UNIWERSYTET WSB MERITO W GDAŃSKU

WYDZIAŁ INFORMATYKI I NOWYCH   
TECHNOLOGII

**ANALIZA PORÓWNAWCZA KONWOLUCYJNYCH SIECI NEURONOWYCH INCEPTION RESNET V2 I XCEPTION W ZASTOSOWANIU ICH DO KATEGORYZOWANIA ZMIAN SKÓRNYCH**

Praca inżynierska

na kierunku Informatyka

Praca napisana pod kierunkiem

dr inż. Pawła Tomkiewicza

Marek Małek, 56572, Gdańsk 2024

Spis treści

[WSTĘP 3](#_Toc154646696)

[Rozdział 1 – Historia Sieci Neuronowych 5](#_Toc154646697)

[1.1 Wprowadzenie 5](#_Toc154646698)

[1.2 Konwolucyjne Sieci Neuronowe (CNN) 6](#_Toc154646699)

[1.3 Mechanizm propagacji wstecznej 7](#_Toc154646700)

[1.4 Operacja splotu 8](#_Toc154646701)

[Rozdział 2 – Architektury InceptionResNetV2 i Xception 9](#_Toc154646702)

[2.1 Architektura InceptionResNetV2 9](#_Toc154646704)

[2.2 Architektura Xception 9](#_Toc154646705)

[Rozdział 2 11](#_Toc154646706)

[3.1 Rider 11](#_Toc154646708)

[3.2 DataGrip 11](#_Toc154646709)

[3.3 Visual Studio Code 11](#_Toc154646710)

[3.4 Apache Cassandra 11](#_Toc154646711)

[3.5 ElasticSearch 12](#_Toc154646712)

[3.6 Kafka 12](#_Toc154646713)

[Rozdział 3 – Wzbogacanie Zestawu Danych 13](#_Toc154646714)

[4.1 Przyczyna i cel wzbogacania danych 13](#_Toc154646716)

[4.2 Wymagania techniczne 14](#_Toc154646717)

[Rozdział 4 15](#_Toc154646718)

[5.1 Implementacja 15](#_Toc154646720)

[5.2 Testowanie 20](#_Toc154646721)

[5.3 Wdrożenie 21](#_Toc154646722)

[BIBLIOGRAFIA 22](#_Toc154646723)

[Opracowania książkowe 22](#_Toc154646724)

[Netografia 22](#_Toc154646725)

[SPIS RYSUNKÓW 23](#_Toc154646726)

# WSTĘP

Celem pracy jest wykonanie analizy porównawczej dwóch, klasycznych już architektur konwolucyjnych sieci neuronowych, którymi są InceptionResNetV2 oraz Xception. Obie te architektury służyć będą za rdzeń kilkunastu wariacji mających na celu osiągnięcie wyników lepszych, niż byłoby to możliwe przy użyciu niezmodyfikowanych sieci i surowych danych. W jej skład wchodzić będą następujące części:

* Krótka, eksploracyjna analiza danych (en. EDA[[1]](#footnote-1)) – jej celem jest sprawdzenie rozkładu klas obrazów
* Wyszukiwanie modelu bazowego – aby wyraźnie zobaczyć przewagę sieci neuronowych w danym zagadnieniu, dobrze jest najpierw zastosować prostsze i szeroko znane techniki oraz sprawdzić jaką one mają wydajność
* Sprawdzenie jak dobrze na zestawie danych HAM10000[[2]](#footnote-2) działa architektura InceptionResNetV2[[3]](#footnote-3), na razie ignorując dysbalans klas; sprawdzenie, czy będzie w stanie pobić model bazowy
* Dostrajanie wybranej architektury tak, aby lepiej radziła sobie na zbiorze danych HAM10000
* Wytrenowanie na tym samym zestawie klas architektury Xception[[4]](#footnote-4) i porównanie jej wyników z InceptionResNetV2

Poza wymienionymi składowymi, wykonane zostaną także skrypty ułatwiające zarządzanie kodem oraz automatyzujące wielokrotne uruchamianie procesów treningowych. Jest to potrzebne, aby móc wyciągnąć poprawne wnioski, ponieważ w szkoleniu każdej sieci neuronowej jest miejsce na element losowości. Stąd dla zrozumienia, która modyfikacja miała pozytywny wpływ na trening potrzebne jest kilkukrotne uruchomienie go oraz uśrednienie wyników, a następnie analiza statystyczna.

W rozdziale pierwszym i drugim została zawarta część teoretyczna pracy. Opisano w nich krótko historię sieci neuronowych, w szczególności splotowych sieci neuronowych (CNN[[5]](#footnote-5)), przybliżono w nich informacje dotyczące technik, które sprawiły że są tak skuteczne w rozpoznawaniu obrazów, a także opisano dwie wybrane architektury porównując je w miejscach, w których najmocniej się różnią.

Na początku rozdziału trzeciego opisano proces i skutki trenowania modeli bazowych, które ostatecznie posłużyły jako punkt wyjścia do poszukiwań właściwych architektur neuronowych. Następnie opisano modyfikacje mające usprawnić sieci InceptionResNetV2 oraz Xception i analogicznie do początku rozdziału – opisano proces i skutki trenowania tych modeli.

W rozdziale czwartym przedstawiono proces analizy statystycznej stanowiącej sposób dojścia do opisanych dalej wniosków. Opisano tu też problemy, z którymi spotkano się w trakcie prac.

# Rozdział 1 – Historia Sieci Neuronowych

## Wprowadzenie

Większość wytwarzanego oprogramowania od samego początku była algorytmiczna. Program miał zawierać procesy i reguły biznesowe danego przedsiębiorstwa i w ten sposób automatyzować powtarzalne czynności, które można przelać na kod w postaci szeregu instrukcji. Oprogramowanie takie zawsze jest specyficzne, rzadziej dla całej domeny, w której porusza się firma, częściej dla konkretnego jej wycinku, specyfiki działania samego przedsiębiorstwa. Oczywiście istnieją gotowe produkty, takie jak SAP ERP[[6]](#footnote-6), które operują w obrębie całych domen, jednak wciąż ich zasada działania jest taka, jak mniejszych programów – do aplikacji trafiają dane zewnętrzne, które po przetworzeniu przez ściśle określone reguły biznesowe dają spodziewany rezultat. Od zawsze istniała jednak potrzeba istnienia oprogramowania posługującego się logiką rozmytą[[7]](#footnote-7) tak, aby dla pewnych klas problemów nie trzeba było tworzyć tak sztywnych i dokładnych reguł przetwarzania danych. W istocie, być może dla większości tych klas problemów nie byłoby to nawet możliwe. Dla przykładu, w praktyce niemożliwe jest napisanie dobrego klasyfikatora pisma odręcznego.

Różne algorytmy uczenia maszynowego znane były od połowy ubiegłego stulecia, a wciąż krótką historię samych sieci neuronowych podzielić można na kilka okresów:

* Wczesne koncepcje lat 40-tych i 50-tych: wtedy to Warren McCulloch i Walter Pitts tworzą pierwszy „obliczeniowy” model biologicznego neuronu[[8]](#footnote-8).
* Wynalezienie Perceptronu[[9]](#footnote-9) w późnych latach 50-tych: była to wczesna sieć neuronowa zdolna nauczyć się rozpoznawania prostych wzorców, miała jednak spore ograniczenia, np. nie była w stanie rozwiązywać problemów, które nie były liniowo rozdzielne (np. problem XOR[[10]](#footnote-10))
* „Pierwsza Zima Sztucznej Inteligencji” w późnych latach 60-tych i 70-tych: przez ograniczenia perceptronu oraz niewielkiej dostępnej wtedy badaczom mocy obliczeniowej zainteresowanie inwestorów mocno ucierpiały przez co rozwój AI bardzo spowolnił.
* Wynalezienie algorytmu Propagacji Wstecznej[[11]](#footnote-11): umożliwiło to sieciom neuronowym uczenie się bardziej skomplikowanych wzorców.
* „Druga Zima Sztucznej Inteligencji” w późnych latach 80-tych i 90-tych: nastała z powodu zbyt dużych oczekiwań i ograniczonej dostępności mocy obliczeniowej. Innymi słowy sprzęt komputerowy wciąż był zbyt słaby, aby sieci neuronowe mogły pokazać swoje prawdziwe możliwości. Wtedy też na popularności zyskały mechanizmy uczenia maszynowego niezwiązane z sieciami neuronowymi, takie jak Support Vector Machines[[12]](#footnote-12)
* Ponowna faza wzrostu zainteresowania AI oraz czas przełomów: od początku lat 2000 ponownie widać było rosnące zainteresowanie sieciami neuronowymi przez badaczy i wielkie firmy technologiczne. W okolicach początku drugiego dziesięciolecia lat dwutysięcznych zaczęły się pojawiać kolejne przełomowe odkrycia i czas ten trwa do teraz. Kilka największych z nich to np. AlexNet[[13]](#footnote-13), która to sieć stanowiła przełom w dziedzinie rozpoznawania obrazów, odkrycie architektur GAN[[14]](#footnote-14) oraz Transformer[[15]](#footnote-15) (która to jest używana np. w święcącym sukcesy ChatGPT).

## Konwolucyjne Sieci Neuronowe (CNN)

Chociaż historia rozpoznawania obrazów przy pomocy sieci neuronowych, przynajmniej na poziomie koncepcji, sięga lat 80-tych, to pierwszym najgłośniejszym zastosowaniem tego typu sieci była architektura LeNet-5, stworzona w późnych latach 90-tych przez obecnie szeroko znanego badacza sztucznej inteligencji – Yanna LeCuna. Celem jej stworzenia było umożliwienie Amerykańskiemu Biuru Pocztowemu automatycznego odczytu kodów pocztowych z kopert. Była to dość prosta (z dzisiejszego punktu widzenia) architektura, której składowe to warstwa wejścia o wymiarach 32x32x1 (gdzie pierwsze dwie liczby to wysokość i szerokość obrazów, a ostatnia to ilość kanałów), warstwa splotowa, warstwa uśredniająca, jeszcze jedna warstwa splotowa, po niej uśredniająca i ostatnia splotowa, po której znajdują się dwie warstwy gęsto połączone oraz warstwa wyjścia, służąca do klasyfikacji liczb. Warstwy splotowe wykrywają wzorce przestrzenne, natomiast warstwy uśredniające zmniejszają rozmiar danych pochodzących z ich poprzedników. Zostało to zilustrowane na Rysunku 1.

Rysunek 1 – Architektura Sieci LeNet-5

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

Źródło: https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

Architektura ta stała się punktem wyjścia dla twórców bardziej złożonych sieci neuronowych takich jak AlexNet, VGGNet, czy ResNet.

Być może największym wkładem Yanna LeCuna w rozwój sztucznej inteligencji było zastosowanie mechanizmu tzw. Propagacji Wstecznej. Nie była to całkowicie nowa idea, jednak była świeża w świecie wielowarstwowych modeli konwolucyjnych. Jako, że jest to technika leżąca u podstaw każdego rodzaju sieci, nie tylko CNN, zostanie ona opisana jako pierwsza.

## Mechanizm propagacji wstecznej

Mechanizm ten składa się z kilku kroków:

* Przejście wprzód: warstwa po warstwie, aż do wyjścia przekazywane są dane. W przypadku sieci konwolucyjnej będą to coraz bardziej zmienione – zmniejszone, a na końcu spłaszczone do postaci wektora, dane obrazu.
* Obliczenie straty: funkcja straty (która jest jednym z tzw. hiperparametrów) oblicza różnicę między predykcją modelu, a stanem faktycznym – tym, co chciano by, aby model zwrócił.
* Przejście w tył (propagacja wsteczna): w tym miejscu algorytm oblicza gradienty funkcji straty względem każdej z wag w sieci. Każda z owych wag przyczynia się do uzyskania ostatecznego rezultatu, dlatego też sygnał błędu musi zostać rozpropagowany wstecz.
* Gradienty używane są do aktualizacji wag sieci.
* Cały cykl powtarza się na nowo, w każdym przejściu minimalizując wynik funkcji straty.

## Operacja splotu

Tak jak mechanizm propagacji wstecznej jest kluczowym elementem dla wszystkich rodzajów sieci neuronowych, to operacja splotu jest takim elementem w sieciach konwolucyjnych. Dzięki niej, sieci splotowe są w stanie lepiej, od np. sieci gęsto połączonych wychwytywać złożone wzorce, ale też, do pewnego stopnia są w stanie znajdywać je w różnych miejscach obrazów wejściowych. Składowymi operacji splotu są:

* Filtry (lub kernele): są to małe macierze wag, które podobnie jak wagi warstw gęstych ustawione są początkowo na losowe wartości.
* „Aplikowanie” filtrów na danych wejściowych: w tym momencie każda z wag mnożona jest przez wartość danego piksela. Kiedy mnożenia na obecnej pozycji filtra zostają zakończone, ich wyniki są do siebie dodawane – wynik tych operacji to wynik działania filtra na bieżącym obszarze obrazu.
* „Przesuwanie” filtrów: w tym momencie następuje przesunięcie kerneli o określoną ilość pikseli, aż do „końca” obrazu.
* Wyjściem z tej operacji są tzw. mapy cech, które reprezentują to, co w danym przejściu udało się wykryć warstwie splotowej.
* Na końcu, na każdej mapie cech wywoływana jest nieliniowa funkcja aktywacji (np. ReLU), po to, żeby model był zdolny wychwytywać skomplikowane wzorce.

# Rozdział 2 – Architektury InceptionResNetV2 i Xception



## Architektura InceptionResNetV2

Największą zmianą wprowadzoną przez autorów tej architektury w stosunku do jej poprzedników jest użycie modułów incepcyjnych oraz połączeń skrótowych (residual connections). Obie te techniki istniały już wcześniej, jednak użycie ich kombinacji umożliwiło utworzenie dużo głębszych struktur, które oprócz cechowania się wyższą dokładnością, mniej cierpiały z powodu problemu tzw. zanikającego gradientu.

Problem ten został zauważony przez badaczy, kiedy próbowali uczyć bardzo głębokie sieci. Opisany wcześniej mechanizm propagacji wstecznej może cechować się tym, że wagi warstw położonych najdalej są w procesie treningu dostosowywane w najmniejszym stopniu, co z kolei skutkuje tym, że warstwy te przestają się uczyć i sieć przykłada nieproporcjonalnie wyższą uwagę to położonych wyżej. To z kolei sprawia, że stosowanie głębokich sieci neuronowych może nie przynieść żadnych korzyści, lub wręcz sprawić, że ich dokładność będzie niższa od ich prostszych rodzajów. W poszukiwaniu architektur zdolnych rozpoznawać coraz bardziej skomplikowane wzorce, narodziła się konieczność zapobieżenia temu zjawisku. Połączenia skrótowe (pomijające) działają tak, że warstwa poprzedzająca warstwę bezpośrednio nad nią, zwraca swój rezultat do warstwy położonej jeszcze wyżej – oba wyniki są dodawane i w ten sposób mechanizm propagacji wstecznej może w bardziej wydajny sposób rozpropagować gradient funkcji straty.

Jeśli zaś chodzi o ideę, która stała za zastosowaniem modułów incepcyjnych, to badaczom chodziło o to, aby móc wykrywać skomplikowane wzorce w różnych skalach. Stąd, moduły incepcyjne przeprowadzają operację splotu używając jednocześnie filtrów o różnych wielkościach (np. 5x5, 3x3).

Na rysunku 2 zaprezentowane zostały oba omawiane moduły. Jest to bardzo podstawowa prezentacja tej idei, a w jej konkretnych implementacjach występują różne bloki incepcyjne oraz pomijające, bardziej rozbudowane od poniższych.

Po lewej stronie widać, że blok incepcyjny rozgałęzia się na kilka „ramion” przechodzących przez dane filtrami różnych wielkości. Na końcu wyniki tych równoległych operacji łączone są do postaci jednej mapy cech.

Rysunek 2 - przykład modułu incepcyjnego

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, linia, Czcionka

Opis wygenerowany automatycznie

Źródło: https://www.researchgate.net/figure/Basic-architecture-of-inception-block\_fig4\_330511306  
Źródło: https://arxiv.org/pdf/1602.07261v2.pdf? https://arxiv.org/pdf/1602.07261v2.pdf?

Po prawej stronie widać schema bloku zawierającego połączenie pomijające. W tym przykładzie, dla uproszczenia użyto dwóch warstw splotowych, jednak nic nie stoi na przeszkodzie, aby w ich miejscu pojawił się blok incepcyjny taki, jak na schemacie z lewej strony.

## Architektura Xception

C# jest obiektowym językiem ogólnego przeznaczenia. Sprawdza się w wielu zastosowaniach – od programowania serwerowego w aplikacjach przetwarzających zarówno małe jak i duże ilości danych opartych o ASP.NET Core[[16]](#footnote-16), przez tworzenie przeglądarkowych frontendów opartych na silniku Blazor[[17]](#footnote-17), czy aplikacji mobilnych opartych o środowisko Xamarin[[18]](#footnote-18), aż do gier pisanych w oparciu o framework Unity[[19]](#footnote-19). Historycznie bazował na Javie i faktycznie składnia jego pierwszych wersji bardzo mocno ją przypominała. Jego wielkim ograniczeniem było też to, że oprogramowanie w nim napisane można było wytwarzać i uruchamiać jedynie pod systemem operacyjnym Windows. Z czasem, według opinii społeczności, prześcignął swój pierwowzór w zakresie wygody programowania i szybko wdrażanych, przydatnych usprawnień, a z wejściem na rynek środowiska uruchomieniowego .Net Core, język C#, a także inne języki uruchamiane na tej platformie (VB, F#), stał się wieloplatformowy i obecnie bez problemu napisane w nim oprogramowanie można uruchamiać zarówno na Windowsie, jak i systemach Linux, czy Unix.

Środowisko .Net z kolei jest tym wirtualną maszyną zapewniającą programom pisanym przy użyciu C# odpowiednie abstrakcje, zarządzanie i automatyczne odśmiecanie pamięci [[20]](#footnote-20)oraz inne mechanizmy, których nie może zabraknąć w nowoczesnej platformie programistycznej.

# Rozdział 2

**WYKORZYSTANE NARZĘDZIA I TECHNOLOGIE**



## Rider

IDE (Integrated Development Environment) Rider [[21]](#footnote-21)jest konkurencyjnym względem Visual Studio środowiskiem programistycznym. Jego twórcą jest znana na rynku firma JetBrains, która specjalizuje się w wytwarzaniu tego typu oprogramowania. Inne ich produkty, to np. IDE szeroko znane w społeczności programistów Python – PyCharm, czy w środowisku data scientist’ów – DataSpell. Wybrane zostało z uwagi na dużo większą responsywność i szybkość działania względem domyślnego w środowisku programistów C# IDE jakim jest Visual Studio.

## DataGrip

Jest to kolejne IDE pomocne w wytwarzaniu oprogramowania integrującego się z mniej popularnymi, a jednocześnie bardziej specjalistycznymi bazami danych typu MongoDB, czy użyta tutaj Cassandra.

## Visual Studio Code

Jest to bardzo lekkie, proste środowisko programistyczne ułatwiające pracę z językami skryptowymi, takimi jak Powershell.

## Apache Cassandra[[22]](#footnote-22)

Jest to rozproszony system zarządzania bazą danych o charakterze open source. Został zaprojektowany do obsługi dużej ilości rozproszonych danych na wielu serwerach, który będzie nadal funkcjonował nawet jeśli jeden z serwerów przestanie działać. Baza danych została nazwana po wieszczce Kasandrze z mitologii greckiej. Jest to rozwiązanie typu NoSQL, które zostało pierwotnie stworzone przez Facebooka. Udostępnione jako open source w 2008. W styczniu 2009 Facebook przekazał ją do Apache Software Foundation, pełnoprawnym projektem została rok później.

W niniejszym projekcie traktowana jest jako główny magazyn, z którego w razie potrzeby można będzie odtworzyć pomocnicze agregaty rezydujące w ElasticSearch.

## ElasticSearch

W niniejszym projekcie ElasticSearch służy jako magazyn pomocniczy, przechowujący, oraz ułatwiający wyszukiwanie zagregowanych z dzienną rozdzielczością danych pochodzących z turbin wiatrowych.

## Kafka

Apache Kafka początkowo zaprojektowano w LinkedIn, a następnie udostępniono jako oprogramowanie opensource na początku 2011 roku. Projekt opuścił Apache Incubator 23 października 2012 roku. W listopadzie 2014 roku kilku inżynierów, którzy pracowali nad Kafką w LinkedIn, założyło nowe, silnie związane z Kafką przedsiębiorstwo o nazwie Confluent.

# Rozdział 3 – Wzbogacanie Zestawu Danych



## Przyczyna i cel wzbogacania danych

Sieci neuronowe w najlepszy sposób odnajdują rozmaitości, jeśli dysponują dużymi i zróżnicowanymi zestawami danych. Mogą wyuczyć się rozpoznawania niektórych cech, na mniejszych zestawach, ale zasady ich działania sprawiają, że tym lepiej będą to robić, im lepsze dane dostaną w trakcie uczenia. W przypadku tego konkretnego problemu – rozpoznawania zmian skórnych na podstawie obrazów dermatoskopowych – zastosowano kilka popularnych technik augmentacji, takich jak:

* Przesunięcie w pionie lub poziomie – wiele obrazów pokazuje zmianę skórną mniej więcej w centrum, podczas gdy w rzeczywistości, osoba wykonująca zdjęcie może nie być tak dokładna. Aby więc zapewnić dobrą generalizację modelu na obrazach zmian skórnych obecnych w innym, niż centralne miejsce obrazu, można zastosować ten rodzaj augmentacji.
* Zmiana skali obrazu – czyli jego powiększenie, lub pomniejszenie. Przyczyna zastosowania tej techniki jest taka sama, jak w poprzednim podpunkcie.
* Zmiana jasności obrazu – aby symulować różne warunki oświetlenia, w których zdjęcia mogłyby być robione.

Istnieje również wiele innych technik wzbogacania obrazów, jednak uznano, że dla rozpatrywanego problemu będą one nieskuteczne, lub wręcz szkodliwe. Kilka z nich to:

* Obrót obrazu – zmiany skórne nie posiadają elementów, które powtarzają się w tych samych miejscach. Nie ma więc znaczenia pod jakim kątem zdjęcie zostało wykonane, ponieważ sieć neuronowa i tak powinna je poprawnie rozpoznać. Obroty obrazów mają więcej sensu, kiedy trenowana jest sieć rozpoznająca twarze, przedmioty, czy istoty żywe – obiekty, które wykazują sporą regularność.
* Odbijanie obrazu w pionie, czy poziomie – jw.
* Przesunięcie wartości kanałów – w przypadku rozpoznawania zmian skórnych ta technika mogłaby być szkodliwa dla dokładności modelu, ponieważ kolor takiej zmiany jest jednym z czynników używanych podczas klasyfikacji, Przesuwając wartości kanałów, istotna informacja zostałaby zniszczona.

## Wymagania techniczne

Aplikacja ta została przygotowana do używania na urządzeniach typu PC (w tym, przede wszystkim wszelkiej klasy środowiska serwerowe) i laptop. Z uwagi na wysokie wymagania dotyczące ilości rdzeni procesora i dostępnej pamięci, tylko te dwa środowiska zdolne będą ją uruchomić.

Jeśli chodzi o wymagania sprzętowe stawiane środowisku lokalnemu, takiemu jak laptop, są to:

* system operacyjny zdolny uruchamiać skrypty Powershell – Windows lub Linux z zainstalowanymi stosownymi pakietami
* zainstalowane oprogramowanie Docker Desktop
* procesor zdolny uruchomić 5 wątków
* 14gb pamięci RAM

Są to wymagania minimalne, takie które sprawią, że uruchomienie tej aplikacji nie sprawi, że komputer przestanie być responsywny.

Jeśli chodzi o wymagania sprzętowe stawiane środowisku serwerowemu, są to:

* system operacyjny Linux na serwerze uruchamiającym aplikację
* co najmniej trzy niezależne serwery hostujące węzły bazy danych Cassandra (4-rdzeniowe procesory, co najmniej 128gb pamięci RAM)
* co najmniej trzy niezależne serwery hostujące węzły bazy danych ElasticSearch (technicznie, jw.)

# Rozdział 4

**IMPLEMENTACJA APLIKACJI**



## Implementacja

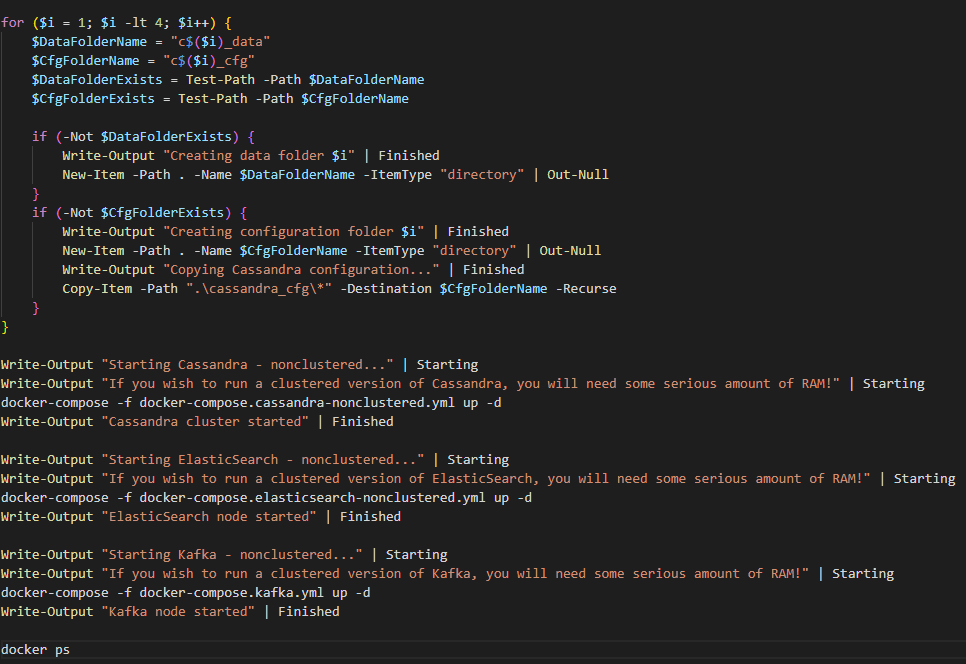
Tworzenie aplikacji tego typu należy rozpocząć od instalacji niezbędnych komponentów infrastrukturalnych. Pierwszym tego typu elementem jest oprogramowanie Docker Desktop. Dla systemu Windows jego autorzy przewidzieli dedykowany instalator, który należy ściągnąć z internetu i uruchomić. Podczas instalacji należy zaakceptować domyślnie wybrane opcje.

Po zainstalowaniu Dockera należy uruchomić w nim bazy danych wymagane przez aplikację do działania. W rzeczywistości, kiedy aplikacja uruchomiona byłaby na serwerze, Docker wymagany byłby tylko po to, aby móc ją uruchamiać, natomiast same bazy danych uruchamiane byłyby dla zapewnienia najwyższej wydajności w trybie tzw. „bare metal” – bezpośrednio na serwerze.

Projekt zawiera skrypt automatyzujący instalację wymaganego oprogramowania. Znajduje się on w ścieżce (zaczynając od głównego folderu projektu): /devops/runme.ps1.

Rysunek 3

Skrypt automatyzujący infrastrukturę

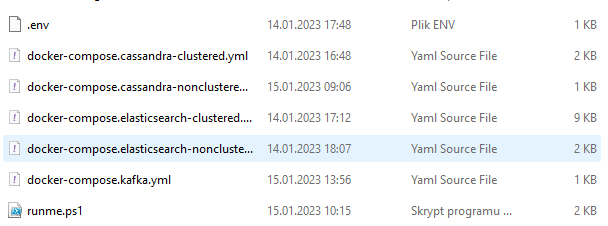


Źródło: opracowanie własne

Jego jedyną rolą jest uruchomienie oprogramowania docker, które automatycznie zainstaluje elementy zdefiniowane w plikach docker compose widocznych poniżej:

Rysunek 4

Lista plików docker-compose

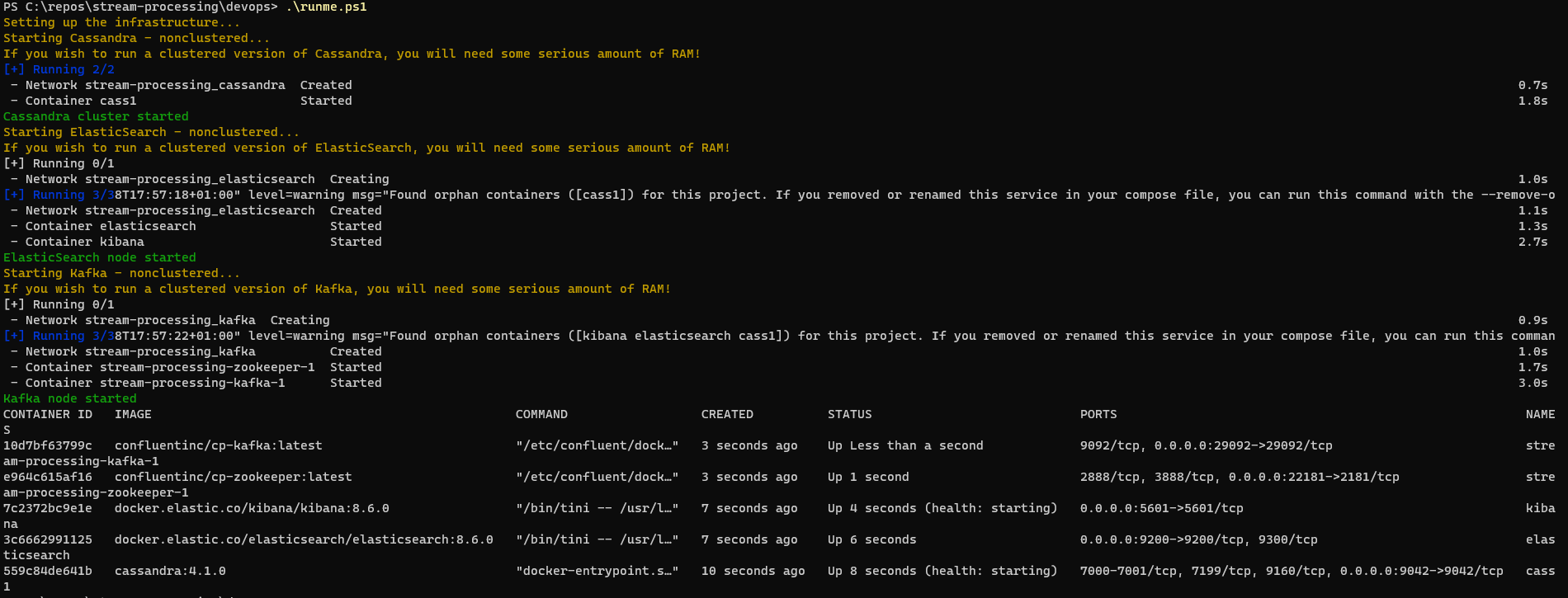


Źródło: opracowanie własne

Powszechną praktyką, w wypadku konfiguracji skomplikowanej infrastruktury przy użycia Dockera jest użycie oddzielnych plików definiujących obrazy[[23]](#footnote-23) dla ułatwienia zarządzania nimi. Po zakończeniu pracy skryptu, w konsoli powinniśmy zobaczyć rezultat podobny do poniższego:

Rysunek 5

Rezultat uruchomienia skryptu runme.ps1

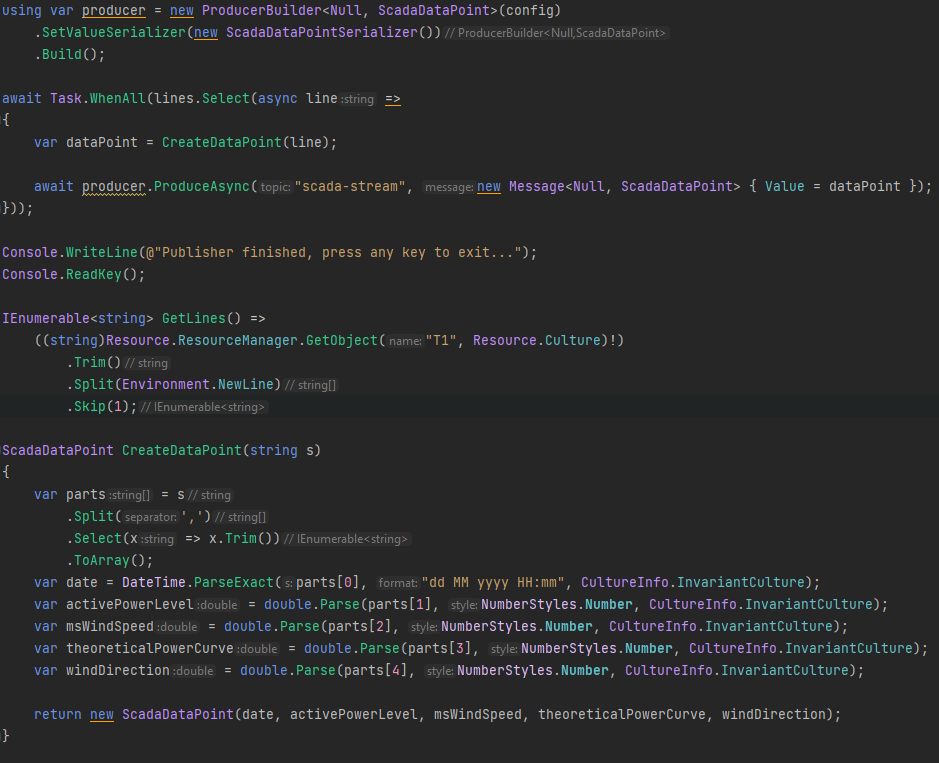


Źródło: opracowanie własne

Kolejnym krokiem jest stworzenie aplikacji umieszającej dane w kolejce Kafka. Potrzebujemy jej z uwagi na testowanie – tworząc aplikację nie mamy możliwości interakcji z prawdziwym systemem SCADA. Nie mamy jej aż do momentu wejścia na środowisko preprodukcyjne, chociaż równie często spotykana sytuacja wygląda tak, że nie ma się styczności z faktycznymi turbinami aż do wejścia na produkcję. Stąd, aby móc przetestować cały proces, musimy jego początek emulować. Nie jest to trudne, a mając dobrze zdefiniowane wymagania biznesowe dotyczące tego, co mamy składować w bazach danych kod takiego emulatora (po zignorowaniu importu bibliotek i początkowych linii konfiguracyjnych) wygląda następująco:

Rysunek 6

Kod programu - producenta danych



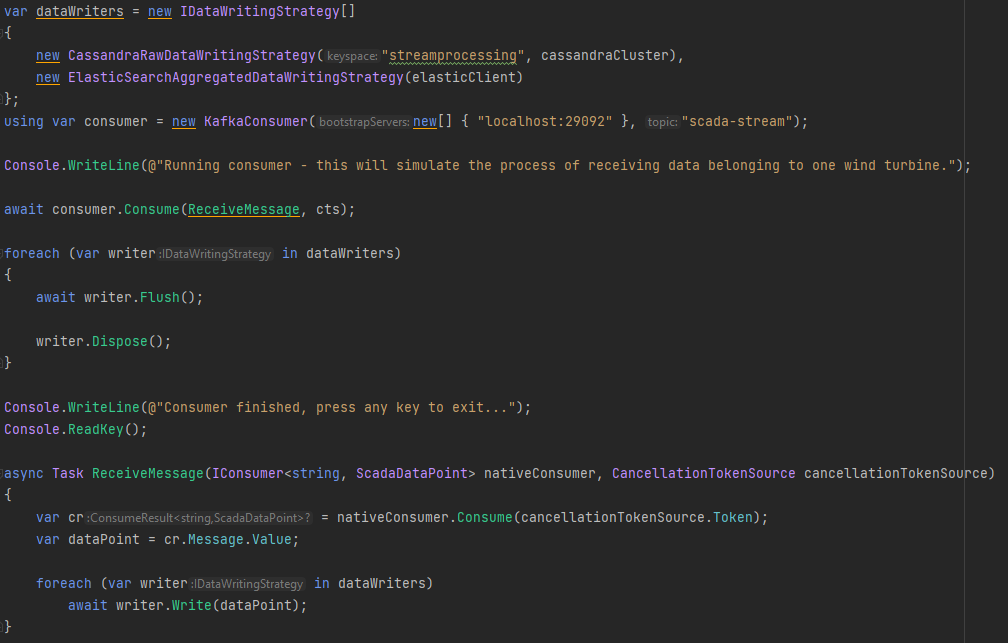
Źródło: opracowanie własne

Pierwsze linie widoczne na powyższym rysunku starają się równolegle wysłać wiele wierszy danych. Odczyt owych linii (z pliku) definiuje funkcja GetLines, następnie obiekt typu ScadaDataPoint tworzony jest z kolejnych linii przy użyciu metody CreateDataPoint.

Jak zostało wspomniane wcześniej, program ten jest tylko emulatorem – pomocą developerską istniejącą po to, aby lokalny rozwój aplikacji był możliwy i sam nie będzie umieszczany na produkcji. Dużo ciekawszym zatem programem jest aplikacja konsumenta. Kod jej punktu wejścia zaprezentowany został na kolejnym rysunku:

Rysunek 7

Punkt wejściowy aplikacji - konsumenta



Źródło: opracowanie własne

Pierwsze linie definiują dwie strategie[[24]](#footnote-24) zapisu danych. Aby wykorzystać polimorfizm[[25]](#footnote-25), a więc uprościć kod programu i uczynić go gotowym na ewentualne, przyszłe zmiany, a jednocześnie bardziej ogólnym, wzmiankowane strategie implementują interfejs IDataWritingStrategy. W tej prototypowej aplikacji powstały jedynie dwie strategie, ale z tak napisanym kodem, łatwo można rozszerzyć ich listę o kolejne.

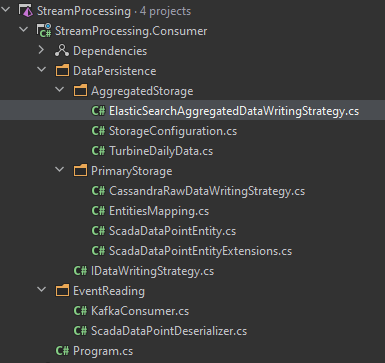
Pierwszą z nich jest CassandraRawDataWritingStrategy. Ma ona na celu zapis surowych danych – w niezmienionej formie. Powodem jej istnienia jest to, że nasza firma chce, aby jej analitycy mogli z takich surowych danych budować własne modele dynamicznie. Innymi słowy dzisiaj dla firmy może najbardziej istotny będzie poziom produkowanej mocy, a więc wszystkie analizy powinny być prowadzone na tej tylko kolumnie. W przyszłości jednak wiemy, że istotne będzie też badanie awaryjności turbin wiatrowych. Oprócz poziomu mocy istnieją też inne wskaźniki, które mogą analitykom i ich modelom uczenia maszynowego sugerować, że dla danej turbiny istnieje pewne wysokie prawdopodobieństwo nadchodzącej awarii.

Drugą zaimplementowaną strategią jest ElasticSearchAggregateDataWritingStrategy. Ta ma na celu budowanie dziennych agregatów danych. Oczywiście analitycy mogliby je sami budować, jednak tutaj cel jest nieco inny. Wymaganie biznesowe jest takie, aby aplikacja frontendowa potrafiła generować pewne typy prostych raportów na żądanie, bez udziału analityków. Aby temu wymaganiu sprostać, powstał pomysł dziennych agregacji danych.

Prócz tych dwóch, kluczowych elementów aplikacji konsumenckiej, zawiera ona kolejny – KafkaConsumer. Nie zapisuje on danych, lecz odczytuje je z kolejki Kafka i przekazuje dwóm opisywanym wcześniej strategiom. Aby podsumować kształt aplikacji konsumenta, załączony zostaje kolejny rysunek:

Rysunek 8

Wysokopoziomowy rzut na elementy aplikacji



Źródło: opracowanie własne

## Testowanie

Podczas testowania zostały przeprowadzone:

* Testy funkcjonalne
  + Sprawdzenie poprawności wysyłki danych
  + Sprawdzenie poprawności składowania danych w obu bazach
* Testy wydajnościowe
  + Sprawdzenie wydajności aplikacji pod kątem wolumenu danych, który zdolna jest obsłużyć

Dzięki poprawnemu użyciu technologii przeznaczonych dokładnie dla opisywanego scenariusza udowodniono, że cały wybrany stos technologiczny będzie skalował się liniowo wraz z dodawaniem do systemu kolejnych danych.

## Wdrożenie

Sugerowanych jest tutaj kilka rozwiązań. Aplikacja została przygotowana w taki sposób, aby mogła operować zarówno w środowisku „on-premise[[26]](#footnote-26)”, jak i chmurowym. Poza infrastrukturą w postaci wymienionych już bazy danych Cassandra, ElasticSearch oraz systemu wysyłki zdarzeń Kafka, aplikacja nie ma zewnętrznych zależności. Zatem, jeśli poda się jej odpowiednie dane konfiguracyjne takie jak hasła dostępowe i adresy serwerów, bez zmian samego kodu możliwe będzie jej przeniesienie na dowolny serwer, komunikujący się z dowolną ilością serwerów hostujących wymienione części infrastruktury.

# BIBLIOGRAFIA

## Opracowania książkowe

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku.** – Wydawnictwo Helion
2. Jeff Carpenter, Eben Hewitt: **Cassandra: The Definitive Guide, 3rd Edition.** – Wydawnictwo O’Reilly
3. Clinton Gormley: **Elasticsearch: The Definitive Guide**. – Wydawnictwo O’Reilly
4. Neha Narkhede, Gwen Shapira, Todd Palino: **Kafka: The Definitive Guide.** – Wydawnictwo O’Reilly

## Netografia

1. https://en.wikipedia.org/wiki/SCADA (data odczytu 17.01.2023)
2. https://learn.microsoft.com/pl-pl/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (data odczytu 17.01.2023)
3. https://learn.microsoft.com/pl-pl/aspnet/core/blazor/?view=aspnetcore-7.0 (data odczytu 17.01.2023)
4. https://learn.microsoft.com/pl-pl/xamarin/ (data odczytu 17.01.2023)
5. https://unity.com/how-to/learning-c-sharp-unity-beginners (data odczytu 17.01.2023)
6. https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals (data odczytu 17.01.2023)
7. https://www.jetbrains.com/rider/ (data odczytu 17.01.2023)
8. https://cassandra.apache.org/\_/index.html (data odczytu 17.01.2023)
9. https://pl.wikipedia.org/wiki/Elasticsearch (data odczytu 17.01.2023)
10. https://lucene.apache.org/ (data odczytu 17.01.2023)
11. https://blog.packagecloud.io/what-is-a-docker-image/ (data odczytu 18.01.2023)
12. https://refactoring.guru/design-patterns/strategy (data odczytu: 18.01.2023)
13. https://www.modestprogrammer.pl/co-to-jest-polimorfizm-w-programowaniu-obiektowym (data odczytu: 18.01.2023)
14. https://www.forbes.com/sites/forbestechcouncil/2019/03/22/cloud-based-vs-on-premise-servers/?sh=a86be3f79e20 (data odczytu: 18.01.2023)

# SPIS RYSUNKÓW

Rysunek 1 9

Rysunek 2 10

Rysunek 3 10

Rysunek 4 11

Rysunek 5 12

Rysunek 6 13

1. https://en.wikipedia.org/wiki/Exploratory\_data\_analysis [↑](#footnote-ref-1)
2. https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000 [↑](#footnote-ref-2)
3. https://arxiv.org/pdf/1602.07261v2.pdf [↑](#footnote-ref-3)
4. https://arxiv.org/pdf/1610.02357.pdf [↑](#footnote-ref-4)
5. https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html [↑](#footnote-ref-5)
6. https://pl.wikipedia.org/wiki/SAP\_ERP [↑](#footnote-ref-6)
7. https://towardsdatascience.com/a-very-brief-introduction-to-fuzzy-logic-and-fuzzy-systems-d68d14b3a3b8 [↑](#footnote-ref-7)
8. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 [↑](#footnote-ref-8)
9. https://maelfabien.github.io/deeplearning/Perceptron/#the-classic-model [↑](#footnote-ref-9)
10. https://www.niser.ac.in/~smishra/teach/cs460/2020/lectures/lec19/ [↑](#footnote-ref-10)
11. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd [↑](#footnote-ref-11)
12. https://pl.wikipedia.org/wiki/Maszyna\_wektor%C3%B3w\_no%C5%9Bnych [↑](#footnote-ref-12)
13. https://proceedings.neurips.cc/paper\_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf [↑](#footnote-ref-13)
14. https://arxiv.org/pdf/1406.2661.pdf [↑](#footnote-ref-14)
15. https://arxiv.org/pdf/1706.03762.pdf [↑](#footnote-ref-15)
16. https://learn.microsoft.com/pl-pl/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (data odczytu 17.01.2023) [↑](#footnote-ref-16)
17. https://learn.microsoft.com/pl-pl/aspnet/core/blazor/?view=aspnetcore-7.0 (data odczytu 17.01.2023) [↑](#footnote-ref-17)
18. https://learn.microsoft.com/pl-pl/xamarin/ (data odczytu 17.01.2023) [↑](#footnote-ref-18)
19. https://unity.com/how-to/learning-c-sharp-unity-beginners (data odczytu 17.01.2023) [↑](#footnote-ref-19)
20. https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals (data odczytu 17.01.2023) [↑](#footnote-ref-20)
21. https://www.jetbrains.com/rider/ (data odczytu 17.01.2023) [↑](#footnote-ref-21)
22. https://cassandra.apache.org/\_/index.html (data odczytu 17.01.2023) [↑](#footnote-ref-22)
23. https://blog.packagecloud.io/what-is-a-docker-image/ (data odczytu 18.01.2023) [↑](#footnote-ref-23)
24. https://refactoring.guru/design-patterns/strategy (data odczytu: 18.01.2023) [↑](#footnote-ref-24)
25. https://www.modestprogrammer.pl/co-to-jest-polimorfizm-w-programowaniu-obiektowym (data odczytu: 18.01.2023) [↑](#footnote-ref-25)
26. https://www.forbes.com/sites/forbestechcouncil/2019/03/22/cloud-based-vs-on-premise-servers/?sh=a86be3f79e20 (data odczytu: 18.01.2023) [↑](#footnote-ref-26)