przetwarzanie danych
    - Wpływ rozmiaru zdjęcia na dokładność
    - Treshold – progowanie - doprowadzenie do zdjęcia 0 / 255
    - Wyostrzenie
* System webowy
  + Potrzeba skalowalności, dużej elastyczności i prostoty wprowadzania zmian
  + Zdecentralizowany model podejmujący decyzję
  + Prostota działania
  + Flask
  + Jenkins
  + Gitlab
  + Colab
    - Platforma do uczenia
    - Duże zasoby
  + Http
* Typy sieci neuronowych
* Badane paramettry sieci neuronowej

Badania przeprowadzić na przyjętej architekturze, po ich zakończeniu skiąć rezultaty w jedno i porównać wyniki!

Poprawić treść strony na polską !!!!!!

# Notatki - wycięte

## 4.1. Historia sieci neuronowych

W branży medycznej sztuczne sieci neuronowe wykorzystywane są najczęściej do zadań klasyfikacji, pomagając lekarzom w podejmowaniu decyzji od których zależy zdrowie pacjęta. Mając na uwadze wielkość kosztów związanych z błędnym działaniem modelu przeprowadzono szereg badań mających na celu dobór jak najlepszych parametrów modelu w celu maksymalizacji dokładności przeprowadzanej przez niego klasyfikacji.

# Argumentacja wyboru oprogramowania:

Zdecydowano sie na wykożystanie w pracy systemu operacyjnego Linux Ubuntu 22.04 ze względu na brak kosztów licencyjnych oraz wsparcie społęczności, rozwijającej projekt systemu operacyjnego od 2004 roku.

W projekcie zdecydowano się na wykożystanie systemu operacyjnego Linux Ubuntu, ze względu na:

* brak kosztów licencyjnych
* wsparcie dla pozostałych oprogramowania, modułów i bibliotek wykożystanych w daleszej części pracy
* wbudowany firewall wystarczający do potrzeb projektu
* dużą społeczność udzielającą się na forach dyskusyjnych

W projekcie zdecydowano się na wykożystanie oprogramowania Oracle Virtual Box, ze względu na:

* możliwość tworzenia migawek i kopi bezpieczeństwa całego systemu
* ochrona systemu hosta przed błędami i uszkodzeniami które moą wystąpić w środowisku wykonawczym
* brak ingerencji systemu wirtualnego w ustawienia hosta
* wygodna możliwość wydzielenia zasobów

W projekcie zdecydowano się na wykożystanie frameworka Flask, ze względu na:

* niewielkie wymagania sprzętowe
* brak kosztów licencyjnych
* dobrą dokumentację oraz wsparcie społeczności
* udostępnia możliwość programowania w języku python

W projekcie zdecydowano się na wykożystanie biblioteki Tensor Flow, ze względu na:

* brak kosztów licencyjnych
* możliwość programowania w języku python
* wygodne i szybkie tworzenie modeli sieci neuronowych
* możliwość wydzielenia procesu uczenia modelu do dedykowanych usług chmurowych
* duża społeczność oraz liczne artykuły naukowe pomocne na etapie tworzenia aplikacji

W projekcie zdecydowano się również na skożystanie z usług serwisu Google Colaboratory (w skrócie Colab) ze względu na:

* możliwość uruchamiania wybranych fragmentów kodu napisanego w języku python na wyspecjalizowanej maszynie za pośrednictwem przeglądarki internetowej
* bezpłatny dostęp do procesorów graficznych i jednostek TPU do 12 godzin dzięki czemu jesteśmy w stanie szybciej wytrenować model sieci

W projekcie zdecydowano się na wykożystanie oprogramowa ze względu na:

# Kawałek o licencjach

Oprogramowanie VirtualBox dostępne jest na licencji GNU General Public Licens, dzięki czemu możemu wykożystać je za darmo do użytku osobistego w celu stworzenia projektu do pracy dyplomowej.

System operacyjny Linux Ubuntu dystrybuowany jest na licencji Creative Commons CC-BY-SA wersja 3.0 UK, która zapewnia możliwość bezpłatnego kożystania z wspomnianego systemu operacyjnego.

Framework Flask dostępny jest za darmo pod warunkami zawartymi w licencji Licencja BSD-3-Clause.

Biblioteka Tensor Flow dostępna jest na darmowej licencji Apache License 2.0