



**WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ**
Uniwersytet Łódzki

Maciej Pracucik

Kierunek: informatyka

Specjalność: informatyka stosowana

Ścieżka dydaktyczna: systemy mobilne

Numer albumu: 410731

Wykorzystanie sieci typu GAN na potrzeby generowania gestów rąk.

Praca magisterska

wykonana pod kierunkiem
dr Krzysztof Podlaski
w Katedrze Informatyki WFiIS UŁ

Łódź 2025

Spis treści

1 Wprowadzenie	4
1.1 Problematyka	4
1.2 Cel i zakres pracy	4
1.3 Struktura pracy	4
2 Sieci GAN, analiza konkurencji i techniczne aspekty realizacji	5
2.1 Sieci neuronowe i AI	5
2.2 Budowa sieci neuronowej	6
2.3 Sieci typu GAN	7
2.4 GestureGAN	8
2.5 PoseGAN	10
3 Narzędzia i technologie wybrane do realizacji projektu	12
3.1 Python	12
3.2 PyTorch	13
3.3 MediaPipe	13
3.4 OpenCV	13
4 Proces tworzenia projektu	14
4.1 Dobór zbioru danych	14
4.2 Wybór architektury sieci	16
4.3 Preprocesowanie danych	18
4.4 Model sieci	20
4.5 Flow i animacja	23
5 Wyniki i dyskusja	25
6 Podsumowanie	25
6.1 Zalety i wady przyjętych rozwiązań	25
6.2 Napotkane trudności	25
6.3 Możliwości rozwoju	25
6.4 Wnioski końcowe	25
Bibliografia	26

1 Wprowadzenie

1.1 Problematyka

W dzisiejszych czasach bardzo modnym tematem jest sztuczna inteligencja, która zaczyna się wkradać w każdy aspekt naszego życia. Możemy ją spotkać w formie chatów, podpowiedzi do pisanego kodu, asystentów internetowych, systemów rozpoznawania głosów, czy nawet we własnej lodówce! Każda firma żeby zaistnieć i pozostać istotną inwestuje w tę część technologii. Jedną z gałęzi rozwoju jest generowanie zdjęć, czy filmów. AI już jest w stanie wygenerować przeróżne obrazy, jednakże to z czym AI radzi sobie najgorzej są ręce.

W tym projekcie chodzi o stworzenie sieci typu GAN, która pozwoli na generowanie obrazów rąk, w jak najlepszej jakości. Dodatkowo sieć ma za zadanie nauczyć się, żeby modyfikować istniejące zdjęcia i nadawać im zupełnie inny gest, przy zachowaniu jak najlepszej jakości i realizmu. Szczególnie ten drugi aspekt pozostaje dla sztucznej inteligencji problematyczny. Myślę, że każdy z nas spotkał się ze zdjęciami, które dosłownie wyglądają, jak żywe, ale to co najczęściej zdradza, że jednak to AI maczało w nim palce są ręce. Za długie palce, dziwne ich ułożenie, ilość, czy nawet całkowicie odrealniony wygląd. Przeróżne firmy, jak i naukowcy stale ulepszają sieci, i rozwiązania, żeby i to przestało być problemem. Niniejsza praca również podejmuje się tego niełatwego zadania.

1.2 Cel i zakres pracy

Celem niniejszej pracy jest stworzenie sieci neuronowej typu GAN, która pozwoli na generowanie obrazów gestów rąk, w jak najlepszej jakości.

Docelowo również, wygenerowane zdjęcia będą wykorzystywane do stworzenia animacji przechodzenia z jednego gestu w inny.

1.3 Struktura pracy

Pierwszy rozdział przybliży to czym są sieci neuronowe, a dokładniej typu GAN, jakie są analogiczne rozwiązania, oraz o samym generowaniu zdjęć. Następny opowie jakie narzędzia, biblioteki i technologie zostały wykorzystane do realizacji projektu. Trzeci zaś mówi o tym jak wyglądał proces tworzenia projektu. Co po kolejci zostało zrobione, jakie po drodze wystąpiły komplikacje, oraz jak zostały rozwiązane i finalnie jak wygląda pro-

jekt. Przedostatni rozdział to przedstawienie wyników, rezultatów realizowanego projektu, analiza i omówienie ich. Ostatni rozdział zawiera wnioski końcowe i podsumowanie.

2 Sieci GAN, analiza konkurencji i techniczne aspekty realizacji

2.1 Sieci neuronowe i AI

Człowiek od samego początku swojego istnienia jest istotą niesamowicie ciekawą. To ona sprawia, że w głowie ludzi pojawiają się pytania, a co jeśli? A co jeśli to co wiemy to jest tylko część prawdy? Co jeśli jest coś więcej? To dzięki zadawaniu sobie przeróżnych pytań przez różne osoby, najczęściej przez największe umysły jakie chodziły po tej ziemi tak dużo udało nam się osiągnąć. Poczynając od wynalezienia koła, silniki parowe, elektryczność, internet aż po loty w kosmos. Niektóre z tych wynalazków łączy kolejny aspekt, to że człowiek chce sobie ułatwiać codzienne zadania. Żeby jak najbardziej zwiększyć swoją produktywność, żeby codziennie czynności nie wchodziły w drogę, albo żeby je po prostu ułatwić czy przyspieszyć.

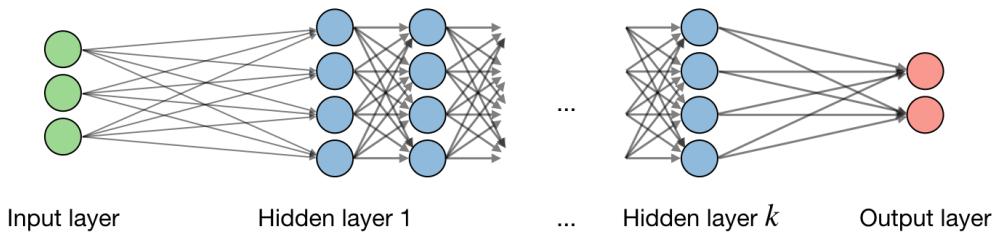
Kolejnym taki wynalazkiem, który łączy te dwa aspekty jest sztuczna inteligencja. Już od dawien dawna, przeróżne filmy czy książki science-fiction rozbudzały naszą wyobraźnię, o istnieniu robotów, sztucznej inteligencji równej ludzkiej, czy nawet przewyższającej ją. Do stworzenia pierwszych wzorów, inspiracją była ludzka sieć neuronowa, to jak działa nasz mózg. Takie coś starano się przekuć w pierwszej wzory, pierwsze sieci. Początkowo był to prosty matematyczny opis komórki neuronowej przez McCullocha i Pittsa w 1943 [8]. W połączeniu z zagadnieniem przetwarzania danych mógł modelować proste funkcje logiczne. Dopiero w 1949 roku Hebb sformułował regułę, którą uznaje się za pierwszą regułę uczenia sztucznych sieci neuronowych[8].

Obecnie sieci i modeli sieci jest tysiące. Do najpopularniejszych należą np CNN - Convolutional Neural Network, czy RNN - Recurrent Neural Network. W tym projekcie wykorzystujemy sieci typu GAN, czyli Generative Adversarial Network, o której więcej w kolejnym podrozdziale.

2.2 Budowa sieci neuronowej

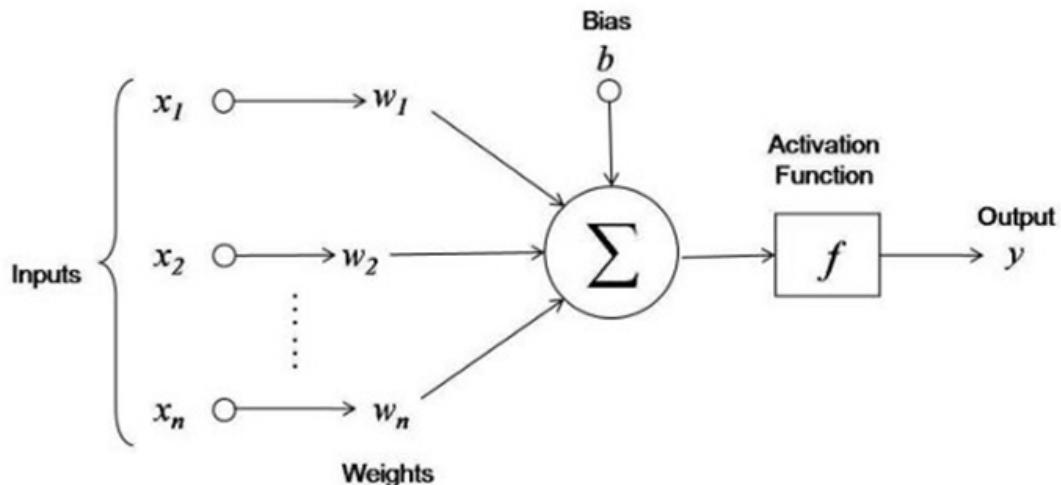
Najmniejszą składową sieci neuronowej są oczywiście neurony, te neurony są poukładowane po kilka w tak zwane warstwy. Najmniejsza ilość warstw to 3. Składają się z warstwy wejściowej, reprezentuje dane wejściowe w postaci numerycznej, warstwy ukryte, liczba mnoga tu jest celowa, ponieważ ich może wystąpić nieskończonie wiele, to one wykonują obliczenia. Ostatnim typem warstwy jest warstwa wyjściowa, ona generuje dane wyjściowe.

Tylko no właśnie dane. Jako, że sieci neuronowe to matematyka, to muszą działać na liczbach także nasze dane muszą być zmienione na liczby, a następnie znormalizowane do zakresu między 0 a 1.



Rysunek 1: Budowa sieci neuronowej [12]

Zajmijmy się teraz najmniejszą częścią sieci, czyli neuronem.



Rysunek 2: Budowa neuronu [12]

”W neuronie wartości sygnałów wejściowych (oznaczone na rysunku jako x_1, x_2, \dots, x_n) mnożone są przez wagi (oznaczone jako w_1, w_2, \dots, w_n), a iloczyny są ze sobą

dodawane (do wyniku dodawana jest wartość b, która nie jest mnożona przez wartość sygnału wejściowego). Wynik rachunków poddawany jest funkcji aktywacji, która decyduje o ostatecznej wartości wysyłanego sygnału. W najprostszym przypadku może to być funkcja progowa (1 dla wartości dodatnich, 0 dla wartości niedodatnich), która przypomina działanie ludzkiego neuronu: „wysyłam sygnał lub nie”.[12]

2.3 Sieci typu GAN

Sieci GAN czyli Generative Adversarial Network, a tłumacząc na polski, generatywne sieci współzawodnicze. Jak sama nazwa wskazuje są to sieci generatywne, czyli generują dane, najczęściej są to obrazy, ale nie ma ograniczeń, mogą być to również np filmiki. Sieci GAN składają się tak naprawdę z dwóch sieci, które ze sobą równocześnie współzawodniczą. Bardzo często są porównywane do fałszowania pieniędzy, mamy jedną sieć, która uczy generować jak najbardziej fałszywki, a drugą, która ma za zadanie odrzucać fałszywki. Podzielone są na:

- Generator uczy się generować wiarygodne dane i docelowo oszukać dyskryminator, że to co generuje jest prawdziwe
- Dyskryminator, uczy się odróżniać prawdę od fikcji



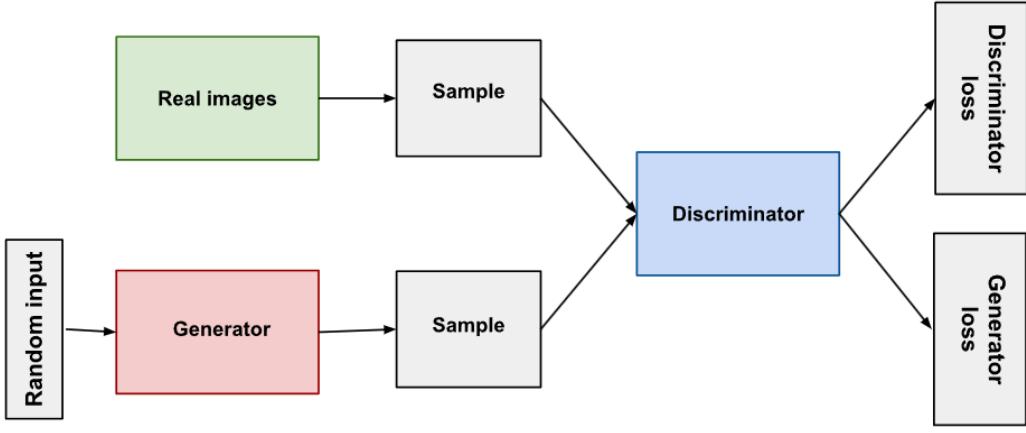
W miarę postępów w treningu generator coraz lepiej tworzy dane wyjściowe, które mogą oszukać dyskryminator:



Na koniec, jeśli szkolenie generatora przebiega dobrze, funkcja dyskryminacyjna coraz gorzej odróżnia prawdziwe obrazy od fałszywych. Zaczyna klasyfikować fałszywe dane jako prawdziwe, przez co jego dokładność spada.



Rysunek 3: Generative Adversarial Network [4]



Rysunek 4: Ogólny model sieci GAN [4]

Sieci GAN mają oczywiście swoje pod typy, czy też może framework, czy jeszcze inaczej już wcześniej stworzone typy tej sieci, które są powszechnie znane i używane. Wyróżnić można:

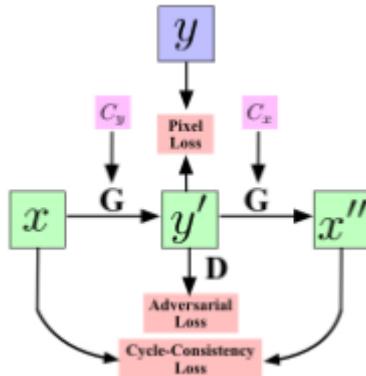
- Pix2Pix jest to sieć, która przetwarza jedno zdjęcie w drugie, co jest istotne w tej sieci to to, że dane muszą być sparowane, czyli jeden z obrazów jest obrazem wejściowym, a drugim obrazem wyjściowym
- CycleGAN jest bardzo podobny do Pix2Pix tylko zbiór danych nie jest sparowany. Można za jego pomocą robić takie translacje jak np. z koni zebry

Ten pierwszy został wykorzystany do stworzenia tego projektu, choć początkowo to ten drugi był używany, ale o tym w oddzielnym rozdziale.

2.4 GestureGAN

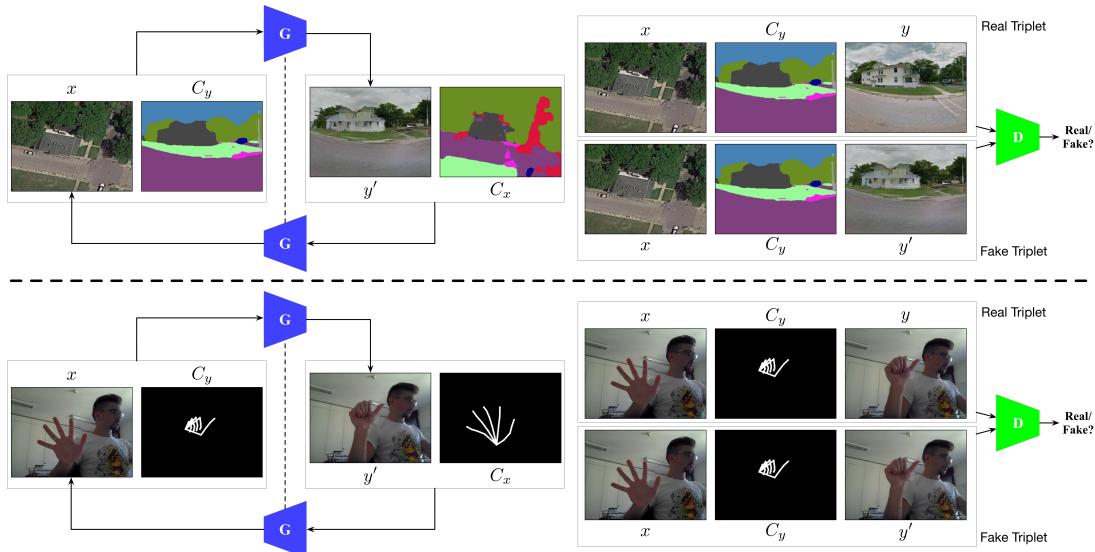
GestureGAN[6] jest to propozycja stworzona przez specjalistów z różnych uczelni, takich jak Oxford czy Texas State University. W swoim założeniu ma robić to samo co sieć stworzona na potrzeby tego projektu, jednakże to jej budowa jest tym co je rozróżnia. Była to bardzo duża inspiracja przy tworzeniu projektu, podobnie jak niniejsza sieć wykorzystywany jest mediapipe do ekstrakcji szkieletu. Co je np. rozróżnia to, że GestureGAN otrzymuje dwa zdjęcia na wejściu, jedno prawdziwe, drugie samego szkieletu docelowego i ma wyjściu otrzymujemy wygenerowane zdjęcie i pierwotny szkielet. Takie rozwiązanie było ok w przypadku ichniego zbioru danych, ponieważ był sparowany, to znaczy ta sama osoba wykonywała wszystkie gesty. Niestety w wykorzystanym przeze mnie zbiorze

danych nie było takiej możliwości, dane stały się sparowane nieco sztucznie, ale o tym później. Bardzo dużą zaletą tego projektu jest to, że jest bardzo uniwersalny, z każdego gestu jesteśmy w stanie uzyskać każdy gest.



Rysunek 5: Model GestureGAN [6]

i tu jeszcze przykładowe efekty i możliwości GestureGAN:

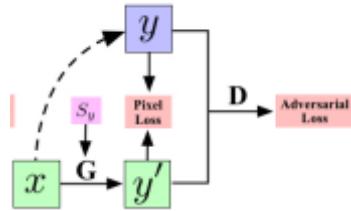


Rysunek 6: Zdjęcie wygenerowane przez GestureGAN [6]

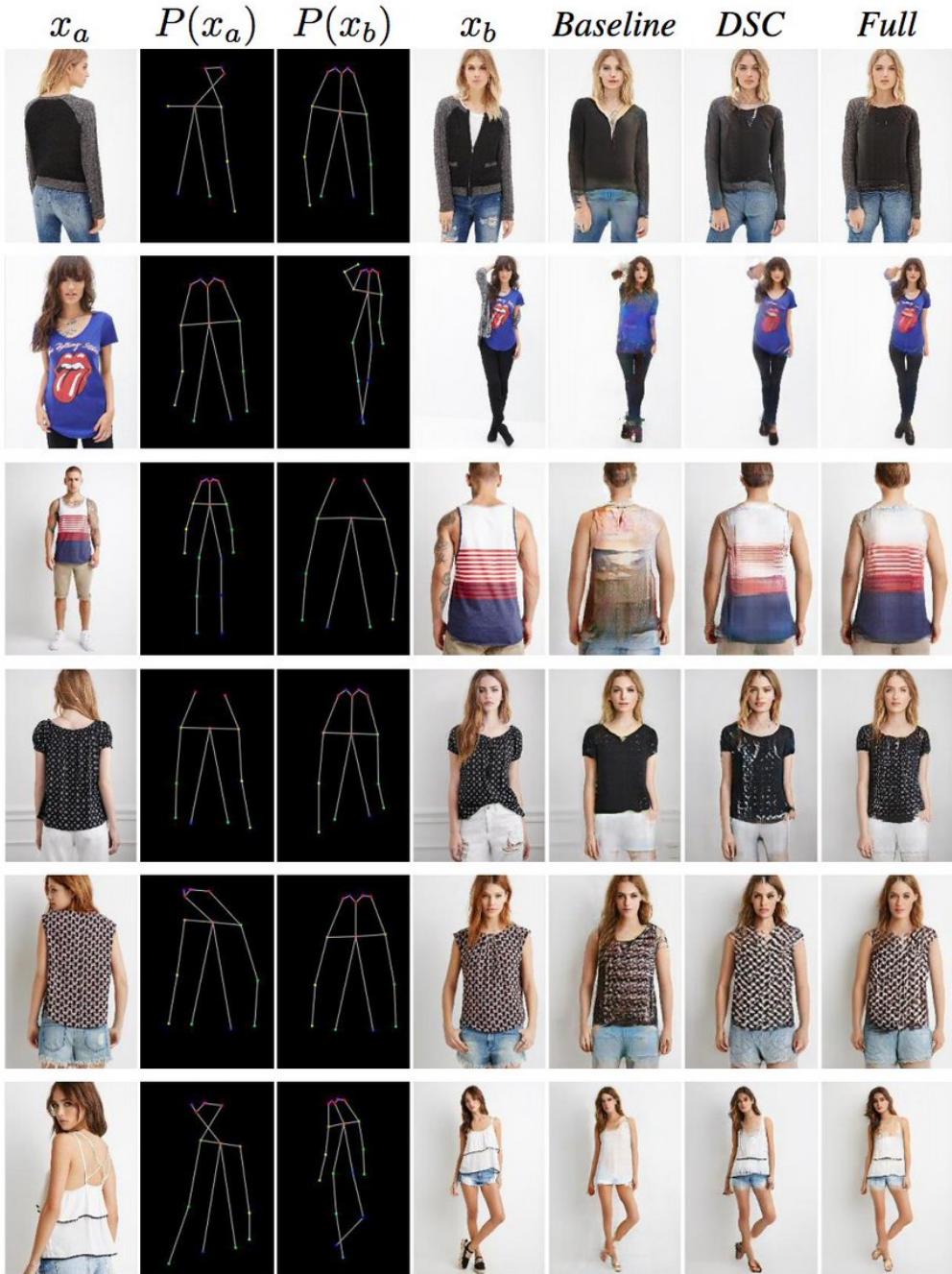
Ten projekt stanowiłby bardzo dobrą bazę, generuje wyraźnie i zróżnicowane gesty rąk, jest bardzo uniwersalny, ale problem zaczyna się już na początku. Zbiór danych, który tutaj jest wykorzystywany jest niedostępny, albo wymaga dodatkowych dostępów. Nie wszystkie dane też są wygenerowane, choćby zdjęcia szkieletów. Te dane, które były dostępne, było ich po prostu za mało. Samego projektu też ciężko zrozumieć i nie wiadomo jak z rozszerzalnością kodu. Dlatego najlepszym rozwiązaniem było przygotowanie własnego zbioru danych, własnego modelu i metodą prób i błędów, dochodzenie do rozwiązania.

2.5 PoseGAN

PoseGAN[1] jest dość zbliżony do GestureGAN, jednakże ma zasadniczą różnicę w swoim zastosowaniu. Tak jak GestureGAN, jak sama nazwa wskazuje służy do generowania gestów rąk, tak PoseGAN służy do generowania całych poz ludzkich. Także zakres jest większy jednakże przez fakt, że nie ma skupienia na dłoniach dokładność wykonywanego gestu nie ma tu znaczenia. Jednakże jest to zdecydowanie najbliższe wykonywanemu projektowi.



Rysunek 7: Model PoseGAN [1]



Rysunek 8: Zdjęcie wygenerowane przez PoseGAN [1]

Projekt jest niewątpliwie bardzo imponujący jednakże no właśnie skupia się na czymś kompletnie innym. Tak jak widać na rysunku 8 dlonie nawet nie są wyszczególnione na szkielecie, a przerabianie, dorabianie takiego elementu byłoby zbyt czasochłonne. Ciężko też ocenić czy przyniosłoby zadowalające rezultaty, przy innym zbiorze danych skupionych czysto na dloniach. Sieć i tak by musiała się uczyć wszystkiego od nowa, plus no właśnie nie do końca interesuje nas cały ludzki szkielet. Dlatego niezbędne było stworzenie własnego

projektu.

3 Narzędzia i technologie wybrane do realizacji projektu

3.1 Python

Językiem programowania wykorzystanym do realizacji projektu jest Python. Jest to bardzo oczywisty wybór, ponieważ jest to najpopularniejszy język programowania do tworzenia sieci neuronowych, w dzisiejszych czasach. Przybliźmy go jednak. Według oficjalnej dokumentacji, jest ”łatwy do nauczenia się i potężnym językiem programowania. Posiada wydajne struktury danych wysokiego poziomu oraz proste, ale skuteczne podejście do programowania obiektowego” [2]. Co niejako wyróżnia go na tle innych języków, to fakt, że jest dynamicznie typowany oraz jest językiem interpretowanym. To oznacza, że nie posiada kompilatora a interpreter, który nie kompiluje programu do pliku wykonywalnego, a kod jest wykonywany w czasie rzeczywistym. Czyni to Python językiem łatwym w testowaniu, kompilowaniu czy wykonywaniu, przenośny, co oznacza, że ten sam kod uruchomi się niezależnie od systemu operacyjnego czy urządzenia. Sam język Python jest napisany w języku C, co oznacza, że jest bardzo szybki i wydajny. Został stworzony przez Guido van Rossum w roku 1991. Co ciekawe jego nazwa wywodzi się z starych serii skeczów grupy Monty Python’s Flying Circus, a zyskał na popularności w momencie gdy firma Google, powiedziała, że wykorzystuje go do własnych, wewnętrznych celów.

Jednakże czemu akurat to on jest najczęściej wykorzystany do AI? Odpowiedź jest tak prosta jak sam Python jest prosty. Wynika to z tego, że Python ma bardzo prostą i czytelną składnię, co pozwala developerom, czy naukowcom skupianie się na logice i samym problemie, a nie na składni[13]. Dodatkowo Python sam w sobie nie wymaga dużej ilości kodu. Najprostsza sieć neuronowa może zostać stworzona i uruchomiona w zaledwie 4 linijki! Kolejnym powodem przemawiającym dlaczego to właśnie Python jest najczęściej używany, jest bardzo duża ilość bibliotek, czy to wbudowanych, czy stworzonych przez społeczność. Biblioteki takie jak NumPy, SciPy, Matplotlib, czy wykorzystane w tym projekcie PyTorch czy MediaPipe, o których będzie w kolejnych podrozdziałach. Także kolejnym i ostatnim już aspektem, o którym warto wspomnieć, jest wyżej wymieniona społeczność. To dzięki dużej liczbie osób i ogromnym zebranym doświadczeniu, tworzenie

dowolnego projektu staje się znacznie prostsze. Praktycznie każdy projekt, aspekt projektu czy problem, ktoś już natrafił, dzięki czemu możemy szybko i sprawnie rozwiązywać problem.

3.2 PyTorch

”PyTorch to w pełni funkcjonalny framework do tworzenia modeli do deep learningu, który jest typem machine learningu, najczęściej wykorzystywanym w aplikacjach takich jak rozpoznawanie obrazów czy procesowanie języka.” [10] Biblioteka ta została napisana w Pythonie, przez developerów z Facebook AI Research, oraz ma doskonałe wsparcie do wykorzystywania GPU, szczególnie dla GPU od firmy Nvidia. Co czyni ją idealną biblioteką dla tego projektu. Jednak projekt dotyczy generowania obrazów rąk, czyli dobre wykorzystanie GPU jest bardzo wskazane. Początkowo jednak używana była konkurencyjna biblioteka, a dokładniej Tensorflow, jednakże była wolniejsza, miała gorsze wykorzystanie GPU i była mniej klarowna od PyTorch, co zaważyło na finalnym wyborze.

3.3 MediaPipe

”MediaPipe Solutions to zestaw bibliotek i narzędzi, które umożliwiają szybkie stosowanie w aplikacjach technik sztucznej inteligencji (AI) i uczenia maszynowego (ML).” [5] Opis ten pochodzi z oficjalnej dokumentacji Mediapipe, jednakże sam opis jest zbyt ogólny. Ta biblioteka ma masę możliwości i zastosowań. Do nich można zaliczyć np. rozpoznawanie obrazu. Nie chcemy pisać sieci do rozpoznawania, czy na danym obrazku jest pies czy kot? Mediapipe daje gotowe rozwiązanie! Poza tym do wyboru jest też klasyfikacja obrazu, segmentacja obrazu, wykrywanie twarzy, rozpoznawanie gestu, oraz to co było niezbędne w tym projekcie to wykrywanie rąk i tworzenie szkieletu. To dzięki tej bibliotece udało się wyłuskać szkielet ręki na każdym z obrazów, ze zbioru danych, a następnie nałożyć go na zdjęcie wraz z maską i przekazane do modelu. Później była przydatna do tworzenia kolejnych klatek animacji, przechodzenia z jednego gestu, w drugi.

3.4 OpenCV

OpenCV jest to biblioteka do machine learningu oraz computer vision. Posiada w swoim arsenale dostęp do takich narzędzi jak wykrywanie i rozpoznawanie twarzy, identyfikowanie obiektów, śledzenie ruchów kamery, śledzenie obiektów 3D, usuwanie czerwonych

oczu ze zdjęć, czy nawet łączenie zdjęć razem, by uzyskać obraz całej sceny o wysokiej rozdzielczości.[11] Jednak to żadna z tych funkcji nie została użyta w projekcie. Wykrywanie i tworzenie szkieletu to zadanie Mediapipe, tworzenie modelu to działka PyTorch. OpenCV miał znacznie prostsze zadanie, został wykorzystany do ładowania zdjęć, czy to dla preprocessingu, czy już bezpośrednio do modelu. Dalej zapisywał każdy kolejny obraz w zależności od epoki, dzięki czemu można było śledzić poczynania i na koniec sklejał wszystkie klatki i tworzył z nich animację.

4 Proces tworzenia projektu

4.1 Dobór zbioru danych

Pierwszym i bardzo ważnym krokiem jest wybór zbioru danych. Zbiór danych to podstawa projektu, jakość modelu ma bezpośrednie przełożenie, w zależności od danych, jak również ich przygotowania. To od niego będzie zależeć zarówno to jak długo sieć będzie się uczyć, ale przede wszystkim jakie osiągnie rezultaty. Gdy zbiór zdjęć jest wyraźny, najlepiej dane są sparowane, to sieć bardzo szybko zrozumie na czym polega różnica i na czym powinna się skupiać.

Poszukiwania były zaskakująco problematyczne, głównie ze względu na słabą współpracę z biblioteką Mediapipe. Często zdjęcia z innych zbiorów, to były białe plamy na czarnym tle, przez co ciężko było nawet rozróżnić ustawienie dloni czy choćby palce. Inny dataset prezentował zdjęcie robione kamerą nocną, co w zasadzie prowadziło do takich samych rezultatów, jak w poprzednio wspomnianym zbiorze. Mediapipe nie umiał wychwycić gdzie znajduje się dłoń, palce czy stawy.

Kolejne próby były już na zbiorze przedstawiającym zdjęcia liter przedstawianych w języku migowym. Ten zbiór był dobry, ze względu na to, że przedstawiał same dłonie, na jednolitym tle, był sparowany. Jednakże co skreśliło go, to fakt, że tych zdjęć było po prostu za mało. Dla problemu takiego jak generowanie gestów, tych zdjęć musi być dużo. Tam było ich mniej więcej 30 na gest. Jest to dużo za mało

Finalnie wybór padł na zbiór HaGrid [9].

Ten zbiór nie jest niestety sparowany, jednakże posiada na tyle dużo zalet, że warto było go wybrać.

Po pierwsze jest ogromny. Cały posiada aż 1.5TB, 1 086 158 zdjęć w FullHD, podzielono

nych na 33 klasy gestów. Co więcej, do zbioru jest dołączony plik json dla każdego gestu, poniżej przykładowy fragment dla jednego zdjęcia:

```
"04c49801-1101-4b4e-82d0-d4607cd01df0": {
    "bboxes": [
        [0.0694444444, 0.3104166667, 0.2666666667, 0.2640625],
        [0.5993055556, 0.2875, 0.2569444444, 0.2760416667]
    ],
    "labels": [
        "thumb_index2",
        "thumb_index2"
    ],
    "united_bbox": [
        [0.0694444444, 0.2875, 0.7868055556, 0.2869791667]
    ],
    "united_label": [
        "thumb_index2"
    ],
    "user_id": "2fe6a9156ff8ca27fbce8ada318c592b",
    "hand_landmarks": [
        [
            [0.37233507701702123, 0.5935673528948108],
            [0.3997604810145188, 0.5925499847441514],
            ...
        ],
        [
            [0.37388438145820907, 0.47547576284667353],
            [0.39460467775730607, 0.4698847093520443],
            ...
        ]
    ],
    "meta": {
        "age": [24.41],
        "gender": ["female"],
        "race": ["White"]
    }
}
```

Jak widzimy znajduje się tu bardzo dużo przydatnych informacji. Do najważniejszych należą:

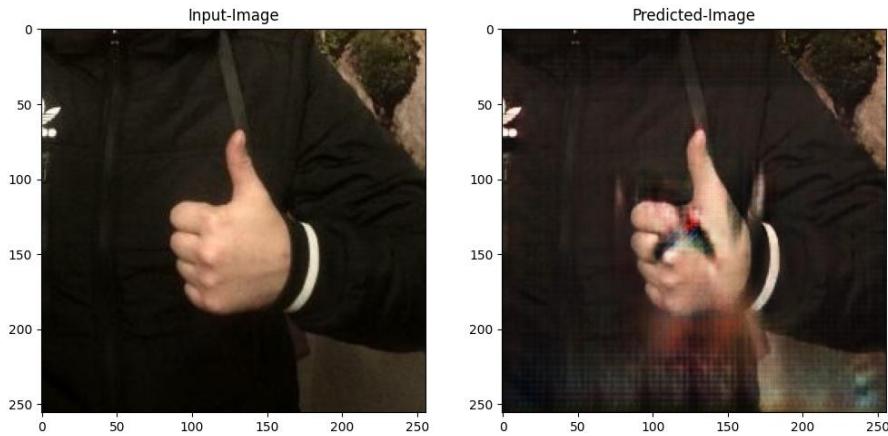
- bboxes - są to koordynaty na jakim obszarze znajdują się dlonie, przydało się to do przycinania zdjęć do wielkości 256x256, która jest standardem dla sieci GAN
- labels - dzięki temu wiadomo było którą dłoń wykorzystać do analizy czy też przetwarzania obrazu
- hand landmarks - to koordynaty położenia palca, stawów, oraz dloni

Tak jak można zauważyć jest to doskonały zbiór danych, do ideału brakuje jednego, żeby był sparowany, ale i to nie jest problem, o czym więcej w rozdziale o preprocessingu.

4.2 Wybór architektury sieci

Wybór architektury był nieco uproszczony, bo z założenia projekt wymaga użycia sieci GAN, jednakże podtypów sieci jest dużo. Początkowo wybrany został CycleGAN, w związku z tym, że wybrany zbiór nie był sparowany, a CycleGAN dzięki dodatkowej funkcji straty, wykorzystuje utratę spójności cyklu żeby trening nie musiał się odbywać na sparowanych danych. Jako bazę posłużył tutorial z oficjalnej dokumentacji Tensorflow [14] (wtedy jeszcze ta właśnie ta biblioteka była wybrana). Z założenia wszystko wydawało się że powinno zadziałać. Na tutorialu robione są z koni zebry, czy z jabłek pomarańcze. To w zasadzie czemu miałoby sobie nie poradzić z gestami?

Rzeczywistość okazała się być bardzo brutalna.



Rysunek 9: Zdjęcie wygenerowane przez CycleGAN, po 45 epokach [14]

Na pozór wydawać by się mogło, że to zdjęcie jest ok, sieć kombinuje i dosyć wiernie odtworzyła ręce, jedynie pojawiają się artefakty. Zgadza się, ale nie takie było jej zadanie. Powinna nałożyć nowy gest. Ponownie wydawać by się mogło, że zaczyna dobrze kombinować, bo artefakty pojawiają się na dłoni. Tylko niestety przy przeróżnych konfiguracjach parametrów, ilości puszczaanych epok, czy zmian, nie udało się zmusić sieci do manipulacji gestami. Z tego powodu trzeba było coś zmienić, najwyraźniej zbiór jest zbyt zróżnicowany, tła są inne, przedstawiają innych ludzi itd, żeby CycleGAN sobie poradził.

Więc wybrane zostało inne podejście. Nastąpiła zmiana architektury na Pix2Pix i zmieniony pulowany został zbiór żeby uczynić go sparowanym. Dokładnie co zostało zrobione jest w kolejnym rozdziale. W dużym skrócie, na ręce z danym gestem, została, na dłoń, nakładana maska i na nią szkielet gestu. Co czyni nasz zbiór sparowanym i idealnym kandydatem dla Pix2Pix.

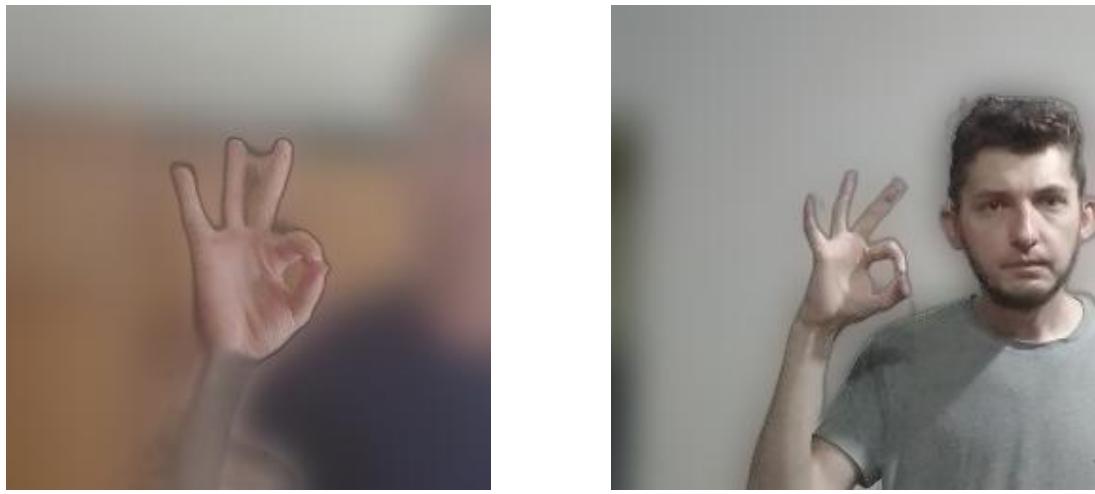


Rysunek 10: Przykłady przed i po przycięciu

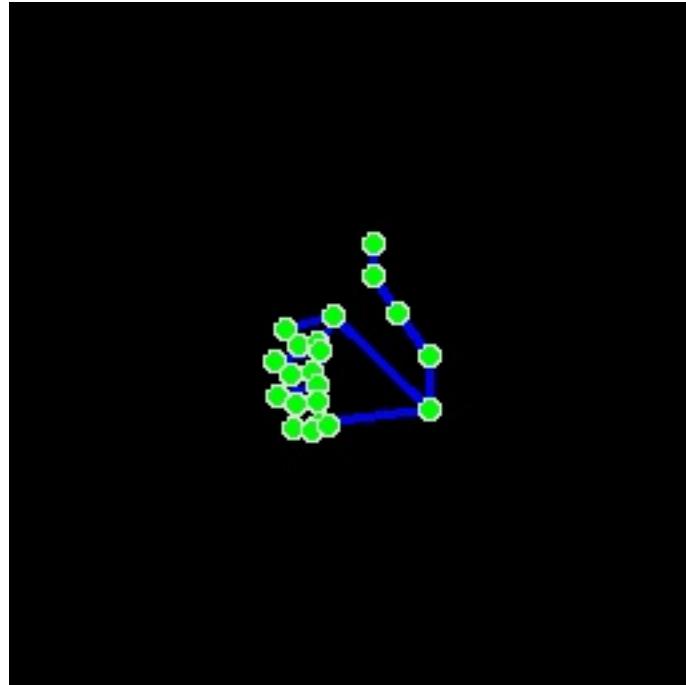
4.3 Preprocesowanie danych

Preprocesowanie zaczęło się od najprostszej, a zarazem bardzo ważnej części. Przyjęcia zdjęć do wielkości 256x256, czyli standardu dla sieci GAN. Takie zdjęcia stanowiły początkowy input i zdjęcie docelowe dla sieci. Jednakże jak zostało opisane w podrozdziale o wyborze architektury, to było za mało.

Następnie idąc za przykładem z GestureGAN [6], kolejną próbą było przekazywanie sieci zarówno zdjęcia oryginalnego, jak i oddziennie zdjęcia docelowego gestu, ale w formie szkieletu na czarnym tle.



Rysunek 12: Przykłady po obróbce tła



Rysunek 11: Wygenerowane zdjęcie szkieletu dłoni

Kolejnymi próbami obróbki zdjęć było usuwanie tła, a w zasadzie rozmazywanie go, przez wycięcie za pomocą biblioteki rembg [3]. Która to pozwala na wykrycie i usunięcie tła z obrazu. Dłoń i część, która powinna zostać jest wykrywana przy wykorzystaniu MediaPipe.

Tu przykład po takiej obróbce: Jak widzimy na obrazkach powyżej, efekt jest bardzo różny. Ciężko było o jednoznaczne wykrywanie ręki. Raz zostawała sam ręka, raz wraz z człowiekiem, jeszcze innym razem potrafił zostać sam człowiek. Efekty były mało zadowalające, jednakże, mimo wszystko takie dane również zostały wypróbowane, ponownie

nie przynosząc żadnych rezultatów.

Pomimo licznych niepowodzeń i ślepych zaułków, narodził się jeszcze jeden, finalny pomysł. Stworzenie zbioru danych sparowanego, w nieco sztuczny sposób. Każde zdjęcie dłoni zostało przepuszczane przez OpenCV i MediaPipe, w celu wykrycia, nałożenia maski a następnie szkieletu gestu. Dając następujący przykład:



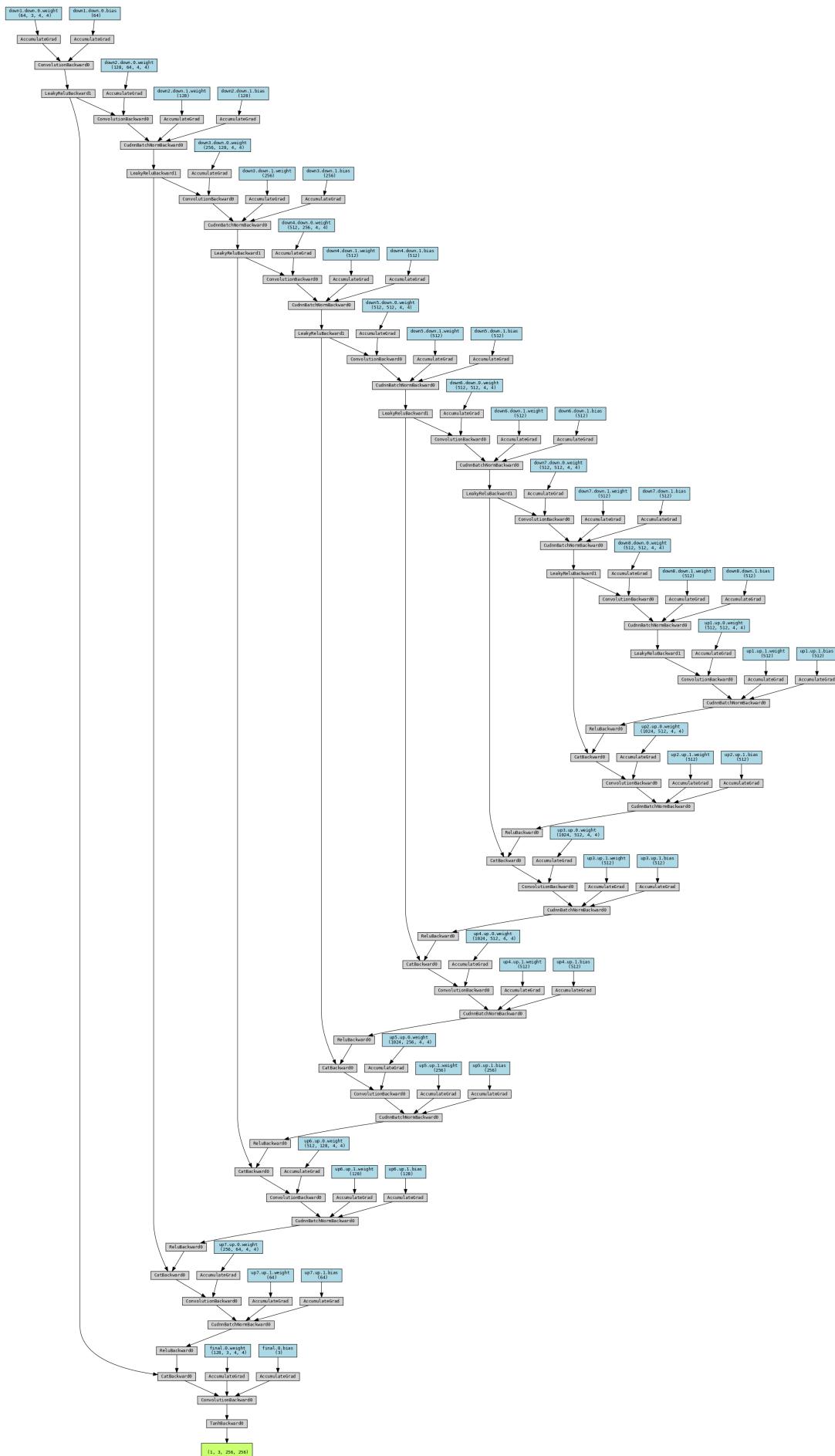
Rysunek 13: Przykład zdjęcia z nałożoną maską i szkieletem gestu

Funkcja nakładająca maski jest na tyle uniwersalna, że może przyjąć dowolny kolor i możemy przekazać jej jako parametr, jak duży obszar ma zająć. Jednakże każde zdjęcie jest różne dlatego niezbędne było dobranie jednego rozmiaru maski, która będzie pasować dla znacznej większości obrazów. Tak przygotowane zdjęcia, a w zasadzie zbiór danych posłużył do treningu i testowania sieci.

4.4 Model sieci

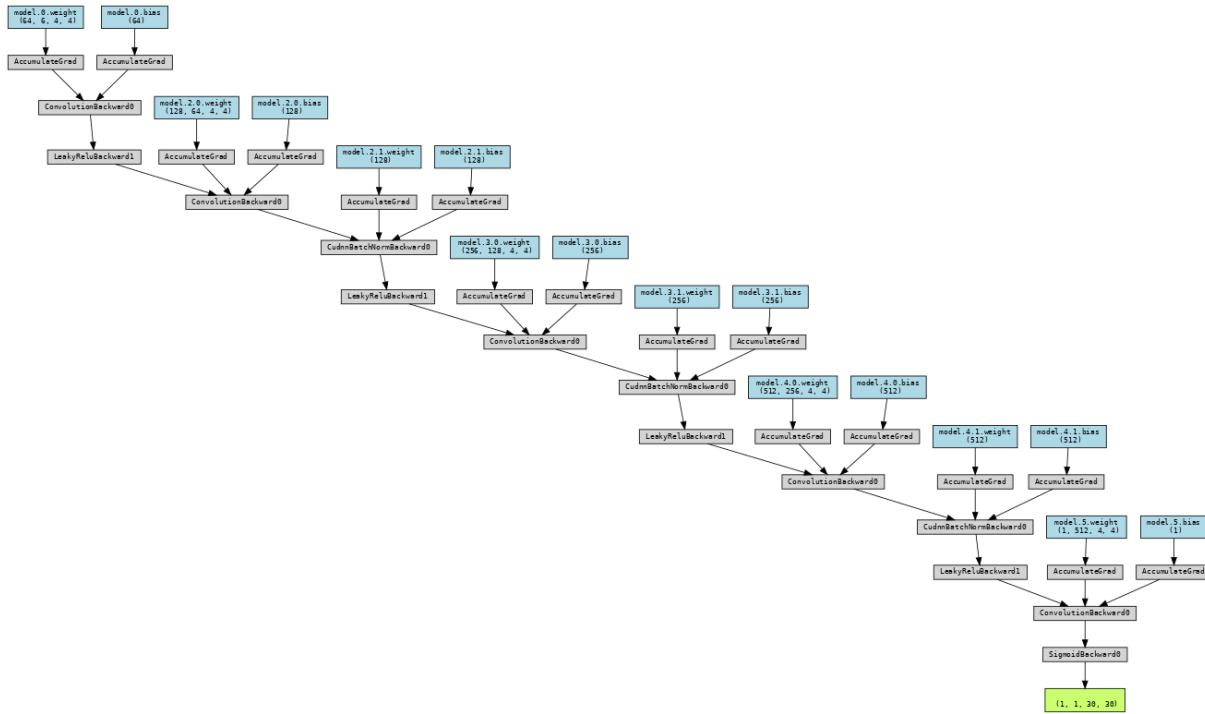
Architektura Pix2Pix jest typem warunkowej, generatywnej sieci adwersarza (cGAN), ma za zadanie nauczyć się mapowania pomiędzy obrazami, przy sparowanych danych. Pix2Pix jest bardzo uniwersalnym narzędziem, można go zastosować przykładowo do generowania kolorowych zdjęć z czarno białych, przekształcania szkiców w zdjęcia [15], czy robienia z dnia nocy. Architektura, w swojej bazowej formie składa się z generatora, opartego na U-Net, oraz discriminatora, opartego na PatchGAN.

U-Net jest siecią, w głównej mierze stworzoną do segmentacji obrazów, czyli przypisywania klasy do pikseli na obrazku, w celu np poznania kształtu danego obiektu [16]. Przed jej powstaniem klasyfikacja dotyczyła jedynie całego obrazka, ale dopiero dopiero Ronneberger, w 2015, zaproponował U-Net i to popchnęło dalej cały rozwój translacji image to image. Składa się z dwóch ścieżek, które razem składają się na kształt litery "U", stąd nazwa. Pierwsza z nich, to enkoder, jest zbliżona swoją budową do klasycznych sieci konwolucyjnych i odpowiada za klasyfikację. Zbudowany jest z bloków konwolucji, normalizacji wsadowej i LeakyRelu. Druga z kolei to dekoder[17]. Zbudowany jest z konwolucji transponowanej, normalizacji wsadowej, odrzucenia i LeakyRelu.



Rysunek 14: Generator U-Net
22

Dyskryminator ma bardzo proste zadanie i założenie, ma po prostu sklasyfikować czy każda łatka(patch) jest prawdziwa czy nie.[15]



Rysunek 15: Dyskryminator PatchGAN

W zasadzie zaproponowane rozwiązanie nie różni się znacznie od klasycznego Pix2Pix. Jednakże co go wyróżnia i dało kolosalną różnicę w wyglądzie i jakości to zastosowanie straty percepcyjnej VGG [7].

Jest to stosunkowo nowo zaproponowane podejście. Klasycznym podejściem było sprawdzanie straty pomiędzy wygenerowanym pixelem a prawdziwym obrazem. W tym, porównuje się ich reprezentacje cechowe uzyskane z warstw konwolucyjnych sieci VGG, wytrenowanej na ImageNet. Co daje znacznie bardziej realistyczne rezultaty.

$$\mathcal{L}_{\text{perc}} = \sum_l \lambda_l \|\phi_l(I_{\text{gen}}) - \phi_l(I_{\text{ref}})\|_2^2$$

4.5 Flow i animacja

Flow uczenia sieci jest następujący:

1. Przycinane są zdjęcia pobrane z datasetu [9]
2. Generowane są obrazy z nałożoną maską i szkieletem ręki

3. Zdjęcia są ładowane, normalizowane i przekazywane do sieci
 4. Sieć jest inicjalizowana, jeśli istnieją to ładowane są informacje z poprzednich uruchomień.
 - Generator
 - Dyskryminator
 - Optymalizatory generatora i dyskryminatora
 - Wartości straty dla generatora i dyskryminatora
 5. Generator generuje obraz
 6. Dyskryminator najpierw odbiera prawdziwy obraz i fałszywy
 7. Obliczana jest strata dyskryminatora
 8. Propagacja wsteczna straty dyskryminatora
 9. Obliczana jest strata generatora
 10. Obliczana jest strata VGG
 11. Wyliczana jest całościowa strata generatora
-

```
g_loss_total = g_loss + self.lambda_vgg * loss_vgg
```

12. Propagacja wsteczna straty generatora
13. Sumowana jest strata generatora i dyskryminatora dla każdej kolejnej pary zdjęć
14. Wyliczana jest średnia strata generatora i dyskryminatora, dla epoki
15. Generowane są przykładowe obrazki i robiony jest checkpoint

Flow animacji jest następujący:

1. Brany jest przykładowy, pasujący położeniem gest docelowy i wraz z maską nakładany na obrazek, który ma być przerobiony
2. Generowane są zdjęcia, z interpolacją szkieletu z gestu startowego do docelowego.
3. Zdjęcia są przekazywane do wyuczonej sieci i zapisywane
4. Zdjęcia są sklejane w animacje dzięki OpenCV

5 Wyniki i dyskusja

wyniki

6 Podsumowanie

6.1 Zalety i wady przyjętych rozwiązań

Zalety

6.2 Napotkane trudności

trudności

6.3 Możliwości rozwoju

rozwoj

6.4 Wnioski końcowe

wnioski

Bibliografia

- [1] Aliaksandr Siarohin i Enver Sangineto i Stephane Lathuiliere i Nicu Sebe. “Deformable GANs for Pose-based Human Image Generationn”. In: (2018).
- [2] Python Software Foundation. *The Python Tutorial*. 2025. URL: <https://docs.python.org/3/tutorial/index.html>.
- [3] Daniel Gatis. *Rembg*. URL: <https://github.com/danielgatis/rembg>.
- [4] Google. *Kurs zaawansowany Machine Learning GAN*. URL: <https://developers.google.com/machine-learning/gan?hl=pl>.
- [5] Google. *Przewodnik po rozwiązańach MediaPipe*. URL: <https://ai.google.dev/edge/mediapipe/solutions/guide?hl=pl>.
- [6] Hao Tang i Hong Liu i Nicu Sebe. “Unified Generative Adversarial Networks for Controllable Image-to-Image Translation”. In: (2020).
- [7] Li Fei-Fei Justin Johnson Alexandre Alahi. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: (2016).
- [8] Bohdan Macukow. “SIECI NEURONOWE, HISTORIA BADAŃ I PODSTAWOWE MODELE”. In: (). URL: <https://pages.mni.edu.pl/~macukowb/wspolne/PNEiT1.pdf>.
- [9] Anton Nuzhdin et al. *HaGRIDv2: 1M Images for Static and Dynamic Hand Gesture Recognition*. 2024. arXiv: 2412.01508 [cs.CV]. URL: <https://arxiv.org/abs/2412.01508>.
- [10] Nvidia. *PyTorch*. URL: <https://www.nvidia.com/en-us/glossary/pytorch/>.
- [11] OpenCV. *OpenCV is the world's biggest computer vision library*. URL: <https://opencv.org/about/>.
- [12] Blog CSI Uniwersytet Im. Adama Mickiewicza w Poznaniu. *Sieci neuronowe w NLP*. 2025. URL: <https://csi.amu.edu.pl/blog-csi/sieci-neuronowe-w-nlp>.
- [13] Dhruvitkumar Talati. “Python: The alchemist behind AI’s intelligent evolution”. In: (2021).
- [14] Tensorflow. *CycleGAN*. URL: <https://www.tensorflow.org/tutorials/generative/cyclegan?hl=pl>.

- [15] Tensorflow. *pix2pix: Tłumaczenie obrazu na obraz z warunkowym GAN*. URL: https://www.tensorflow.org/tutorials/generative/pix2pix?hl=pl#import_tensorflow_and_other_libraries.
- [16] Tensorflow. *Segmentacja obrazu*. URL: https://www.tensorflow.org/tutorials/images/segmentation?hl=pl#what_is_image_segmentation.
- [17] Jocelyn Chanussot Xin Wu Danfeng Hong. “UIU-Net: U-Net in U-Net for Infrared Small Object Detection”. In: (2022).

Spis rysunków

1	Budowa sieci neuronowej [12]	6
2	Budowa neuronu [12]	6
3	Generative Adversarial Network [4]	7
4	Ogólny model sieci GAN [4]	8
5	Model GestureGAN [6]	9
6	Zdjęcie wygenerowane przez GestureGAN [6]	9
7	Model PoseGAN [1]	10
8	Zdjęcie wygenerowane przez PoseGAN [1]	11
9	Zdjęcie wygenerowane przez CycleGAN, po 45 epokach [14]	17
10	Przykłady przed i po przycięciu	18
12	Przykłady po obróbce tła	19
11	Wygenerowane zdjęcie szkieletu dloni	19
13	Przykład zdjęcia z nałożoną maską i szkieletem gestu	20
14	Generator U-Net	22
15	Dyskryminator PatchGAN	23

Spis tabel

Listingi