WYŻSZA SZKOŁA BANKOWA W GDAŃSKU
WYDZIAŁ FINANSÓW I ZARZĄDZANIA

Natalia Cybulska
nr albumu 43240

GENEROWANIE OBRAZÓW IMPRESJONISTYCZNYCH ZA POMOCĄ GENERATYWNYCH SIECI WSPÓŁZAWODNICZĄCYCH

Praca inżynierska
na kierunku Informatyka

Praca napisana pod kierunkiem
dr inż. Elżbiety Zamiar

Gdańsk 2021

Spis treści

Wstęp 3
Rozdział 1. Część teoretyczna 5
1.1. Sieci neuronowe 5
1.2. Generatywne sieci współzawodniczące 8
1.2.1. Klasyfikator 10
1.2.2. Generator 13
1.2.3. Teoria gier w GAN 16
1.3. Styl impresjonistyczny 17
Rozdział 2. Wybór i integracja technologii 19
2.1. Wybór architektury GAN 19
2.2. Przygotowanie maszyny wirtualnej 22
2.2.1. Wymagania sprzętowe 22
2.2.2. Środowisko programistyczne 24
2.2.3. Biblioteki języka Python spoza standardowej biblioteki 26
Rozdział 3. Implementacja generatora zdjęć impresjonistycznych 28
3.1. Przygotowanie bazy danych 28
3.1.1. Wstępne przetworzenie danych 29
3.1.2. Konwersja obrazów 32
3.2. Trening sieci neuronowych 34
3.2.1. Dobór parametrów treningu 34
3.2.2. Proces uczenia 36
Perspektywy rozwoju 43
Zakończenie 45
Spis wykorzystanych źródeł 50
Spis obrazków 53
Spis tabel 56
Spis wykresów 57
Spis schematów 58
Spis załączników 59

Wstęp

Współczesne badania nad wizją komputerową i uczeniem maszynowym ukazały że maszyny potrafią nauczyć się rozróżniać kategorie stylów, takich jak np. Renesans, Barok, Impresjonizm czy Kubizm z sensowną dokładnością[1][2][3] .

Słowo „styl" używane jest, by opisać indywidualną manierę czy sposób w jaki ktoś coś wykonuje. Styl oznacza również grupę prac, które mają podobną typologię charakterystyk, tak jak na przykład styl impresjonistyczny.

Umiejętność klasyfikowania stylów przez maszynę implikuje, że maszyna nauczyła się reprezentacji tego, czym jest obraz impresjonistyczny. Poznała istotne dyskryminatywne cechy poprzez wizualną analizę obrazów[4]. Dzięki postępom techniki coraz łatwiej klasyfikować dane, jednak przez dłuższy czas przed specjalistami od uczenia maszynowego stał problem: jak generować dane, które mogłyby podrabiać te istniejące naprawdę. Jako przykład można tu podać zdjęcia, które wyglądałyby na tyle realistycznie, by oszukać ludzkie oko. Odpowiedzią na ten problem są m.in. generatywne sieci współzawodniczące (ang. generative adversarial networks).

Stworzenie modelu generatywnego jest cięższym zadaniem niż przy zwykłym dyskryminacyjnym modelu. Modele generatywne muszą posiąść wiedzę na temat skomplikowanych korelacji pomiędzy obiektami występującymi na obrazie. Jeśli rozważyć by przykład generatora tworzącego obrazy przedstawiające auta, musiałby on posiąść wiedzę, w jakich miejscach może się znajdować samochód. Próbuje uniknąć próbek przedstawiających auta unoszące się nad powierzchnią wody, a zamiast tego pożądane są obrazy aut znajdujących się na bardziej nadających się dla nich nawierzchniach.

Model dyskryminacyjny nauczony różnicy pomiędzy „auto" lub „nie-auto", może zignorować wiele korelacji, które generator musi odgadnąć. Klasyfikator próbuje nakreślić granice w przestrzeni danych, natomiast generator próbuje wymodelować sposób rozmieszczenia danych w przestrzeni.

W świecie technologii coraz częściej stosuje się rozwiązania oparte na sztucznej inteligencji. Jest to ciągle rozwijający się rynek, który kryje w sobie duży potencjał. Chciałabym dzięki tej pracy rozwinąć swoje umiejętności i wiedzę na temat uczenia maszynowego.

Celem pracy inżynierskiej jest implementacja generatywnej sieci współzawodniczącej. Jej zadaniem jest tworzenie obrazów przypominających te wywodzące się z nurtu impresjonistycznego, jednak nieistniejące nigdy wcześniej.

Praca składa się z pięciu rozdziałów. W rozdziale teoretycznym opisano kluczowe zagadnienia potrzebne do zrozumienia projektu, przedstawiono budowę i działanie sieci neuronowych i ich integrację w architekturze GAN. Przedstawiono również charakterystykę stylu impresjonistycznego, użytą później do empirycznej analizy obrazów wytworzonych przez generator.

W rozdziale drugim opisano dostępne architektury GAN, największą uwagę poświęcając na StyleGAN2, na której opierał się ten projekt. Omówiono również konfigurację środowiska programistycznego, użyte technologie i narzędzia. Przetworzono zbiór obrazów impresjonistycznych i użyto ich do treningu GAN. Przedstawiono rozwój generowanych obrazów podczas procesu uczenia maszynowego. Po zakończeniu treningu przeanalizowano zebrane wyniki.

W rozdziale trzecim opisano proces przetworzenia zbioru obrazów impresjonistycznych, wykonane na nim operacje. Następnie przedstawiono dobór parametrów i przebieg treningu sieci neuronowych, wraz z ukazaniem rozwoju generowanych obrazów podczas procesu uczenia maszynowego.

W rozdziale perspektywy rozwoju przybliżono możliwe metody na usprawnienie projektu praktycznego i dodatkowe sposoby na przeprowadzenie testów jakości generatora, które nie zostały wykonane przez ograniczone zasoby sprzętowe.

W zakończeniu przedstawiono wyniki końcowe i poddano analizie obrazy tworzone przez generator. Omówiono przyczyny zniekształceń znajdujących się na niektórych obrazach, zwrócono też uwagę na najlepsze atrybuty, które zdołał posiąść generator.

Słowa kluczowe: sieci neuronowe, generatywne sieci współzawodniczące, uczenie maszynowe, modele generatywne

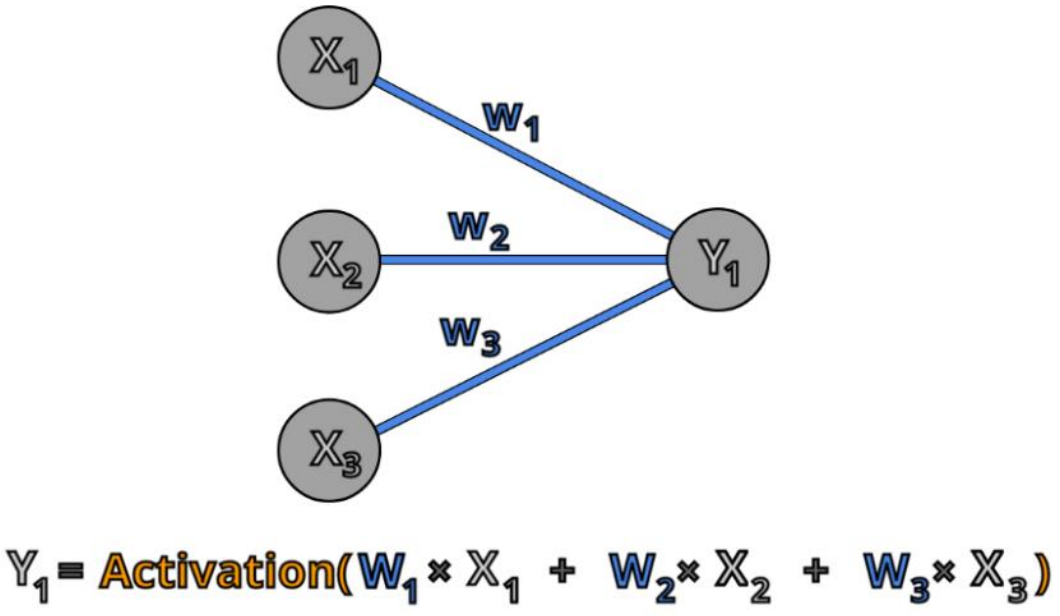
Rozdział 1. Część teoretyczna

1.1. Sieci neuronowe

Neuron jest matematyczną operacją, która przyjmuje na wejście dane, mnoży je przez swoje wagi i przekazuje do następnej warstwy w sieci neuronowej. Celem neuronu jest dostosowanie wag na podstawie analizy wielu przykładów danych wejściowych. Na ich podstawie neuron decyduje, które cechy są istotne, które nie mają znaczenia, a także nadaje im pozytywną lub negatywną wartość[5].

Wyjściem neuronu jest wartość funkcji aktywacji, która przyjmuje jako argument sumę ważoną z wejść neuronu. Funkcja aktywacji umożliwia ograniczenie przedziału liczb na których będą przeprowadzane operacje. W sztucznych neuronach dane wejściowe przekładają się na wagi, z których obliczana jest suma ważona. Jeśliby wyrzucić funkcję aktywacji z powyższego modelu, suma ważona wag mogłaby przyjmować dowolną wartość, co utrudniłoby obliczenia i analizę numeryczną[5].

Schemat 1. Zasada działania neuronu



Źródło: B. Mehlig, *Artificial Neural Networks*, Gothenburg University 2019

W tym przykładzie działanie neuronu Y1 wygląda w następujący sposób: mnoży on X1 przez wagę W1, X2 przez W2, X3 przez W3, a potem je wszystkie sumuje. Następnie poddaje tę liczbę funkcji aktywacji.

Algorytmy sieci neuronowej są inspirowane architekturą i dynamiką sieci znajdujących się w ludzkim mózgu.

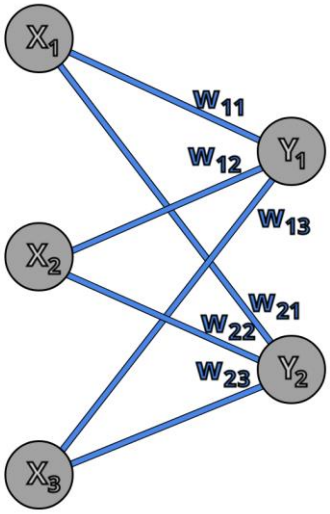
Sieć składa się z połączonych ze sobą neuronów, dzięki czemu może wyliczać różne wartości. Neurony są połączone ze sobą wagami, to znaczy każda jednostka oblicza średnią ważoną swoich wejść.

Sieć uczy się, jak klasyfikować dane wejściowe, poprzez dostosowywanie swoich wag i progu w oparciu o poprzednie przypadki. Optymalne wartości tych parametrów są dostosowywane w oparciu o gradient błędu. Występują różne algorytmy uczenia, a w tym projekcie wartość gradientu błędu jest wyliczana za pomocą algorytmu propagacji wstecznej[6].

Sztuczne sieci neuronowe składają się z trzech typów warstw: wejściowej (zbiera dane i przekazuje je dalej), ukrytej (tu szukane są powiązania między neuronami, czyli zachodzi proces uczenia się) i wyjściowej (gromadzi wnioski, wyniki analizy)[6].

Do pierwszej warstwy – analogicznie jak w przypadku obrazów rejestrowanych np. przez nerwy wzrokowe u człowieka – trafiają nieprzetworzone dane wejściowe. Każda kolejna warstwa otrzymuje dane będące wynikiem przetworzenia danych w poprzedniej warstwie.

Schemat 2. Zasada działania sieci neuronowej



Źródło: B. Mehlig, *Artificial Neural Networks*, Gothenburg University 2019

W tym przykładzie każdy z neuronów z pierwszej warstwy jest połączony z każdym neuronem z drugiej warstwy. Neuron Y1 jest połączony z neuronami X1, X2, X3 za pomocą wag W11, W12, W13. Neuron Y2 jest połączony z neuronami X1, X2, X3 za pomocą wag W21, W22, W23. Wzory na obliczenie sumy ważonej na wejściu neuronów wyglądają następująco:

Równanie 1. Wzory na sumę ważoną na wejściach neuronów

$$Y_1 = W_{11} \times X_1 + W_{12} \times X_2 + W_{13} \times X_3$$

$$Y_2 = W_{21} \times X_1 + W_{22} \times X_2 + W_{23} \times X_3$$

Źródło: B. Mehlig, *Artificial Neural Networks*, Gothenburg University 2019

Po poddaniu tych wartości funkcji aktywacji otrzymamy wartości wyjściowe na neuronach Y1, Y2.

Jednym z przykładów wykorzystania sieci neuronowej jest tłumaczenie tekstu przez maszynę w Google Translate. Zastosowano tam przykład sieci rekurencyjnej, która przyjmuje na wejście sekwencje słów lub pojedyncze litery. W trakcie podawania danych wejściowych, słowo po słowie, sieć zwraca sekwencję przetłumaczonych słów. Sieci rekurencyjne potrafią być wydajnie wytrenowane na dużym zbiorze danych w postaci zdań wejściowych i odpowiadającym im translacji[7].

1.2. Generatywne sieci współzawodniczące

Generatywne sieci współzawodniczące (ang. generative adversarial networks) – GAN to algorytmiczna architektura. Rozwijanie generatywnego frameworku można podzielić na następujące etapy[6]:

1. Istnieje zbiór danych składający się z pewnych obserwacji
2. Zakładamy że obserwacje zostały wygenerowane według pewnej nieznanej dystrybucji
3. Generatywny model próbuje stworzyć dane podobne do tych w zbiorze danych, utworzone według reguły która przypomina tę z zebranych obserwacji
4. Model tworzy dane odmienne od tych ze zbioru danych.

GAN zawiera co najmniej dwie niezależne sieci neuronowe, pierwszą zwaną dyskryminatorem, która uczy się rozpoznawać obrazy i drugą, która uczy się generować obrazy (generatora). Występują również architektury, w których dodaje się dodatkowy, wcześniej wytrenowany model, np. VGG16. VGG16 to model sieci neuronowej przedstawiony przez K. Simonyan i A. Zisserman z Uniwersytetu Oksfordzkiego w pracy „Very Deep Convolutional Networks for Large-Scale Image Recognition”. Model osiągnął dokładność w rozpoznawaniu obrazów równą 92.7% w teście na bazie ImageNet – składającego się z 1000 obrazów z 1000 różnych kategorii.

W momencie gdy generator zacznie tworzyć tak realistyczne zdjęcia, że dyskryminator nie będzie w stanie odróżnić ich od prawdziwych, model generatywny zostanie w pełni wytrenowany.

Klasyfikator ma za zadanie odnaleźć wszelkie mankamenty w wytworzonym obrazie, które będą w stanie nam powiedzieć, że dany obraz nie jest prawdziwy. Jeśli znajdzie jakiś błąd w obrazie, odrzuci go. Kiedy już odnajdzie ten błąd, będzie dalej go odrzucał, aż nie zostanie naprawiony. Klasyfikator zdradza generatorowi, w jaki sposób powinien dostosować swoje wagi, w jakim kierunku pójść, by generowane obrazy były coraz lepsze.

Sieci trenuje się następującym cyklem – dyskryminator trenuje się na x prawdziwych zdjęciach z bazy danych, następnie przeprowadza inferencję na y wygenerowanych zdjęć. Inferencja to przewidywanie przez model sieci neuronowej nowych wyników na podstawie analizy poprzednich wyników. Parametry x i y są zmienne w zależności od architektury i wybranych parametrów. Generator jest w ten sposób trenowany przez klasyfikator. Ważnym czynnikiem jest to, by generator i klasyfikator były trenowane równocześnie. Zbyt duża różnica w wytrenowaniu którejś z sieci może spowodować drastyczny spadek krzywej uczenia się drugiej sieci[8].

Znanym przykładem GAN stworzonym za pomocą architektury StyleGAN2, jest "This Person Does Not Exist" (ta osoba nie istnieje). To strona internetowa, która pokazuje zdjęcia realistycznie wyglądających ludzkich twarzy, które zostały wygenerowane komputerowo.

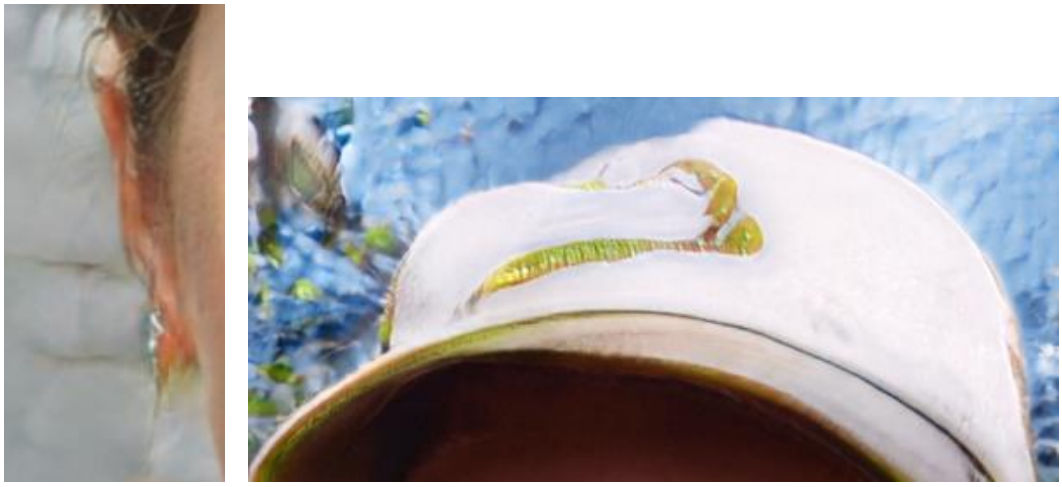
Do wad GAN zalicza się problem z tworzeniem wyraźnych linii, ostrych i gładkich gradientów. Nie radzi sobie z przetwarzaniem monochromatycznych obrazów i grafiką liniową[9].

Obraz 1. Przykładowe zdjęcia umieszczone na stronie This Person Does Not Exist

Źródło: thispersondoesnotexist.com [dostęp z dnia 05.07.2020]

Pierwsze zdjęcie z lewej strony przedstawia istniejącą osobę, dwa pozostałe zostały wygenerowane komputerowo. Jakość wygenerowanych twarzy jest na tyle wysoka, że niewyćwiczone oko mogłoby nie wyłapać, że nie są one prawdziwe.

Obraz 2. Zakłócenia w generowanych zdjęciach twarzy



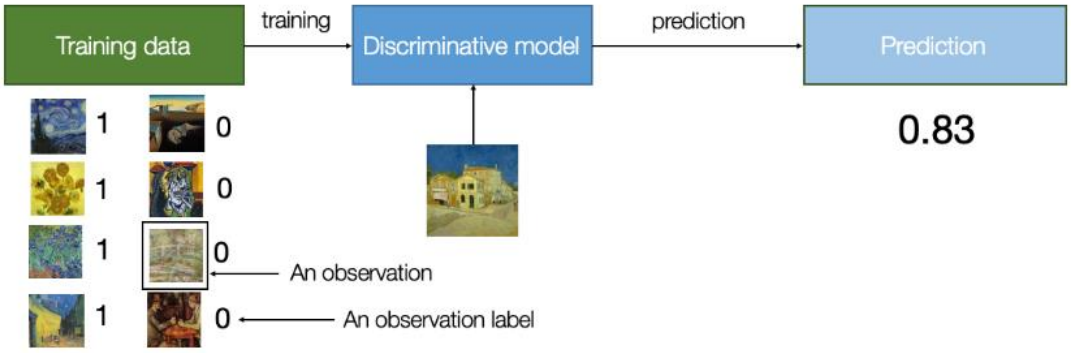
Źródło: thispersondoesnotexist.com [dostęp z dnia 05.07.2020]

Największe zakłócenia można dostrzec w detalach, które najrzadziej pojawiają się na zdjęciach w bazie danych. Generator nie poradził sobie z utworzeniem prawidłowego ucha z kolczykiem. Czapka znajdująca się na prawym zdjęciu ma nienaturalne zniekształcenia. Kolejną rzeczą jest tło – w przypadku wygenerowanych zdjęć przedstawia szum, plamy, bliżej nieokreślone kształty, nie jest to częste tło prawdziwych zdjęć.

1.2.1. Klasyfikator

Klasyfikator służy do przetworzenia danych i podzielenia zestawu danych na osobne klasy. W przypadku generatywnych sieci współzawodniczących operuje on na dwóch klasach: obraz z oryginalnej bazy danych, obraz wytworzony przez generator. Jak działa klasyfikator? Poniższy obraz przedstawia sieć neuronową, która stwierdza czy dany obraz został namalowany przez Vincenta van Gogha.

Schemat 3. Schemat sieci neuronowej zwracającej prawdopodobieństwo, z jakim dany obraz został namalowany przez Vincenta Van Gogha



Źródło: David Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*, O'Reilly 2019

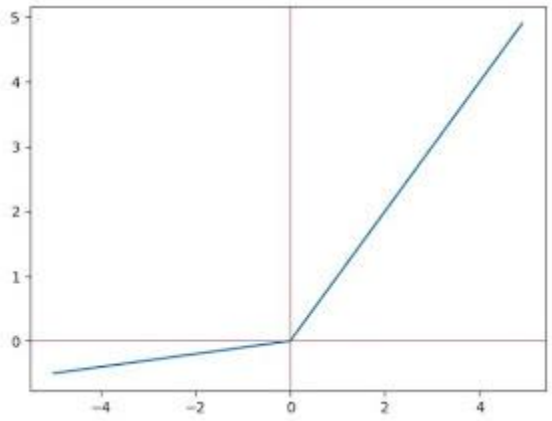
Na początku sieć neuronowa jest trenowana za pomocą obrazów – kiedy jest to obraz namalowany przez van Gogha, obraz ma etykietę – 1. Kiedy nie jest to obraz stworzony przez Vincenta van Gogha, dostanie etykietę 0. Jest to przykład uczenia nadzorowanego, jako że na wejściu oprócz zdjęcia znajduje się odpowiadająca mu etykieta.

Sieć jest trenowana tak długo, aż będzie w stanie zauważyć różnicę pomiędzy danymi obrazami – wytworzy model, który będzie zawierał w sobie pewnego rodzaju esencję obrazów van Gogha.

Co się dzieje po wytrenowaniu sieci? Po podaniu obrazu wejściowego i dokonaniu inferencji za pomocą modelu, na wyjściu otrzymujemy liczbę od 0 do 1 – im bliższa 1, tym pewniejsze jest, że dany obraz został namalowany przez van Gogha.

W implementacji klasyfikatora w architekturze GAN korzysta się z funkcji aktywacji LeakyReLU (ang. Leaky Rectified Linear Unit) na wszystkich warstwach sieci neuronowej[9].

Wykres 1. Wykres funkcji LeakyReLU



Źródło: A. Mass [et al.], *Rectifier nonlinearities improve neural network acoustic models*, Stanford University 2013

Nachylenie wzoru funkcji dla wartości ujemnych wynosi 0.01 a dla wartości dodatnich 1.

Kiedy argument funkcji jest dodatni, wartość funkcji równa się jej argumentowi; kiedy argument wynosi zero, wartość wynosi zero; kiedy argument jest ujemny, wartość funkcji to argument przemnożony przez 0.01. Funkcja jest monotoniczna i różniczkowalna[10].

Fragment kodu 1. Funkcja LeakyReLU zapisana pseudokodem

```
if input >= 0:
    return input
else:
    return input * 0.1
```

Źródło: opracowanie własne

Należy zwrócić uwagę, że następuje dowolność postawienia znaku równości przy zwróceniu wartości wyjściowej – w każdym przypadku zostanie zwrócone 0.

1.2.2. Generator

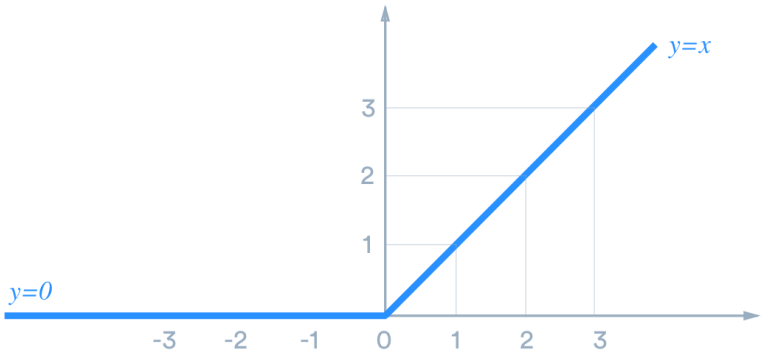
Głównym celem modelu generatywnego jest stworzenie sztucznych danych, które zostaną przez dyskryminator uznane za prawdziwe. Model dodaje do obrazów szum, zwykle szum gaussowski, dzięki czemu tworzy unikatowe obrazy[6].

Obraz 3. Porównanie generowanych obrazów w przeciągu lat

Źródło: David Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*, O'Reilly 2019

W implementacji generatora w architekturze GAN korzysta się z funkcji aktywacji ReLU (ang.Rectified Linear Unit) na wszystkich warstwach oprócz ostatniej.

Wykres 2. Wykres funkcji ReLU



Źródło: S. Behnke, *Hierarchical Neural Networks for Image Interpretation. Lecture Notes in Computer Science*, Springer 2003

Dla wszystkich argumentów niedodatnich przyjmuje ona wartość zero. Kiedy argument funkcji jest dodatni, wartość funkcji równa się jej argumentowi, nachylenie dla wartości ujemnych wynosi 0, a dla wartości dodatnich 1. Funkcja ReLU jest różniczkowalna we wszystkich punktach z wyjątkiem 0. Dla wartości większych niż 0, wyliczane jest po prostu maksimum funkcji.

Fragment kodu 2. Funkcja ReLU zapisana pseudokodem

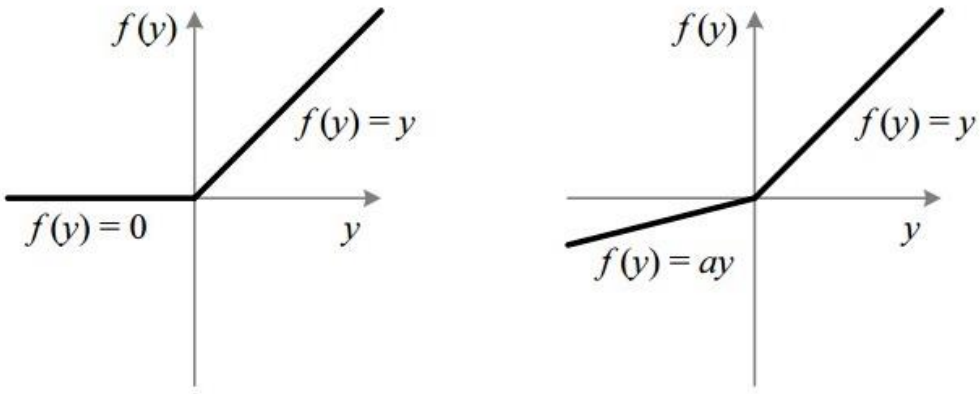
```
if input > 0:
    return input
else:
    return 0
```

Źródło: opracowanie własne

Zaletą funkcji ReLU jest prostota obliczeniowa. Dzięki liniowemu zachowaniu łatwiej zoptymalizować sieć neuronową[11]. Kolejną z zalet jest możliwość przesłania wartości równej 0.0.

Do wad zalicza się fakt, że wszystkie negatywne wartości stają się od razu zerem, przez co negatywne przypadki nie są poddawane analizie. Zerkając na graf nie można odróżnić zera od negatywnej wartości, wszystkie mapowane są do 0[11].

Wykres 3. Porównanie funkcji aktywacji ReLU i Leaky ReLU



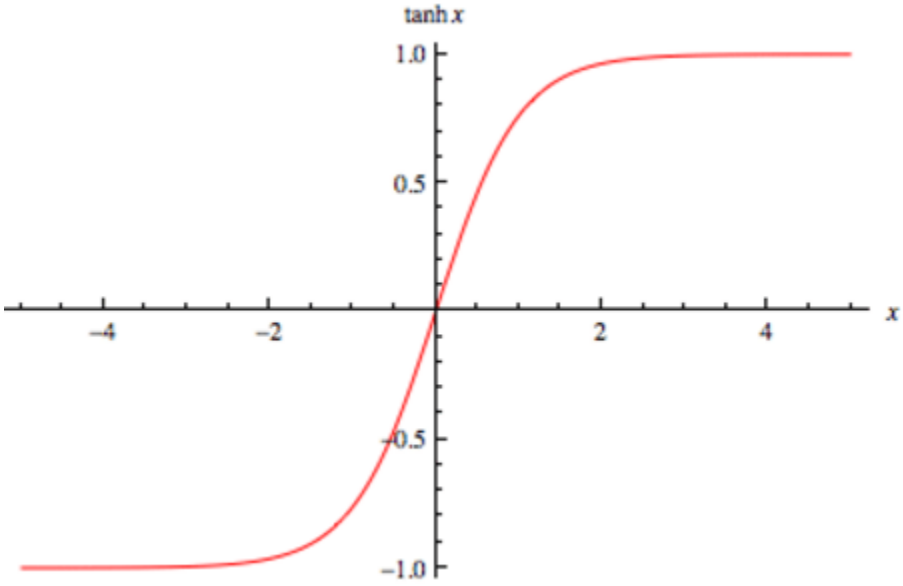
Źródło: X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* International Conference on Artificial Intelligence and Statistics 2010

Suma ważona na ostatniej warstwie sieci generatora jest poddawana funkcji aktywacji Tanh. Jej zaletą jest to, że negatywne wartości będą podkreślone na grafie, a także występuje rozróżnienie wartości negatywnych od zera. Minusem tej funkcji jest relatywna złożoność obliczeniowa[12]. Funkcja zdefiniowana jest wzorem:

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Jest to funkcja sigmoidalna, jej zbiór wartości to (-1, 1). Funkcja jest monotoniczna i różniczkowalna.

Wykres 4. Wykres funkcji aktywacji Tanh



Źródło: X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* International Conference on Artificial Intelligence and Statistics 2010

1.2.3. Teoria gier w GAN

GAN to przykład generatywnego modelu, jednak można spojrzeć na to zagadnienie poprzez teorię gry. Z racji tego, że wynik każdego z graczy zależy bezpośrednio od strategii jego przeciwnika, GAN można uznać za grę. Występuje w niej dwóch graczy: generator i klasyfikator.

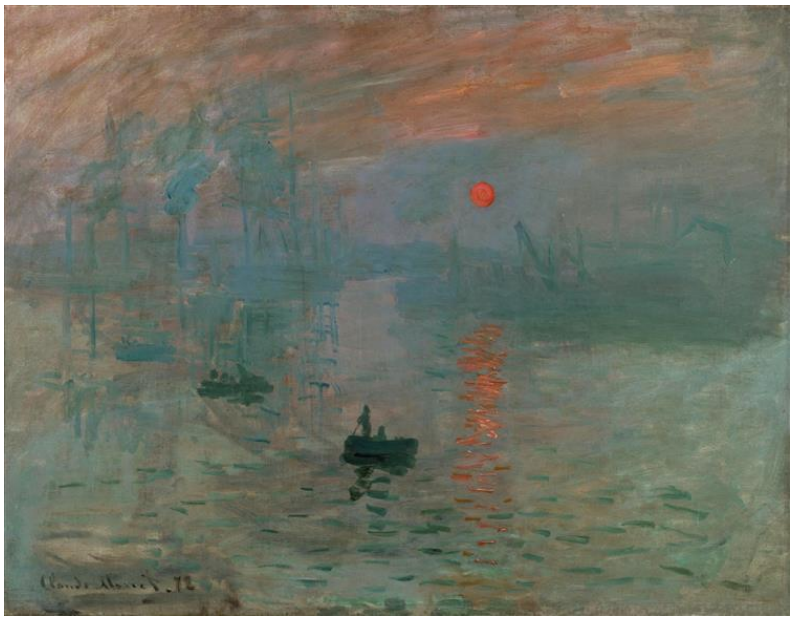
Celem gry jest osiągnięcie równowagi Nasha (ang. Nash equilibrium)[13]. Generator tworzy próbki danych, które mają podrabiać dane znajdujące się w oryginalnej bazie danych, natomiast klasyfikator ma zadanie rozróżnić, czy dana próbka jest częścią oryginalnej bazy danych czy została stworzona przez generator. Celem każdego z graczy jest zminimalizowanie spodziewanej wartości funkcji kosztu (ang. cost function), wyznaczającej „koszt" pewnego wydarzenia. Walczą o liczbę – jedno chce, by ona była jak najmniejsza, drugie jak największa. Ta liczba to współczynnik błędu klasyfikatora[14].

Idealnym zakończeniem eksperymentu, jeśli wziąć pod uwagę jakość generatora, jest zwrócenie przez klasyfikator prawdopodobieństwa o wartości 0.5 przy każdej inferencji. Oznacza to, że niemożliwe jest dokonanie przez klasyfikator decyzji, czy obraz podany na wejściu jest wytworzony przez generator czy pochodzi z oryginalnej bazy danych.

1.3. Styl impresjonistyczny

Nazwa impresjonizm (fr. impressionisme) oznacza „odbicie”, „wrażenie”. Zapoczątkował ją obraz Claude Monet – Impresja, wschód słońca (fr. Impression, soleil levant).

Obraz 4. Claude Monet, *Impresja, wschód słońca*



Źródło: WikiArt

Impresjonizm został wybrany do realizacji projektu z powodu swojej charakterystyki. Cechuje się krótkimi pociągnięciami pędzla, próbą „złapania uciekających chwil”. Skupia się on na powierzchowności, życiu codziennym, nie posiada głębokiego przesłania. Obrazy w stylu impresjonistycznym nie posiadają wielu detali, podkreślano grę naturalnego światła[15].

Często stosowano technikę impasto, w której kładziono farbę grubą warstwą. Kolejne warstwy mokrej farby nakładano na siebie bez czekania na wyschnięcie warstw, dzięki czemu uzyskiwano efekt miękkich krawędzi i przenikania się kolorów. Kolory nakładano obok siebie, przy jak najmniejszym mieszaniu; wykorzystywano zasadę jednoczesnego kontrastu, aby nadać obrazowi bardziej żywy wygląd[15].

Wpływ na powstanie takiego gatunku jak impresjonizm miał rozwój technologiczny w XIX wieku, kiedy opracowano syntetyczne pigmenty do farb dla artystów. Farby stały się bardziej wyraziste, zaczęły być dostępne w intensywnych odcieniach zieleni, niebieskiego i żółtego. Nowe procesy technologiczne umożliwiły zakup materiałów malarskich w niższych cenach. Połączenie tych rzeczy dodało popularności malowaniu w terenie, zamiast jak do tej pory w specjalnej pracowni. Częstym motywem stało się malowanie natury i krajobrazów[15].

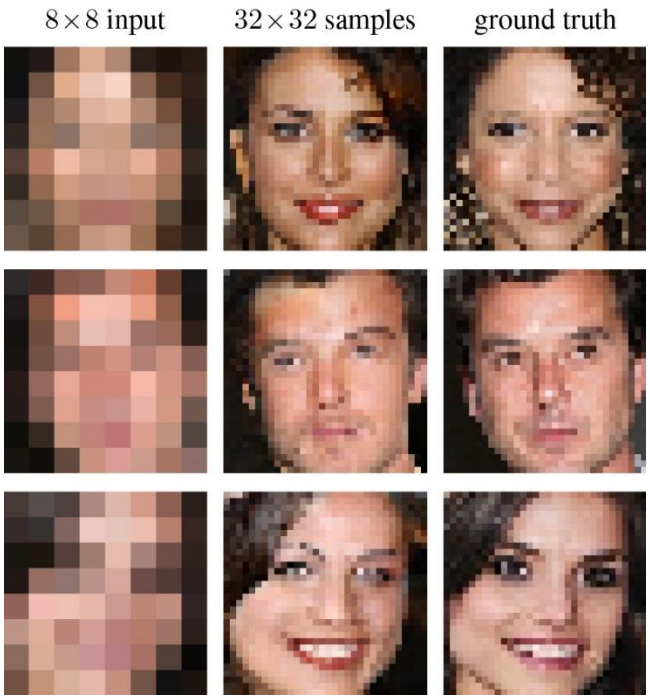
Rozdział 2. Wybór i integracja technologii

Wraz z wynalezieniem GAN modele generatywne zaczęły przynosić obiecujące wyniki w dziedzinie przetwarzania obrazu. Udostępniono wiele architektur, które różnią się od siebie możliwościami i efektywnością.

2.1. Wybór architektury GAN

SRGAN (ang. Super Resolution GAN) to architektura służąca do zwiększania rozdzielczości obrazów zadanych na wejściu. Działanie jest całkiem podobne jak wyostrzanie zdjęć w serialu CSI: Kryminalne zagadki Miami. Składa się z trzech sieci neuronowych – generatora, klasyfikatora i przetrenowanej sieci VGG-16, używanej do klasyfikacji i detekcji obiektów[16].

Obraz 5. Zwiększenie rozdzielczości zdjęcia za pomocą SRGAN



Źródło: Christian Ledig [et al.], *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, Conference on Computer Vision and Pattern Recognition 2017

Z lewej strony znajdują się obrazy składające się z 8x8 pikseli. Zostały one stworzone przez zmniejszenie oryginalnego zdjęcia z bazy danych do mniejszej rozdzielczości.

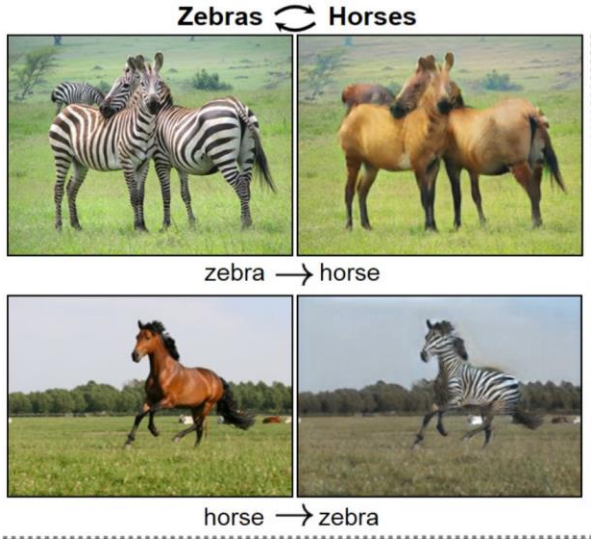
Po środku znajdują się twarze, które zostały wygenerowane przez SRGAN na podstawie próbki z lewej strony. Zwiększono rozdzielczość zdjęcia do 32x32 pikseli.

Z prawej strony znajduje się oryginalne zdjęcie zmniejszone do 32x32 pikseli, które jest wykorzystane do sprawdzenia jakości obrazów generowanych przez SRGAN.

CycleGAN to architektura umożliwiająca translację obrazu na obraz (ang. image-to-image translation); wydzielenie pewnego fragmentu jednego obrazu i podmiana innego elementu w innym obrazie[17].

Jeden z przykładów użycia CycleGAN: jeśli poda się na wejście zdjęcie z końmi, CycleGAN potrafi zamienić konia na zebrę (i na odwrót) z pozostawieniem tego samego tła. Wymagany był trening sieci neuronowych na bazie zdjęć koni i zebr.

Obraz 6. Translacja obrazu na obraz na przykładzie zebr i koni



Źródło: CycleGAN Github

Przy implementacji pracy inżynierskiej skorzystano z StyleGAN2, architektury GAN opartej na stylu (ang. style-based GAN architecture).

Obraz 7. Zdjęcia twarzy wygenerowane za pomocą StyleGAN2

Źródło: Tero Karras [et al.], *Analyzing and Improving the Image Quality of StyleGAN*, Conference on Computer Vision and Pattern Recognition 2019

Architektura ta umożliwia nauczanie z rozdzieleniem na wysokopoziomowe atrybuty, zapewnia stochastyczną wariację w generowanych obrazach[18]. Mimo że oryginalnie została przeznaczona do generowania zdjęć ludzkich twarzy, StyleGAN2 umożliwia trening na bazach zawierających inne obiekty. Przykład użycia odmiennej bazy można znaleźć nawet w oryginalnym repozytorium StyleGAN2 na Githubie, gdzie udostępniono wstępnie wytrenowane sieci na zdjęciach aut, kotów czy koni. Aby wygenerować obrazy należy użyć odpowiedniego zrserializowanego pliku pickle (.pkl) utworzonego za pomocą paczki „pickle" języka Python[19].

Obraz 8. Zdjęcia aut wygenerowane za pomocą StyleGAN2

Źródło: Tero Karras [et al.], *Analyzing and Improving the Image Quality of StyleGAN*, Conference on Computer Vision and Pattern Recognition 2019

2.2. Przygotowanie maszyny wirtualnej

Głównym narzędziem wykorzystanym przy implementacji projektu był StyleGAN2. Konsekwencją tego była potrzeba skonfigurowania środowiska do jego uruchomienia.

StyleGAN2 wymaga zasobów maszynowych przekraczających możliwości standardowego komputera do użytku domowego. Wymagany jest wysokiej klasy GPU NVIDIA wraz z sterownikami NVCC, architekturą CUDA w wersji 10.0 oraz biblioteką cuDNN w wersji 7.5. Zalecane jest GPU posiadające przynajmniej 16GB DRAMu. StyleGAN2 może zostać uruchomiony na obu systemach: Linux i Windows, jednak Linux jest polecany ze względu na większą stabilność. Wymagana jest 64-bitowa wersja Pythona 3.6 i TensorFlow w wersji 1.14 lub 1.15[19].

By osiągnąć wyniki podobne do tych prezentowanych przez twórców StyleGAN2, należy dostarczyć sporych zasobów sprzętowych. Twórcy technologii przeprowadzili pomiary na kartach graficznych Tesla V100, z których wyliczono orientacyjny czas treningu sieci neuronowych. Przy rozdzielczości 1024x1024 pikseli i przy użyciu 8 kart graficznych czas treningu wyniosi 9 dni i 18 godzin, zaś dla 1 karty 69 dni i 23 godziny[18].

2.2.1. Wymagania sprzętowe

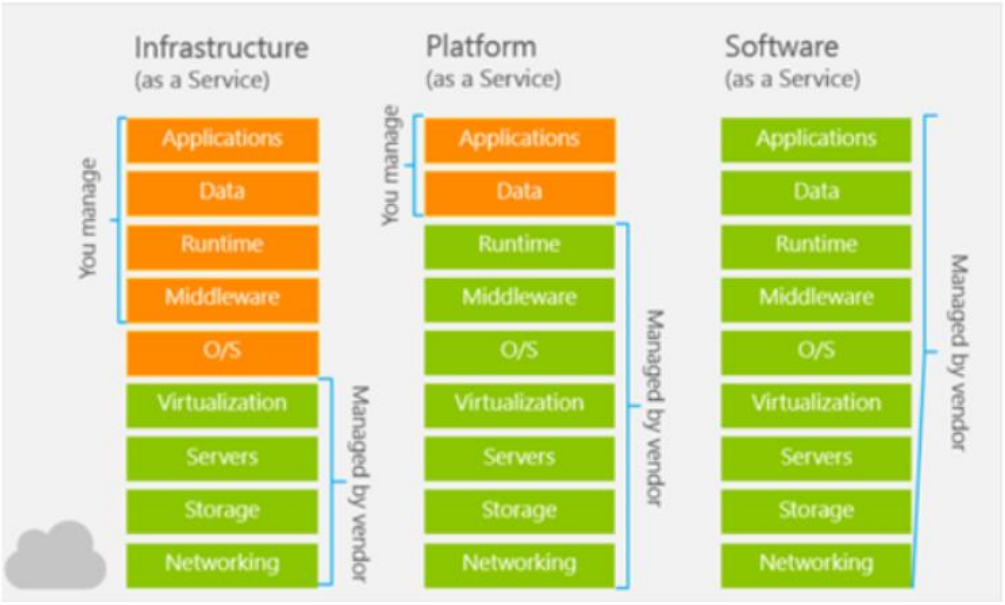
By zrealizować projekt pracy inżynierskiej zdecydowano się na wybór platformy Azure.

Azure jest platformą chmurową firmy Microsoft. Zdecydowano się właśnie na tę platformę z tego powodu, że udostępnia ona wszystkie zasoby potrzebne do realizacji pracy inżynierskiej w ramach pakietu dla studentów. Microsoft udostępnia swoje wszystkie zasoby do maksymalnie 100 dolarów amerykańskich na usługi w ramach platformy Azure. Dostępny był również pakiet dla dowolnego użytkownika, nie tylko studentów, wynosił on 175 dolarów amerykańskich, jednak ten pakiet nie obejmował rezerwacji maszyn o vcpu >= 6, a Azure wymaga właśnie takiej maszyny, by podłączyć do niej GPU[20].

Skorzystano z usługi IaaS – infrastruktura jako usługa (ang. Infrastructure as a Service). Dzięki temu nie zaistniała potrzeba kupna sprzętu do realizacji części praktycznej.

W momencie zaprzestania pracy wystarczyło zwolnić zajmowane zasoby by zlikwidować większość naliczanych kosztów. W momencie kontynuowania pracy należało wejść na platformę Azure i jednym kliknięciem alokować z powrotem zasoby sprzętowe, aby maszyna była gotowa do pracy. Odszedł również problem uaktualniania oprogramowania i sprzętu oraz rozwiązywania problemów ze sprzętem.

Obraz 9. Porównanie usług platformy Azure



Źródło: Dokumentacja Azure

IaaS obejmuje szereg różnych usług, zdecydowano się na wynajęciu zasobów sprzętowych, zarządzanych i udostępnianych za pomocą Internetu.

Wybrana maszyna wirtualna składała się z:

– dysku zawierającego system operacyjny Ubuntu Server 18.04 LTS

– tymczasowego dysku SSD, używanego do krótkoterminowego przechowywania danych aplikacji

– dysku HDD o pojemności 512GiB, który służy do długoterminowego przechowywania danych

– dodatkowego sterownika NVIDIA GPU Driver Extension

– 6 procesorów wirtualnych

– pamięci RAM o pojemności 56GiB

– akceleratora obliczeniowego Nvidia Tesla K80 posiadającego 24GB pamięci GDDR5, pozwalającej na przesył danych z przepustowością 580GB/s

– procesora Intel Xeon E5-2690 v3 o 124 wątkach, częstotliwości bazowej 2,6GHz oraz 30MB pamięci cache.

Architektura CUDA została wdrożona przez platformę Azure przy wybraniu wtyczki NVIDIA GPU Driver Extension.

Wybrano łączenie się z maszyną po SSH (ang. Secure shell, służącego do terminalowego łączenia się ze zdalnymi komputerami) na standardowym porcie TCP 22. Sposobem uwierzytelniania był klucz publiczny SSH wygenerowany przez platformę Azure.

Potrzebne było manualne dołączenie nowego dysku HDD na przechowywanie danych do pracy inżynierskiej. W tym celu wykonano następujące kroki:

1. Dodano nowy dysk na platformie Azure.
2. Połączono się do maszyny wirtualnej poprzez SSH.
3. Odnaleziono dysk dodany przez platformę Azure.
4. Przeprowadzono partycjonowanie nowego dysku.
5. Utworzono nowy katalog o nazwie „/datadrive”.
6. Wykonano mount dysku do katalogu „/datadrive”.

2.2.2. Środowisko programistyczne

Środowisko programistyczne złożone było z wielu technologii, które opisano poniżej.

CUDA (ang. Compute Unified Device Architecture) – architektura procesorów wielordzeniowych, umożliwiająca wydajniejsze rozwiązywanie problemów numerycznych. W przemyśle gier używana do renderowania grafiki, obliczeń fizycznych efektów takich jak dym, ogień i płyny.

NVCC (ang. NVIDIA CUDA Compiler) to kompilator przeznaczony do użytku wraz z architekturą CUDA.

cuDNN (ang. CUDA Deep Neural Network library) to biblioteka funkcji używanych w głębokich sieciach neuronowych takich jak np. propagacja wsteczna, czy pooling. Jest to biblioteka działająca na GPU.

Git – system kontroli wersji.

Python3 – wysokopoziomowy język programowania, posiada rozbudowany pakiet bibliotek standardowych.

ImageMagick – pakiet do obróbki grafiki.

Conda – manager paczek.

Zainstalowano narzędzie Git i za jego pomocą sklonowano repozytorium StyleGAN2.

Następnym krokiem było sprawdzenie czy instalacja NVCC przebiegła pomyślnie. Instalacja NVCC i CUDA nastąpiła po stronie Azure.

StyleGAN2 opiera się na zmodyfikowanej wersji TensorFlow Ops (zwane również TensorFlow Operations). TensorFlow Ops to węzły w formie grafu, które wykonują zdefiniowane operacje na Tensorach[21]. Są one kompilowane na początku treningu sieci neuronowych.

Wykonano test działania kompilatora udostępniony w repozytorium StyleGAN2, jednak nie przebiegł on pomyślnie i został zwrócony błąd o braku zainstalowanego NVCC. Problem rozwiązało dodanie następujących linii do pliku .bashrc.

Fragment kodu 3. Dodane linie do pliku .bashrc

```
export PATH="/usr/local/cuda-10.0/bin:$PATH"
export LD_LIBRARY_PATH="/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH"
```

Źródło: Opracowanie własne

Dodają one referencje architektury CUDA do zmiennych środowiskowych PATH i LD_LIBRARY_PATH. Nowa wartość zmiennych będzie równała się starej wartości poszerzonej o ścieżkę do CUDA.

W zmiennej środowiskowej PATH przechowywane są ścieżki do wykonywalnych programów.

W zmiennej LD_LIBRARY_PATH przechowywane są ścieżki do współdzielonych pomiędzy programami bibliotek.

W systemie Ubuntu 18.04 Python 3.6 jest domyślnie zainstalowany. Kod z repozytorium StyleGAN2 korzysta z modułów języka Python, które nie należą do standardowej biblioteki. W trakcie realizacji projektu stworzono wirtualne środowisko na dwa sposoby.

1. Stworzono wirtualne środowisko za pomocą narzędzia virtualenv i instalowano paczki za pomocą systemu pip.
2. Stworzono wirtualne środowisko i instalowano paczki za pomocą narzędzia Conda.

Główną różnicą jest to, że narzędzie Conda oprócz obsługiwania zależności między paczkami, sprawdza również zależności pomiędzy systemowymi bibliotekami i pakietami.

2.2.3. Biblioteki języka Python spoza standardowej biblioteki

TensorFlow to biblioteka używana w uczeniu maszynowym, metodach numerycznych i głębokich sieciach neuronowych. TensorFlow umożliwia przedstawianie obliczeń w formie grafu. Wierzchołki grafu przedstawiają działania matematyczne, takie jak dodawanie, mnożenie macierzy, obliczanie pochodnych funkcji. Krawędzie grafu to tensory łączące wierzchołki. Tensor może być sumą, wektorem, obrazem itd. Każdy wierzchołek grafu przybiera więc na początku różne tensory, przeprowadza obliczenia, i oddaje nowe tensory[21]. Użyto wersji ze wsparciem dla GPU.

Keras to biblioteka oprogramowania typu Open Source (otwarte oprogramowanie), która zapewnia interfejs Pythona dla sztucznych sieci neuronowych. Keras działa jako interfejs dla biblioteki TensorFlow.

NumPy to biblioteka przydatna przy pracy z tablicami i macierzami. Tablice NumPy są przechowywane w jednym miejscu w pamięci, dzięki czemu praca na nich jest bardziej wydajna. W standardowym obiekcie Pythonowym typu lista przechowywane są referencje do obiektów. Dzięki tej zmianie stosowany jest stos zamiast sterty, co przekłada się na wzrost prędkości wykonywania obliczeń.

Requests biblioteka służąca do komunikacji za pomocą protokołu HTTP (ang. Hypertext Transfer Protocol).

Pillow biblioteka służąca do otwierania i edycji plików graficznych.

SciPy biblioteka służąca do naukowych i technicznych obliczeń, zawiera moduły do optymalizacji, liniowej algebry, interpolacji, przetwarzania obrazów, etc.

OpenCV-Python (Open Source Computer Vision Library) biblioteka funkcji wykorzystywanych podczas uczenia maszynowego, obróbki obrazu, przetwarzania obrazu w czasie rzeczywistym.

Rozdział 3. Implementacja generatora zdjęć impresjonistycznych

Implementację generatora zdjęć impresjonistycznych podzielono na dwa etapy. Pierwszym było wstępne przetworzenie bazy danych (ang. data pre-processing), którego efektem była konwersja wszystkich zdjęć w bazie. Drugim było wykorzystanie przetworzonej bazy przy treningu sieci neuronowych za pomocą narzędzia StyleGAN2.

3.1. Przygotowanie bazy danych

Ważnym krokiem przy pracy nad sieciami neuronowymi jest odpowiednie przygotowanie danych. Rozpoczęto od zaznajomienia się bazą, zrozumienia jej struktury.

Przy implementacji projektu skorzystano z bazy WikiArt, przygotowanej w ramach realizacji projektu w pracy naukowej „Ceci n'est pas une pipe: A Deep Convolutional Network for Fine-art Paintings Classification", udostępnionej w repozytorium cs-chan/ArtGAN na Githubie. Zawiera ona w sobie 13 006 obrazów. Do bazy były załączone pliki konfiguracyjne, w których przyporządkowano obrazom jedną z klas „gatunek". Adnotacje zostały wykonane w pliku CSV (ang. comma-separated values) i były zawarte w dwóch plikach: pierwszy z danymi przeznaczonymi na trening, drugi z danymi przeznaczonymi do walidacji, swoim zakresem obejmowały większość zbioru obrazów WikiArt. Napisano skrypt w Pythonie, który zebrał i posegregował wyniki. Z 13 006 obrazów 11 532 miało przypisany gatunek. Najwięcej obrazów w bazie danych przedstawia krajobraz i pejzaż miejski.

Tabela 1. Tabela przedstawiająca ilość obrazów w bazie danego gatunku

Gatunek obrazu	Ilość obrazów
Obraz abstrakcyjny	1545
Pejzaż miejski	2669
Malarstwo rodzajowe	1
Krajobraz	4365
Akt	360
Portret	1899
Obraz religijny	18
Szkic	244
Martwa natura	431

Źródło: opracowanie własne

3.1.1. Wstępne przetworzenie danych

Za pomocą pakietu ImageMagic zamieniono przestrzeń barw obrazów na RGB, na wypadek gdyby któryś obrazów był w skali szarości. Był to niezbędny krok do zamiany obrazów na pliki TFRecord.

StyleGAN2 wymaga od zdjęć, żeby były w kształcie kwadratu. Obrazy zawarte w bazie miały tak różne proporcje, że niemożliwym była zmiana obrazu na kwadrat bezstratnie. Rozważano dwa podejścia do tego problemu – jednym było wycięcie centralnego kwadratu z obrazu. Drugą opcją była zmiana kształtu obrazu bez zachowania oryginalnych proporcji zdjęcia. Dla lepszego zobrazowania zmian naniesionych na obrazy, poniżej umieszczono i omówiono zdjęcie szczeniaka poddane obu metodom.

Obraz 10. Nieprzetworzone zdjęcie psa

Źródło: zasoby własne

Obraz 11. Zdjęcie z którego wycięto centralny kwadrat

Źródło: zasoby własne

Wycięcie centralnego kwadratu spowodowało to utratę fragmentów znajdujących się po bokach.

Obraz 12. Zdjęcie którego kształt zmieniono na kwadrat nie zachowując oryginalnych proporcji zdjęcia

Źródło: zasoby własne

Zdecydowano się na zachowanie wszystkich elementów obrazów. Zmieniono proporcje zdjęć, przez co niektóre obrazy były w małym stopniu zniekształcone, jednak zachowały one swoją czytelność. Kolejną zaletą jest zachowanie kompozycji obrazu. Przy tak dużej ilości zdjęć nie było opłacalne ręczne przycinanie każdej sztuki do kwadratu. Nie jest wiadome, jaka część obrazu jest najważniejsza i czy po przycięciu pewnych fragmentów nie straci on swoich ważnych elementów.

By osiągnąć jak najlepsze efekty przy mniejszym nakładzie zasobowym pod względem sprzętu, zdecydowano się na rozdzielczość obrazu 256x256 pikseli.

3.1.2. Konwersja obrazów

StyleGAN2 nie operuje na bitmapie, na przykład na obrazie o rozszerzeniu .jpg. Korzysta on z formatu TFRecord, który służy do przechowywania zestawów danych w formie sekwencji binarnych rekordów. Dzięki wstępnemu przetworzeniu danych zostaną one przedstawione w formie, która będzie szybsza do załadowania i w ciągu dalszej pracy nie będzie trzeba ponownie przetwarzać oryginalnych obrazów.

Po przygotowaniu bazy danych należało skonwertować obrazy do formatu TFRecord. StyleGAN2 zawiera w sobie specjalne służące do tego narzędzie - dataset_tool. Za jego pomocą skonwertowano bitmapy na pliki TFRecords. Wówczas pojawił się błąd przestawiony poniżej.

Obraz 13. Błąd pojawiający się podczas konwesji bitmapy

```
(.venv) student@gan-ubuntu:/datadrive/stylegan2$ python dataset_tool.py
create_from_images /datadrive/stylegan2/Impressionism_tf
/datadrive/stylegan2/Impressionism
Loading images from "/datadrive/stylegan2/Impressionism"
Traceback (most recent call last):
  File "dataset_tool.py", line 643, in <module>
    execute_cmdline(sys.argv)
  File "dataset_tool.py", line 638, in execute_cmdline
    func(**vars(args))
  File "dataset_tool.py", line 509, in create_from_images
    resolution = img.shape[0]
IndexError: tuple index out of range
```

Źródło: opracowanie własne

Postanowiono zmienić sposób tworzenia wirtualnego środowiska. Skorzystano z managera pakietów Conda. Do pełnego środowiska wystarczyło zainstalować paczkę TensorFlow w wersji 1.15, a manager paczek sam zainstalował wszystkie zależności, które były potrzebne. Manager paczek również odpowiednio zmienił wersje innych paczek, by nie nastąpiły konflikty.

Obraz 14. Paczki zainstalowane i zaktualizowane za pomocą managera paczek Conda

```
The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    _anaconda_depends-2020.07  |           py36_0           6 KB
    _tflow_select-2.1.0        |             gpu            2 KB
    absl-py-0.11.0             |     pyhd3eb1b0_1         103 KB
    anaconda-custom            |           py36_1           3 KB
    astor-0.8.1                |           py36_0          47 KB
    c-ares-1.17.1              |       h27cfd23_0         108 KB
    certifi-2020.12.5          |   py36h06a4308_0         140 KB
    cudatoolkit-10.0.130       |                0       261.2 MB
    cudnn-7.6.5                |       cuda10.0_0       165.0 MB
    cupti-10.0.130             |                0         1.5 MB
    gast-0.2.2                 |           py36_0         155 KB
    google-pasta-0.2.0         |             py_0          46 KB
    grpcio-1.31.0              |   py36hf8bcb03_0         2.0 MB
    keras-applications-1.0.8   |             py_1          29 KB
    keras-preprocessing-1.1.0  |             py_1          37 KB
    libprotobuf-3.13.0.1       |       hd408876_0         2.0 MB
    markdown-3.3.3             |   py36h06a4308_0         128 KB
    openssl-1.1.1i             |       h27cfd23_0         2.5 MB
    opt_einsum-3.1.0           |             py_0          54 KB
    protobuf-3.13.0.1          |   py36he6710b0_1         633 KB
    tensorboard-1.15.0         |     pyhb230dea_0         3.2 MB
    tensorflow-1.15.0          |gpu_py36h5a509aa_0           4 KB
    tensorflow-base-1.15.0     |gpu_py36h9dcbed7_0       156.5 MB
    tensorflow-estimator-1.15.1|     pyh2649769_0         271 KB
    tensorflow-gpu-1.15.0      |       h0d30ee6_0           3 KB
    termcolor-1.1.0            |           py36_1           8 KB
    werkzeug-0.16.1            |             py_0         258 KB
    ------------------------------------------------------------
                                          Total:       595.9 MB
```

Źródło: opracowanie własne

Po zmianie wirtualnego środowiska, pomyślnie skonwertowano obrazy na format TFRecord. Dane skonwertowane w ten sposób zajmują około 18 razy więcej pamięci niż oryginalna baza zdjęć.

3.2. Trening sieci neuronowych

Trening klasyfikatora i generatora w architekturze StyleGAN2 należy do czasochłonnych procesów. W ninejszym podrozdziale ukazano rozwój generowanych obrazów w różnych odstępach czasowych, rozpoczynając od inicjalnego stanu sieci neuronowych. Obrazy zostały poddane analizie empirycznej, skupiając uwagę na zmieniających się atrybutach pomiędzy wyszczególnionymi próbkami.

3.2.1. Dobór parametrów treningu

Ważnym elementem pracy był dobór parametrów przy technologii StyleGAN2.

Tabela 2. Parametry użyte przy technologii StyleGAN2

parametr	wartość	opis
data-dir	/datadrive/stylegan2	ścieżka do katalogu z bazą danych w formacie TFRecord
dataset	Impressionism_tf	nazwa bazy danych
config	config-f	config którego należy użyć do treningu
num-gpus	1	ilość dostępnych GPU do treningu
total-kimg	13000	długość treningu w tysiącach obrazów
metrics	fid50k/None	lista metryk do zebrania, rozdzielana przecinkiem

Źródło: opracowanie własne

Początkowo chciano zbierać metrykę FID - odległości początkowej Frécheta. Dzięki niej można efektywnie mierzyć jakość generatora[13]. Była jednak zbyt wyczerpująca zasobowo, co 3,5 godziny treningu marnowano 80 minut na zbieranie metryki. Podjęto decyzję, by wykorzystać wszystkie dostępne zasoby na trening generatora.

Kolejne parametry wymagały ręcznej zmiany kodu w repozytorium StyleGAN2 w pliku stylegan2/run_training.py.

Parametr train.image_snapshot_ticks oznacza częstość generowania obrazu na aktualnym stanie wytrenowania sieci neuronowych. Jedynką jest tutaj zakończenie jednego cyklu treningowego, który trwa średnio 3,5 godziny.

Parametr train.network_snapshot_ticks oznacza częstość zapisywania stanu sieci neuronowej. Tutaj również jedynka oznacza jeden cykl treningowy.

Fragment kodu 4. Zmienione wartości parametrów w repozytorium StyleGAN2

```
train.image_snapshot_ticks = 1
train.network_snapshot_ticks = 2
```

Źródło: opracowanie własne

Przy testach parametrów wystąpiła potrzeba wznowienia przerwanego treningu. Nie jest to pokryte dokumentacją StyleGAN2, jednak taka opcja istnieje. Należy edytować plik stylegan2/training/training_loop.py

Fragment kodu 5. Fragment kodu odpowiedzialny za wznowienie treningu

```
resume_pkl              = None,     # Network pickle to resume training from,
None = train from scratch.
resume_kimg             = 0.0,      # Assumed training progress at the
beginning. Affects reporting and training schedule.
```

Źródło: Repozytorium StyleGAN2

I zmienić następujące parametry:

resume_pkl – ścieżka do sieci neuronowej w postaci zserializowanego pliku .pkl

resume_kimg – określa etap uczenia od którego należy wznowić trening, wartość zmiennoprzecinkowa

3.2.2. Proces uczenia

Kod źródłowy w architekturze StyleGAN2 jest bardzo czytelnie napisany, co umożliwiło sprawną pracę przy jego użyciu. Nazwy funkcji i zmiennych były utrzymane w poprawnej konwencji, z punktu widzenia programisty ułatwiło to wdrożenie się do projektu. Twórcy dodali również bardzo wartościowe komentarze przy definiowaniu parametrów użytych w treningu.

Do minusów architektury StyleGAN2 należy mało rozwinięta dokumentacja. Szerszy opis środowiska programistycznego, a chociażby pełny spis wymagań, oszczędziłby dużo czasu pracy przy tym narzędziu. Konfiguracja środowiska polegała bowiem w sporej części na ręcznym przeglądaniu kodu w poszukiwaniu niespisanych wymagań.

Przy próbie uruchomienia treningu pojawił się następujący błąd na próbie skompilowania wtyczki upfirdn_2d.cu.:

Obraz 15. Błąd kompilacji wtyczki TensorFlow

```
Setting up TensorFlow plugin "fused_bias_act.cu": Preprocessing... Compil-
ing... Loading... Done.
Setting up TensorFlow plugin "upfirdn_2d.cu": Preprocessing... Compiling...
Error
tensorflow.python.framework.errors_impl.NotFoundError: /datadrive/style-
gan2/dnnlib/tflib/_cudacache/fused_bias_act_ab9576f7069b67e28efa01980fb6e5c8.
so: undefined symbol:  _ZN10tensorflow12OpDefBuilder6OutputESs
```

Źródło: opracowanie własne

Zdecydowano się na wypróbowanie innej wersji paczki TensorFlow, tym razem w wersji 1.14. Przygotowano środowisko za pomocą managera paczek Conda. Skończyło się to jednak identycznym błędem.

Potrzebna była zmiana w pliku repozytorium stylegan stylegan2/dnnlib/tflib/custom_ops.py, w którym m.in. definiuje się flagi używane do kompilowania wtyczek. Zamieniono „-D_GLIBCXX_USE_CXX11_ABI=0” na „D_GLIBCXX_USE_CXX11_ABI=1”.

Fragment kodu 6. Fragment pliku custom_ops.py, w którym zdefiniowano flagi używane do kompilowania wtyczek

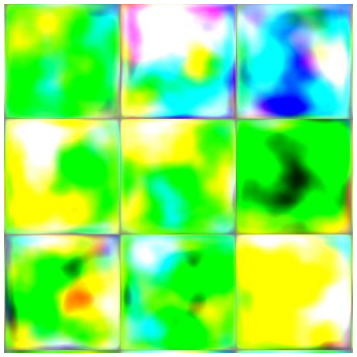
```
compile_opts += '"%s"' % os.path.join(tf.sysconfig.get_lib(), 'python',
'_pywrap_tensorflow_internal.so')
compile_opts += ' --compiler-options \'-fPIC -D_GLIBCXX_USE_CXX11_ABI=0\''
```

Źródło: Repozytorium StyleGAN2

Wynikało to z rozbieżności pomiędzy wersją TensorFlow i gcc (ang. GNU Compiler Collection), używanego do budowania wtyczki.

Możliwość generowania obrazów pojawiła się tuż po zainicjalizowaniu sieci neuronowej. Wystarczyło użyć skryptu stylegan2/run_generator.py, wskazując jako parametr ścieżkę do pliku pickle, który zawierał w sobie zserializowane sieci neuronowe.

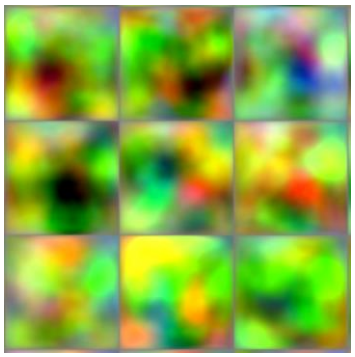
Obraz 16. Obrazy wygenerowane przed procesem treningu



Źródło: opracowanie własne

Uczenie generatora rozpoczyna się losowym szumem, nieukierunkowanym w żadną stronę. Jest to ślepy traf generatora na to, jak może potencjalnie wyglądać baza danych, którą próbuje odtworzyć.

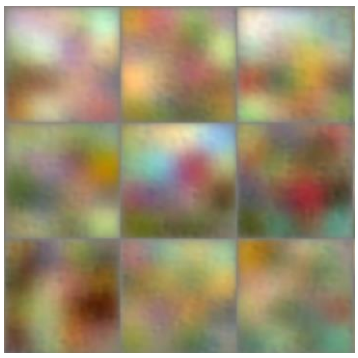
Obraz 17. Obrazy wygenerowane po 4 godzinach treningu



Źródło: opracowanie własne

Generator zaczął tworzyć obrazy z nakładającymi się na siebie plamami, poszerzyła się gama kolorystyczna.

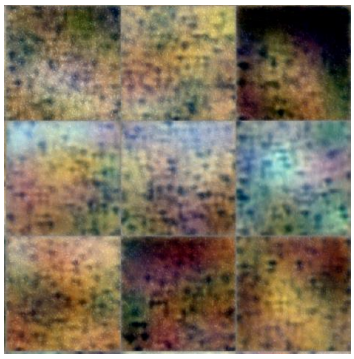
Obraz 18. Obrazy wygenerowane po 11 godzinach treningu



Źródło: opracowanie własne

Zaczęła być dostrzegalna wyraźna szachownica kolorów, które zostały stonowane. Na obrazach pojawiły się gęsto rozmieszczone, lekko widoczne kropki.

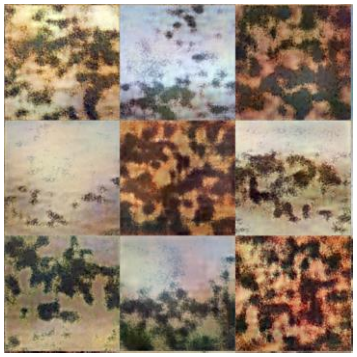
Obraz 19. Obrazy wygenerowane po 21 godzinach treningu



Źródło: opracowanie własne

Pojawiły się dostrzegalne plamy, rozpoczął się proces tworzenia kształtów.

Obraz 20. Obrazy wygenerowane po 39 godzinach treningu



Źródło: opracowanie własne

Obrazy zaczęły przyjmować kolorystykę bardziej zbliżoną do tej zajmującej największą powierzchnię w bazie obrazów impresjonistycznych. Plamy ciemnego koloru zaczęły się grupować, tło uzyskało gładkie przejścia między kolorami.

Obraz 21. Obrazy wygenerowane po 53 godzinach treningu

Źródło: opracowanie własne

Nastąpił rozwój w mapowaniu kształtów i tekstury w generowanych obrazach.

Obraz 22. Obrazy wygenerowane po 81 godzinach treningu

Źródło: opracowanie własne

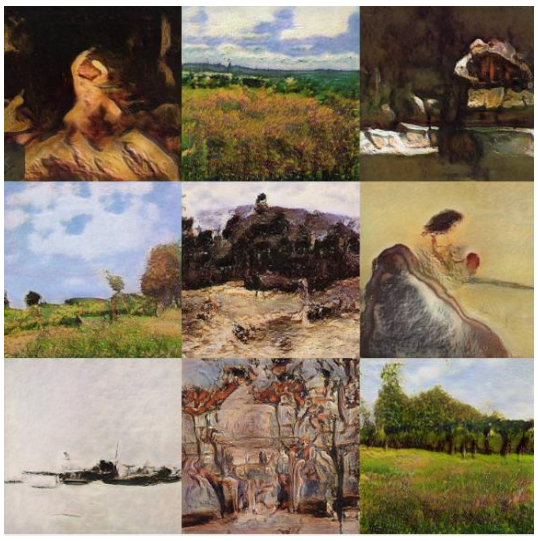
Proces treningu zaczął przynosić mniejszy efekt wizualny, różnice w generowanych obrazach co iterację były coraz mniejsze.

Obraz 23. Obrazy wygenerowane po 133 godzinach treningu

Źródło: opracowanie własne

Na obrazach zaczęły pojawiać się widoczne kształty różnych obiektów, trening powoli zmierzał w sprzyjającą stronę.

Obraz 24. Obrazy wygenerowane po 203 godzinach treningu



Źródło: opracowanie własne

Na obrazach zaczęły być dostrzegalne drzewa, rośliny, a także zarysy ludzi.

Powyższe próbki nie przestawiają postępu w równych ramach czasowych. Im dłużej trwał trening, tym mniejszy wizualnie wydawał się postęp od poprzedniego tika zegara. Wybrano próbki w taki sposób, by najlepiej pokazać następujące po sobie zmiany.

Perspektywy rozwoju

Przez niewystarczającą ilość zasobów sprzętowych nie dokonano szerokich testów stworzonego generatora. Do oceny jego jakości użyto badania empirycznego, polegającego na porównaniu atrybutów istotnych w impresjonizmie i tych obecnych w generowanych obrazach.

Do monitorowania jakości generatora można skorzystać z szerokiej gamy metryk zaimplementowanych w StyleGAN2: odległość początkowa Frécheta, inception score, perceptual path length, liniowa niezależność, precyzja i czułość[19].

StyleGAN2 nie osiąga najlepszych wyników na zbyt różnorodnych zestawach danych. By poprawić jakość generowanych obrazów można dokonać operacji zwanej discriminator rejection sampling[22]. Polega ona na użyciu klasyfikatora wytrenowanego przy architekturze GAN. Za jego pomocą skanowane są wszystkie obraz znajdujące się w bazie, klasyfikator nadaje im wartość – na ile jest pewien, że obraz jest impresjonistyczny. Następnie odrzuca się najbardziej odstające próbki w bazie – jeśli dyskryminator nadał im niską wartość, mogą psuć rozwój generatora. Należy potem wznowić trening na pomniejszonej bazie danych.

Rozważano podejścia do problemu generowania próbek słabszej jakości w przypadku generowanych obrazów przedstawiających ludzi. Ciekawym eksperymentem byłoby przeprowadzenie strojenia sieci neuronowej z użyciem obrazów impresjonistycznych z ludźmi. Podstawą są tutaj wcześniej wyliczone wagi sieci neuronowych, które przechowują swego rodzaju esencję czym są obrazy impresjonistyczne. GAN skupiłby się na udoskonaleniu tworzenia ludzi na tworzonych obrazach. Aplikacja generująca obrazy składałaby się z dwóch sieci neuronowych generujących obrazy: jednej generującej obrazy z ludźmi, drugiej będącą generatorem wytworzonym podczas tej pracy praktycznej.

Rozważano przeprowadzenie testów automatycznych, które sprawdzałyby, czy generowane obrazy rzeczywiście należą do stylu impresjonistycznego. Jedną z możliwych metod jest skorzystanie z modelu, który klasyfikuje style obrazów. Zautomatyzowanie nastąpiłoby poprzez wyserwowanie tego modelu na serwerze OpenVINO™ Model Server, skalowalnym i wydajnym rozwiązaniu do obsługi modeli uczenia maszynowego[23]. Poprzez użycie klientów udostępnionych w repozytorium openvinotoolkit/model_server, można

dokonać inferencji na modelu poprzez podanie obrazów wytworzonych przez generator. Analiza stylów przypisanych przez klasyfikator wygenerowanym próbkom, dałaby wstępne potwierdzenie, że obrazy rzeczywiście należą do stylu impresjonistycznego. Nie skorzystano z tego rozwiązania, gdyż nie znaleziono gotowego modelu do klasyfikowania stylów obrazów, który miałby dokładność większą niż 60%. Wytrenowanie takiego modelu samemu byłoby natomiast materiałem na osobną pracę inżynierską.

Zakończenie

Cel pracy inżynierskiej został osiągnięty. Stworzono generator obrazów impresjonistycznych, który tworzy obrazy przypominające te wywodzące się z nurtu impresjonistycznego, jednak nie istniejące nigdy wcześniej. Pełen czas treningu generatora wyniósł 8 dni i 3 godziny. Po tym okresie wyczerpała się pula 100 dolarów amerykańskich udostępnionych studentom w ramach platformy Azure.

W zdjęciach przedstawiających naturę doskonale widoczna jest gra światłem tak charakterystyczna dla stylu impresjonistycznego.

Obraz 25. Wygenerowane obrazy przedstawiające taflę wody

Źródło: opracowanie własne

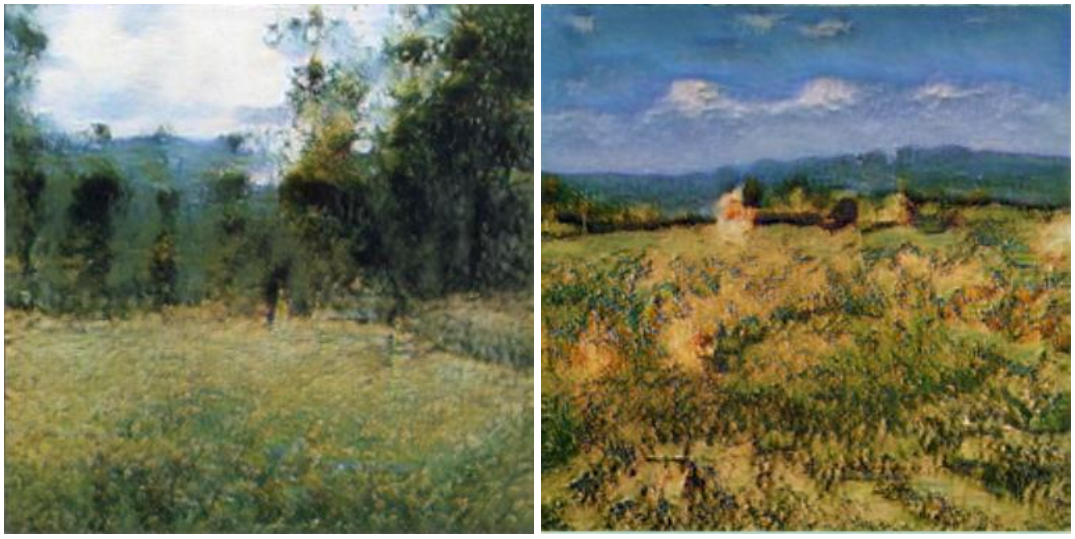
Generator w zdjęciach przedstawiających taflę wody nie zapomniał o odbiciu tego, co znajduje się ponad nią. Niektóre elementy zostały zniekształcone lub pominięte, ale sieć neuronowa dosyć dobrze pojęła wygląd i zachowanie wody.

Obraz 26. Wygenerowane obrazy przedstawiające krajobraz

Źródło: opracowanie własne

Obrazy mają efekt krótkich pociągnięć pędzla, rozmytych kontur. Jest to zasługą połączenia dwóch rzeczy. Dane wejściowe przyjmowały taką formę, więc sieć neuronowa zaczęła kopiować te atrybuty. Dodatkowo architektura GAN ma w zwyczaju skupiać się na teksturach, więc generator priorytetowo zajął się odtworzeniem efektu malowania farbami i pędzlem.

Obraz 27. Wygenerowane obrazy przedstawiające łąkę



Źródło: opracowanie własne

Generowane obrazy przedstawiały zróżnicowane lokalizacje.

Obraz 28. Wygenerowane obrazy przedstawiające krajobraz

Źródło: opracowanie własne

Generowanie ludzi nie poszło już tak dobrze jak w przypadku krajobrazów. Pierwszym czynnikiem, który na to wpłynął, to fakt że człowiek jest relatywnie ciężkim elementem do odtworzenia. Wszelkie odchylenia od normy są widoczne na pierwszy rzut oka.

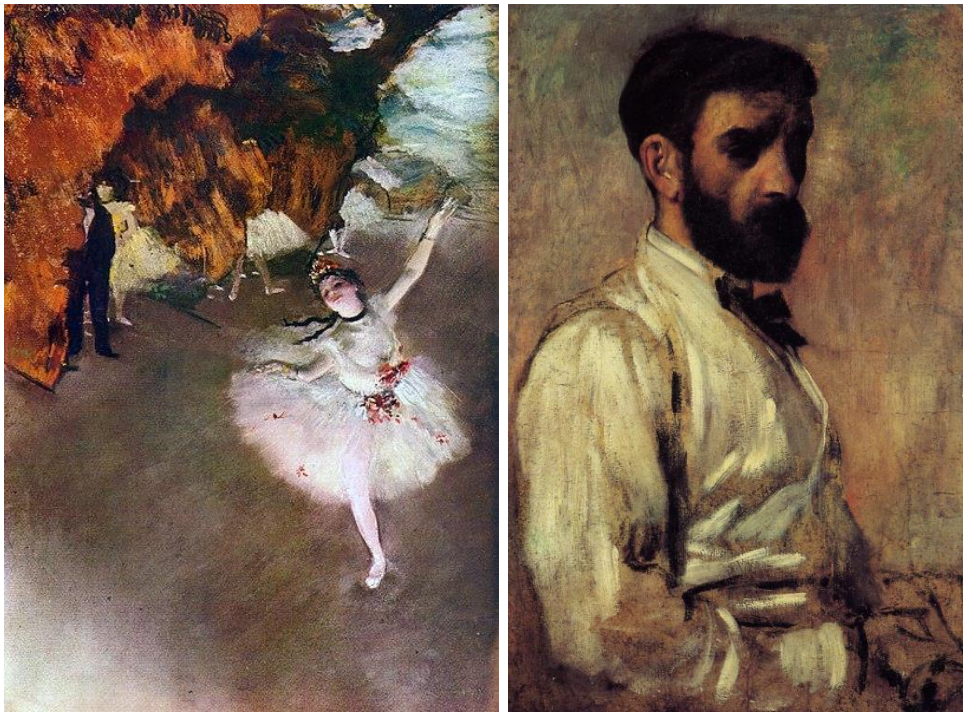
Obraz 29. Wygenerowane obrazy przedstawiające ludzi

Źródło: opracowanie własne

GAN działa najlepiej przy bazach danych, które są mało urozmaicone. Dobrym przykładem są wygenerowane zdjęcia twarzy z „This Person Does Not Exist”. Baza danych, na której był uruchomiony trening, składała się z kwadratowych zdjęć twarzy, a sama twarz zajmowała większą część zdjęcia[24]. Najbardziej zróżnicowanym elementem zdjęcia było jego tło, w pokazanych przykładach wyglądające jak bliżej nieokreślony szum.

W bazie WikiArt występowały obrazy ludzi o różnym wyglądzie, pozach, namalowanych w zupełnie inny sposób. Poniżej pokazano dwa obrazy z bazy WikiArt w stylu impresjonistycznym. Wygląd, pozy, umiejscowienie namalowanych postaci są zróżnicowane. Są one również namalowane w inny sposób, podkreślone zostały inne atrybuty.

Obraz 30. Obrazy Edgar Degas (od lewej): *Primaballerina* oraz *Portret Leona Bonnat*



Źródło: WikiArt

Istnieje parę sposobów na to by rezultaty generowane przez sieć neuronową były bardziej przypominające te z oryginalnej bazy danych. Na poziomie obecnej technologii eksperymenty związane z generowaniem grafiki należą do tych wymagających kosztownego sprzętu. Był to czynnik ograniczający możliwy rozwój generowanych obrazów. Wydłużenie czasu treningu, zwiększenie zasobów sprzętowych – są to rzeczy, które sprawiłyby, że osiągnięte wyniki byłyby bardziej impresywne.

Spis wykorzystanych źródeł

[1] L. Shamir [et al.], *Impressionism, Expressionism, Surrealism: Automated Recognition of Painters and Schools of Art*, Intramural Research Program of the NIH 2010

[2] S. Karayev [et al.], Recognizing image style, arXiv:1311.3715 2013

[3] B. Saleh, A. Elgammal, *Large-scale Classification of Fine-Art Paintings: Learning The Right Metric on The Right Feature*, arXiv:1505.00855 2015

[4] B. Saleh [et al.], *Toward automated discovery of artistic influence, Multimedia Tools and Applications*, The State University of New Jersey 2016

[5] B. Mehlig, *Artificial Neural Networks*, Gothenburg University 2019

[6] David Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*, O'Reilly 2019

[7] Yonghui Wu [et al.], *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, Cornell University 2016

[8] Tero Karras [et al.], *A Style-Based Generator Architecture for Generative Adversarial Networks*, Conference on Computer Vision and Pattern Recognition 2019

[9] Tim Salimans [et al.], *Improved Techniques for Training GANs*, arXiv:1606.03498 2016

[10] A. Mass [et al.], *Rectifier nonlinearities improve neural network acoustic models*, Stanford University 2013

[11] S. Behnke, *Hierarchical Neural Networks for Image Interpretation. Lecture Notes in Computer Science*, Springer 2003

[12] X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, International Conference on Artificial Intelligence and Statistics 2010

[13] Martin Heusel [et al.], *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, Johannes Kepler University Linz 2018

[14] Barbara Franci [et al.], *A game-theoretic approach for Generative Adversarial Networks*, Cornell University 2020

[15] www.metmuseum.org

[16] Christian Ledig [et al.], *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, Conference on Computer Vision and Pattern Recognition 2017

[17] Jun-Yan Zhu [et al.], *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, International Conference on Computer Vision 2017

[18] Tero Karras [et al.], *Analyzing and Improving the Image Quality of StyleGAN*, Conference on Computer Vision and Pattern Recognition 2020

[19] Repozytorium NVlabs/stylegan2 na Githubie

[20] Dokumentacja Azure

[21] Dokumentacja TensorFlow

[22] S. Azadi [et al.], *Discriminator Rejection Sampling*, International Conference on Learning Representations 2019

[23] Repozytorium *openvinotoolkit/model_server* na Githubie

[24] V. Varkarakis [et al.], *Re-Training StyleGAN - A First Step Towards Building Large, Scalable Synthetic Facial Datasets*, National University of Ireland Galway 2020

Spis obrazków

[1] Przykładowe zdjęcia umieszczone na stronie This Person Does Not Exist, thispersondoesnotexist.com [dostęp z dnia 05.07.2020]

[2] Zakłócenia w generowanych zdjęciach twarzy, thispersondoesnotexist.com [dostęp z dnia 05.07.2020]

[3] Porównanie generowanych obrazów w przeciągu lat, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*, O'Reilly 2019

[4] Claude Monet, *Impresja, wschód słońca*, WikiArt

[5] Zwiększenie rozdzielczości zdjęcia za pomocą SRGAN, Christian Ledig [et al.], *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, Conference on Computer Vision and Pattern Recognition 2017

[6] Translacja obrazu na obraz na przykładzie zebr i koni, CycleGAN Github

[7] Zdjęcia twarzy wygenerowane za pomocą StyleGAN2, Tero Karras [et al.], *Analyzing and Improving the Image Quality of StyleGAN*, Conference on Computer Vision and Pattern Recognition 2019

[8] Zdjęcia aut wygenerowane za pomocą StyleGAN2, Tero Karras [et al.], *Analyzing and Improving the Image Quality of StyleGAN*, Conference on Computer Vision and Pattern Recognition 2019

[9] Porównanie usług platformy Azure, Dokumentacja Azure [dostęp z dnia 12.08.2020]

[10] Nieprzetworzone zdjęcie psa, zasoby własne

[11] Zdjęcie z którego wycięto centralny kwadrat, zasoby własne

[12] Zdjęcie którego kształt zmieniono na kwadrat nie zachowując oryginalnych proporcji zdjęcia, zasoby własne

[13] Błąd pojawiający się podczas konwesji bitmapy, opracowanie własne

[14] Paczki zainstalowane i zaktualizowane za pomocą managera paczek Conda, opracowanie własne

[15] Błąd kompilacji wtyczki TensorFlow, opracowanie własne

[16] Obrazy wygenerowane przed procesem treningu, opracowanie własne

[17] Obrazy wygenerowane po 4 godzinach treningu, opracowanie własne

[18] Obrazy wygenerowane po 11 godzinach treningu, opracowanie własne

[19] Obrazy wygenerowane po 21 godzinach treningu, opracowanie własne

[20] Obrazy wygenerowane po 39 godzinach treningu, opracowanie własne

[21] Obrazy wygenerowane po 53 godzinach treningu, opracowanie własne

[22] Obrazy wygenerowane po 81 godzinach treningu, opracowanie własne

[23] Obrazy wygenerowane po 133 godzinach treningu, opracowanie własne

[24] Obrazy wygenerowane po 203 godzinach treningu, opracowanie własne

[25] Wygenerowane obrazy przedstawiające taflę wody

[26] Wygenerowane obrazy przedstawiające krajobraz

[27] Wygenerowane obrazy przedstawiające łąkę

[28] Wygenerowane obrazy przedstawiające krajobraz

[29] Wygenerowane obrazy przedstawiające ludzi

[30] Obrazy Edgar Degas (od lewej): *Primaballerina* oraz *Portret Leona Bonnat*

Spis tabel

[1] Tabela przedstawiająca ilość obrazów w bazie danego gatunku

[2] Parametry użyte przy technologii StyleGAN2

Spis wykresów

[1] Wykres funkcji LeakyReLU, A. Mass [et al.], *Rectifier nonlinearities improve neural network acoustic models*, Stanford University 2013

[2] Wykres funkcji ReLU, S. Behnke, *Hierarchical Neural Networks for Image Interpretation. Lecture Notes in Computer Science*, Springer 2003

[3] Porównanie funkcji aktywacji ReLU i Leaky ReLU, X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, International Conference on Artificial Intelligence and Statistics 2010

[4] Wykres funkcji aktywacji Tanh, X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, International Conference on Artificial Intelligence and Statistics 2010

Spis schematów

[1] Przedstawienie działania neuronu, B. Mehlig, *Artificial Neural Networks* Gothenburg University 2019

[2] Przedstawienie działania sieci neuronowej, B. Mehlig, *Artificial Neural Networks*, Cornell University 2019

[3] Schemat sieci neuronowej zwracającej prawdopodobieństwo, z jakim dany obraz został namalowany przez Vincenta Van Gogha, David Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*, O'Reilly 2019

Spis załączników

parse_csv_files.py – skrypt przetwarzający pliki konfiguracyjne z bazy danych WikiArt

generator.pkl – zserializowana sieć neuronowa, za pomocą której można generować obrazy