

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Informatyka w multimediacach

Poprawa jakości przekazu wideo po kompresji kodekiem H.266 przy użyciu
generatywnych sieci neuronowych GAN

Piotr Jan Domański

Numer albumu 293102

promotor
dr hab. inż. Grzegorz Pastuszak

WARSZAWA 2024

Poprawa jakości przekazu wideo po kompresji kodekiem H.266 przy użyciu generatywnych sieci neuronowych GAN

Streszczenie. W prezentowanej pracy zbadano efektywność sieci neuronowych, w szczególności generatywnych sieci przeciwnieckawnych GAN, w procesie poprawy jakości wideo po jego kompresji kodekiem H.266/VVC. Analizie poddano wydajność modeli bazujących na architekturach ResNet oraz DenseNet, oceniając jednocześnie wpływ zastosowania sieci GAN. Aby zwiększyć skuteczność modeli, wprowadzono dodatkowe dane wejściowe, takie jak profil i parametry kompresji, co umożliwiło znaczne polepszenie jakości odtwarzanej ramki. Wyniki te, porównywane z najlepszymi rozwiązaniami z literatury, dowodzą potencjału zaproponowanego podejścia w kontekście optymalizacji jakości transmisji wideo.

Słowa kluczowe: GAN, sieci neuronowe, kompresja danych, VVC, H.266, ResNet, DenseNet

Enhancement of video data compressed using H.266 codec with usage of generative GAN neural networks

Abstract. In the presented work, the efficiency of neural networks, particularly generative adversarial GAN ones, is investigated in the process of enhancing video quality post its compression with the H.266/VVC codec. The performance of models based on ResNet and DenseNet architectures is analyzed while simultaneously assessing the impact of employing generative adversarial networks. To increase the models' effectiveness, additional input data, such as the compression profile and parameters, is introduced, which enable a significant improvement in the quality of the reconstructed frame. These results, comparable with the best solutions from the literature, demonstrate the potential of the proposed approach in the context of optimizing video transmission quality.

Keywords: GAN, neural networks, compression, VVC, H.266, ResNet, DenseNet



Politechnika Warszawska

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Warszawa, 09.02.2024 r.

miejscowość i data

Piotr Domański
.....
imię i nazwisko studenta
293012.....
numer albumu
Informatyka.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanego z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

Piotr Domański
czytelny podpis studenta

Spis treści

1. Wstęp	11
2. Kompresja wideo	12
2.1. Podstawy kompresji wideo	12
2.1.1. Podstawy kompresji ramek	13
2.2. Standardy kompresji wideo	14
2.2.1. Wczesne standardy kompresji	15
2.2.2. Advanced Video Coding (H.264/AVC)	15
2.2.3. High Efficiency Video Coding (H.265/HEVC)	17
2.2.4. Versatile Video Coding (VVC) i Przyszłe Standardy	18
2.3. Wyzwania i artefakty kompresji wideo	19
2.3.1. Artefakty blokowe	19
2.3.2. Ringing Artifacts	20
2.3.3. Wyzwanie utraty jakości przy wysokim stopniu kompresji	20
2.4. Metody poprawy jakości wideo przy kompresji kodekiem VVC	21
2.4.1. Filtr deblokujący	21
2.4.2. Filtr SAO	21
2.4.3. Filtr ALF	21
2.4.4. Inne metody poprawy jakości wideo	22
3. Sieci neuronowe	23
3.1. Sieci konwolucyjne	23
3.2. ResNet	24
3.3. DenseNet	25
3.4. Generatywne sieci przeciwnostawne	25
3.5. Sieci neuronowe w poprawie jakości wideo po kompresji	26
3.5.1. Metody przetwarzające pojedyncze ramki	27
3.5.2. Metody przetwarzające grupy ramek	28
3.5.3. Metody wykorzystujące dodatkowe informacje o przekazie wideo	28
4. Opis zbioru danych	29
4.1. Charakterystyka wykorzystanego zbioru danych	29
4.2. Wybór danych i podział na podzbiory treningowe i walidacyjne	29
4.3. Format YUV420	30
4.4. Przygotowanie danych	31
4.4.1. Przygotowanie sekwencji wideo	31
4.4.2. Kodowanie i dekodowanie sekwencji wideo	31
4.4.3. Normalizacja danych	35
5. Metodologia rozwiązania	37
5.1. Opis wykorzystanych sieci neuronowych	37

5.1.1. Wspólne cechy badanych sieci neuronowych	37
5.1.2. Sieci ResNet(ang. Residual Netowrks)	38
5.1.3. Sieci DenseNet (ang. Densely Connected Convolutional Networks	39
5.1.4. Sieci konwolucyjne	40
5.1.5. GAN - Generatywne sieci przeciwnostawne	41
5.2. Parametryzacja i optymalizacja	42
5.2.1. Funkcja straty	42
5.2.2. Techniki optymalizacja uczenia	43
5.2.3. Parametry treningu	46
6. Przebieg treningu	48
6.1. Trening rozdzielny	48
6.1.1. Napotkane problemy	48
6.1.2. Przebieg funkcji straty	50
6.2. Trening sieci GAN	50
6.2.1. Problem stabilności treningu sieci GAN	51
6.2.2. Przebieg funkcji straty	53
7. Szczegóły implementacyjne	54
7.1. Implementacja modeli sieci neuronowych	55
7.1.1. Implementacja warstw konwolucyjnych	55
7.1.2. Implementacja struktur ResNet	58
7.1.3. Implementacja struktur DenseNet	61
7.1.4. Implementacja warstw przetwarzających informacje o parametrach kompresji i rodzaju ramki	65
7.2. Szczegóły implementacyjne procesu treningu	66
7.2.1. Zalety PyTorch Lightning	66
7.2.2. Struktura Projektu w PyTorch Lightning	66
7.2.3. Implementacja modułu danych	67
7.2.4. Implementacja modułu głównego	71
7.3. Konfiguracja eksperymentów	74
7.4. Proces predykacji i obliczania metryk danych po poprawie jakości	76
8. Uzyskane wyniki	79
8.1. Wyniki dla treningu w metodologii GAN	79
8.2. Porównanie z filtrami wbudowanymi w kodik VVC	82
8.3. Porównanie uzyskanych wyników na tle literatury	85
8.4. Wpływ dodatkowych informacji na wejściu sieci na wyniki	88
9. Wnioski	89
9.1. Możliwości zastosowania w praktyce	89
9.2. Ograniczenia i kierunki dalszych badań	90
9.2.1. Zbadanie wyników w warunkach określonych w JVET CTC	90

9.2.2. Wykorzystanie zależności czasowych	90
9.2.3. Wykorzystanie innych informacji	90
9.2.4. Zaawansowane techniki uczenia głębokiego	90
9.2.5. Optymalizacja dla różnych rodzajów treści wideo	90
9.2.6. Badanie efektów percepcyjnych	91
9.2.7. Optymalizacja pod kątem wykorzystania w czasie rzeczywistym	91
9.2.8. Zastosowania w specjalistycznych dziedzinach	91
10. Podsumowanie	92
Bibliografia	93
Wykaz symboli i skrótów	96
Spis rysunków	97
Spis tabel	98

1. Wstęp

W dobie cyfrowej rewolucji jakość przesyłanych danych wideo stała się kluczowym aspektem w wielu dziedzinach, począwszy od transmisji telewizyjnej, a skończywszy na usługach strumieniowania. Wysoka rozdzielcość obrazu często wiąże się z dużymi rozmiarami plików, co stanowi wyzwanie dla efektywnej transmisji i przechowywania danych. W tym kontekście, kodeki takie, jak Versatile Video Coding (H.266/VVC), odegrały istotną rolę w poprawie efektywności kompresji, jednocześnie zachowując wysoką jakość wizualną. Jednakże, kompresja ta nie jest pozbawiona wad, a jednym z głównych wyzwań jest dalsza poprawa jakości obrazu po kompresji.

Celem niniejszej pracy jest zbadanie możliwości wykorzystania sieci neuronowych Generative Adversarial Networks (GAN) przy wykorzystaniu informacji o profilu kodowania, QP, użytych filtrów i rodzaju ramki do poprawy jakości wideo po kompresji przy użyciu kodeka H.266. Praca koncentruje się na analizie różnych architektur sieci neuronowych, takich jak sieci konwolucyjne, ResNet oraz DenseNet, w celu zidentyfikowania najbardziej efektywnej metody poprawy jakości obrazu. Ma charakter badawczy, gdzie głównym źródłem danych jest zbiór Xiph [1], przetwarzany w formacie YUV 420. Analizie poddane zostaną dane po kompresji różnymi konfiguracjami kodeka H.266/VVC w wersji referencyjnej [2]: profile AI (ang. *All Intra*) oraz RA (ang. *Random Access*), parametr QP (32, 37, 42, 47), a także z różne kombinacje użytych filtrów ALF, DB, SAO.

Celem było nie tylko zrozumienie wpływu poszczególnych architektur sieci neuronowych i wykorzystania informacji dotyczących parametrów kompresji na poprawę jakości wideo po dekompresji, ale także zidentyfikowanie najlepszych praktyk i strategii, które mogą być zastosowane w przyszłych badaniach i implementacjach w tej dziedzinie.

2. Kompresja wideo

Rozdział teoretyczny niniejszej pracy ma na celu przedstawienie kluczowych koncepcji i technologii związanych z kompresją wideo, które stanowią fundament dla przeprowadzanych badań. W dobie cyfryzacji i rosnącego zapotrzebowania na treści wideo, efektywna kompresja danych staje się niezbędna, by sprostać ograniczeniom przestrzeni dyskowej oraz przepustowości transmisji danych. Jednocześnie wyzwaniem pozostaje zachowanie wysokiej jakości przekazu, co jest kluczowe dla użytkowników i aplikacji końcowych.

W tym rozdziale szczegółowo omówiono standardy kodeków wideo, ze szczególnym uwzględnieniem najnowszego kodeka H.266/VVC (Versatile Video Coding) [3], który wprowadza szereg innowacji w dziedzinie kompresji. Skupiono się również na tym, jak różne techniki kompresji wpływają na jakość wideo i jakie się z tym wiążą wyzwania, między innymi artefakty kompresji i utrata detali obrazu.

Ponadto niniejszy rozdział zawiera przegląd metod i technologii stosowanych do poprawy jakości wideo po kompresji, w tym roli zaawansowanych technik przetwarzania obrazu i uczenia maszynowego. Szczególna uwaga zostanie poświęcona wykorzystaniu sieci neuronowych, które otwierają nowe możliwości w zakresie optymalizacji wideo.

Przedstawienie bazy teoretycznej jest niezbędnym wprowadzeniem do dalszych części pracy, które koncentrują się na eksperymentalnym badaniu i analizie wybranych metod poprawy jakości wideo. Stanowi to fundament dla zrozumienia zarówno problemów, jak i możliwych rozwiązań w obszarze przetwarzania wideo w dobie nowoczesnych technologii komunikacyjnych.

2.1. Podstawy kompresji wideo

Kompresja wideo jest procesem, którego celem jest redukcja ilości danych użytych do reprezentowania wideo, przy jednoczesnym dążeniu do minimalizowania wpływu tej redukcji na jakość obrazu. Polega na zastosowaniu różnych technik i algorytmów, które pozwalają na efektywne kodowanie informacji wizualnych, usuwając lub zmniejszając elementy mniej istotne z punktu widzenia percepcji ludzkiego wzroku. Kluczowym wyzwaniem w tym procesie jest znalezienie balansu między stopniem redukcji danych, a zachowaniem wysokiej jakości obrazu, aby użytkownik końcowy mógł oglądać treści wideo bez zauważalnych degradacji jakości.

Kompresja dzieli się na bezstratną i stratną. W przypadku bezstratnej, dane są kodowane tak, aby móc odtworzyć je w oryginalnej postaci bez utraty informacji. Natomiast kompresji stratna cechuje się tym, że dane są kodowane w sposób, który umożliwia ich odtworzenie w postaci zbliżonej do oryginalnej, ale z pewną utratą informacji. Przy kodowaniu przekazu wideo, kompresja bezstratna jest rzadko stosowana, ponieważ nie pozwala na uzyskanie znaczącej redukcji rozmiaru pliku.

Kompresja wideo wykorzystuje zarówno zależności przestrzenne między pikselami

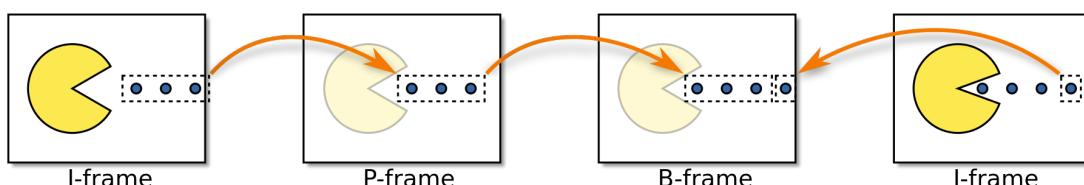
w obrębie jednej klatki do zmniejszenia redundancji przestrzennej, jak i zależności czasowe między kolejnymi klatkami do redukcji redundancji czasowej.

2.1.1. Podstawy kompresji ramek

Podstawowym pojęciem w kontekście kompresji wideo jest rodzaj ramek. Wyróżnia się trzy rodzaje klatek:

1. Klatki I (ang. *Intra*) - klatki niezależne, które nie wymagają informacji z innych klatek do dekodowania. Są one kodowane jako pojedyncze klatki
2. Klatki P (ang. *Predicted*) - klatki przewidywane, które wymagają informacji z klatki I lub P do dekodowania. Są one kodowane jako różnice między klatkami
3. Klatki B (ang. *Bi-directional*) - klatki dwukierunkowe, które wymagają informacji z klatek I, P lub B do dekodowania. Są one kodowane jako różnice między klatkami

Graficzna reprezentacja zależności między klatkami została przedstawiona na rysunku 2.1.



Rysunek 2.1. Porównanie różnych rodzajów ramek, źródło: Wikipedia

Klatki I są kodowane jako pojedyncze obrazy, niezależnie od innych klatek. Ich reprezentacja bitowa jest największa, ponieważ nie wykorzystują one żadnej redundancji czasowej, natomiast zastosowanie samych ramek intra gwarantuje najwyższą jakość każdej ramki. Profil wykorzystujący jedynie taki rodzaj klatek oznaczany jest jako AI - All Intra (ang. *same intra*). Jest on szeroko wykorzystywany w profesjonalnym sprzęcie do nagrywania wideo i archiwizacji, każda ramka może być bowiem przetwarzana niezależnie, umożliwiając szybki dostęp do klatek i ich przetwarzania. To ułatwia przycinanie, wycinanie lub inne formy edycji, bo nie wymaga re-kodowania sąsiednich ramek.

Kompresja ramek intra korzysta z podobnych technik, jak kompresja obrazów statycznych. Klatki I są kodowane w formacie YUV, a następnie poddawane transformacji kosinusowej (DCT) w blokach, której współczynniki częstotliwościowe ulegają kwantyzacji. Uzyskane wartości są następnie kodowane za pomocą kodowania entropijnego, takiego jak kodowanie Huffmana lub kodowanie arytmetyczne.

W procesie kodowania ramek typu inter, stosowanym w nowoczesnych kodekach wideo, wykorzystuje się informacje zawarte w innych klatkach w celu efektywnego kodowania danej ramki. Zamiast kodować każdą ramkę niezależnie, jak w przypadku ramek typu intra, w kodowaniu inter wykorzystuje się różnice między klatkami, co pozwala na znaczną redukcję ilości danych.

2. Kompresja wideo

Kluczowym elementem tego procesu jest predykcja ruchu, polegająca na analizie, jak obiekty lub sceny poruszają się między kolejnymi klatkami. Algorytmy predykcji ruchu generują tzw. wektory ruchu, które opisują te zmiany. Na przykład, jeśli element obrazu przesunął się o kilka pikseli między klatkami, zamiast ponownego kodowania tego elementu, kodowane jest przesunięcie. Następnie kodowane są różnice, zwane resztami, między rzeczywistą ramką, a jej predykcją. W przypadkach, gdzie predykcja jest dokładna, te różnice są minimalne, co pozwala na znaczne oszczędności w zakresie danych.

Kodowanie ramek typu inter obejmuje dwa główne typy ramek: P (predictive frame) i B (bi-directional frame). P są kodowane na podstawie informacji z poprzednich ramek, czy to I, czy innych P. Z kolei B wykorzystują informacje zarówno z poprzednich, jak i następujących ramek, co pozwala na jeszcze efektywniejszą kompresję.

Podsumowując, kodowanie typu inter jest niezwykle ważne dla efektywnej kompresji wideo. Pozwala ono na znaczącą redukcję rozmiaru pliku poprzez inteligentne wykorzystanie informacji międzyklatkowych i predykcji ruchu. Dzięki temu wideo może być przechowywane i transmitowane w sposób znacznie bardziej wydajny, bez konieczności rezygnacji z wysokiej jakości obrazu.

2.2. Standardy kompresji wideo

Od początków ery cyfrowej kompresja wideo stanowiła kluczowy element w rozwoju technologii multimedialnych. Technologia ta pojawiła się w latach 80. i 90. XX wieku, kiedy to rosnące zapotrzebowanie na cyfrowe przechowywanie i transmisję wideo spowodowało potrzebę opracowania efektywnych metod kompresji. Powstające w tym okresie standardy, takie jak MPEG-1 i MPEG-2, zrewolucjonizowały branżę, umożliwiając cyfrowe przechowywanie wideo na dyskach CD i DVD oraz transmisję telewizji cyfrowej.

W miarę postępu technologicznego i wzrostu oczekiwania, co do jakości wideo, szczególnie z nadaniem ery HD, pojawiła się potrzeba tworzenia coraz bardziej zaawansowanych standardów kompresji. To doprowadziło do powstania takich kodeków, jak H.264/AVC (Advanced Video Coding), który stał się fundamentem dla większości współczesnych aplikacji wideo, od strumieniowania online, po zastosowania mobilne.

Ostatnie lata przyniosły kolejne wyzwania, związane z pojawieniem się wideo o rozdzielcości 4K i wyżej, co wymagało jeszcze większej efektywności kompresji. W odpowiedzi na te potrzeby, powstał standard H.265/HEVC (High Efficiency Video Coding), oferujący znaczne ulepszenia w zakresie efektywności, przy jednoczesnym zachowaniu, a nawet poprawie, jakości obrazu.

Najnowsze prace nad standardem H.266/VVC sygnalizują kolejny krok naprzód, oferując wsparcie dla najnowszych trendów w technologii wideo, w tym dla rozszerzonej rzeczywistości i wideo 360 stopni. Wszystkie te innowacje pokazują, jak postęp technologiczny nieustannie napędza rozwój standardów kompresji wideo, umożliwiając tworzenie, przetwarzanie i dystrybucję treści wideo w coraz nowych formach.

2.2.1. Wczesne standardy kompresji

Standard MPEG-1, zaprezentowany na początku lat 90. XX wieku, był jednym z pierwszych, który umożliwił efektywną kompresję cyfrowego wideo i audio, otwierając drogę do ery multimedialnej. Jego głównym celem było umożliwienie przechowywania niewielkich fragmentów wideo na CD-ROMach, z jakością porównywalną do systemu VHS. MPEG-1 został zaprojektowany do pracy przy około 1.5 Mb/s, odpowiadając ówczesnym dyskom CD-ROM. Standard ten obsługiwał stosunkowo niską rozdzielczość obrazu, około 352x240 pikseli w systemie NTSC i 352x288 w PAL, wystarczających do celów konsumenckich tamtych czasów. MPEG-1 znalazł swoje zastosowanie głównie w formacie Video CD (VCD), który stał się popularny w wielu krajach, jako dostępna alternatywa dla VHS.

MPEG-2, który pojawił się w połowie lat 90., zrewolucjonizował branżę, wprowadzając znaczne ulepszenia w porównaniu do swojego poprzednika. Ten standard był kluczowy dla rozwoju telewizji cyfrowej i formatu DVD. Zapewniał lepszą jakość obrazu, co było niezbędne do obsługi nowych form mediów i transmisji. MPEG-2 obsługiwał wyższe rozdzielczości, sięgając standardów SD (480i/576i) i częściowo HD, umożliwiając znacznie lepszą jakość obrazu, niż MPEG-1, co przyczyniło się do jego sukcesu w telewizji cyfrowej oraz w przemyśle DVD. Standard ten szybko stał się podstawą dla transmisji cyfrowej w różnych formach – naziemnej, satelitarnej, jak i kablowej – i był podstawą dla formatu DVD, który przyczynił się do globalnej popularności cyfrowego wideo w domach na całym świecie.

Te dwa standardy, MPEG-1 i MPEG-2, miały duży wpływ na rozwój cyfrowej kompresji wideo, ustanawiając podstawy, na których oparto późniejsze innowacje w tej dziedzinie. Ich wprowadzenie umożliwiło przechowywanie i dystrybucję treści wideo w sposób, który był wcześniej niemożliwy, otwierając nowe możliwości dla przemysłu rozrywkowego i informacyjnego.

2.2.2. Advanced Video Coding (H.264/AVC)

H.264 [4], znany również jako Advanced Video Coding (AVC), to standard kompresji wideo, który zrewolucjonizował branżę, stając się jednym z najbardziej wpływowych kodeków począwszy od wprowadzenia go w 2003 roku. Jego rozwój i implementacja były odpowiedzią na rosnące zapotrzebowanie na wydajniejszą kompresję wideo w obliczu coraz wyższej popularności strumieniowania wideo w Internecie oraz większych rozdzielczości transmisji, takich jak High Definition (HD).

H.264 wprowadził szereg kluczowych innowacji technologicznych, które znacząco poprawiły efektywność kompresji w porównaniu do poprzednich rozwiązań, takich jak MPEG-2. Jedną z najważniejszych nowości było ulepszenie technik predykcji ruchu. Wektory ruchu w AVC wyznaczane są z dokładnością do 1/4 piksela mogą wykraczać poza granice obrazu. H.264 używa bardziej zaawansowanych i elastycznych metod predykcji, pozwalając na dokładniejsze śledzenie ruchu i zmian w obrazie między klatkami. Dzięki

2. Kompresja wideo

temu możliwe było znaczne zmniejszenie ilości danych potrzebnych do opisania ruchu w sekwencji wideo.

Kolejną istotną zmianą w H.264 było wprowadzenie predykcji dla ramek typu Intra. Wcześniej standardy, takie jak MPEG-2, nie wykorzystywały predykcji dla ramek intra, co powodowało znaczne straty w zakresie efektywności kompresji. H.264 używa predykcji ramek intra, co pozwala na znaczne zmniejszenie rozmiaru pliku.

Kodek H.264 stosuje również ulepszone techniki adaptacyjnego kodowania entropijnego, w tym kodowanie Cabac (Context-adaptive binary arithmetic coding) i Cavlc (Context-adaptive variable-length coding). Te metody pozwalają na bardziej efektywne kodowanie współczynników kwantyzacji i innych informacji, co przekłada się na dalsze zwiększenie efektywności kompresji.

Standard wspiera tryby kodowania bez przeplotu (ramkowy) oraz z przeplotem (polowy), umożliwiając wybór trybu kodowania na poziomie makrobloku lub ramki. Zastosowano też transformację całkowitoliczbową wymagającą arytmetyki 16-bitowej.

Dzięki tym innowacjom, H.264 stał się normą w wielu aplikacjach, od strumieniowania wideo na żywo, przez zastosowania mobilne, po nagrywanie wideo wysokiej rozdzielczości. Jego elastyczność, wysoka wydajność i doskonała jakość obrazu sprawiły, że stał się on jednym z najważniejszych kodeków wideo w nowoczesnym przemyśle multimedialnym.

Zastosowania H.264 H.264 odegrał kluczową rolę w rozwoju strumieniowania wideo online. Dzięki swojej wysokiej efektywności kompresji i zdolności do dostarczania wysokiej jakości obrazu przy stosunkowo niskiej przepływności danych, stał się wyznacznikiem w serwisach takich jak YouTube, Netflix, i innych platformach VOD (Video on Demand). Jego zdolność do adaptacji do różnych prędkości połączeń internetowych sprawiła, że okazał się niezastąpiony w dostarczaniu płynnego wideo w różnorodnych warunkach sieciowych.

W świecie mobilnym, gdzie ograniczenia dotyczące przepustowości i przestrzeni dyskowej są kluczowe, H.264 zapewnia idealną równowagę między jakością, a wielkością pliku. Jest on szeroko stosowany w nagrywaniu i odtwarzaniu wideo na smartfonach i tabletach, umożliwiając użytkownikom korzystanie z wysokiej jakości treści filmów bez obciążania pamięci urządzenia.

H.264 był jednym z pierwszych kodeków, który umożliwił szeroko dostępną dystrybucję wideo w wysokiej rozdzielczości (HD). Dzięki swojej wydajności stał się standardem w transmisjach telewizyjnych HD, a także w produkcji Blu-Ray. Jego zdolność do obsługi obrazu 1080p bez znaczących strat jakościowych, przy jednoczesnym utrzymaniu rozsądnej wielkości pliku, sprawiła, że stał się preferowanym wyborem dla HD.

H.264 znalazł również zastosowanie w profesjonalnej produkcji i nadawaniu, oferując odpowiednią jakość dla transmisji telewizyjnych oraz elastyczność potrzebną w nowoczesnej produkcji wideo. Dzięki temu, że potrafił się dostosować do różnych przepływności

danych i rozdzielczości, jest wysoce ceniony przez producentów i nadawców. Stosowany jest w używanym w Polsce jeszcze do niedawna standardzie telewizji naziemnej DVB-T.

Podsumowując, H.264 zrewolucjonizował sposób, w jaki wideo jest nagrywane, przetwarzane, przechowywane i dystrybuowane. Jego uniwersalność sprawia, że jest on niezwykle popularny w różnych dziedzinach, od użytku domowego po profesjonalne zastosowania. Bez względu na to, czy chodzi o strumieniowanie treści online, nagrywanie wideo na urządzeniach mobilnych, czy nadawanie telewizyjne w wysokiej rozdzielczości, H.264 pozostaje jednym z najbardziej wpływowych i wszechstronnych standardów kompresji wideo.

2.2.3. High Efficiency Video Coding (H.265/HEVC)

High Efficiency Video Coding (HEVC) [5], znany również jako H.265, jest następcą H.264 i reprezentuje kolejny znaczący krok naprzód w technologii kompresji wideo. Wprowadzono go w 2013 roku i od tego czasu zyskał uznanie za swoją zdolność do znacznej poprawy efektywności kompresji, szczególnie przy wyższych rozdzielczościach obrazu.

HEVC został zaprojektowany, aby sprostać rosnącym wymaganiom nowoczesnej transmisji wideo, w tym potrzebie obsługi wyższych rozdzielczości, takich jak 4K (Ultra HD) i 8K. Dzięki ulepszonym algorytmom kompresji, HEVC jest w stanie zredukować rozmiar pliku o około 50% w porównaniu do H.264, przy tej samej jakości obrazu. To stanowi istotną innowację, biorąc pod uwagę rosnące zapotrzebowanie na wysokiej jakości treści wideo i ograniczenia przepustowości sieciowej.

Jednym z kluczowych ulepszeń w HEVC jest zastosowanie bardziej zaawansowanej techniki predykacji ruchu i podziału obrazu na bloki. HEVC wykorzystuje tzw. jednostki kodowania drzew (Coding Tree Units, CTUs), które mogą być znacznie większe niż bloki używane w H.264, pozwalając na bardziej efektywną analizę i kompresję dużych obszarów obrazu. Ponadto HEVC oferuje wyższą elastyczność w wyborze rozmiarów bloków i podbloków, co pozwala na lepsze dostosowanie do różnorodnych cech obrazu.

HEVC wprowadza również ulepszenia w zakresie kwantyzacji, filtracji i kodowania entropijnego, co przyczynia się do zwiększenia stopnia kompresji. Te ulepszenia pozwalają na redukcję artefaktów kompresji i zapewnienie wyższej jakości obrazu, nawet przy niższych przepływnościach danych.

Wprowadzenie HEVC miało duże znaczenie dla branży, oferując bardziej efektywne strumieniowanie i dystrybucję wideo w wysokiej rozdzielczości. Jest to szczególnie ważne w kontekście rosnącej popularności wideo 4K i 8K, gdzie tradycyjne metody kompresji nie są wystarczające. Dzięki HEVC, możliwe jest dostarczanie treści o bardzo wysokiej rozdzielczości bez wymagania proporcjonalnego wzrostu przepustowości sieciowej, co otwiera nowe możliwości dla przemysłu rozrywkowego, nadawców i konsumentów treści wideo.

2. Kompresja wideo

Zastosowania HEVC High Efficiency Video Coding (HEVC) odgrywa kluczową rolę w kształtowaniu przyszłości transmisji wideo, zwłaszcza w kontekście rosnącego zapotrzebowania na filmy o wysokiej rozdzielczości. Jego zdolność do znacznej redukcji rozmiaru plików bez pogarszania jakości obrazu czyni go idealnym rozwiązaniem dla wyzwań, z jakimi boryka się branża wideo w dobie szybkiego rozwoju technologii i zmieniających się oczekiwani konsumentów.

W miarę, gdy coraz więcej konsumentów oczekuje treści wideo w rozdzielczości 4K i 8K, HEVC staje się bardziej istotny. Standard ten umożliwia strumieniowanie treści o wysokiej rozdzielczości przy znacznie niższej przepływności danych w porównaniu do poprzednich kodeków, takich jak H.264. Jest to szczególnie ważne dla platform strumieniujących, które muszą balansować między oferowaniem wysokiej jakości obrazu, a ograniczeniami przepustowości sieciowej.

HEVC jest bardziej elastyczny w obsłudze różnych formatów i rozmiarów obrazu, co jest kluczowe w środowisku, gdzie różnorodność urządzeń odtwarzających jest ogromna, od smartfonów po duże ekrany telewizyjne. Zdolność do dostosowania się do szerokiego zakresu rozdzielczości i rodzajów treści czyni HEVC idealnym wyborem dla licznych zastosowań, od strumieniowania na żywo po wideo na żądanie (VOD).

HEVC ma również znaczący wpływ na przemysł telewizyjny i nadawczy, umożliwiając transmisję w rozdzielczości HD i Ultra HD przy użyciu obecnej infrastruktury transmisyjnej. To pozwala nadawcom na oferowanie wyższej jakości treści bez konieczności kosztownej rozbudowy sieci. Przykładem może być wprowadzany w 2022 roku w Polsce drugi standard telewizji naziemnej DVB-T2 stosujący kompresję H.265.

2.2.4. Versatile Video Coding (VVC) i Przyszłe Standardy

Versatile Video Coding (VVC), znany również jako H.266 [3] i ISO/IEC 23080-3 [6], to najnowszy standard w dziedzinie kompresji wideo, który ma na celu dalsze ulepszanie efektywności kompresji oraz dostosowanie się do rozwijających technologii i formatów. Jego pierwsza wersja została opublikowana w 2020, jako następca HEVC (H.265), oferując obiecujące perspektywy dla przyszłości przetwarzania i dystrybucji treści.

Głównym celem VVC jest zwiększenie efektywności kompresji, co jest kluczowe w obliczu stale rosnących wymagań dotyczących jakości i rozdzielczości filmów. VVC ma potencjał do redukcji rozmiaru pliku o około 50% w porównaniu do HEVC przy tej samej jakości obrazu, co ma znaczące implikacje zarówno dla konsumentów, jak i dla przemysłu. Zmniejszenie szybkości transmisji bez utraty jakości jest niezwykle ważne, szczególnie w kontekście rosnącej popularności wideo w rozdzielczości 8K, gdzie wielkość plików może stanowić wyzwanie.

Standard H.266 wprowadził również znaczne usprawnienia w kompresji ramek intra, zwiększając liczbę modeli predykacji z 35 w standardzie HEVC do 67. Pozwala to na redukcję przepływności dla ramek typu intra o 25% w stosunku do standardu H.265 kosztem większej złożoności obliczeniowej podczas kodowania.

Jednak to, co wyróżnia VVC, to nie tylko jego zdolności kompresyjne, ale także wsparcie dla nowych i wschodzących formatów wideo. Standard ten został zaprojektowany z myślą o obsłudze treści 360-stopniowych oraz wykorzystywanej w rozszerzonej rzeczywistości (AR), co jest coraz bardziej istotne w miarę rozwoju tych technologii. Umożliwienie efektywnej kompresji dla wskazanych formatów jest kluczowe, ponieważ wymagają one znacznie większych ilości danych, niż tradycyjne filmy, ze względu na ich złożoność i wymagania dotyczące rozdzielczości.

VVC dzięki swojej zaawansowanej technologii ma potencjał do znacznego wpływu na sposób, w jaki przekaz jest strumieniowany, przechowywany i konsumowany. Jego zdolność do radzenia sobie z wysokimi rozdzielczościami i złożonymi formatami otwiera nowe możliwości dla przemysłu wideo, od produkcji filmów i programów telewizyjnych, po gry komputerowe i aplikacje AR/VR. W miarę, jak technologia ta będzie wdrażana i adaptowana, możemy oczekiwać, że stanie się ona nowym standardem w przetwarzaniu i dystrybucji przekazu, napędzając innowacje i poprawiając doświadczenia użytkowników na całym świecie.

Standard H.266 jest obecnie w fazie wdrażania i nie jest jeszcze szeroko stosowany w przemyśle. Jednak jego potencjał jest ogromny, a wpływ na przyszłość wideo może być znaczący. W miarę jak technologia ta będzie wdrażana i adaptowana, możemy oczekiwać, że stanie się ona nowym standardem w przetwarzaniu i dystrybucji przekazu, napędzając innowacje i poprawiając doświadczenia użytkowników na całym świecie. Z nadchodzących projektów, w 2024 w Brazylii ma zostać wprowadzony standard telewizji TV 3.0 korzystający z kodeka VVC [7], a organizacja DVB od 2022 roku dodała standard H.266 do swojej głównej specyfikacji [8].

2.3. Wyzwania i artefakty kompresji wideo

Kompresja wideo, mimo swoich licznych zalet, wiąże się z różnymi wyzwaniami technicznymi i jakościowymi. Jednym z głównych problemów są artefakty, pojawiające się w wyniku stratnej kompresji wideo. Jest to efekt utraty informacji, który jest nieunikniony w procesie redukcji rozmiaru pliku. Wyzwaniem dla twórców kodków i algorytmów kompresji jest zminimalizowanie tych artefaktów, przy jednoczesnym utrzymaniu jak najwyższej efektywności kompresji.

2.3.1. Artefakty blokowe

Najbardziej powszechnymi artefaktami są artefakty blokowe, zauważalne jako widoczne granice między blokami kompresji. Stają się szczególnie widoczne w obszarach o jednolitym kolorze i wynikają z kwantyzacji i ograniczenia przepływności danych. Na rysunku 2.2 przedstawiono porównanie klatki przed i po kompresji kodekiem VVC przy użyciu silnej kompresji w profilu AI bez użycia filtra przeciwdziałającemu tym artefaktom.



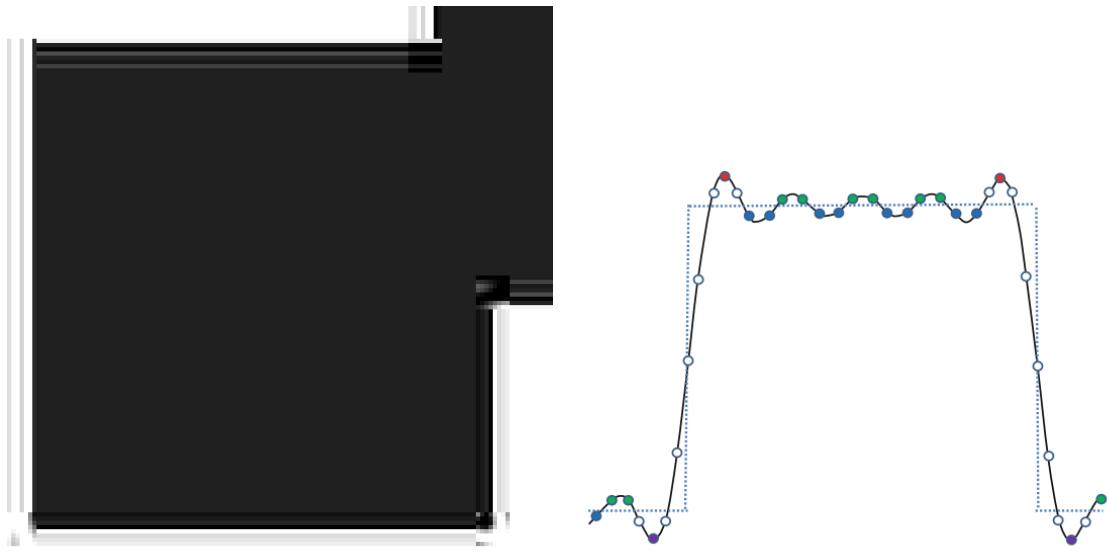
(a) klatka przed kompresją

(b) klatka po kompresji

Rysunek 2.2. Efekt blokowy przy QP=50

2.3.2. Ringing Artifacts

Zastosowanie transformacji DCT powoduje występowanie tzw. artefaktów dzwoniących (ang. *ringing artifact*, rys.2.3), ukazujących się jako echo na krawędziach. Zjawisko to znane jest jako efekt Gibbsa - funkcja aproksymacji nadmiernie oscyluje w punktach nieciągłości. Na rysunku 2.3 przedstawiono przykład takiego artefaktu oraz efektu Gibbsa.



(a) Przykład artefaktu

(b) Efekt Gibbsa [9]

Rysunek 2.3. Ringing artifact

2.3.3. Wyzwanie utraty jakości przy wysokim stopniu kompresji

Kolejnym wyzwaniem jest utrzymanie akceptowalnej jakości obrazu przy wysokim stopniu kompresji. Staje się to szczególnie istotne dla zastosowań, gdzie przepustowość

sieciowa jest ograniczona, jak w przypadku strumieniowania wideo na żywo lub mobilnego przesyłu danych.

Głównym parametrem kontrolującym stopień kompresji jest QP, czyli parametr kwantyzacji (ang. *Quantization Parameter*). Odpowiada on za kontrolę stopnia kwantyzacji, czyli redukcję dokładności danych, niezbędnej do kompresji wideo. Im wyższa wartość QP, tym większa redukcja rozmiaru pliku, ale także znaczniejsza utrata jakości obrazu. Wartość QP bywa zwykle wyrażana w skali logarytmicznej od 0 do 51, gdzie 0 oznacza brak kompresji, a 51 maksymalną kompresję. Na rysunku 2.4 przedstawiono różnice pomiędzy tą samą klatką przed kompresją i po kompresji kodekiem VVC w trybie AI o parametrze QP o wartości 50 oraz wykres przedstawiający zależność pomiędzy parametrem QP oraz przepływnością dla tego standardu.



Rysunek 2.4. Wpływ parametru QP na jakość obrazu oraz przepływność

2.4. Metody poprawy jakości wideo przy kompresji kodekiem VVC

W celu poprawy jakości wideo przy kompresji kodekiem VVC można zastosować szereg metod, które zostały przedstawione w kolejnych podrozdziałach.

2.4.1. Filtr deblokujący

Filtr deblokujący redukuje efekt blokowy wygładzając krawędzie między makro-blokkami.

2.4.2. Filtr SAO

Sample Adaptive Offset post-filter został dodany wraz ze standardem H.265. Pozwala na redukcję ringing artifacts powstających w wyniku zastosowania DCT i pojawiących się szumów, poprzez adaptacyjne wyrównywanie wartości. Kategoryzuje zrekonstruowane próbki na podstawie wyszukiwania krawędzi (porównywanie z sąsiednimi próbkkami) oraz pasma (analiza wartości próbki) i dopasowuje na tej podstawie wartości filtru (offset).

2.4.3. Filtr ALF

Adaptive Loop Filter dodano w standardzie H.266. Jest to adaptacyjny filtr wykonujący ostateczne korekty zakodowanego sygnału tak, aby zredukować błąd średnio-kwadratowy pomiędzy zakodowanymi i oryginalnymi próbkkami. Filtrowanie odbywa się oddzielnie dla

2. Kompresja wideo

jasności i barwy używając różnej wielkości filtrów dla tych kanałów. Specjalny komponent ALF-CC dodatkowo wykorzystuje korelację między luminacją i chrominancją.

Wykorzystanie filtrów takich jak SAO, DB i ALF w procesie dekodowania wideo potrafi w pewnych przypadkach negatywnie wpływać na metryki takie jak PSNR (ang. *Peak Signal-to-Noise Ratio*) i SSIM (*Structural Similarity Index Measure*). Choć głównym celem tych filtrów jest poprawa percepcyjnej jakości obrazu poprzez redukcję artefaktów kompresji, ich działanie na wspomniane metryki może być zróżnicowane.

2.4.4. Inne metody poprawy jakości wideo

Wyłączenie niektórych filtrów i zwiększenie parametru QP pozwala znacznie zredukować przepływność transmisji danych kosztem utraty szczegółowości i omawianymi wcześniej artefaktami, co znacznie zniża wskaźnik QoE (ang. *quality of experience*, jakość doświadczenia) zdekodowanych informacji. Dlatego istotne jest zbadanie możliwości poprawy jakości plików VVC po stronie dekodera.

Sieci konwolucyjne do poprawy jakości wideo Sieci konwolucyjne (CNN, ang. *Convolutional Neural Networks*) są obecnie jednym z najbardziej obiecujących podejść do poprawy jakości wideo. W ostatnich latach pojawiło się wiele prac naukowych, które wykorzystują modele splotowe w tym celu. Głównym obszarem badań jest ich zastosowanie przy kompresji stratnej, co jest szczególnie istotne w przypadku kodeków wideo, takich jak H.266 i H.265, które są szeroko używane w różnych zastosowaniach.

Przykłady rozwiązań stosujących sieci neuronowe do poprawy jakości wideo przy kompresji kodekiem VVC zostały przedstawione w rozdziale 3.5 poświęconym sieciom neuronowym.

3. Sieci neuronowe

Sztuczne sieci neuronowe to jedno z najbardziej fascynujących i dynamicznie rozwijających się narzędzi w dziedzinie sztucznej inteligencji i uczenia maszynowego. Ich historia sięga lat 40. i 50. XX wieku, kiedy to naukowcy zaczęli eksplorować ideę tworzenia maszyn zdolnych do naśladowania procesów zachodzących w ludzkim mózgu. Te wcześnie koncepcje ewoluowały przez dekady, prowadząc do powstania współczesnych sieci neuronowych, które są teraz nieodłącznym elementem zaawansowanych systemów AI.

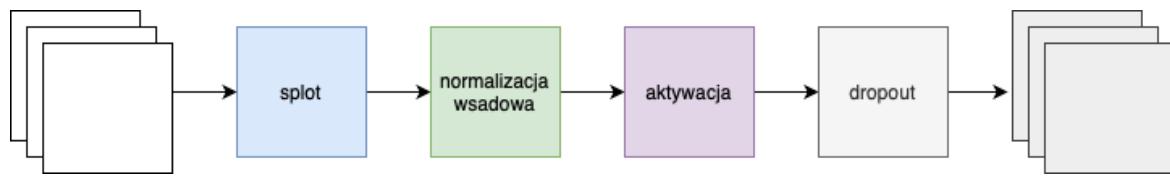
Podstawowym założeniem sztucznych sieci neuronowych jest próba emulacji działania ludzkiego mózgu. Tak, jak nasz mózg składa się z sieci połączonych ze sobą neuronów, przetwarzających i przekazujących informacje, tak sztuczne sieci neuronowe wykorzystują szereg połączonych ze sobą "neuronów" - jednostek obliczeniowych, pracujących wspólnie nad rozwiązywaniem złożonych problemów. Każdy "neuron" w sieci otrzymuje dane wejściowe, przetwarza je z wykorzystaniem zdefiniowanej funkcji matematycznej i przekazuje wynik do kolejnych "neuronów".

Sztuczne sieci neuronowe uczą się z doświadczenia. Oznacza to, że ich zdolność do wykonywania zadań, takich jak klasyfikacja, rozpoznawanie wzorców czy prognozowanie, poprawia się w miarę przetwarzania coraz większej ilości danych. Proces ten jest analogiczny do uczenia się ludzkiego mózgu, który adaptuje się i rozwija na podstawie napływających do niego informacji i doświadczeń.

W ciągu ostatnich dekad, dzięki postępowi w obliczeniach i dostępności dużych zbiorów danych, sieci neuronowe stały się coraz bardziej zaawansowane i skuteczne. Obecnie są wykorzystywane w szerokim spektrum aplikacji, od rozpoznawania mowy i obrazów, po autonomiczne pojazdy i personalizowane systemy rekomendacji. Ich zdolność do modelowania skomplikowanych wzorców i uczenia się z danych czyni je niezwykle potężnym narzędziem w rękach naukowców i inżynierów, otwierając nowe horyzonty w dziedzinie sztucznej inteligencji.

3.1. Sieci konwolucyjne

Sieci konwolucyjne CNN stanowią kluczowy rodzaj sztucznych sieci neuronowych, które zdobyły ogromną popularność w dziedzinie przetwarzania obrazów. Zaprojektowane specjalnie do efektywnego analizowania wzorców w danych przestrzennych, CNN wprowadzają koncepcję warstw splotowych, umożliwiając skuteczne wykrywanie lokalnych cech, takich jak krawędzie czy tekstury. Każda warstwa konwolucyjna składa się z filtrów, przesuwających się po danych wejściowych, stosując operację splotu. Ponadto, sieci splotowe często wykorzystują warstwy aktywacji, jak ReLU (ang. *Rectified Linear Unit*), oraz warstwy poolingowe, które zmniejszają rozmiar przestrzenny sygnału, zwiększając jednocześnie abstrakcyjność informacji. Dzięki zdolności do hierarchicznego uczenia się cech o różnym



Rysunek 3.1. Diagram struktury warstwy konwolucyjnej

poziomie złożoności, CNN są niezwykle skuteczne w zadaniach związanych z analizą obrazów, takich jak klasyfikacja czy detekcja obiektów.

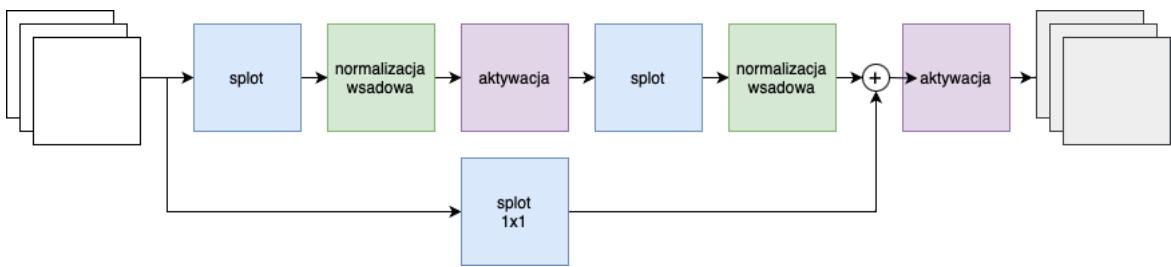
Każda warstwa splotowa typowo składa się z następujących elementów (warstw ukrytych), przedstawionych na rysunku 3.1:

- **Konwolucja** - podstawowa operacja, która przetwarza dane wejściowe, wykorzystując filtr o określonym rozmiarze (np. 3x3, 5x5). Parametry takie, jak rozmiar jądra, padding (dopełnienie) i stride (krok przesunięcia) są dostosowywane w zależności od potrzeb konkretnego modelu i zadania.
- **Normalizacja wsadowa** - po konwolucji często stosuje się normalizację wsadową, która stabilizuje i przyspiesza proces uczenia, normalizując rozkład danych na wyjściu z każdej warstwy konwolucyjnej. BatchNorm redukuje problem wewnętrznych przesunięć kowariantnych, poprawiając efektywność i stabilność uczenia sieci.
- **Funkcja aktywacji** - po normalizacji wsadowej stosuje się funkcję aktywacji, taką jak ReLU (Rectified Linear Unit) lub jej warianty. Funkcja aktywacji wprowadza nieliniowość do procesu uczenia, co jest kluczowe dla efektywnego modelowania złożonych zależności w danych.
- **Dropout** - opcjonalnie, po funkcji aktywacji można zastosować dropout, który polega na losowym “wyłączaniu” pewnego odsetka neuronów podczas treningu. Jest to technika regularyzacji, pomagająca zapobiegać przeuczeniu modelu poprzez zwiększenie jego zdolności do generalizacji.

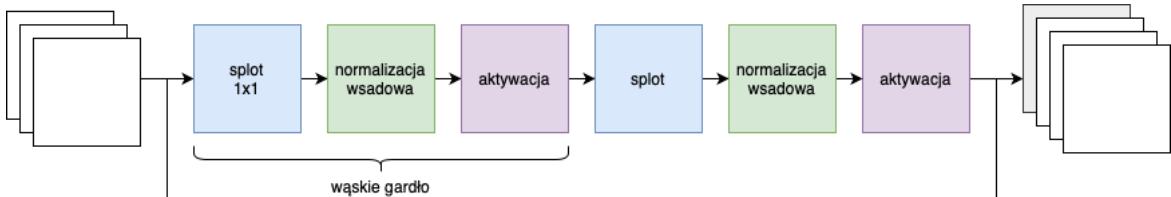
3.2. ResNet

Architektura ResNet [11] wprowadza koncepcję połączeń resztkowych, pozwalających na efektywniejsze przetwarzanie i naukę głębszych sieci neuronowych (rysunek 3.2). Połączenia te pomagają w przeciwdziałaniu problemowi zanikającego gradientu, co jest szczególnie istotne w zadaniach związanych z poprawą jakości przekazu wideo, gdzie subtelne różnice i szczegóły mają kluczowe znaczenie.

Jako, że parametry z pierwszych warstw są przekazywane bezpośrednio do wyjścia, sieci ResNet są szczególnie skuteczne w zadaniach, w których istotne są szczegóły obrazu. W przypadku poprawy jakości wideo po kompresji, sieci ResNet są w stanie efektywnie radzić sobie z redukcją artefaktów kompresji, co zostało potwierdzone w eksperymentach.



Rysunek 3.2. Diagram struktury warstwy ResNet



Rysunek 3.3. Diagram struktury warstwy DenseNet

3.3. DenseNet

DenseNet [12] charakteryzuje się unikalną strukturą, w której każda warstwa jest bezpośrednio połączona z każdą inną. Ta cecha sprawia, że DenseNet jest szczególnie skuteczny w przekazywaniu informacji i uczeniu się na bogatych zestawach cech, co jest korzystne w procesie poprawy jakości wideo.

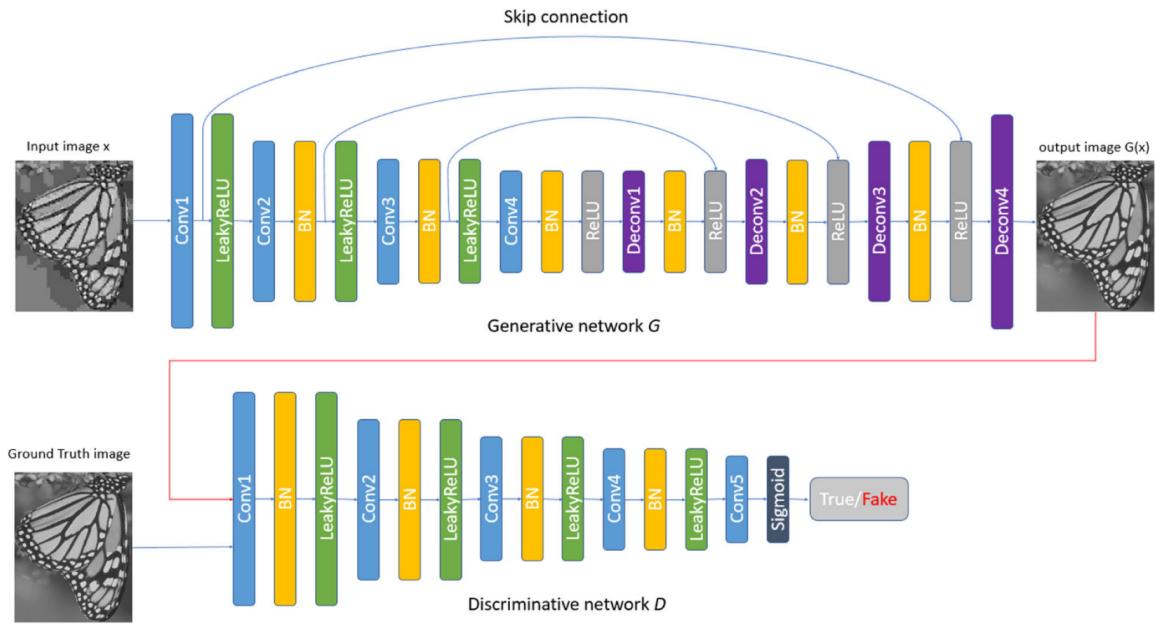
Architektura wprowadza koncepcję połączeń gęstych, które pozwalają na efektywniejsze przetwarzanie i naukę głębszych sieci neuronowych (rysunek 3.3). Działają one podobnie do połączeń resztkowych, natomiast zamiast operacji sumowania wartości wykonywana jest ich konkatenacja. Dlatego w przypadku sieci gęstych liczbę filtrów kolejnych warstw nazywa się przyrostem, bowiem o tę wartość zwiększa się liczba warstw przekazywanych informacji.

Tak narastająca liczba cech dla głębszych modeli może powodować znaczne zwiększenie wymagań sprzętowych, w związku z czym zaproponowano podział sieci na bloki po kilka warstw gęstych, a pomiędzy nimi wykorzystanie tzw. warstwy przejścia, które redukują wymiarowość informacji dwukrotnie przy pomocy splotu o jądrze 1x1.

Twórcy sieci gęstych zaproponowali również użycie warstw typu bottleneck (ang. wąskie gardło), stosujących splot o jądrze 1x1 w celu kompresji ekstrahowanych cech wewnętrz każdej warstwy.

3.4. Generatywne sieci przeciwwstawne

Generative Adversarial Networks (GANs) to kategoria sztucznych sieci neuronowych, wprowadzona przez Iana Goodfellowa i jego zespół w 2014 roku [13]. Modele GAN składają się z dwóch głównych komponentów: generatora i dyskryminatora. Generator ma za zadanie produkować dane, podczas gdy dyskryminator próbuje odróżnić prawdziwe dane



Rysunek 3.4. Architektura sieci ARGAN[15]

od tych stworzonych przez generator. Proces ten przebiega w sposób antagonistyczny, gdzie obie sieci ewoluują, dążąc do osiągnięcia równowagi. GANs znalazły zastosowanie w generowaniu realistycznych obrazów, syntezie danych oraz innych dziedzinach sztucznej inteligencji. Jednak, ze względu na swoją skomplikowaną dynamikę treningową, wymagają starannej optymalizacji i dostosowania parametrów [14].

Sieci GAN znane są ze swoich zdolności do generowania realistycznych danych. W przypadku redukcji artefaktów kompresji, generator może być nauczony tworzyć dane, które są bliskie pierwotnym danym bez występowania defektów wizualnych. Dyskryminator pełni rolę krytyka oceniającego, czy są one zbliżone do rzeczywistych. Taka rywalizacja pomiędzy generatorem a dyskryminatorem może prowadzić do poprawy jakości informacji i jednocześnie redukcji artefaktów kompresji.

Przykładem zastosowania GAN w tym zadaniu może być architektura ARGAN (Artifact Reduction GAN[15]) zaproponowana przez Zengshun Zhao. Generator na wejściu przyjmuje nie szum, a obraz po zdekodowaniu kodekiem JPEG, a jego wyście stanowi predykcja obrazu przed kompresją. Dzięki temu sieć jest uczona odtwarzania realistycznych obrazów z zakodowanych przy silnej kompresji stratnej (rys.3.4).

3.5. Sieci neuronowe w poprawie jakości wideo po kompresji

W tym rozdziale przedstawiono przykłady zastosowania sieci neuronowych w zadaniu poprawy jakości wideo po kompresji. Standard VVC, choć stosunkowo nowy, opiera się na podobnym podstawowym procesie kodowania wideo, jak poprzednie standardy. Dlatego rozwiązania, które zostały zaproponowane dla wcześniejszych standardów, mogą być również stosowane w przypadku VVC.

3.5.1. Metody przetwarzające pojedyncze ramki

Większość istniejących rozwiązań opiera się, analogicznie do proponowanego w niniejszej pracy, o przetwarzanie pojedynczych klatek przekazu wideo. W tym przypadku sieć neuronowa jest uczona na podstawie pary klatek: oryginalnej i zdekodowanej. W wyniku procesu uczenia, sieć jest w stanie nauczyć się poprawiać jakość pojedynczych ramek, redukując artefakty kompresji.

Architektura CNNF [16] jest siecią neuronową o niewielkiej liczbie parametrów do poprawy jakości ramek intra po kompresji kodekiem HEVC, wykorzystując informacje z mapy QP. Autorzy skupili się na wykorzystaniu jednego modelu do redukcji artefaktów niezależnie od wartości parametru QP, natomiast jedynie dla klatek typu I. Uzyskano do 7% redukcji przepływności danych przy zachowaniu jakości na podobnym poziomie dla profilu AI, zastępując jednocześnie filtry przetwarzania końcowego. Ważne jest również, że wykorzystano kompresję sieci neuronowej, co pozwoliło na zmniejszenie rozmiaru modelu i wymagań sprzętowych.

QE-CNN [17] również jest siecią neuronową o małej liczbie parametrów, ale w przeciwieństwie do CNNF, skupia się na jakości zarówno ramek intra, jak i inter. Sieć składa się z dwóch oddzielnych modeli dla obu rodzajów klatek. To rozwiązanie uzyskuje wzmacnianie na poziomie 0.34 dB dla inter i 0.26 dB dla intra, umożliwiając zastosowanie silniejszej kompresji i reducję przepływności danych na poziomie 8.3% przy kompresji kodekiem HEVC.

Kolejnym przykładem jest sieć CVEGAN [18], stworzona w celu poprawy artefaktów kompresji HEVC, natomiast została przetestowana również przy kodowaniu VVC [19]. Aby próbki po poprawie jakości były bardziej podobne do oryginalnych, model na wyjściu generuje mapę różnicową nakładaną na fragment po kompresji oraz zastosowano dodatkowe funkcje celu, które wymuszają podobieństwo cech danych oryginalnych i zrekonstruowanych. Wykorzystano połączenie klasycznych funkcji celu dla zadania poprawy jakości (SSIM, MS-SSIM, l_1 i l_2) oraz specjalnie opracowaną funkcję straty wykorzystującą projekcję cech wydobytych przez sieć dyskryminatora na sferę (ReSphereGAN). Rozwiązanie pozwala na poprawę jakości przekazu wideo, ale wymaga zastosowania skomplikowanej sieci neuronowej po stronie dekodera, co może być utrudnione ze względu na ograniczenia sprzętowe. Autorzy uzyskali poprawę jakości dekompresji na poziomie 0.13 dB dla zbioru danych YouTube User Generated Content [20].

Autorzy WCDANN [21] zaproponowali użycie specjalnych bloków resztkowych, WCDAB, w celu poprawy jakości wideo po kompresji VVC. Uzyskane wyniki na poziomie -4.12% BD-BR.

Jednym z najlepszych rozwiązań dla VVC jest model VVCNN [22]. Autorzy zastosowali prostą sieć wykorzystującą proste połączenia pomijające oraz mapę QP jako dodatkowe informacje. Uzyskano -4.54% BD-rate dla profilu RA oraz -5.21% dla AI.

3.5.2. Metody przetwarzające grupy ramek

Przetwarzanie grupy ramek wideo pozwala na wykorzystanie informacji z wielu klatek, co może być szczególnie przydatne w przypadku poprawy jakości klatek typu B czy P. Jedną z pierwszych sieci proponujących takie rozwiązanie jest STResNet [23], która na wejściu przyjmuje nie tylko aktualną, ale i poprzednie ramki. Architektura pozwoliła na redukcję przepływności danych o 5.1% dla kodeka HEVC.

Architektura MW-GAN [24] proponuje wykorzystanie sieci GAN do poprawy jakości wideo po kompresji. W tym przypadku generator przyjmuje jako wejście nie tylko aktualną ramkę, ale również poprzednią i następną. Dzięki temu sieć jest w stanie nauczyć się poprawiać jakość klatek typu B i P, które są generowane na podstawie informacji z kilku ramek. Autorzy uzyskali poprawę jakości na poziomie 4.55 MOS (ang. *Mean Opinion Score*), co było wówczas najwyższym wynikiem.

Podsumowując, tego typu rozwiązania polegają głównie na przetwarzaniu grup ramek, co znacznie komplikuje proponowane modele sieci i wymaga ogromnych zasobów obliczeniowych w celu poprawy jakości. Modelowanie ruchu przeprowadzane jest już na etapie kodowania danych, informacje z tym związane dostępne są po stronie dekodera i mogą być łatwo używane jako informacja dodatkowa, zamiast przetwarzania pełnych sekwencji ramek.

3.5.3. Metody wykorzystujące dodatkowe informacje o przekazie wideo

Prawie wszystkie wspomniane sieci wykorzystywały mapę QP jako dodatkowa informacja pomagającą uzyskać lepsze rezultaty, natomiast wiele prac skupiło się na wykorzystaniu jeszcze innych dodatkowych informacji, aby uzyskać jak najlepszą poprawę jakości po kompresji.

Kolejną informacją, używaną w celu polepszenia poprawy jakości przez sieci, są dane na temat podziału ramek na bloki, czego przykładem mogą być sieci MMS-net [25] czy BSTN [26] dla HEVC.

W pracy [27] autorzy zaproponowali wykorzystanie charakterystyki GoP (ang. *Group of Pictures*) oprócz mapy QP i rodzaju ramki. Zastosowanie wspomagających informacji pozwoliło na osiągnięcie najlepszych dotychczasowych wyników - 7.31% redukcji przepływności, podczas gdy dla scenariusza bez użycia obrazu predykcyjnego 5.85%. Proponowana architektura poprawiająca jakość po kompresji wykorzystuje bloki rezydualne znane z ResNet [11].

4. Opis zbioru danych

4.1. Charakterystyka wykorzystanego zbioru danych

W pracy skorzystano ze zbioru danych Xiph.org Video Test Media [1]. Jest on szeroko ceniony i wykorzystywany w badaniach nad kompresją danych, oferując bogaty zestaw plików wideo w różnych formatach i rozdzielczościach. Dane te są szczególnie cenne w kontekście badań nad kompresją i dekompresją, gdyż zapewniają reprezentatywne próbki o wysokiej jakości różnorodności.

Większa część plików w zbiorze jest dostępna w formacie YUV420, który jest standardem w przetwarzaniu i kompresji wideo. Jednakże, zestaw zawiera również pliki w innych formatach, takich jak YUV444 i YUV422. Obejmuje również materiały z 10-bitową głębią kolorów.

Istotne w kontekście badania jest to, że zbiór posiada dane w postaci nieskompresowanej. Zatem możliwa jest ich kompresja i dekompresja przy użyciu dowolnego kodeka. W niniejszej pracy skorzystano z VVC w wersji referencyjnej. Dzięki wykorzystaniu nieskompresowanych danych, jako punktu odniesienia, możliwe było dokładne ocenienie skuteczności sieci neuronowych w poprawie jakości wideo po kompresji, co stanowi kluczowy element badania.

4.2. Wybór danych i podział na podzbiory treningowe i walidacyjne

W ramach treningu sieci neuronowych kluczowym aspektem jest normalizacja danych, która zapewnia jednolitość i spójność wejściowych danych treningowych. W niniejszej pracy skupiono się na wykorzystaniu jedynie plików w formacie YUV 420, dostępnych w zbiorze Xiph.org. Ten format został wybrany ze względu na to, że jest on powszechnie używany w transmisjach telewizyjnych, usługach strumieniowania wideo oraz w badaniach nad kompresją.

Zbiór danych Xiph.org, wykorzystany w badaniu, zawiera 59 unikalnych plików w formacie YUV 420. Pliki te charakteryzują się różnorodnością pod względem rozdzielczości i długości, co pozwala na przeprowadzenie wszechstronnych eksperymentów i testów. Wśród rozdzielczości znajdują się formaty takie jak cif, qcif, 480P, 720P, 1080P, 2K i 4K, co umożliwia testowanie sieci na danych o różnych wymaganiach obliczeniowych i szczegółowości obrazu.

Dla celów treningu i walidacji modelu, zbiór został podzielony na dwie grupy: 53 pliki wykorzystano jako sekwencje treningowe, natomiast sześć plików wydzielono jako dane testowe. Pliki wybrano tak, aby reprezentować różne rozdzielczości i typy treści, co pozwala na dokładniejszą ocenę wydajności i skuteczności sieci neuronowej.

Rysunek 4.1 przedstawia po klatce każdej sekwencji testowej. Wybrano je w taki sposób, aby reprezentować różne scenariusze, takie jak: statyczne i dynamiczne sceny, sceny z dużą

4. Opis zbioru danych

i małą ilością ruchu, sceny z dużą i małą ilością szczegółów, sceny z dużą i małą ilością kolorów, itp.



(a) bridge_far_cif



(b) stefan_cif



(c) students_qcif



(d) tractor_1080p25



(e) Johnny_1280x720_60

Rysunek 4.1. Przykładowe klatki sekwencji kontrolnych

Wykorzystanie tego podziału pozwoliło na efektywne trenowanie sieci neuronowej, jednocześnie zapewniając możliwość dokładnej oceny jej wydajności na danych kontrolnych, które odzwierciedlają różnorodność realnych scenariuszy kompresji wideo.

4.3. Format YUV420

Format YUV420 jest szeroko stosowany do reprezentacji przekazu obrazów i wideo. Wykorzystuje chrominancję (kolor) i luminancję (jasność) do opisu kolorów w obrazie, co pozwala na efektywną kompresję danych przy zachowaniu wysokiej jakości wizualnej.

Składowa Y reprezentuje luminancję obrazu, będącą najważniejszym elementem dla percepji człowieka. U i V (Chrominancja) odpowiadają za informacje o kolorze. U reprezentuje składową niebiesko-czerwoną, a V czerwono-zieloną.

W formacie YUV420 stosuje się technikę podpróbkowania 4:2:0, co oznacza, że w pionie i poziomie kolor jest próbowany z połową częstotliwości luminancji. Dzięki temu uzyskuje się redukcję ilości danych, ponieważ ludzkie oko jest mniej wrażliwe na szczegóły koloru niż na szczegóły jasności. Ta metoda pozwala na znaczącą oszczędność przestrzeni przy minimalnej stracie jakości wizualnej.

Format YUV420 jest idealny do zastosowań kompresji wideo, gdyż pozwala na efektywną redukcję rozmiaru pliku przy zachowaniu wysokiej jakości obrazu. Jest powszechnie

używany w różnych standardach kompresji wideo, w tym w popularnych kodekach, takich jak H.264 i HEVC.

Wybór formatu YUV420 w badaniach nad jakością wideo po kompresji jest kluczowy, ponieważ pozwala na realistyczne symulowanie warunków, w jakich większość materiałów wideo jest przetwarzana i rozpowszechniana. Dzięki temu możliwe jest dokładne badanie wpływu różnych technik poprawy jakości przekazu na powszechnie stosowany format danych.

4.4. Przygotowanie danych

Prawidłowe przygotowanie danych to kluczowy krok w procesie uczenia maszynowego, mający ogromny wpływ na skuteczność i wydajność modeli. W kolejnych sekcjach przedstawione zostaną techniki i procesy, które są niezbędne do przetworzenia surowych danych wejściowych w postać użyteczną do treningu, walidacji i testowania modeli sieci neuronowych.

Proces treningu sieci neuronowych na plikach wideo jest bardzo złożony i wymaga dużych zasobów obliczeniowych. W celu zwiększenia jego wydajności i skuteczności, dane wejściowe muszą być odpowiednio przygotowane. W niniejszej pracy skupiono się na dwóch głównych aspektach przygotowania danych: normalizacji i przetwarzaniu wstępny.

4.4.1. Przygotowanie sekwencji wideo

W ramach badania wszystkie pobrane sekwencje wideo zostały zunifikowane pod względem długości, do 64 klatek przy użyciu programu `ffmpeg`. Ta długość została wybrana, aby odpowiadać dwukrotnej pełnej sekwencji ramek w trybie Random Access (RA), który jest jednym z trybów kompresji kodeka VVC. Taka redukcja wielkości plików wideo była konieczna ze względu na ograniczenia sprzętowe, a w szczególności, przez ograniczoną przestrzeń dyskową.

4.4.2. Kodowanie i dekodowanie sekwencji wideo

Sekwencje są kodowane przy użyciu wszystkich badanych kombinacji trybów, filtrów i parametrów QP w wersji referencyjnej kodeka VVC. Proces ten pozwala na wygenerowanie różnorodnych danych po kompresji, odzwierciedlających różne scenariusze i ustawienia kompresji wideo.

Wybrane parametry kompresji W celu dokładnego zbadania wpływu różnych ustawień kodeka VVC na jakość wideo, wybrano specyficzne parametry i konfiguracje do eksperymentów. Parametry kwantyzacji (QP), które mają bezpośredni wpływ na stopień kompresji i jakość obrazu, zostały ustalone na cztery poziomy: 32, 37, 42 i 47. Taki zakres wartości pozwala na zbadanie efektów kodowania w szerokim zakresie, od lżejszej do bardziej intensywnej kompresji.

4. Opis zbioru danych

Dodatkowo skupiono się na dwóch głównych trybach profilach: Random Access (RA) i All Intra (AI). Tryb RA jest zorientowany na efektywną kompresję w scenariuszach z losowym dostępem do ramki, co jest typowe dla transmisji i strumieniowania wideo. Z kolei tryb AI skupia się na kodowaniu każdej klatki wideo niezależnie od innych, wykorzystując tylko metody przetwarzania wewnętrzkalatkowego, bez odwoływanego się do informacji z innych klatek. Stosuje się go często w scenariuszach wysokiej jakości archiwizacji wideo.

Aby uzyskać pełny obraz wpływu różnych technik filtracji w kodeku VVC na jakość przekazu, przeprowadzono eksperymenty z wszystkimi kombinacjami włączonych i wyłączonego filtrów: Deblocking Filter (DB), Sample Adaptive Offset (SAO) oraz Adaptive Loop Filter (ALF). Każdy z tych filtrów służy do redukcji określonych typów artefaktów kompresji i może znaczaco wpływać na percepcyjną jakość. Przeanalizowanie wszystkich kombinacji tych filtrów pozwoliło na zrozumienie, jak każdy z nich indywidualnie, jak i w kombinacji, wpływa na utratę informacji przy kodowaniu.

Poprzez ten szczegółowy dobór parametrów kwantyzacji, trybów kodowania i filtracji, badanie oferuje kompleksowe spojrzenie na różne aspekty wpływające na jakość wideo po kompresji kodekiem VVC, co jest kluczowe dla opracowania efektywnych metod jej poprawy.

W ramach pracy badano kombinacje następujących parametrów:

- tryby kodowania: Random Access (RA) i All Intra (AI)
- filtry: Deblocking Filter (DB), Sample Adaptive Offset (SAO) i Adaptive Loop Filter (ALF)
- parametr QP: 32, 37, 42 i 47

Podsumowując, dla każdej sekwencji wygenerowano 64 pliki po kompresji, które są następnie dekodowane do postaci YUV420. Odpowiada to kombinacji 2 trybów kodowania, 3 filtrów i 4 parametrów QP, co daje 64 możliwych zestawów parametrów kompresji. W efekcie kodowania kodekiem VVC powstaje łącznie 3776 plików wideo po kompresji, które są następnie dzielone na zbiór 3392 treningowych oraz 384 walidacyjnych.

Proces kodowania i dekodowania kodekiem VVC W celu kompresji oraz dekomprezji danych użyto kodu VVC w wersji referencyjnej[2]. Pozwoliło to na wykorzystanie gotowych plików konfiguracyjnych definiujących parametry kompresji, takie jak: tryb kodowania, parametr QP, filtry, itp. Fragment przykładowego pliku konfiguracyjnego, ilustrujący najistotniejsze zmienne, został przedstawiony na listingu 1. Zarówno dla konfiguracji trybu AI, jak i RA skorzystano z przykładowych konfiguracji załączonych do referencyjnego kodu, zmieniając jedynie potrzebne parametry, takie jak użycie poszczególnych filtrów.

Listing 1. Fragment przykładowego pliku konfiguracyjnego kodu VVC, profil AI z QP 32 z wyłączeniami wszystkimi filtrami

```
1 ===== Coding Structure =====
2 IntraPeriod :1 #Period of I-Frame (-1 =only first)
3
```

```

4 | ====== Quantization ======
5 | QP :32 #Quantization parameter(0–51)
6 |
7 |
8 | # General
9 | CTUSize :64
10 |
11 | # Tools configuration
12 | ALF :0 #Adaptive Loop Filter: 0: disabled, 1: enabled
13 | SAO :0 #Sample adaptive offset, 0: disabled, 1: enabled
14 |
15 | ====== Deblock Filter ======
16 | LoopFilterDisable :1 #Disable deblocking filter (0=Filter, 1=No Filter)

```

Po przygotowaniu plików konfiguracyjnych dla każdej możliwej kombinacji parametrów, proces kodowania i dekodowania sekwencji wideo został zautomatyzowany przy użyciu skryptów bash. Skrypty te wykorzystują skompilowany wcześniej koder i dekoder w wersji referencyjnej kodeka VVC w celu kompresji sekwencji wideo, a następnie dekompresji zakodowanych danych. Fragment przykładowego skryptu kodującego sekwencję wideo został przedstawiony na listingu 2. Warto zaznaczyć, że wartość QP można przekazać jako parametr przy wywołaniu programu kompresującego, dzięki czemu liczba potrzebnych plików konfiguracyjnych dla eksperymentu wynosi jedynie 16.

Listing 2. Przykładowa komenda kodująca sekwencję wideo

```

1 | ./vvenc/bin/release-static/vvencFFapp \
2 | -c ${CONFIGFILE} \
3 |   --InputFile=$FILE \
4 |   --BitstreamFile=$ODIR/$DESTINATION \
5 |   --ReconFile=$ODIR/$RECON_FILE \
6 |   --FrameRate $FRAMERATE \
7 |   --FramesToBeEncoded $END_FRAME \
8 |   --SourceWidth $WIDTH \
9 |   --SourceHeight $HEIGHT \
10 |   --QP=$QP \
11 | >$ODIR/$LOG_FILE

```

Dane kompresowane są jednowątkowo, w związku z tym wykorzystano specjalnie przygotowany skrypt w języku Python w celu umożliwienia uruchomienia wielu instancji dowolnego skryptu jednocześnie, analogicznie do GNU Parallel [28], z tą różnicą, że przygotowany skrypt zapisywał listę ukończonych oraz nieukończonych zadań przy przerwaniu. Dzięki temu, w przypadku przerwania procesu, możliwe jest wznowienie pracy od ostatniego zakończonego zadania. Pętlę główną skryptu przedstawiono na listingu 3.

Listing 3. Pętla główna programu do równoległego wykonywania zadań

4. Opis zbioru danych

```
1 print("Spawning_processes...")
2
3 for _ in range(self.count_limit):
4     self.spawn_next()
5
6 print("Entering_loop...")
7
8 while self.running():
9     try:
10         done_processes = [
11             p for p in self.active_processes if p.poll() is not None
12         ]
13
14         self.active_processes = [
15             p for p in self.active_processes if p not in done_processes
16         ]
17
18         for p in done_processes:
19             if p.returncode != 0:
20                 self.failed.append(p.args)
21             else:
22                 self.done.append(p.args)
23                 self.spawn_next()
24
25             time.sleep(1)
26             if done_processes:
27                 print(f"Total_done_after_this_iteration: {len(self.done)}")
28                 print(f"Total_left_after_this_iteration: {len(self.jobs)}")
29         except KeyboardInterrupt:
30             for p in self.active_processes:
31                 os.killpg(os.getpgid(p.pid), signal.SIGTERM)
32                 self.failed.append(p.args)
33                 self.active_processes = []
34             break
```

Po procesie kodowania, zakodowane sekwencje są dekodowane, aby umożliwić dalszą analizę i porównanie z oryginalnymi danymi. Analogicznie do procesu kodowania, proces dekodowania został zautomatyzowany przy użyciu równolegle wykonywanych skryptów bash. Fragment przykładowego skryptu dekodującego sekwencję video został przedstawiony na listingu 4.

Listing 4. Przykładowa komenda kodująca sekwencję video

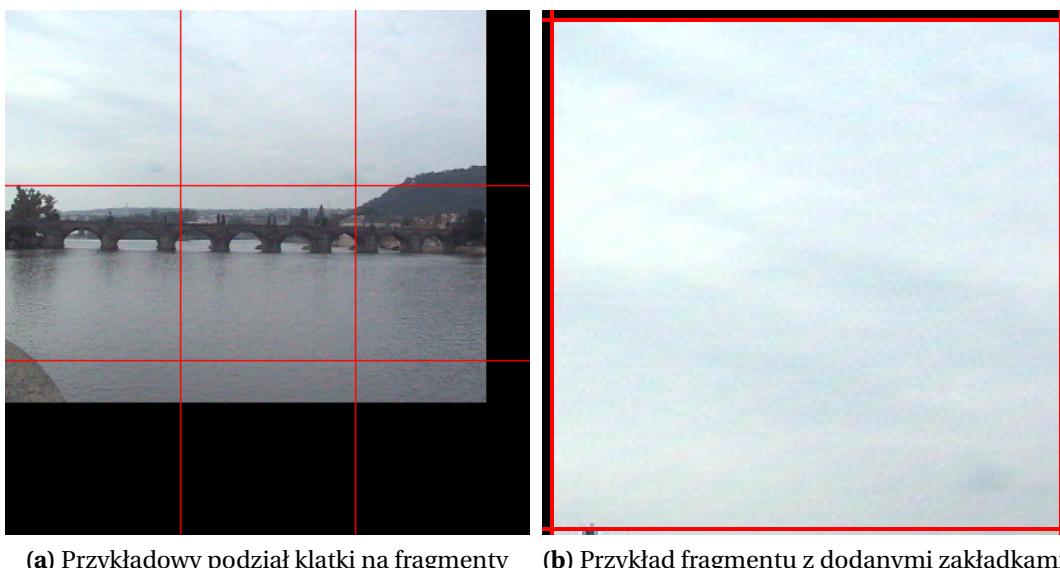
```
1 ./vvdec/bin/release-static/vvdecapp \
2 -b ${ENCODED_PATH} \
3 -o ${DECODED_PATH} \
```

```
4 > ${DECODED_PATH}.log
```

4.4.3. Normalizacja danych

Aby móc przetwarzać dane w sieciach neuronowych, konieczne jest odpowiednie przygotowanie danych wejściowych. W przypadku wideo w formacie YUV420, konieczna jest konwersja do YUV444. Podobnie do istniejących rozwiązań, takich jak VVCNN [22], skorzystano w tym celu z narzędzia `ffmpeg`, wykonującego skalowanie składowych U i V przy pomocy interpolacji liniowej.

Następnie pliki wideo podzielono na pojedyncze klatki, a następnie na fragmenty o stałym rozmiarze 132 na 132 pikseli - 128 na 128 pikseli plus piksele obszarów sąsiadujących (zakładki). Taki rozmiar został wybrany, aby zoptymalizować proces przetwarzania danych, jednocześnie zapewniając wystarczającą ilość informacji dla efektywnej analizy i treningu sieci neuronowej, ponieważ odpowiada wielkości jednego CTU kodera VVC. Dodatkowo, zastosowanie stałego rozmiaru niweluje problem różnych rozdzielczości plików wideo na etapie treningu sieci neuronowych. Przykładowy podział ramki przedstawiono na rysunku 4.2a przy pomocy czerwonych linii w miejscach podziału.



Rysunek 4.2. Metodologia normalizacji danych poprzez podział na fragmenty o stałym rozmiarze

Zastosowano technikę tworzenia zakładek, dodając zewnętrzne 2 piksele każdego fragmentu sąsiadującego, co pozwala na lepsze zachowanie kontekstu przestrzennego i redukcję efektów brzegowych podczas przetwarzania obszarów przez sieć neuronową. Przykład został przedstawiony na rysunku 4.2b. Kolorem czerwonym oznaczono linie podziału ramki. Przedstawiona część klatki znajduje się w lewym górnym roku, zatem zakładki górna i lewa wypełnione są zerami.

4. Opis zbioru danych

Struktura danych treningowych Każdy fragment wideo posiada w ścieżce pliku zakodowane informacje na temat użytego profilu kodowania VVC, rodzaju ramki (intra czy inter), wartości parametru QP, zastosowanych filtrów, numeru klatki oraz lokalizacji w obrębie oryginalnego wideo. Ta metoda nomenklatury plików ułatwia identyfikację i selekcję danych do analizy i treningu modelu. Przykładowo, prawa dolna część pierwszej ramki pliku wideo bridge_far_cif zakodowana profilem AI z wyłączonymi wszystkimi filtrami przy $QP = 32$ znajduje się w folderze z walidacyjnymi danymi po kompresji pod lokalizacją bridge_far_cif/AI_QP32_ALF0_DB0_SA00/0_True/256_256_rb.

Dane testowe W procesie przygotowania danych do testowania modeli sieci neuronowych podjęto decyzję o przechowywaniu klatek wideo w ich pełnych formatach, zamiast dzielenia ich na mniejsze fragmenty. Ta strategia ma kluczowe znaczenie dla zapewnienia spójności i jakości końcowych wyników. Przetwarzając ramki w całości, unika się konieczności ponownego składania obrazu z poszczególnych segmentów podczas fazy predykcji, co mogłoby potencjalnie prowadzić do niespójności i utraty jakości, szczególnie na granicach sklejanych fragmentów.

Zachowanie integralności całych ramek w trakcie testowania modelu pozwala na bardziej precyzyjną i rzetelną ocenę jego skuteczności. Eliminuje to potencjalne komplikacje wynikające z procesu rekonstrukcji, zapewniając, że wyniki predykcji odzwierciedlają rzeczywistą skuteczność modelu w poprawie jakości pełnych klatek wideo. Ta metoda nie tylko upraszcza proces testowania, ale również zapewnia wiarygodność wyników, co jest kluczowe w kontekście rzetelnej oceny i wdrażania zaawansowanych technologii przetwarzania obrazów.

5. Metodologia rozwiązania

Celem tej sekcji jest przedstawienie ogólnej metodyki, która została zastosowana w celu osiągnięcia głównych celów niniejszej pracy. Zakłada ona zastosowanie sieci trenowanych w architekturze generatywnych sieci przeciwnych (ang. Generative Adversarial Networks), z wykorzystaniem informacji na temat profilu kodeka, QP, użytych filtrów oraz rodzaju poprawianej ramki do poprawy jakości wideo po kompresji z użyciem kodeka VVC.

W pracy przyjęto podejście eksperymentalne, które pozwala na empiryczne testowanie wydajności badanych sieci w kontekście poprawy jakości wideo. Eksperymenty te zostały zaprojektowane w taki sposób, aby dostarczyć wiarygodnych i porównywalnych wyników, które mogą być wykorzystane do wnioskowania o skuteczności poszczególnych metod.

W pierwszym etapie badania przetestowano sieci o różnych głębokościach i liczbach parametrów. W tym celu zastosowano sieci o różnej liczbie warstw oraz różnej liczbie filtrów na każdej warstwie. Dzięki temu można było określić, która konfiguracja sieci jest najbardziej efektywna w zadaniu poprawy jakości wideo po kompresji. Następnie dokonano porównań najskuteczniejszych modeli konwolucyjnych, resztowych i gęstych.

W drugim etapie badania skupiono się na wykorzystaniu architektury generatywnych sieci przeciwnych w zadaniu poprawy jakości wideo po kompresji. W tym celu zastosowano sieć GAN, w której generator stanowiła wybrana na podstawie wyników sieć z pierwszego etapu (trenowana od zera), a dyskryminator - sieć DenseNet. Pozwoliło to na określenie, czy wykorzystanie sieci GAN w zadaniu poprawy jakości wideo po kompresji jest skuteczne.

W pracy przedstawiono także metodykę oceny wyników, co obejmuje wykorzystanie standardowych metryk jakości wideo i analizę statyczną. To podejście ma na celu zapewnienie, że wnioski wyciągnięte z badań są obiektywne i oparte na solidnych danych.

5.1. Opis wykorzystanych sieci neuronowych

W ramach niniejszej pracy skupiono się na porównaniu różnych architektur sieci neuronowych w kontekście poprawy jakości wideo po kompresji kodekiem VVC: sieci ResNet, DenseNet oraz tradycyjnej konwolucyjnej. Każda z tych architektur została wybrana ze względu na swoje unikalne cechy i potencjalną skuteczność w zadaniu poprawy jakości obrazu.

5.1.1. Wspólne cechy badanych sieci neuronowych

Istniejące rozwiązania udowodniły, że sieci neuronowe są w stanie poprawić jakość danych po kompresji. W szczególności architektury takie, jak QE-CNN [17] czy CNNF [16], są w stanie znacznie poprawić wideo po kompresji, a mała liczba ich parametrów sprawia, że potencjalnie można by wykorzystać je na odbiornikach o wystarczającej mocy obliczeniowej. Dlatego w pracy skupiono się na zbadaniu różnych modeli o możliwie

5. Metodologia rozwiązania

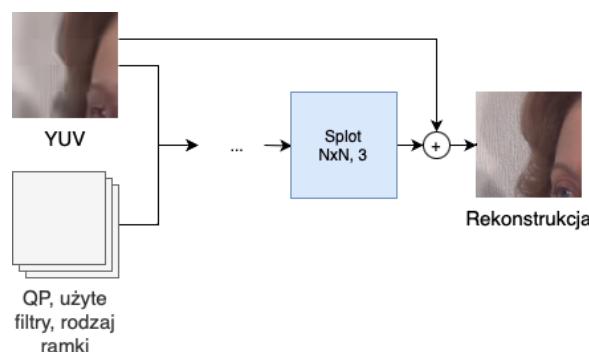
małej liczbie parametrów. Wybrana architektura powinna skutecznie poprawiać jakość wideo po kompresji w akceptowalnym czasie.

Wejście sieci stanowią fragmenty 132 na 132 piksele ramki po dekompresji: 128 na 128 piksele odpowiadających wielkości CTU oraz po 2 piksele z sąsiadujących CTU. Dodatkowo na wejściu podawane są parametry kompresji: QP, użyte filtry, rodzaj ramki (intra/inter) oraz profil kodowania (RA/AI). Te informacje mają za zadanie nie tylko ułatwić zadanie poprawy jakości, prowadząc do potencjalnie lepszych wyników, ale również mogą pomóc w wyszkoleniu sieci uniwersalnej, którą można wykorzystać do plików skompresowanych przy użyciu różnych parametrów i profili kodowania.

Pomimo wykorzystania jedynie plików w formacie YUV420 w procesach szkolenia, walidacji i testowania, jako dane wejściowe wykorzystane są fragmenty klatek w formacie YUV444. Wynika to ze specyfiki przetwarzania modeli konwolucyjnych, które spodziewają się sygnału o stałym rozmiarze. Z tego samego powodu przekazywane informacje dodatkowe skalowane są do rozdzielczości 132 na 132 piksele przy pomocy interpolacji liniowej, czego implementacja została pokazana w rozdziale 7.1.4, natomiast sposób konwersji wideo do formatu YUV444 oraz podziału na obszary w rozdziale 4.4.3.

Wyjście ostatniej warstwy tworzy mapę poprawek. W ten sposób uzyskiwana jest rekonstrukcja ramki po poprawie jakości, stanowiąca wyjście sieci. Zastosowanie mapy poprawek do zdekodowanych ramek pozwoliło znacznie poprawić rezultaty, co przedstawiono w rozdziale 6.1.1 poświęconym problemom napotkanym w trakcie treningu. W kolejnych podrozdziałach przedstawiono szczegółowe informacje na temat architektur sieci, które zostały wykorzystane w badaniu.

Ostatnia warstwa konwolucyjna z trzema filtrami, jak i charakterystyka danych wejściowych oraz wyjściowych, stanowią cechy wspólne wszystkich badanych sieci, co zostało przedstawione na rysunku 5.1. Trzema kropkami oznaczono część sieci różną dla modeli.



Rysunek 5.1. Diagram wspólnej części architektur sieci użytych do poprawy jakości

5.1.2. Sieci ResNet(**ang. Residual Networks**)

Połączenia resztkowe sieci ResNet są realizowane poprzez dodawanie wejścia do wyjścia każdego bloku. W ten sposób informacje z poprzednich warstw są lepiej propagowane

do kolejnych, co pozwala na efektywniejsze przetwarzanie i przeciwdziałanie problemowi zanikającego gradientu. Prowadzi to w teorii do lepszej efektywności trenowanych parametrów, umożliwiając wykorzystanie głębszych sieci oraz uzyskanie lepszych wyników wzmacnienia PSNR i SSIM w stosunku do sieci konwolucyjnych bez połączeń resztkowych.

Jako punkt wyjściowy wybrano architekturę CNNF [16], osiągającą dobre wyniki w zakresie poprawy jakości wideo po kompresji kodekiem HEVC. Warstwy konwolucyjne tej sieci zastąpiono warstwami resztkowymi (oprócz ostatniej) oraz zastosowano funkcję aktywacji PReLU [29] w celu poprawy stabilności treningu. Ten wybór uzasadniono w rozdziale 6.2.1 poświęconemu rozwiązańom problemów napotkanych przy szkoleniu generatywnych sieci GAN. Następnie stopniowo zwiększano głębokość sieci oraz liczbę filtrów na każdej warstwie, aż do momentu, gdy dalsze dodawanie trenowanych parametrów nie przynosiło już wyraźnej poprawy w wynikach. Taki punkt nasycenia był kluczowy w określeniu ostatecznej struktury sieci, pozwalając na wybór konfiguracji, która nie tylko efektywnie radzi sobie z zadaniem poprawy jakości wideo po kompresji, ale także jest wydajna obliczeniowo.

Na podstawie eksperymentów wybrano sieć ResNet o siedmiu warstwach, a więc o takiej samej liczbie, jak w przypadku sieci CNNF. Zwiększyły natomiast liczbę filtrów na wszystkich warstwach do wartości przedstawionych w tabeli 5.1. Parametry te zostały dobrane eksperymentalnie na podstawie wyników uzyskanych przez sieci o różnych głębokościach i liczbie filtrów na warstwach. Kolejne warstwy resztkowe oznaczono jako R_i , ostatnia warstwa (C_7) nie posiada połączenia resztkowego, ponieważ na jej wyjściu jest trójkanałowa mapa poprawek. Sieć ma 989707 trenowanych parametrów.

Warstwa	R_1	R_2	R_3	R_4	R_5	R_6	C_7
Liczba filtrów	64	64	96	64	48	32	3
Rozmiar jądra splotu	9×9	7×7	5×5	3×3	3×3	3×3	3×3
Liczba parametrów	21249	401666	390722	98690	51746	24770	864

Tabela 5.1. Liczba filtrów oraz rozmiar jądra splotu na warstwach sieci ResNet

5.1.3. Sieci DenseNet (ang. Densely Connected Convolutional Networks)

Połączenia między warstwami gęstymi są realizowane poprzez konkatenację wyjść. W ten sposób każda warstwa otrzymuje na wejściu informacje ze wszystkich poprzednich. Połączenia gęste realizują zatem funkcję podobną do połączeń resztkowych - przekazywanie informacji z poprzednich warstw sieci do warstw kolejnych.

Analogicznie do poprzedniego eksperymentu, proces dobioru architektury sieci DenseNet przebiegał poprzez zwiększanie liczby warstw, bloków oraz modyfikowaniu współczynnika przyrostu, czyli liczby filtrów na każdej warstwie. Zdecydowano się również wykorzystać inne cechy modeli gęstych, oprócz samych funkcji skrótu, takich, jak warstwy wąskiego gardła, struktury blokowej o stałym współczynniku przyrostu (*gr*, ang. *growth*

5. Metodologia rozwiązania

rate), czyli liczba filtrów kolejnych warstw, i warstw przejścia. Dokonano modyfikacji warstw przejścia, aby móc korzystać z funkcjonalności redukcji liczby cech informacji pomiędzy blokami bez zmiany jej wymiarowości. W tym celu pozostawiono jedynie konwolucję o kernelu 1×1 , czyli pominięto operację average pooling. Podobnie jak w przypadku ResNet, zastosowano funkcję aktywacji PReLU w celu poprawy stabilności treningu GAN.

Jako punkt wyjściowy do wyboru odpowiedniej architektury sieci posłużył model podobny do sieci ResNet, czyli sieć o siedmiu warstwach, gdzie odpowiednio warstwy zostały zastąpione blokami gęstymi. Następnie stopniowo zwiększano głębokość sieci oraz liczbę filtrów na każdej warstwie, obserwując przebieg treningu i wyniki. W ten sposób można było określić, która konfiguracja sieci jest najbardziej efektywna w zadaniu poprawy jakości wideo po kompresji. Jako, że nie zaobserwowano znaczących różnic wyników dla zwiększania głębokości sieci lub liczby filtrów, otrzymane modele gęste mają takie same liczby filtrów w kolejnych blokach, co sieć resztkowa w kolejnych warstwach. Dzięki temu można porównać wpływ samych rozwiązań znanych z sieci DenseNet i ResNet na otrzymane wyniki.

W podobny sposób, eksperymentalnie, wybrano odpowiedni współczynnik bn , czyli mnożnika liczby filtrów na warstwach typu wąskie gardło (ang. *bottleneck*), oraz gr . Użytkowane wartości wynoszą 1.5 dla bn oraz 16 dla gr .

Liczبę warstw w kolejnych blokach oraz ich współczynnik przyrostu proponowanego modelu gęstego przedstawiono w tabeli 5.2. Podobnie, jak w przypadku sieci ResNet, kolejne bloki oznaczono jako D_i , a ostatnia warstwa (C_7) nie posiada połączenia gęstego, ponieważ na jej wyjściu jest trójkanałowa mapa poprawek. W pierwszym bloku sieci, odpowiedzialnym za wstępную ekstrakcję cech z sygnału wejściowego, zastosowano pojedynczą warstwę. Proponowana sieć ma 342239 parametrów.

Warstwa	D_1	D_2	D_3	D_4	D_5	D_6	C_7
Liczba warstw	1	4	6	4	3	2	1
Wsp. przyrostu / liczba filtrów	64	16	16	16	16	16	3
Rozmiar jądra splotu	9×9	7×7	5×5	3×3	3×3	3×3	3×3
Liczba parametrów	20865	121365	111025	42849	29887	13764	2484

Tabela 5.2. Architektura proponowanej sieci DenseNet

5.1.4. Sieci konwolucyjne

Jako, że sieci konwolucyjne są najbardziej podstawowym rodzajem sieci neuronowych, zdecydowano się na zbadanie ich skuteczności w zadaniu poprawy jakości wideo po kompresji. W tym celu zastosowano model o architekturze podobnej do ResNet, jednak bez połączeń resztkowych. W ten sposób można było określić, czy połączenia resztkowe i gęste są kluczowe w zadaniu poprawy jakości wideo po kompresji. Podobnie jak w

przypadku ResNet i DenseNet, zastosowano funkcję aktywacji PReLU w celu poprawy stabilności treningu GAN.

Liczę filtrów oraz rozmiar ich jądra na kolejnych warstwach przedstawiono w tabeli 5.3. Sieć ma 473414 trenowanych parametrów.

Warstwa	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Liczba filtrów	64	64	96	64	48	32	3
Rozmiar jądra splotu	9×9	7×7	5×5	3×3	3×3	3×3	3×3
Liczba parametrów	20865	200833	153793	55425	27745	13889	864

Tabela 5.3. Liczba filtrów oraz rozmiar jądra splotu na warstwach sieci konwolucyjnej

5.1.5. GAN - Generatywne sieci przeciwnostawne

Generatywne sieci przeciwnostawne (GAN) mogą mieć kilka istotnych zastosowań w zakresie poprawy jakości wideo po kompresji. W szczególności, sieci GAN mogą zostać użyte poprzez wykorzystanie generatora do generowania rekonstrukcji, a dyskryminatora do oceny jej jakości.

Jako generatora wykorzystano sieć typu DenseNet bez stałego współczynnika przyrostu, opisaną w poprzednim podrozdziale. Została ona wybrana na podstawie wyników pierwszego etapu badania przedstawionych w rozdziale 8.

Dyskryminator Odpowiedni dobór architektury sieci dyskryminatora okazał się kluczowy. Specyfika eksperymentu wymagała, aby sieć dyskryminatora była w stanie rozpoznawać artefakty kompresji, natomiast w zbiorze danych na każdy fragment ramki przed zakodowaniem kodikiem VVC przypada 64 kombinacji różnych parametrów kompresji. Powoduje to dysproporcję w treningu dyskryminatora, polegającą na tym, że przy pełnym przejściu jednej epoki treningu w metodologii GAN, sieć dyskryminatora będzie optymalizowana pod kątem rozpoznawania danego oryginalnego fragmentu aż 64 razy - każdy pakiet danych jest bowiem zestawem fragmentu po kompresji oraz fragmentu oryginalnego, a ten sam fragment oryginalny jest załączany do 64 różnych fragmentów po kodowaniu.

Jako dyskryminator wykorzystano sieć wzorowaną na DenseNet-121, czyli o najmniejszej liczbie parametrów spośród modeli proponowanych przez autorów modelu gęstego [12], z współczynnikiem przyrostu ustawionym na 8. Również dla tego modelu zastosowano funkcję aktywacji PReLU. W trakcie eksperymentów proponowana architektura pozwalała na uzyskanie stabilnego uczenia w metodologii GAN, natomiast trenowana rozdzielnie od generatora nie wykazała znaczących problemów takich, jak nadmierne dopasowanie (ang. *overfit*). Liczba warstw kolejnych bloków została przedstawiona w tabeli 5.4. Poprzez D_i oznaczono bloki gęste, C_i konwolucyjne, a FC_i w pełni połączone (ang. *fully-connected*). Sieć ma 252671 trenowanych parametrów.

5. Metodologia rozwiązania

Blok	C_1	D_2	D_3	D_4	D_5	FC_6
Liczba warstw	1	6	12	24	16	1
Wsp. przyrostu / liczba filtrów	64	8	8	8	8	1
Rozmiar jądra splotu	7×7	3×3	3×3	3×3	3×3	-
Liczba parametrów	4769	15485	39937	124349	67872	259

Tabela 5.4. Architektura sieci DenseNet-121

Wyjściem sieci dyskryminatora jest wartość, określająca prawdopodobieństwo, że dany fragment jest oryginalny. Jest ona wykorzystywana jako funkcja straty w metodologii GAN, co pozwala na trenowanie generatora w celu poprawy jakości wideo po kompresji. Sieć uczona jest, aby w przypadku obszarów oryginalnych na wyjściu produkowała 1, natomiast w przypadku fragmentów po poprawie jakości 0.

Warto zaznaczyć, że testowano też głębsze sieci gęste, natomiast w przypadku ich zastosowania na etapie treningu rozdzielonego można było zauważać znaczne nasilenie się problemu nadmiernego dopasowania. Oznaczało to, że sieci były w stanie nauczyć się rozpoznawać fragmenty oryginalne ze zbioru treningowego na pamięć, tracąc zdolność generalizacji.

5.2. Parametryzacja i optymalizacja

Optymalizacja i parametryzacja są kluczowymi elementami procesu trenowania sieci neuronowych, ponieważ mają bezpośredni wpływ na skuteczność i efektywność uczenia się modelu. Prawidłowe ustawienie parametrów, takich jak szybkość uczenia (ang. *learning rate*), rozmiar partii (ang. *batch size*), czy wybór optymalizatora, decyduje o tym, jak szybko i efektywnie sieć jest w stanie nauczyć się złożonych wzorców zawartych w danych. Zbyt wysoka szybkość uczenia może prowadzić do niestabilności i przeskakiwania optymalnych rozwiązań, podczas gdy zbyt niska grozi zatrzymaniem się w minimach lokalnych. Równie istotna jest architektura modelu, w tym liczba warstw i neuronów, które muszą być dostosowane do specyfiki zadania. Nieodpowiednia parametryzacja może prowadzić do problemów takich, jak przeuczenie (ang. *overfitting*) lub niedouczenie (ang. *underfitting*), a także do nieefektywnego wykorzystania zasobów obliczeniowych. Optymalizacja procesu uczenia poprzez staranne dobieranie i dostosowywanie parametrów jest zatem kluczowa dla osiągnięcia wysokiej jakości modeli, które są zarówno precyzyjne, jak i efektywne.

5.2.1. Funkcja straty

Ze względu na charakter eksperymentu, konieczny był wybór trzech funkcji straty.

W pierwszym etapie, gdzie sieci trenowane były bez wykorzystania metodologii GAN, jako funkcję straty zastosowano złożenie metryk: strukturalnego podobieństwa (*SSIM*, ang. *structural similarity*), średnio kwadratowego strukturalnego podobieństwa (*MS* –

SSIM, ang. *mean squares structural similarity*), funkcji straty $l1$ oraz $l2$. Współczynniki wagowe tych metryk zostały dobrane eksperymentalnie. Kompletna funkcja straty dla tego etapu została przedstawiona na równaniu 1.

$$l = 0.1 * MS - SSIM_loss + 0.1 * SSIM_loss + 0.5 * l1 + 0.3 * l2 \quad (1)$$

Zastosowanie złożenia wielu metryk pozwoliło na optymalizację wyników pod wieloma kątami. Metryki $MS - SSIM$ oraz $SSIM$ są szczególnie przydatne w zadaniu poprawy jakości wideo po kompresji, ponieważ pozwalają na ocenę strukturalnego podobieństwa wideo w sposób zbliżony do ludzkiego postrzegania. Metryki $l1$ oraz $l2$ pozwalają na ocenę subtelnych różnic pikselowych pomiędzy obrazami.

W drugim etapie badania, gdzie wykorzystano metodologię GAN, jako funkcję straty zastosowano binarną entropię skośną (BCE , ang. *binary cross entropy*). Na równaniu 2 przedstawiono pełną funkcję straty dyskryminatora, oznaczając jego predykcję dla fragmentów oryginalnych jako *real*, natomiast po poprawie jakości, jako *fake*.

$$\begin{aligned} l_{d_real} &= BCE(real, 1) \\ l_{d_fake} &= BCE(fake, 0) \\ l_d &= (l_{d_real} + l_{d_fake}) / 2 \end{aligned} \quad (2)$$

Jako funkcję straty w drugim etapie wykorzystano dla generatora złożenie tych samych metryk, co dla treningu rozdzielonego, rozszerzonych o funkcję straty wynikającą z predykcji dyskryminatora dla fragmentów po poprawie jakości l_{dg} (równanie 3).

$$l_{dg} = BCE(fake, 1) \quad (3)$$

Współczynniki wagowe tych metryk zostały dobrane eksperymentalnie. Kompletna funkcja straty przedstawiono na równaniu 4.

$$l_g = 0.08 * MS - SSIM_loss + 0.08 * SSIM_loss + 0.45 * l1 + 0.29 * l2 + 0.1 * l_{dg} \quad (4)$$

5.2.2. Techniki optymalizacja uczenia

Szczególną uwagę zwrócono na optymalizację procesu uczenia. W tym celu zastosowano szereg metod, które miały na celu poprawę stabilności oraz zapobieganie zatrzymywaniu się w lokalnym minimum.

Współczynnik uczenia Wykorzystano mechanizm regulacji współczynnika uczenia, pozwalającą na dynamiczne dostosowanie współczynnika uczenia w trakcie treningu. W szczególności zastosowano metodę redukcji tego współczynnika przy zbyt długim okresie zastoju, co pozwoliło na zmniejszenie współczynnika uczenia w przypadku braku poprawy

5. Metodologia rozwiązania

wyników przez określoną liczbę epok. W ten sposób można było zapewnić, że proces uczenia nie zatrzyma się w lokalnym minimum, a sieć będzie w stanie osiągnąć lepsze wyniki.

Dodatkowo wykorzystano też manualną regulację współczynnika uczenia, która pozwala na odgórne planowanie zmian współczynnika uczenia w trakcie treningu, gdyby istniała taka potrzeba. Skorzystano z tej opcji jedynie przy treningu w metodologii GAN.

Rozmiar partii Podczas, gdy w przypadku oddzielnego treningu sieci neuronowych rozmiar partii może nie wywierać istotnego wpływu na wyniki procesu uczenia, w kontekście treningu sieci GAN jego rola staje się znacząca. W metodologii GAN rozmiar partii ma bezpośredni wpływ na stabilność i efektywność treningu. Wybór mniejszych rozmiarów partii często prowadzi do stabilniejszych i bardziej kontrolowanych procesów treningowych, co jest kluczowe w kontekście delikatnej równowagi pomiędzy siecią generującą (generatorem) a dyskryminującą (dyskryminatorem). Chociaż mniejsze partie mogą wydłużać czas potrzebny na przeszkolenie modelu, to przyczyniają się one do zwiększenia precyzji i jakości końcowego wyniku, poprzez bardziej skuteczną regulację i dostosowanie procesu uczenia. Taki podejście jest zatem istotnym elementem strategii optymalizacyjnych w treningu sieci GAN, gwarantującym lepszą równowagę i stabilność w procesie szkolenia modelu.

W ramach badań przeprowadzonych w pracy dokonano eksperymentalnego doboru rozmiaru partii, przy czym zastosowano różne wielkości w zależności od etapu. W fazie treningu GAN najlepsze wyniki osiągnięto dla rozmiaru partii wynoszącego 8, co pozwoliło na skuteczne uczenie się modelu, przy zachowaniu odpowiedniej stabilności i wydajności. W przypadku walidacji i treningu rozłącznego, wybrano strategię wykorzystania największego możliwego rozmiaru partii, czyli 64, co umożliwiało szybsze przetwarzanie i efektywniejszą ocenę modelu.

Natomiast w procesie predykcji rozmiar partii został dostosowany do specyfiki przetwarzanych danych. W sytuacji, gdy ramki analizowane we fragmentach o wymiarach 128x128 pikseli, zastosowano rozmiar partii 64. Z kolei, gdy klatki były poprawiane w całości, zdecydowano się na użycie rozmiaru partii równego 1. Takie podejście pozwala na zachowanie integralności i wysokiej jakości obrazu, natomiast uniemożliwia zastosowanie większych partii, ze względu na wysokie rozdzielczości danych oraz ich zróżnicowanie.

Podsumowując, dokonany dobór rozmiaru partii odzwierciedla zróżnicowane potrzeby poszczególnych etapów procesu uczenia i predykcji, pozwalając na optymalizację wydajności oraz jakości wyników w różnych scenariuszach przetwarzania danych.

Przerywanie i wznowianie treningu Zastosowano metodę przerywania i wznowiania treningu, która pozwala na zatrzymanie go w dowolnym momencie i wznowienie w późniejszym czasie. W szczególności, zapewniono zapisanie się wag sieci, numeru epoki oraz

wszystkich innych atrybutów procesu szkolenia, w przypadku próby przerwania procesu (np. poprzez sygnał SIGKILL).

Wykorzystano metodę wczesnego przerywania (ang. *early stopping*), która pozwala na zatrzymanie treningu w sytuacji braku poprawy wyników przez określoną liczbę epok. W ten sposób można było zapobiec zatrzymywaniu się w lokalnym minimum, a sieć była w stanie osiągnąć lepsze wyniki.

Selektywne trenowanie sieci w metodologii GAN Trening sieci GAN jest szczególnie narażony na brak stabilności, czyli równowagi pomiędzy dyskryminatorem a generatorem. Jednym ze sposobów na poprawę stabilności szkolenia jest selektywne trenowanie sieci. Polega ono na wyłączaniu treningu jednej z sieci, gdy zacznie ona osiągać o wiele lepsze wyniki od drugiej, destabilizując proces treningu.

Zdecydowano się na zastosowanie podejścia, w którym, w przypadku, kiedy wartość funkcji straty wynikającej z predykcji dyskryminatora l_d jest mniejsza, niż zadana wartość graniczna (określana w konfiguracji eksperymentu), trening dyskryminatora był wyłączany, aż do wyszkolenia sieci generatora w takim stopniu, aby l_d znowu znalazło się w zakresie powyżej wartości progowej.

Analogicznie, w przypadku kiedy wartość funkcji straty wynikającej z predykcji dyskryminatora dla fragmentów po poprawie jakości l_{dg} była wyższa niż zadana wartość graniczna (określana w konfiguracji eksperymentu), trening generatora wyłączano, aż do wyszkolenia sieci dyskryminatora w takim stopniu, aby l_{dg} znowu znalazło się w zakresie poniżej wartości progowej.

Selektywne wyłączanie trenowania sieci zależało od uśrednionej wartości tych funkcji straty dla określonej w konfiguracji liczby ostatnich kroków treningowych.

Dobór optymalizatora Dla generatora, czyli sieci odpowiedzialnej za poprawę jakości obrazu, wybrano optymalizator Adam. Jest on powszechnie stosowany ze względu na swoją efektywność w szybkim i stabilnym konwergowaniu w procesie uczenia, szczególnie w zadaniach związanych z przetwarzaniem obrazów, gdzie jest w stanie dobrze radzić sobie z różnorodnością i złożonością danych. Z drugiej strony, dla dyskryminatora, który ma za zadanie ocenić jakość wygenerowanych obrazów, zastosowano optymalizator SGD (Stochastic Gradient Descent). SGD jest znany ze swojej skuteczności w zapewnianiu solidnej i stabilnej konwergencji, co jest kluczowe dla roli dyskryminatora w procesie treningu GAN. Ten rozróżniony wybór optymalizatorów pozwala na lepsze dopasowanie procesu uczenia do specyficznych wymagań i dynamik każdej z części sieci GAN, co może przyczynić się do poprawy ogólnej skuteczności i efektywności trenowania modelu.

5.2.3. Parametry treningu

W tym podrozdziale przedstawiono najważniejsze parametry treningu dla obu etapów badania, takie jak liczba epok, rozmiar partii, współczynnik uczenia, czy współczynnik korekty parametru uczenia. Wartości te zostały dobrane eksperymentalnie.

Trening bez wykorzystania metodologii GAN W przypadku treningu bez wykorzystania metodologii GAN, zastosowano następujące parametry:

- Współczynnik uczenia: 0.0001
- Liczba epok: 1000
- Rozmiar partii: 64
- Współczynnik korekty parametru uczenia: 0.5
- Automatyczne dostosowywanie współczynnika uczenia na podstawie wartości funkcji straty: tak
- Manualnie wskazane punkty zmiany współczynnika uczenia: brak

Na tym etapie również trenowano rozdzielnie sieć dyskryminatora, dla której zastosowano następujące parametry:

- Współczynnik uczenia: 0.001
- Liczba epok: 1000
- Rozmiar partii: 64
- Współczynnik korekty parametru uczenia: 0.5
- Automatyczne dostosowywanie współczynnika uczenia na podstawie wartości funkcji straty: tak
- Manualnie wskazane punkty zmiany współczynnika uczenia: brak

Trening z wykorzystaniem metodologii GAN Dla treningu z wykorzystaniem metodologii GAN, zastosowano następujące parametry:

- Współczynnik uczenia generatora: 0.0001
- Współczynnik uczenia dyskryminatora: 0.001
- Liczba epok: 1000 (pierwsze 100 epok trening rozdzielny, następne 900 epok w metodologii GAN)
- Rozmiar partii: 8
- Współczynnik korekty parametru uczenia: 0.5 (dla obu sieci)
- Automatyczne dostosowywanie współczynnika uczenia na podstawie wartości funkcji straty: tak (monitorowanie funkcji straty dla dyskryminatora l_d i l_{gd})
- Manualnie wskazane punkty zmiany współczynnika uczenia: 50, 100, 150, 200, 300, 400, 500, 600, 800 (dla obu sieci)
- Wartość progowa funkcji straty dyskryminatora: 0.25
- Wartość progowa funkcji straty generatora: 0.65

Taka sama liczba epok w pierwszym i drugim etapie badania pozwala na porównanie wyników uzyskanych w obu metodologiach. Zastosowano jednak 100 epok rozdzielonego treningu, w celu wstępniego wyszkolenia sieci, co pozwoliło na uzyskanie jeszcze lepszych wyników w metodologii GAN, stabilniejszy trening oraz skrócenie całkowitego czasu treningu na tym etapie.

6. Przebieg treningu

Trening sieci neuronowych jest procesem złożonym, wymagającym wielu eksperymentów i prób. W tej sekcji opisane zostaną napotkane problemy oraz metody ich rozwiązania, które znacząco wpłynęły na stabilność i skuteczność treningu oraz uzyskane wyniki.

6.1. Trening rozdzielny

W pierwszym etapie sieci szkolone były rozdzielnie. Miało to na celu zbadanie możliwości różnych architektur w zadaniu poprawy jakości przed rozpoczęciem uczenia generatywnego sieci przeciwnych GAN. Skupiono się na doborze parametrów uczenia, głębokości modeli i liczb filtrów na kolejnych warstwach tak, aby zapewnić stabilny trening i uzyskać jak najlepszą poprawę jakości na zbiorze testującym. Metodę wyznaczania najlepszej architektury ResNet, DenseNet i konwolucyjnej przedstawiono w rozdziale 5 poświęconym metodologii rozwiązania.

Do monitorowania postępu szkolenia sieci neuronowych wykorzystano narzędzie *Weights and Biases* [30]. Pozwala ono na śledzenie postępu, wizualizację metryk i parametrów oraz porównywanie wyników różnych doświadczeń. Monitorowano nie tylko przebieg funkcji straty, ale również wartości wzmacnienia PSNR i SSIM na próbce danych testujących po każdej epoce.

Już na wstępnie zastosowano metody, takie jak dropout [31], normalizacja wsadu [32] czy automatyczne dostosowywanie współczynnika uczenia, które miały na celu zwiększenie stabilności treningu oraz poprawę zdolności do generalizacji sieci. W trakcie treningu testowano różne wartości współczynnika uczenia oraz innych parametrów, aby znaleźć optymalne ustawienia.

6.1.1. Napotkane problemy

Początkowo zastosowano funkcję straty łączącą metryki SSIM i $MS - SSIM$, przedstawioną na rysunku 5, a sieci trenowane pod kątem uzyskania rekonstrukcji na ich wyjściu. Niezależnie od badanej sieci, zauważono bardzo niskie wzmacnienie metryki PSNR w stosunku do poprawy SSIM oraz problem z odwzorowaniem kolorów rekonstruowanych ramek. Na rysunku 6.1 przedstawiono rekonstrukcję pojedynczej ramki, na której można zauważać różnicę w kolorze niebieskiego tła względem oryginału i klatki po dekomprezji.

$$l = 0.5 * MS - SSIM_loss + 0.5 * SSIM_loss \quad (5)$$

Rozszerzenie funkcji straty Aby zniwelować problem słabego wzmacnienia PSNR zdecydowano się rozszerzyć funkcję straty o dodatkowe składowe. Wprowadzono metryki l_1 i l_2 pomiędzy rekonstrukcją a oryginałem, sugerując się rozwiązaniem CVEGAN [18]. Proporcje pomiędzy składowymi zostały dobrane eksperymentalnie na podstawie ich średnich wartości w trakcie eksperymentów tak, aby normalizować ich gradienty.



Rysunek 6.1. Problem z odwzorowaniem kolorów przy poprawie jakości przez sieć

Dodatkowe składowe pozwoliły na zwiększenie wzmacnienia PSNR, jednak problem z odwzorowaniem kolorów pozostał. W celu poprawy tego problemu zdecydowano się na zastosowanie kolejnych usprawnień. Proporcje pomiędzy składowymi zostały dobrane tak, aby normalizować ich gradienty, na podstawie wartości obserwowanych w trakcie eksperymentów.

Zastosowanie mapy poprawek do zdekodowanych ramek Autorzy sieci CVEGAN [18] zauważyl, że zastosowanie mapy poprawek do zdekodowanych ramek pozwala na lepsze odwzorowanie kolorów i ogólną poprawę jakości rekonstrukcji. W związku z tym zdecydowano się na wykorzystanie tej metody.

Zmiana ta pozwoliła na zniwelowanie problemu odwzorowania kolorów oraz pozwoliła na znaczną poprawę uzyskanych wyników, jednak w dalszym ciągu wzmacnienie PSNR oraz SSIM dla kanału Y było niższe niż dla U i V.

Skalowanie funkcji straty dla składowych YUV Analiza danych treningowych oraz testowych wykazała, że dla kanału Y wartości metryk PSNR i SSIM po kompresji kodekiem H.266/VVC były znacznie niższe, niż dla kanałów U i V. W tabeli 6.1 przedstawiono średnie wartości tych metryk dla zbioru testowego. Co za tym idzie, gradienty dla tych składowych są znacznie niższe, niż dla luminancji. Mogło to prowadzić do gorszego przystosowania się sieci do poprawy jakości chrominancji.

metryka	kanał Y	kanał U	kanał V
PSNR	33.3684	38.0436	38.2393
SSIM	0.8892	0.9030	0.9078

Tabela 6.1. Porównanie średnich wartości PSNR i SSIM składowych YUV danych testowych po kompresji kodekiem H.266/VVC

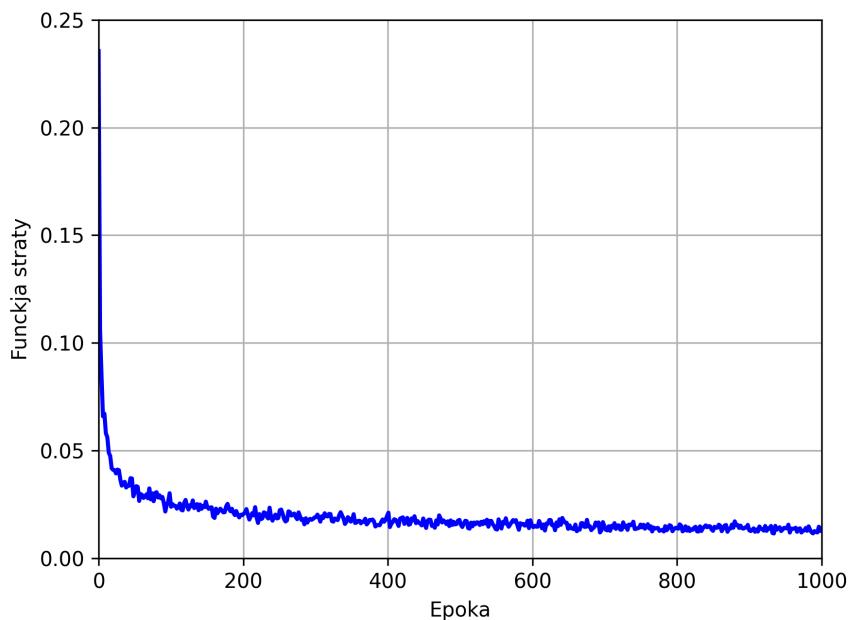
Starano się skorygować ten problem poprzez zastosowanie mechanizmu skalowania funkcji straty. Testowano podejście automatyczne, polegające na obliczeniu średnich wartości dla każdej składowej YUV, a następnie normalizujące te wartości, oraz podejście manualne, poprzez podanie współczynników wagowych w konfiguracji. Nie uzyskano

6. Przebieg treningu

jednak poprawy wyników. Warto zaznaczyć, że pomimo mniejszego wzmacnienia metryk PSNR i SSIM, redukcja przepływności danych obliczona przy pomocy delty Bjøntegaard'a dla U i V jest porównywalna do składowej Y.

6.1.2. Przebieg funkcji straty

Rysunek 6.2 przedstawia przebieg funkcji straty podczas treningu sieci DenseNet po zastosowaniu wszystkich usprawnień. Jej wartość maleje wraz z kolejnymi epokami, co oznacza, że sieć poprawia jakość rekonstrukcji. Wykres ten pokrywa się ze spodziewanym jego przebiegiem - wartość funkcji straty dla pierwszych epok szybko maleje, natomiast dla późniejszych spadek jest zdecydowanie wolniejszy, osiągając na końcu treningu stabilny poziom.



Rysunek 6.2. Przebieg funkcji straty podczas treningu sieci DenseNet poprawiającej jakość

6.2. Trening sieci GAN

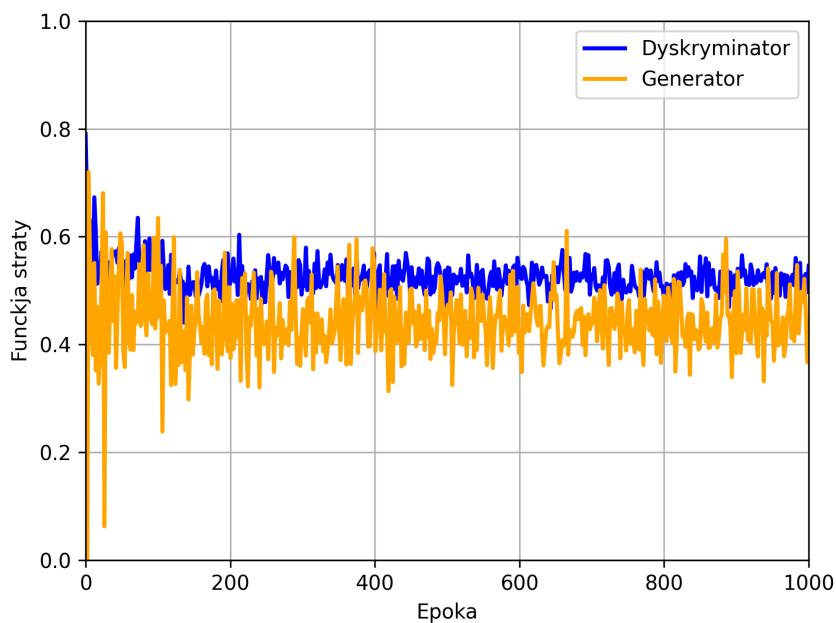
Drugim etapem badania był trening generatywnej sieci przeciwnostawnej GAN. Jako generatora wykorzystano sieć DenseNet, która uzyskała najlepsze wyniki w pierwszym etapie, co przedstawiono w rozdziale 8, a dyskryminatora - DenseNet-121 o współczynniku przyrostu $lr = 8$. Wybór architektury dyskryminatora był podyktowany chęcią zwiększenia jego zdolności do wykrywania subtelnych różnic w jakości rekonstrukcji, co miało na celu zwiększenie stabilności treningu oraz poprawę jakości generowanych ramek.

Zastosowano te same metody mające na celu zwiększenie stabilności treningu sieci, co w pierwszym etapie. W trakcie treningu testowano różne wartości współczynnika uczenia oraz innych parametrów, aby znaleźć optymalne ustawienia.

Na początku jako funkcja straty zastosowano funkcję straty GAN, zaproponowaną przez autorów tej architektury.

6.2.1. Problem stabilności treningu sieci GAN

Główym problemem napotkanym podczas treningu sieci GAN była jej niestabilność i brak zbieżności. W pierwszych próbach sieć nie była w stanie nauczyć się poprawy jakości rekonstrukcji, a funkcja straty oscylowała wokół stałej wartości. Przykładowy przebieg funkcji straty podczas niestabilnego treningu przedstawiono na rysunku 6.3.



Rysunek 6.3. Przebieg funkcji straty podczas treningu sieci GAN dla pierwszego stabilnego treningu

W trakcie treningu zauważono też, że sieć dyskryminatora nie jest w stanie nauczyć się rozróżniania poprawionych ramek od oryginalnych dla danych testowych, pomimo wysokich wartości metryki dokładności (ang. *accuracy*) w trakcie treningu. Jest to typowy objaw tzw. problemu nadmiernego dopasowania (ang. *overfitting*).

Ponadto, nawet w przypadkach kiedy udawało się osiągnąć stabilny trening, jakość generowanych ramek była niewystarczająca, a sieć nie była w stanie nauczyć się poprawy jakości rekonstrukcji lepiej niż w pierwszym etapie badania.

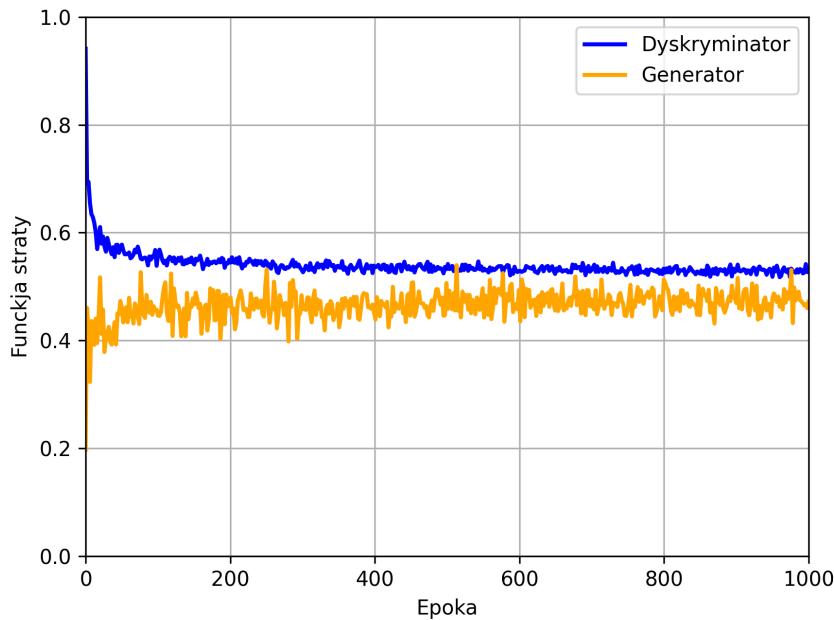
Zmniejszenie współczynnika przyrostu dla dyskryminatora Rozwiązaniem problemu nadmiernego dopasowania okazało się zmniejszenie współczynnika przyrostu dla dyskryminatora. Zastosowano wartość $lr = 8$, co pozwoliło na zwiększenie zdolności dyskryminatora do generalizacji rozpoznawanych artefaktów.

Trening selektywny W celu poprawy stabilności treningu zastosowano mechanizm treningu selektywnego. Jest on szeroko stosowany w literaturze [14], a polega na czasowym

6. Przebieg treningu

wyłączeniu szkoleniu jednej z sieci w przypadku, kiedy ona zaczyna sobie radzić lepiej od drugiej, destabilizując proces uczenia.

Wykorzystanie tej metody pozwoliło na znaczne ustabilizowanie treningu sieci GAN, co przedstawiono na rysunku 6.4.



Rysunek 6.4. Przebieg funkcji straty po zastosowaniu selektywnego treningu

Funkcja straty Mimo osiągnięcia stabilnego treningu, jakość generowanych ramek wciąż nie zaobserwowano znacznej poprawy jakości w stosunku do pierwszego etapu. W celu poprawy tego problemu zdecydowano się rozszerzyć funkcję straty sieci generatora o metryki wykorzystane w pierwszym etapie badania, bezpośrednio optymalizujące metryki takie jak SSIM i MSE, wpływający na PSNR. Takie podejście jest szeroko wykorzystywane w literaturze, na przykład w przypadku sieci CVEGAN [18].

Zastosowanie łączonej funkcji straty pozwoliło na znaczne poprawienie jakości generowanych ramek, natomiast nie wpłynęło na stabilność treningu.

Dostosowanie parametrów uczenia Pomimo zastosowania wszystkich usprawnień, w dalszym ciągu jakość generowanych ramek była niewystarczająca, a trening sieci GAN był stabilny lecz nie zbieżny. Dlatego zdecydowano się na wprowadzenie dalszych metod go usprawniających.

Pierwszą z nich było zastosowanie optymalizatora SGD zamiast ADAM dla dyskryminatora. Optymalizator SGD jest mniej zaawansowany niż ADAM, jednak w przypadku problemów z zbieżnością sieci GAN może okazać się skuteczniejszy. Jest to metoda po-wszechnie stosowana w literaturze [14]. SGD, będąc prostszym i bardziej przewidywalnym

optymalizatorem, może przyczynić się do stabilizacji procesu uczenia dyskryminatora, który ma za zadanie klasyfikować dane jako prawdziwe lub wygenerowane. Stabilność ta jest kluczowa, aby dyskryminator nie stał się zbyt silny za szybko, co mogłoby zahamować uczenie generatora. Ten ma zazwyczaj bardziej trudne zadanie – musi generować dane, które będą na tyle przekonujące, by oszukać dyskryminator. ADAM, będąc bardziej zaawansowanym optymalizatorem z adaptacyjnymi wskaźnikami uczenia dla różnych parametrów, lepiej radzi sobie z tą złożonością, dostosowując proces uczenia w bardziej dynamiczny sposób.

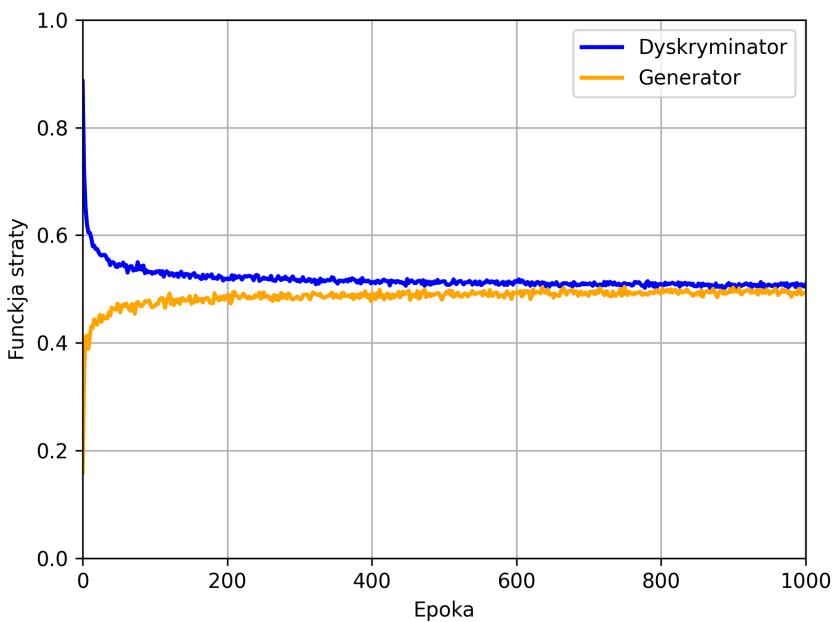
Następnie wykorzystano metodę manualnego zarządzania współczynnikiem uczenia. W trakcie treningu zauważono, że automatyczne dostosowywanie tego parametru jest niewystarczające, prowadząc do braku zbieżności lub destabilizacji uczenia. Dlatego wykorzystano metodę manualnego zarządzania tych współczynnikiem, a jego wartości dla kolejnych epok zostały dobrane na podstawie obserwacji przebiegu eksperymentów.

Funkcja aktywacji Kolejnym usprawnieniem stosowanym często w literaturze jest zastosowanie funkcji aktywacji PReLU zamiast ReLU. Funkcja ta pozwala na uczenie się większej liczby cech w sieciach neuronowych, co może przyczynić się do poprawy jakości generowanych ramek. PReLU minimalizuje ryzyko "umierania neuronów", szczególnie ważne dla szkolenia generatywnych sieci GAN, gdzie utrzymanie efektywnego przepływu gradientu jest kluczowe dla uczenia generatora.

PReLU oferuje dodatkową elastyczność dzięki parametrowi, dostosowywanemu w trakcie uczenia. Ta adaptacyjność pozwala na lepsze dopasowanie funkcji aktywacji do specyficznych cech danych, co może przyczynić się do wydajniejszego generowania bardziej złożonych rozkładów danych przez generator. Możliwość przepuszczania ujemnych wartości wejściowych, nawet w niewielkim stopniu, pozwala na bardziej płynne i zróżnicowane generowanie treści, co jest pożądane w wielu zastosowaniach GAN [14]. W niektórych przypadkach, PReLU może również pomóc w zmniejszeniu ryzyka przetrenowania sieci przez wprowadzenie dodatkowego parametru regularyzującego, co jest korzystne w kontekście konkurencyjnego uczenia się, gdzie łatwo zaburzyć równowagę między generatorem a dyskryminatorem.

6.2.2. Przebieg funkcji straty

Dzięki zastosowaniu wyżej wymienionych usprawnień udało się osiągnąć stabilny trening sieci GAN prowadzący do zbieżności funkcji straty, co przedstawiono na rysunku 6.5.



Rysunek 6.5. Przebieg funkcji straty po zastosowaniu wszystkich usprawnień w treningu sieci GAN

7. Szczegóły implementacyjne

W procesie implementacji modeli sieci neuronowych do zadania poprawy jakości wideo postawiono na wykorzystanie języka programowania Python i frameworka PyTorch [33] wspomaganego przez PyTorch Lightning [34]. Taki wybór narzędzi był motywowany ich elastycznością, wszechstronnością oraz ułatwieniem procesu implementacji i eksperymentowania. Jedną z kluczowych decyzji była rezygnacja z gotowych implementacji bloków DenseNet i ResNet dostępnych w bibliotece PyTorch. Zamiast tego, zdecydowano się na stworzenie własnej implementacji tych architektur. Było to podyktowane potrzebą głębszego zrozumienia szczegółów implementacyjnych tych modeli oraz chęcią uzyskania większej kontroli nad procesem uczenia.

Oryginalnie architektury ResNet i DenseNet zostały opracowane głównie z myślą o zadaniach klasyfikacji obrazów, a w tym projekcie koncentrujemy się na zupełnie innym, jakim jest poprawa jakości obrazu. Własna implementacja pozwoliła na dostosowanie i optymalizację tych architektur pod kątem specyficznych potrzeb.

Początkowo rozważano użycie biblioteki Hydra [35], która oferuje szerokie możliwości konfiguracji eksperymentów. Jednak ostatecznie zdecydowano się na użycie Pydantic Settings, będącym częścią biblioteki Pydantic [36], co okazało się rozwiązaniem prostszym, ale jednocześnie oferującym większą kontrolę nad strukturą i sposobem konfiguracji eksperymentów. Ta decyzja była krokiem w kierunku uproszczenia procesu implementacji, przy zachowaniu elastyczności i precyzji w definiowaniu parametrów eksperymentów.

7.1. Implementacja modeli sieci neuronowych

Biblioteka PyTorch wykorzystuje obiektowy paradygmat programowania do implementacji modeli sieci neuronowych. W tym celu wykorzystuje się klasy dziedziczące po klasie `nn.Module`, które definiują strukturę sieci oraz sposób przetwarzania danych. W ten sposób, każda warstwa sieci jest reprezentowana przez obiekt klasy dziedziczącej po `nn.Module`, a cała sieć jest reprezentowana przez obiekt klasy dziedziczącej po `nn.Module`, agregującym wszystkie warstwy.

Framework znany jest z oferowania szerokiej gamy wysokopoziomowych klas i znacząco ułatwia proces definiowania struktury sieci neuronowych. W pracy zdecydowano się na wykorzystanie klasy `nn.Sequential`, jako klasy bazowej do implementacji struktur (oprócz warstw sieci ResNet, co zostało wyjaśnione w podrozdziale 7.1.2), ponieważ pozwala na tworzenie sekwencji warstw sieci neuronowej w uporządkowany i czytelny sposób. Dzięki temu konstrukcja modelu staje się bardziej intuicyjna, a kod - łatwiejszy do zrozumienia i utrzymania. W praktyce warstwy sieci są definiowane jako sekwencja operacji, przez co model można budować krok po kroku, dodając kolejne warstwy w logicznym porządku.

Ponadto `nn.Sequential` oferuje zoptymalizowane procedury dla sekwencji warstw, co jest istotne dla efektywności obliczeniowej modelu. Oznacza to, że procedury przekazywania danych przez sieć (ang. *forward pass*) są zoptymalizowane pod kątem wydajności, a to jest szczególnie ważne w przypadku złożonych modeli i dużych zbiorów danych. Dzięki temu, trening i testowanie modelu przebiega szybciej, co jest kluczowe w procesie badawczym, gdzie często wymagane są liczne iteracje i eksperymenty.

7.1.1. Implementacja warstw konwolucyjnych

Warstwy konwolucyjne, będące podstawą do implementacji architektur, takich jak ResNet i DenseNet, są złożone z kilku kluczowych komponentów, które współpracują, aby skutecznie przetwarzać i wydobywać cechy z obrazów. W szczególności, można wyróżnić zastosowanie normalizacji wsadowej (ang. *batch normalization*) [32] po operacji splotu, następnie wybranej funkcji aktywacji i ewentualnej operacji dropout [31].

W kontekście przetwarzania obrazów we fragmentach o wymiarach 132x132 pikseli, istotnym elementem jest wybór odpowiedniego rodzaju paddingu. W standardowej konfiguracji często stosuje się zero padding, który dodaje warstwę zer wokół obrazu, aby zachować jego wymiary po konwolucji. Jednak w tej pracy zdecydowano się na możliwość wykorzystania reflection paddingu, polegającego na kopiowaniu wartości z krawędzi obrazu. Takie podejście może pomóc w zachowaniu naturalnych granic obrazu i uniknięciu sztucznych artefaktów na krawędziach przetwarzanych fragmentów.

Implementację warstwy konwolucyjnej przedstawiono na listingu 5. Warto zwrócić uwagę na wykorzystanie dekoratora `@validate_arguments`, będącego częścią wspomnianej już wcześniej biblioteki Pydantic. Zapewnia on mechanizm walidacji typów

7. Szczegóły implementacyjne

parametrów funkcji dla języka Python, który jest językiem dynamicznie typowanym bez wbudowanych mechanizmów sprawdzania poprawności typów parametrów na poziomie interpretera.

Listing 5. Implementacja warstwy konwolucyjnej

```
1 class ConvLayer(nn.Sequential):
2     @validate_arguments
3     def __init__(
4         self,
5         in_channels: int,
6         out_channels: int,
7         kernel_size: int = 3,
8         stride: int = 1,
9         padding: int = 1,
10        dropout: float = 0.0,
11        reflect_padding: bool = True,
12        activation: str = "prelu",
13    ) -> None:
14     super().__init__()
15
16     if reflect_padding and padding > 0:
17         self.add_module(
18             "pad",
19             nn.ReflectionPad2d(
20                 padding,
21             ),
22         )
23
24     self.add_module(
25         "conv",
26         nn.Conv2d(
27             in_channels,
28             out_channels,
29             kernel_size=kernel_size,
30             stride=stride,
31             padding=padding if not reflect_padding else 0,
32             bias=False,
33         ),
34     )
35
36     self.add_module(
37         "bn",
38         nn.BatchNorm2d(out_channels),
39     )
```

```

40
41     if activation != "none":
42         self.add_module("activation", get_activation(activation))
43
44     if dropout > 0:
45         self.add_module(
46             "dropout",
47             nn.Dropout2d(p=dropout),
48         )

```

W wielu modelach sieci neuronowych stosuje się sekwencje samych warstw konwolucyjnych o jednakowych liczbach filtrów, więc zdecydowano się na stworzenie klasy ConvBlock, która agreguje warstwy splotowe w sekwencję. W ten sposób każdy blok konwolucyjny był reprezentowany przez obiekt klasy ConvBlock, łączący wszystkie warstwy splotowe w bloku. Implementację tej klasy przedstawiono na listingu 6.

Listing 6. Implementacja bloku konwolucyjnego

```

1 class ConvBlock(nn.Sequential):
2     @validate_arguments
3     def __init__(
4         self,
5         num_layers: int,
6         in_channels: int,
7         out_channels: int,
8         kernel_size: int = 3,
9         stride: int = 1,
10        padding: int = 1,
11        dropout: float = 0.0,
12        reflect_padding: bool = True,
13        activation: str = "prelu",
14    ) -> None:
15        super0.__init__0
16
17        self.add_module(
18            "conv0",
19            ConvLayer(
20                in_channels=in_channels,
21                out_channels=out_channels,
22                kernel_size=kernel_size,
23                stride=stride,
24                padding=padding,
25                dropout=dropout,
26                reflect_padding=reflect_padding,
27                activation=activation,

```

7. Szczegóły implementacyjne

```
28         ),
29     )
30
31     for i in range(1, num_layers):
32         layer = ConvLayer(
33             in_channels=out_channels,
34             out_channels=out_channels,
35             kernel_size=kernel_size,
36             stride=stride,
37             padding=padding,
38             dropout=dropout,
39             reflect_padding=reflect_padding,
40             activation=activation,
41         )
42         self.add_module(f"conv{i}", layer)
```

7.1.2. Implementacja struktur ResNet

W architekturze sieci ResNet znajdują się warstwy konwolucyjne, które są zorganizowane w bloki resztkowe. Każdy taki blok składa się zwykle z dwóch warstw splotowych. Po drugiej warstwie konwolucyjnej, po zastosowaniu normalizacji wsadowej, dodaje się sygnał wejściowy do bloku. Jest to kluczowy element architektury ResNet, umożliwiający przekazywanie informacji wejściowych bezpośrednio do dalszych warstw, co pomaga w zachowaniu gradientu w procesie uczenia, zwłaszcza w głębokich sieciach.

Implementacja warstwy resztkowej została przedstawiona w listingu 7. Warto zaznaczyć, że jest to jedyny przypadek, gdzie jako klasy bazowej nie wykorzystano `nn.Sequential`, a `nn.Module` ze względu na dwie równoległe sekwencje operacji na informacjach na wejściu warstwy, których wyjścia są sumowane i poddane ostatniej funkcji aktywacji.

Listing 7. Implementacja warstwy resztkowej

```
1 class ResLayer(nn.Module):
2     def __init__(
3         self,
4         in_channels: int,
5         out_channels: int,
6         kernel_size: int = 3,
7         stride: int = 1,
8         padding: int = 1,
9         reflect_padding: bool = True,
10        activation: str = "prelu",
11        dropout: float = 0.0,
12    ) -> None:
13        super().__init__()
```

```
15     self.conv1 = ConvLayer(  
16         in_channels=in_channels,  
17         out_channels=out_channels,  
18         kernel_size=kernel_size,  
19         stride=stride,  
20         padding=padding,  
21         reflect_padding=reflect_padding,  
22         activation=activation,  
23         dropout=dropout,  
24     )  
25  
26     self.conv2 = ConvLayer(  
27         in_channels=out_channels,  
28         out_channels=out_channels,  
29         kernel_size=kernel_size,  
30         stride=1,  
31         padding=padding,  
32         reflect_padding=reflect_padding,  
33         activation="none",  
34     )  
35  
36     self.downsample = None  
37     if in_channels != out_channels or stride != 1:  
38         self.downsample = DownsampleLayer(  
39             in_channels=in_channels,  
40             out_channels=out_channels,  
41             stride=stride,  
42         )  
43  
44     self.activation = get_activation(activation)  
45  
46     self.dropout = None  
47     if dropout > 0:  
48         self.dropout = nn.Dropout2d(p=dropout)  
49  
50  
51     def forward(self, _input: Tensor) -> Tensor:  
52         identity = _input  
53  
54         out = self.conv1(_input)  
55         out = self.conv2(out)  
56  
57         if self.downsample is not None:  
58             identity = self.downsample(_input)
```

7. Szczegóły implementacyjne

```
59     out += identity
60     out = self.activation(out)
61
62
63     if self.dropout is not None:
64         out = self.dropout(_input)
65
66     return out
```

W przypadku, gdy liczba cech wyjściowych z warstwy konwolucyjnej różni się od liczby cech wejściowych bloku, lub gdy ich wymiar jest inny, stosuje się tzw. warstwę downsample. Jej celem jest dopasowanie wymiarów sygnału wejściowego do wymiarów sygnału wyjściowego z drugiego splotu. Downsample zwykle wykorzystuje konwolucję o jądrze 1x1, co pozwala na zmianę liczby cech bez znaczącego wpływu na wymiary przestrzenne obrazu. Dzięki temu możliwe jest sumowanie sygnału wejściowego bloku z wyjściem drugiej warstwy konwolucyjnej, a było to istotne dla zachowania resztowej natury sieci. Listing 8 przedstawia implementację takiej warstwy.

Listing 8. Implementacja warstwy skalującej

```
1 class DownsampleLayer(nn.Sequential):
2     def __init__(
3             self,
4             in_channels: int,
5             out_channels: int,
6             stride: int,
7         ) -> nn.Module:
8         self.add_module(
9             "conv",
10            nn.Conv2d(
11                in_channels,
12                out_channels,
13                kernel_size=1,
14                stride=stride,
15                bias=False,
16            ),
17        )
18
19        self.add_module(
20            "bn",
21            nn.BatchNorm2d(out_channels),
22        )
```

Przy sieciach resztkowych również często stosuje się sekwencje warstw o takiej samej liczbie filtrów, zatem zdecydowano się na stworzenie klasy `ResBlock`, która agregowała je

w sekwencję. W ten sposób każdy blok resztkowy był reprezentowany przez obiekt klasy ResBlock, łączący wszystkie warstwy w bloku. Implementację tej klasy przedstawiono na listingu 9.

Listing 9. Implementacja bloku resztkowego

```

1 class ResBlock(nn.Sequential):
2     @validate_arguments
3     def __init__(
4         self,
5         num_layers: int,
6         in_channels: int,
7         out_channels: int,
8         kernel_size: int = 3,
9         stride: int = 1,
10        padding: int = 1,
11        reflect_padding: bool = True,
12        activation: str = "prelu",
13        dropout: float = 0.0,
14    ) -> None:
15        super().__init__()
16
17        num_channels = in_channels
18
19        for i in range(num_layers):
20            layer = ResLayer(
21                in_channels=num_channels,
22                out_channels=out_channels,
23                kernel_size=kernel_size,
24                stride=stride if i == 0 else 1,
25                padding=padding,
26                reflect_padding=reflect_padding,
27                activation=activation,
28                dropout=dropout,
29            )
30            num_channels = out_channels
31            self.add_module(f"res_layer_{i}", layer)

```

7.1.3. Implementacja struktur DenseNet

Sieci oparte o architekturę DenseNet korzystają tzw. połączeń gęstych, realizowanych poprzez konkatenację wejścia do wyjścia każdej warstwy gęstej. Oryginalna praca sugerowała wykorzystanie stałego współczynnika przyrostu (ang. *growth rate*). Oznacza to, że każda warstwa sieci ma taką samą liczbę filtrów, a liczba warstw sygnału rośnie liniowo wraz z głębokością sieci. W ten sposób, każda warstwa ma dostęp do wszystkich

7. Szczegóły implementacyjne

poprzednich warstw, co pozwala na efektywne wykorzystanie informacji z poprzednich warstw i przeciwdziałanie problemowi zanikającego gradientu. Dodatkowo, sieci te wykorzystują warstwy wąskiego gardła (ang. *bottleneck layer*). Implementacja warstw gęstych została przedstawiona na listingu 10. Warto zaznaczyć, że aby konkatenacja sygnałów była możliwa, konwolucja powinna mieć odpowiednio dobrany rozmiar filtrów, padding oraz stride o wartości równej 1. Pozwala to na zachowanie tego samego rozmiaru wejścia i wyjścia bloku, a zatem zastosowanie wspomnianej operacji.

Listing 10. Implementacja warstwy gęstej

```
1 class DenseLayer(nn.Sequential):
2     @validate_arguments
3     def __init__(
4         self,
5         in_channels: int,
6         growth_rate: int,
7         kernel_size: int = 3,
8         stride: int = 1,
9         padding: int = 1,
10        dropout: float = 0.0,
11        reflect_padding: bool = True,
12        activation: str = "prelu",
13        bn_size: float = 2.0,
14    ) -> None:
15        super().__init__()
16        self.add_module(
17            "bottleneck",
18            ConvLayer(
19                in_channels=in_channels,
20                out_channels=int(bn_size * growth_rate),
21                kernel_size=1,
22                stride=1,
23                padding=0,
24                reflect_padding=reflect_padding,
25                activation=activation,
26            ),
27        )
28
29        self.add_module(
30            "conv",
31            ConvLayer(
32                in_channels=int(bn_size * growth_rate),
33                out_channels=growth_rate,
34                kernel_size=kernel_size,
35                stride=stride,
```

```

36     padding=padding,
37     dropout=dropout,
38     reflect_padding=reflect_padding,
39     activation=activation,
40     ),
41   )
42
43   def forward(self, _input: Tensor) -> Tensor:
44     output = super().forward(_input)
45     return torch.cat((_input, output), 1)

```

Ze względu na szybki przyrost wymiarowości sygnału, autorzy zaproponowali wykorzystanie struktury blokowej - bloki składają się z określonej liczby warstw gęstych, a następnie następuje przejście do kolejnego bloku, w ramach którego stosowana jest kolwolucja o kernelu 1×1 . Blok gęsty jest zatem sekwencją wielu warstw DenseLayer. Implementację takiego bloku przedstawiono na listingu 11.

Listing 11. Implementacja bloku gęstego

```

1 class DenseBlock(nn.Sequential):
2   @validate_arguments
3   def __init__(
4     self,
5     num_layers: int,
6     in_channels: int,
7     growth_rate: int,
8     kernel_size: int = 3,
9     stride: int = 1,
10    padding: int = 1,
11    dropout: float = 0.0,
12    reflect_padding: bool = True,
13    activation: str = "prelu",
14    bn_size: float = 2.0,
15  ) -> None:
16   super().__init__()
17
18   num_features = in_channels
19
20   for i in range(num_layers):
21     layer = DenseLayer(
22       in_channels=num_features,
23       growth_rate=growth_rate,
24       kernel_size=kernel_size,
25       stride=stride,
26       padding=padding,

```

7. Szczegóły implementacyjne

```
27     dropout=dropout,
28     reflect_padding=reflect_padding,
29     activation=activation,
30     bn_size=bn_size,
31 )
32     num_features += growth_rate
33     self.add_module(f"dense_layer_{i}", layer)
```

Konwolucję przy przejściu do kolejnych bloków stosuje się w celu redukcji wymiarowości informacji. W ten sposób, sieć jest w stanie efektywnie przetwarzać informacje, jednocześnie zachowując niską liczbę parametrów. Ze względu na brak ingerencji w rozmiar sygnału na poszczególnych warstwach, a zatem i wewnątrz całych bloków gęstych, to na warstwie przejściowej (ang. *transition layer*) stosuje się operację uśrednionego łączenia (ang. *average pooling*) w celu redukcji rozmiaru informacji. Zdecydowano się udostępnić parametr kontrolujący, czy na danym bloku przejścia stosować tą operację, oraz parametry kontrolujące rozmiar filtra konwolucji tej warstwy. Dzięki temu możliwe jest zarówno zachowanie rozmiaru informacji przy redukcji jej wymiarowości, jak i dowolne manipulowanie tym rozmiarem przy operacji splotu i osiągnięcie np. zwiększenia rozmiaru informacji. Implementacja warstwy przejścia między blokami przedstawiona na listingu 12.

Listing 12. Implementacja warstwy przejścia między blokami gęstymi

```
1 class Transition(nn.Sequential):
2     def __init__(
3         self,
4         in_channels: int,
5         kernel_size: int = 1,
6         stride: int = 1,
7         padding: int = 1,
8         reflect_padding: bool = True,
9         activation: str = "prelu",
10        mode: TransitionMode = TransitionMode.same,
11    ):
12        super().__init__()
13        out_channels = in_channels // 2
14
15        if reflect_padding and padding > 0:
16            self.add_module(
17                "pad",
18                nn.ReflectionPad2d(
19                    padding,
20                ),
21            )
```

```

22     self.add_module(
23         "conv",
24         nn.Conv2d(
25             in_channels,
26             out_channels,
27             kernel_size=kernel_size,
28             stride=stride,
29             padding=padding if not reflect_padding else 0,
30             bias=False,
31             ),
32         ),
33     )
34
35     if mode == TransitionMode.down:
36         self.add_module(
37             "rescale",
38             nn.AvgPool2d(kernel_size=2, stride=2),
39         )
40
41     self.add_module(
42         "bn",
43         nn.BatchNorm2d(out_channels),
44     )
45
46     self.add_module("activation", get_activation(activation))

```

7.1.4. Implementacja warstw przetwarzających informacje o parametrach kompresji i rodzaju ramki

Warstwa przetwarzająca metadane na temat parametrów kompresji i rodzaju ramki ma za zadanie przetworzyć te informacje w taki sposób, aby mogły być one wykorzystane przez sieć neuronową. W tym celu zdecydowano się na zastosowanie interpolacji liniowej. Dane wejściowe są bowiem w formacie wektora pojedynczych cech, natomiast aby móc dodać je do danych konkretnej klatki lub jej fragmentu należy doprowadzić je do wspólnego rozmiaru. Implementację przedstawiono na listingu 13.

Listing 13. Implementacja warstwy przetwarzających metadane ramki

```

1 class MetadataEncoder(nn.Module):
2     @validate_arguments
3     def __init__(
4         self,
5         metadata_size: int = 6,
6         metadata_features: int = 64,
7         size: int = 132,

```

7. Szczegóły implementacyjne

```
8     num_of_blocks: int = 2,
9 ) -> None:
10    super().__init__()
11
12    self.size = size
13
14  def forward(self, x: Tensor) -> Tensor:
15      x = torch.nn.functional.interpolate(x, size=self.size)
16
17      return x
```

7.2. Szczegóły implementacyjne procesu treningu

PyTorch Lightning implementuje wiele warstw abstrakcji dla modułu PyTorch, które zostały zaprojektowane w celu uproszczenia procesu wytwarzania kodu związanego z procesami treningu i predykcji przy jednoczesnym zachowaniu pełnej elastyczności i mocy PyTorch. Oferuje ono bardziej uporządkowaną i modularną strukturę kodu, ułatwiając zarządzanie złożonymi operacjami uczenia maszynowego.

7.2.1. Zalety PyTorch Lightning

1. **Uproszczenie Kodu:** Lightning pozwala na znaczne uproszczenie kodu, przenosząc wiele rutynowych i powtarzalnych zadań, takich jak obsługa pętli treningowych, walidacyjnych i testowych, do wewnętrznych mechanizmów biblioteki.
2. **Modułowość i Czytelność:** Struktura kodu staje się bardziej modularna i czytelna, ponieważ Lightning zachęca do oddzielania logiki biznesowej modelu od kodu inżynierijnego. Ułatwia to zarządzanie skomplikowanymi projektami i współpracę w zespołach.
3. **Skalowalność i Wydajność:** Lightning ułatwia skalowanie eksperymentów na różne środowiska obliczeniowe, od lokalnych GPU po klastry obliczeniowe, bez konieczności modyfikowania kodu źródłowego.
4. **Zaawansowane Funkcje:** Lightning oferuje zaawansowane funkcje, takie jak automatyczne logowanie, wcześnie zatrzymywanie, sprawdzanie punktów kontrolnych i wiele innych, co zwiększa produktywność i efektywność pracy.

7.2.2. Struktura Projektu w PyTorch Lightning

Projekt wykorzystujący PyTorch Lightning zwykle obejmuje kilka kluczowych komponentów:

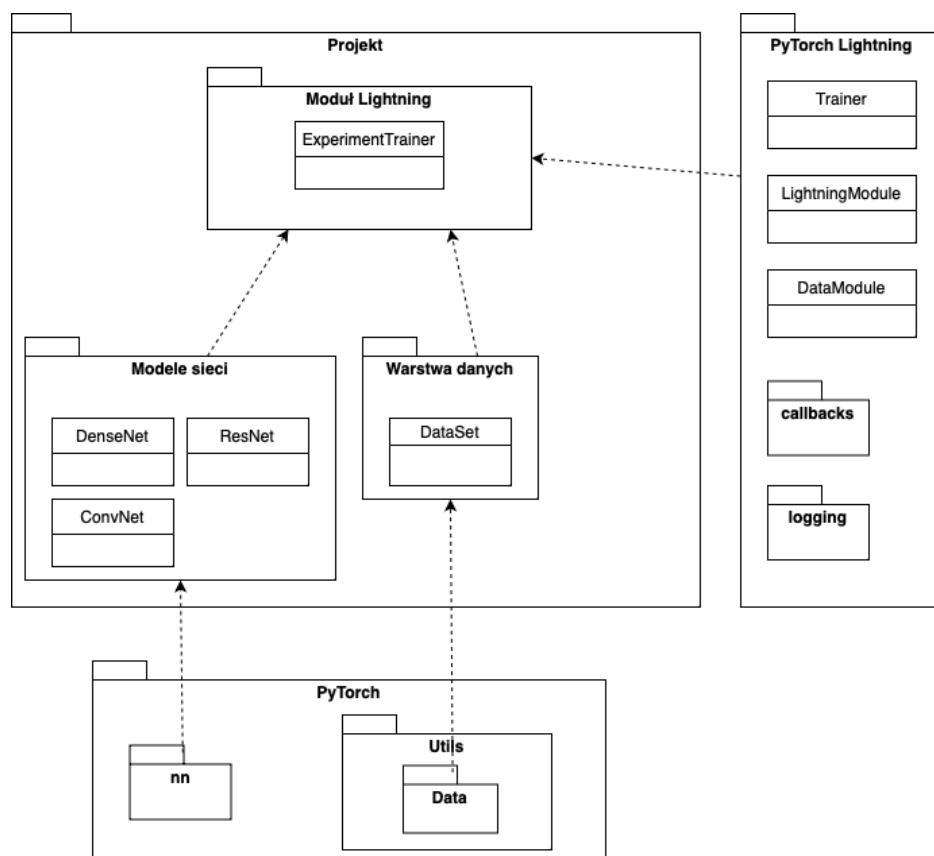
1. **Model LightningModule:** Serce każdego projektu Lightning, LightningModule, jest klasą, gdzie zdefiniowany jest model, metody forward, kroku treningowy, kroku walidacyjny, kroku testowego, kroku predykcji i konfiguracja optymalizatora.
2. **DataModule:** Lightning DataModule to oddzielna klasa, która zarządza danymi. Umożliwia ona łatwą organizację i przetwarzanie danych, definiując metody do

ładowania, przetwarzania i dzielenia danych na zestawy treningowe, walidacyjne i testowe.

3. **Trainer:** Trainer to klasa, która zarządza pętlą treningową. Odpowiada za obsługę wielu zadań, takich jak wczesne zatrzymywanie, sprawdzanie punktów kontrolnych, logowanie, skalowanie eksperymentów i wiele innych. W przypadku większości projektów wystarczy zdefiniować obiekt klasy Trainer i wywołać metodę `fit` na obiekcie klasy `LightningModule`.

Dzięki tym komponentom, PyTorch Lightning pozwala na wydajne i uporządkowane zarządzanie procesem uczenia maszynowego, minimalizując potrzebę pisania boilerplate code i pozwalając na skupienie się na kluczowych aspektach projektu.

Diagram struktury pakietu projektu został przedstawiony na rysunku 7.1. Przedstawia on najważniejsze warstwy modułu w języku Python napisanego w ramach pracy, istotne zależności od omawianego pakietu PyTorch Lightning oraz samego PyTorch.



Rysunek 7.1. Diagram struktury pakietu projektu

7.2.3. Implementacja modułu danych

Sam moduł danych w PyTorch lightning określa jedynie sposób ładowania danych, dzielenia ich na zbiory treningowe, walidacyjne i testowe oraz przetwarzania ich w celu przygotowania do treningu.

7. Szczegóły implementacyjne

Normalizacja danych Ze względu na charakter badania, zdecydowano się nie wykorzystać żadnych normalizacji danych. W zadaniu poprawy jakości wideo po kompresji ważne jest, aby sieć neuronowa uczyła się na danych, które odzwierciedlają rzeczywiste artefakty i cechy sygnału po kompresji. Ujednolicenie danych może zmienić te charakterystyki, uszczuplając istotne informacje o artefaktach kompresji, które sieć powinna nauczyć się korygować.

W przeciwnieństwie do innych zastosowań uczenia maszynowego, gdzie normalizacja pomaga w standaryzacji zakresu danych wejściowych, w poprawie jakości wideo po kompresji ważne jest zachowanie oryginalnego zakresu dynamiki obrazu. Normalizacja mogłaby zakłócić ten zakres, co wpłynęłoby negatywnie na zdolność sieci do efektywnego uczenia się i poprawy jakości obrazu.

Normalizacja może prowadzić do utraty pewnych cech obrazu, które są kluczowe dla procesu poprawy jakości. W zadaniu takim jak poprawa jakości wideo, ważne jest, aby zachować pełną информацию zawartą w danych wejściowych.

Ładowanie danych Ładowanie odpowiednio przygotowanych danych jest jednym z kluczowych elementów procesu treningu. Dane w formacie fragmentów 128x128 pikseli z zakładkami wczytywane są przy pomocy klasy ChunkDataset, dziedziczącej po klasie FrameDataset ładującej pełne ramki w celu wykonania predykcji dla sekwencji wideo. Obie klasy współdzielą większość funkcjonalności, takich jak ładowanie parametrów kompresji i rodzajów ramki, co zostało przedstawione na listingu 14.

Listing 14. Implementacja ładowania parametrów kompresji i rodzajów ramek

```
1 def get_metadata(self, fname: str) -> Metadata:  
2     fname, profiles, frame = fname.split("/")[-3:]  
3     frame = frame.split(".")[0]  
4     _, heigh, width = fname.split("__")  
5     profile, qp, alf, db, sao = profiles.split("_")  
6     frame, is_intra = frame.split("_")  
7  
8     metadata = Metadata(  
9         file=fname,  
10        profile=profile,  
11        qp=int(qp[2:]),  
12        alf=bool(int(alf[3:])),  
13        sao=bool(int(sao[3:])),  
14        db=bool(int(db[2:])),  
15        frame=int(frame),  
16        is_intra=is_intra == "True",  
17        height=int(heigh),  
18        width=int(width),  
19    )
```

20 **return** metadata

Współdzielą one również funkcję `load_data`, która odpowiada za ładowanie fragmentów wideo z plików binarnych. W przypadku klasy `FrameDataset`, funkcja ta wczytuje pełne ramki, a w przypadku klasy `ChunkDataset`, wczytuje fragmenty o rozmiarze 128x128 pikseli. Implementacja tej funkcji dla wczytywania pełnych ramek została przedstawiona na listingu 15. W przypadku ładowania fragmentów klatek jedyna różnica polega na stałym rozmiarze danych: 132x132 pikseli.

Listing 15. Implementacja ładowania danych ramek

```

1 def load_data(self, file_path: str, metadata: Metadata) -> np.ndarray:
2     with open(file_path, "rb") as f:
3         data = np.frombuffer(f.read(), dtype=np.uint8)
4
5     data = np.resize(data, (metadata.height, metadata.width, 3))
6     return data

```

Do treningu sieci wymagane są zarówno fragmenty danych po kompresji, jak i przed, oraz informacje dotyczące tej ramki zawarte w klasie `metadata`. Jako, że dane przechowywane zgodnie z założoną konwencją (opisanej w rozdziale 4.4.3), dostępne dane listowane są za pomocą pakietu `Glob`, pozwalającego na listowanie danych na podstawie tzw. `globów`, czyli wyrażeń regularnych znanych z takich programów jak `ls` i zgodnych z ogólnie przyjętą składnią dla podstawowych pakietów systemu Linux. Przykład listowania wszystkich dostępnych fragmentów został przedstawiony na listingu 16. Listy dostępnych fragmentów w danym folderze i jego podfolderach stanowią zbiór danych.

Listing 16. Implementacja listowania dostępnych fragmentów

```

1 class FrameDataset(torch.utils.data.Dataset):
2     FRAME_GLOB = "{folder}/*_*_*/*.png"
3
4     def __init__(self,
5                  settings: DatasetConfig,
6                  ) -> None:
7         super(self). __init__()
8         self.frame_files = glob(
9             self.FRAME_GLOB.format(
10                 folder=settings.decompressed_folder,
11             )
12         )
13
14     def __len__(self) -> int:
15         return len(self.frame_files)

```

7. Szczegóły implementacyjne

Następnie, dane o określonym indeksie ładowane są przy użyciu funkcji `__getitem__`, gdzie na podstawie ścieżki danych po kompresji obliczane są metadane, następnie ustalana jest lokalizacja danych oryginalnych, na podstawie przyjętego formatu rozmieszczenia danych na dysku. Implementacje tej procedury przedstawiono na listingu 17.

Listing 17. Implementacja dostępu do konkretnych elementów zbioru danych

```
1 FRAME_NAME = "{file} / "
2     "{profile}_QP{qp:d}_ALF{alf:d}_DB{db:d}_SAO{sao:d} / "
3     "{frame}_{is_intra}.png"
4 ORIG_FRAME_NAME = "{file}/{frame}.png"
5
6 def __getitem__(self, idx: int):
7     frame_file = self.frame_files[idx]
8     metadata = self.get_metadata(frame_file)
9
10    orig_frame_name = self.ORIG_FRAME_NAME.format(
11        file=metadata.file,
12        frame=metadata.frame,
13    )
14
15    return (
16        self.frame_transform(self.load_data(frame_file, metadata)),
17        self.frame_transform(self.load_data(orig_frame_name, metadata)),
18        self.metadata_transform(metadata),
19    )
```

Warto dodać, że na wszystkich załadowanych danych wykonywane są odpowiednie transformacje, przetwarzające obiekty do obiektów `pytorch.Tensor`. Zostały one przedstawione na listingu 18.

Listing 18. Implementacja transformacji danych do obiektów `pytorch.Tensor`

```
1 def frame_transform(self):
2     return transforms.ToTensor()
3
4 def metadata_transform(self, metadata: Metadata) -> torch.Tensor:
5     return torch.as_tensor(metadata).float().view(len(metadata), 1, 1)
```

Moduł PyTorch Lightning DataModule W celu ułatwienia zarządzania danymi, zdecydowano się na wykorzystanie modułu `LightningDataModule` z biblioteki PyTorch Lightning. Pozwala on na oddzielenie logiki biznesowej modelu od kodu inżynierijnego, co ułatwia zarządzanie skomplikowanymi projektami i współpracę w zespołach. W przypadku niniejszej pracy, moduł ten został zaimplementowany w klasie `Dataset`, która agreguje

wszystkie klasy odpowiedzialne za ładowanie danych. Najważniejsze szczegóły implementacji tej klasy zostały przedstawione na listingu 19.

Listing 19. Implementacja modułu zarządzającego danymi

```

1  def setup(self, stage=None):
2      if stage == "fit":
3          self.dataset_train = ChunkDataset(
4              settings=self.dataset_config.train,
5          )
6          self.dataset_val = ChunkDataset(
7              settings=self.dataset_config.val,
8          )
9
10     if stage in ("test", "predict"):
11         if self.test_full_frames:
12             self.dataset_test = FrameDataset(
13                 settings=self.dataset_config.test,
14             )
15         else:
16             self.dataset_test = ChunkDataset(
17                 settings=self.dataset_config.test,
18             )
19
20     def train_dataloader(self):
21         data_loader = DataLoader(
22             self.dataset_train,
23             batch_size=self.config.batch_size,
24             shuffle=True,
25             pin_memory=True,
26             num_workers=os.cpu_count(),
27         )
28         return data_loader

```

Przy rozpoczęciu eksperymentu na tej klasie wołana jest metoda `setup(stage)`, która odpowiada za zainicjowanie obiektów zbiorów danych, odpowiednich do trybu treningu, np. "fit" dla operacji treningu modeli. Następnie, moduł główny ładuje odpowiednie obiekty typu `DataLoader`, które zarządzają podziałem całego zbioru na serie treningowe.

7.2.4. Implementacja modułu głównego

Implementacja modułu głównego w przypadku PyTorch Lightning polega na zdefiniowaniu klasy dziedziczącej po `LightningModule`, która definiuje metody kroków treningu, walidacji i predykcji oraz konfiguracji optymalizatorów i metod dostosowywania *lr*. W przypadku niniejszej pracy, zdecydowano się na implementację modułu w postaci klasy `TrainerModule`, agregującą wszystkie komponenty sieci w jedną klasę.

7. Szczegóły implementacyjne

Na listingu 20 przedstawiono implementację głównej funkcji modułu odpowiadającą za pojedynczy krok treningowy. Warto zwrócić uwagę na wykorzystanie funkcji `self.optimizers()`, zwracającą listę optymalizatorów zdefiniowanych w metodzie `configure_optimizer`. Dzięki temu możliwe jest wykorzystanie wielu optymalizatorów w jednym modelu, okazuje się to istotne w przypadku wykorzystania sieci GAN, gdzie stosuje się osobne optymalizatory dla generatora i dyskryminatora.

Dodatkowo, aby umożliwić selektywne trenowanie tylko generatora lub dyskryminatora, zdecydowano się na wykorzystanie specjalnych flag. Sprawia to, że można użyć jednego modułu do trenowania zarówno generatora, jak i dyskryminatora, a także zastosowanie go do trenowania tylko jednego z nich. Sterując tym flagi `train_enhancer` oraz `train_discriminator`, których wartość obliczana jest w funkcji `what_to_train` i zależy od charakteru eksperymentu. Umożliwia to np. rozdzielny trening obu sieci lub selektywne wyłączanie trenowania jednej z sieci w przypadku, gdy dysproporcja pomiędzy sieciami jest zbyt duża, co w przypadku treningu GAN określa się mianem zapaści trybów (ang. *mode collapse*).

Listing 20. Implementacja głównej funkcji modułu odpowiadającą za pojedyńczy krok treningowy

```
1 def training_step(self, batch, batch_idx):
2     g_opt, d_opt = self.optimizers()
3     chunks, orig_chunks, metadata, _ = batch
4
5     train_enhancer, train_discriminator = self.what_to_train()
6
7     # train generator - enhancer
8     if train_enhancer:
9         g_opt.zero_grad()
10        enhanced, preds, g_loss = self.g_step(chunks, orig_chunks, metadata)
11        fake_chunks = enhanced.detach()
12        self.manual_backward(g_loss)
13        g_opt.step()
14
15    elif train_enhancer is not None:
16        # None means we are training only discriminator
17        enhanced, preds, g_loss = self.g_step(chunks, orig_chunks, metadata)
18        fake_chunks = enhanced.detach()
19
20    else:
21        fake_chunks = chunks
22        enhanced = None
23
24    # train discriminator
25    if train_discriminator:
26        d_opt.zero_grad()
```

```

27     fake_preds, real_preds, d_loss = self.d_step(fake_chunks, orig_chunks)
28     self.manual_backward(d_loss)
29     d_opt.step()
30
31 elif train_discriminator is not None:
32     fake_preds, real_preds, d_loss = self.d_step(fake_chunks, orig_chunks)
33
34 else:
35     real_preds = None
36     fake_preds = preds
37
38 if batch_idx % 100 == 0:
39     self.log_images(
40         enhanced,
41         chunks,
42         orig_chunks,
43         fake_preds,
44         real_preds,
45     )

```

Kroki treningowe dla generatora i dyskryminatora zostały zaimplementowane w metodach `g_step` oraz `d_step`. Na listingu 21 przedstawiono implementację kroku dla generatora, natomiast na listingu 22 dla dyskryminatora. Obie funkcje zostały zaimplementowane w taki sposób, aby umożliwić zarówno trening w metodologii GAN, jak i samodzielny każdej z sieci - czy to generatora, czy dyskryminatora.

Listing 21. Implementacja kroku dla generatora

```

1 def g_step(self, chunks, orig_chunks, metadata, stage="train"):
2     enhanced = self(chunks, metadata)
3
4     # ground truth result (ie: all fake)
5     # put on GPU because we created this tensor inside training_loop
6     valid = torch.ones(chunks.size(0), 1)
7     valid = valid.type_as(chunks)
8
9     if (
10         self.mode == TrainingMode.GAN
11         and self.current_epoch > self.separation_epochs
12     ):
13         preds = self.discriminator(enhanced)
14         g_loss = self.g_gan_loss(enhanced, orig_chunks, preds, valid)
15     else:
16         g_loss = self.g_solo_loss(enhanced, orig_chunks)
17

```

7. Szczegóły implementacyjne

```
18     return enhanced, preds, g_loss
```

Listing 22. Implementacja kroku dla dyskryminatora

```
1 def d_step(self, fake_chunks, orig_chunks, stage="train"):
2     valid = torch.ones(orig_chunks.size(0), 1)
3     valid = valid.type_as(orig_chunks)
4     real_pred = self.discriminator(orig_chunks)
5     real_loss = self.adversarial_loss(real_pred, valid)
6
7     # how well can it label as fake?
8     fake = torch.zeros(orig_chunks.size(0), 1)
9     fake = fake.type_as(orig_chunks)
10
11    fake_preds = self.discriminator(fake_chunks)
12    fake_loss = self.adversarial_loss(fake_preds, fake)
13    fake_accuracy = accuracy(fake_preds, fake, task="binary")
14
15    acc = (real_accuracy + fake_accuracy) / 2
16
17    # discriminator loss is the average of these
18    d_loss = (real_loss + fake_loss) / 2
19
20    return fake_preds, real_pred, d_loss
```

7.3. Konfiguracja eksperymentów

Aby umożliwić proste odtwarzanie eksperymentów oraz modyfikację parametrów treningu i architektur sieci bez konieczności ingerencji w kod źródłowy, wykorzystano bibliotekę Pydantic Settings [36]. Pozwala ona na łatwe definiowanie konfiguracji eksperymentów w postaci klas, które następnie mogą być ładowane z plików YAML. W ten sposób możliwe jest zdefiniowanie konfiguracji eksperymentu w postaci pliku YAML, a następnie ładowanie go do obiektu klasy `Settings`, który może być wykorzystany w kodzie źródłowym. Pozwala to na łatwe modyfikowanie parametrów eksperymentu bez konieczności ingerencji w kod źródłowy. Dodatkowo, biblioteka ta pozwala na sprawdzanie poprawności konfiguracji, co jest szczególnie istotne w przypadku złożonych konfiguracji eksperymentów.

Listing 23. Przykładowy plik konfiguracyjny eksperymentu

```
1 test_full_frames: true
2 trainer:
3     mode: enhancer
4     enhancer:
5         epochs: 1000
```

```

6   enhancer_scheduler_gamma: 0.5
7   enhancer_scheduler_milestones: [50, 100, 150, 200, 300, 400, 500, 600, 800]
8   saved_chunk_folder: enhanced/conv
9 enhancer:
10  implementation: conv
11  save_to: 'experiments/enhancer/conv.pth'
12  features:
13    kernel_size: 9
14    padding: 4
15    stride: 1
16    features: 64
17    pool: false
18  structure:
19    blocks:
20      - kernel_size: 7
21        padding: 3
22        stride: 1
23        features: 64
24      - kernel_size: 5
25        padding: 2
26        stride: 1
27        features: 96
28      - kernel_size: 3
29        padding: 1
30        stride: 1
31        features: 64
32      - kernel_size: 3
33        padding: 1
34        stride: 1
35        features: 48
36      - kernel_size: 3
37        padding: 1
38        stride: 1
39        features: 32
40  output_block:
41    kernel_size: 3
42    padding: 1
43    stride: 1
44    features: 3
45    tanh: false

```

Na listingu 23 przedstawiono przykładową konfigurację eksperymentu. W pierwszej kolejności określone są globalne ustawienia, takie jak tryb treningu, czy zapisywanie pełnych ramek (w przypadku uruchomienia procedury predykcji). Następnie w sekcji

7. Szczegóły implementacyjne

enhancer zdefiniowane są parametry treningu generatora, takie jak liczba epok, czy parametry optymalizatora. Sekcji enhancer określa parametry sieci generatora, na przykład strukturę sieci, czy parametry warstwy wyjściowej. W ten sposób możliwe jest łatwe modyfikowanie parametrów eksperymentu oraz architektur sieci bez konieczności ingerencji w kod źródłowy.

7.4. Proces predykcji i obliczania metryk danych po poprawie jakości

Korzystając z modułu trenującego Pytorch Lightning, oprócz metod treningowych można zdefiniować również metody predykcji i walidacji. W przypadku niniejszej pracy zdecydowano się na zdefiniowanie metody predykcji, która pozwala na obliczenie metryk jakości danych po poprawie jakości. W tym celu zdefiniowano metodę `predict_step`, która pozwala na przeprowadzenie predykcji na zbiorze testowym. W przypadku, gdy w konfiguracji eksperymentu ustaliona jest flaga `test_full_frames` na wartość `true`, metoda ta zapisuje pełne ramki po poprawie jakości. W przeciwnym wypadku zapisywane są tylko fragmenty CTU po poprawie jakości. Implementacja tej metody została przedstawiona na listingu 24.

Listing 24. Implementacja funkcji predykcji

```
1 def predict_step(self, batch, batch_idx):
2     chunks, _, metadata, chunk_objs = batch
3
4     # call forward on enhancer
5     enhanced = self(chunks, metadata)
6
7     for i, chunk_data in enumerate(enhanced):
8         chunk = [c[i].cpu() if hasattr(c[i], "cpu") else c[i] for c in chunk_objs]
9         if self.test_full_frames:
10             FrameDataset.save_frame(
11                 chunk, chunk_data.cpu().numpy(), self.config.saved_chunk_folder
12             )
13         else:
14             VVC Dataset.save_chunk(
15                 chunk, chunk_data.cpu().numpy(), self.config.saved_chunk_folder
16             )
```

Za odpowiedni zapis danych po poprawie jakości odpowiada metoda `save_chunk` zaimplementowana w module odpowiadającym za przetwarzanie zbioru danych. Zapisuje ona dane wyjściowe w formacie binarnym.

Ponieważ dane zapisywane są postaci fragmentów lub całości ramek, konieczne jest ich połączenie w sekwencję wideo. W tym celu wykorzystano prosty skrypt w języku Python, którego główną pętlę przedstawiono na listingu 25. W przypadku przetwarzania ramek w całości, w ścieżce pliku załączana jest informacja dotycząca rozdzielczości wideo

po sekwencji `__`, dzięki czemu możliwe jest rozpoznanie przypadków, w których konieczne jest łączenie fragmentów ramek w całość.

Listing 25. Implementacja łączenia klatek po poprawie jakości w sekwencję wideo

```

1  for movie in tqdm(os.listdir(ROOT)):
2      frames = [None] * 64
3      for params in tqdm(os.listdir(os.path.join(ROOT, movie))):
4          if not os.path.isdir(os.path.join(ROOT, movie, params)):
5              continue
6
7      frame_paths = os.listdir(os.path.join(ROOT, movie, params))
8
9      for frame in frame_paths:
10         if "__" not in movie:
11             image = join_frame(movie, params, frame)
12         else:
13             image = read_frame(movie, params, frame)
14
15         frames[int(frame.split("__")[0])] = image.astype(np.ubyte)
16
17     target = f"{ROOT}/{movie}/{params}.yuv"
18     with open(target, "wb") as f:
19         for frame in frames:
20             Y, U, V = cv2.split(frame)
21             f.write(Y.tobytes())
22             f.write(V.tobytes())
23             f.write(U.tobytes())

```

Tak przetworzone dane zapisane są w formacie YUV444, konieczna jest zatem ich konwersja do formatu YUV420, aby móc porównać je z oryginałem. W tym celu wykorzystano narzędzie FFmpeg [37], które pozwala na konwersję plików wideo. Podobnie jak w przypadku przygotowania plików do przetwarzania przez sieci neuronowe opisanego w rozdziale 4.4.3, użyta metoda wykonuje interpolację liniową dla składowy U i V.

Kolejnym etapem przetwarzania danych po poprawie jakości jest obliczenie dla nich metryk PSNR i SSIM. W związku z tym ponownie skorzystano z narzędzia FFmpeg, które pozwala na obliczenie tych metryk dla dwóch plików wideo. Przykładowe wywołanie tego narzędzia przedstawiono na listingu 26. Podając odpowiednio `ssim` oraz `psnr` jako parametr `\$FILTER` obliczane są wartości dla podanej metryki, a wynik zapisywany jest do pliku tekstowego `\$LOGFILE`, skąd z łatwością można odczytać te wartości.

Listing 26. Przykładowe wywołanie narzędzia FFmpeg w celu obliczenia metryk PSNR i SSIM dla dwóch plików wideo

```

1  ffmpeg \\
2      -pix_fmt yuv420p \

```

7. Szczegóły implementacyjne

```
3 -s ${RES} \
4 -i ${REFFILE} \
5 -pix_fmt yuv420p \
6 -s ${RES} \
7 -i ${INFILE} \
8 -filter_complex "${FILTER}" \
9 -f null /dev/null \
10 > ${LOGFILE} \
11 2>&1
```

Powyższe skrypty wykonywane są pojedynczo dla każdego pliku wideo, co jest czasochłonnym procesem. Aby go przyspieszyć, zdecydowano się na wykorzystanie narzędzia do równoległego wykonywania operacji przedstawionego w rozdziale poświęconym przetwarzaniu zbioru danych 4.4.2.

Dla każdego pliku wideo tworzony jest plik z logami zawierającymi wartości metryk PSNR i SSIM dla całej sekwencji. Ostatnim etapem przetwarzania staje się zebranie tych wyników w jednym pliku CSV. W tym celu wykorzystano bibliotekę Pandas [38], pozwalającą na wykonywanie różnych operacji na danych w formacie CSV. W ten sposób, możliwe jest łatwe porównanie wyników pomiędzy eksperymentami.

8. Uzyskane wyniki

Na rysunku 8.1 przedstawiono efekty poprawy jakości pierwszej klatki dla sekwencji Johnny, zakodowanej profilem RA (jest to zatem ramka intra), $qp = 47$, z wyłączonymi wszystkimi filtrami. Wszystkie sieci znakomicie poradziły sobie z redukcją najbardziej widocznych artefaktów blokowych. Maski różnicowe zostały przedstawione na rysunku 8.2, im jaśniejszy piksel tym większa różnica pomiędzy fragmentem oryginalnym a rekonstrukcją po kompresji. Szczególną uwagę należy zwrócić na obszary przedstawiające nos, policzki oraz czoło prezentera, które najlepiej obrazują różnice efektów poprawy uzyskanych przez różne sieci.

Porównanie wyników uzyskanych dla różnych architektur sieci z pierwszego etapu eksperymentu (trening bez zastosowania metodologii GAN) przy profilu kodowania RA przedstawiono w tabeli 8.1 oraz 8.2, dla AI. Wartości metryk dla obu profili potwierdziły, że architektura sieci i wykorzystanie połączeń skrótów ma znaczący wpływ na poprawę jakości po kompresji. Najlepiej w tym zadaniu poradził sobie model DenseNet, uzyskując nawet -7.79% BD-BR dla profilu RA i -12.41% dla AI dla kanału Y. Sieć ResNet wykazała odpowiednio -5.83% i -10.78%, natomiast konwolucyjna, bez jakichkolwiek połączeń skrótu, -1.99% dla RA i -6.86% dla AI.

Architektura	BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
	Y	U	V		Y	U	V	
DenseNet	-7.79	-10.33	-7.70	-7.21	0.46	0.25	0.19	0.0039
ResNet	-5.83	-8.19	-7.47	-3.27	0.38	0.20	0.21	0.0019
Konwolucyjna	-1.99	-5.48	-1.14	-3.70	0.15	0.08	0.06	0.0020

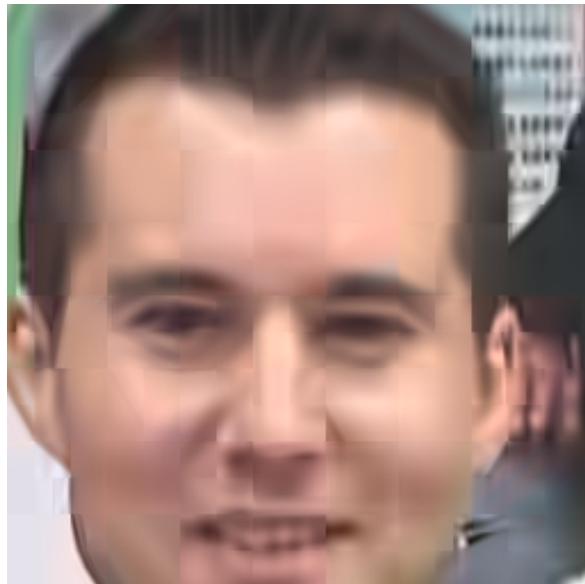
Tabela 8.1. Porównanie wyników w zależności od architektury sieci dla profilu RA

Architektura	BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
	Y	U	V		Y	U	V	
DenseNet	-12.41	-13.38	-10.00	-11.17	0.70	0.26	0.20	0.0053
ResNet	-10.78	-11.58	-9.81	-7.51	0.66	0.25	0.23	0.0036
Konwolucyjna	-6.86	-3.30	-0.92	-7.51	0.39	0.03	0.01	0.0035

Tabela 8.2. Porównanie wyników w zależności od architektury sieci dla profilu AI

8.1. Wyniki dla treningu w metodologii GAN

W drugiej części eksperymentu przeprowadzono trening sieci w metodologii GAN. W tym celu wykorzystano architekturę DenseNet, która osiągnęła najlepsze wyniki w pierwszej części badania.



(a) klatka po kompresji



(b) poprawa jakości siecią DenseNet



(c) poprawa jakości siecią ResNet

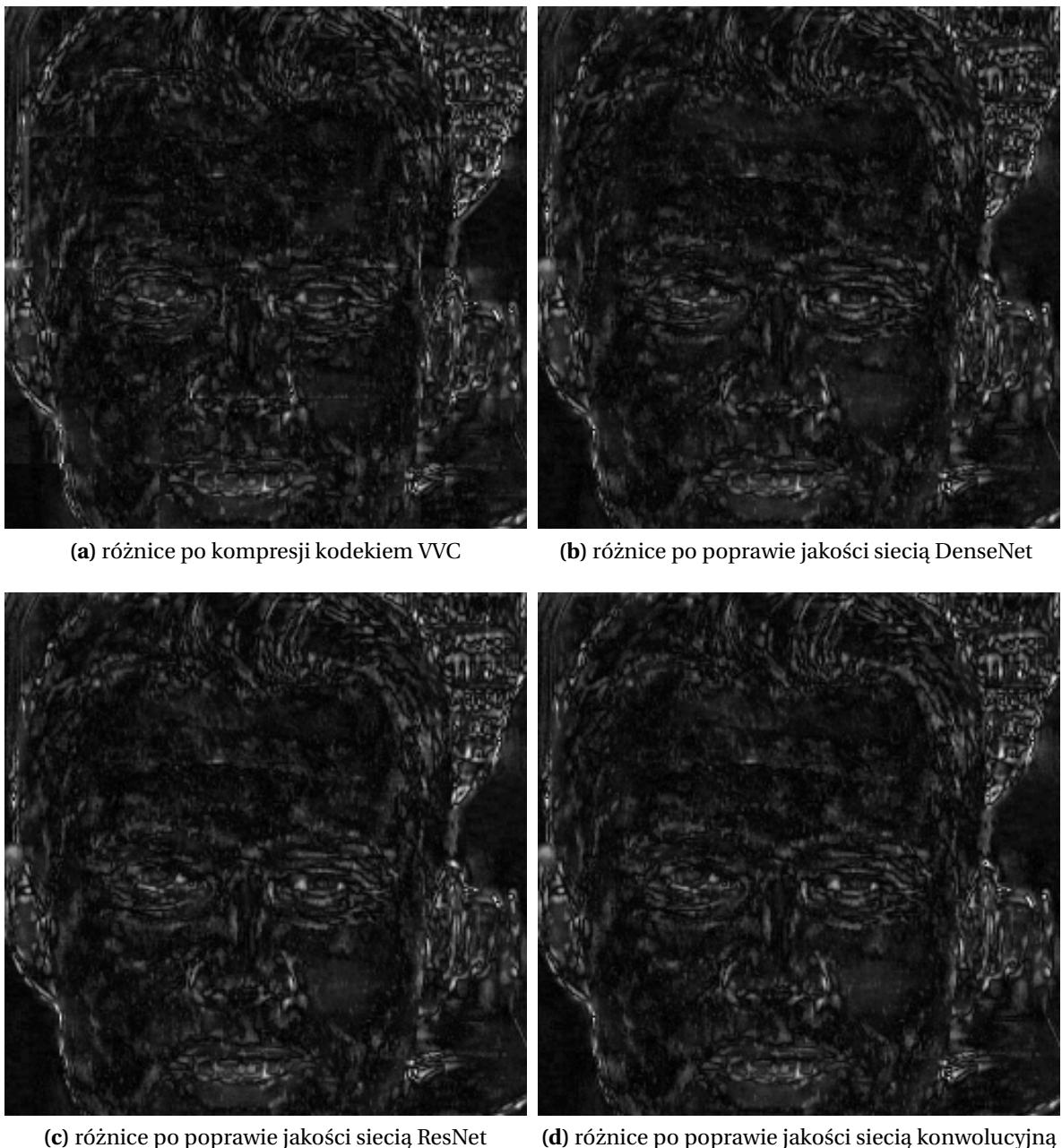


(d) poprawa jakości siecią konwolucyjną

Rysunek 8.1. Porównanie efektów poprawy jakości dla różnych architektur sieci dla sekwencji Johnny ($qp = 47$, filtry wyłączone, profil RA), przybliżenie na fragment 256×256 zawierający głowę prezentera

W tabeli 8.3 przedstawiono pełne wyniki modelu wytrenowanego w metodologii GAN dla profilu RA, natomiast w tabeli 8.4, dla profilu AI. W porównaniu do rezultatów uzyskanych w pierwszej części eksperymentu, odnotowano znaczącą ich poprawę. W przypadku profilu RA dla BD-BR (PSNR-Y) uzyskano wynik lepszy o 2%. Dla obu profili zaobserwowano zdecydowanie lepsze wartości metryk otrzymane dla kanałów U i V PSNR oraz SSIM.

Na rysunku 8.3 przedstawiono porównanie poprawy jakości sieci DenseNet wytreno-



Rysunek 8.2. Porównanie maski różnic pomiędzy oryginałem, ramką po dekompresji i poprawie jakości badanymi architekturami sieci dla sekwencji Johnny ($qp = 47$, filtry wyłączone, profil AI), przybliżenie na fragment $256x256$ zawierający głowę prezentera

wanej z wykorzystaniem metodologii GAN oraz bez, umieszczono też fragment oryginalny oraz po kompresji. Maski różnicowe przedstawione na rysunku 8.4 pokazują, że faktycznie przy użyciu metodologii GAN poprawa fragmentów pozwala uzyskać efekt bliższy oryginalowi i lepiej radzi sobie ona z redukcją artefaktów.

Wykres 8.5 przedstawia wzmacnianie PSNR w zależności od QP dla obu profili kodowania. W przypadku AI poprawa PSNR maleje wraz z parametrem QP, najlepsze wyniki uzyskano przy QP=32, a najgorsze - QP=47. W profilu RA zaobserwowano podobny trend

8. Uzyskane wyniki

Sekwencja	BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
	Y	U	V		Y	U	V	
stefan	-10.72	-15.04	-12.34	-5.81	0.55	0.42	0.37	0.0056
tractor	-10.43	-16.79	-13.57	-7.07	0.44	0.40	0.34	0.0051
Johnny	-8.93	-9.54	-7.70	-15.20	0.42	0.30	0.21	0.0045
bridge far	-8.69	-35.82	-39.90	-13.63	0.46	0.33	0.25	0.0042
students	-6.45	-8.29	-7.03	-3.02	0.44	0.35	0.30	0.0044
Średnia	-9.04	-17.10	-16.11	-8.94	0.46	0.36	0.29	0.0048

Tabela 8.3. Wyniki uzyskane dla sieci wytrenowanej w metodologii GAN, profil RA

Sekwencja	BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
	Y	U	V		Y	U	V	
bridge far	-17.53	-27.86	-23.72	-22.76	0.62	0.40	0.35	0.0057
Johnny	-12.35	-22.27	-21.30	-21.23	0.62	0.39	0.33	0.0060
tractor	-12.14	-29.88	-25.32	-10.94	0.63	0.52	0.47	0.0064
stefan	-11.53	-26.94	-34.76	-8.50	0.90	0.62	0.58	0.0079
students	-10.07	-23.37	-21.81	-5.48	0.63	0.45	0.42	0.0062
Średnia	-12.72	-26.06	-25.38	-13.78	0.68	0.48	0.43	0.0065

Tabela 8.4. Wyniki uzyskane dla sieci wytrenowanej w metodologii GAN, profil AI

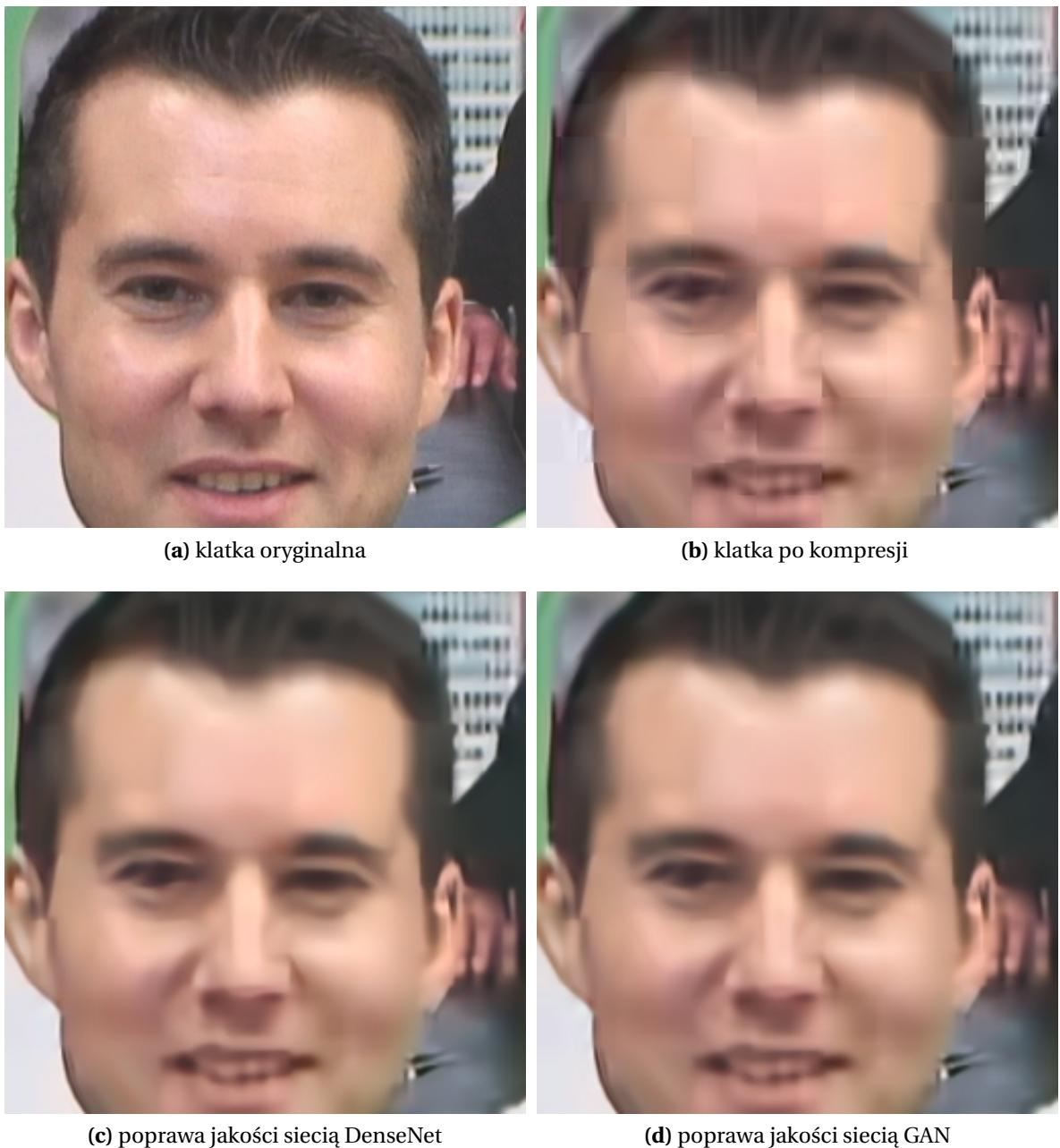
dla wartości QP z zakresu 32-42, natomiast przy QP=47 uzyskano delikatnie lepsze wzmacnienie, niż w przypadku QP=42.

8.2. Porównanie z filtrami wbudowanymi w kodek VVC

W tabelach 8.5 oraz 8.6 przedstawiono wyniki uzyskane przy wykorzystaniu różnych filtrów wbudowanych w kodek VVC, uśrednione dla wszystkich sekwencji testowych. Największą poprawę jakości zaobserwowano, gdy wyłączono wszystkie filtry.

filtry			BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
SAO	ALF	DB	Y	U	V		Y	U	V	
wył.	wył.	wył.	-9.94	-19.20	-17.80	-10.97	0.50	0.39	0.33	0.0059
wył.	wył.	wł.	-9.47	-17.90	-16.89	-9.04	0.48	0.37	0.31	0.0048
wł.	wył.	wył.	-9.23	-17.77	-17.06	-10.36	0.46	0.36	0.30	0.0055
wł.	wył.	wł.	-8.86	-16.61	-16.24	-8.64	0.45	0.34	0.29	0.0045
wył.	wł.	wł.	-8.84	-16.35	-15.33	-7.90	0.46	0.36	0.29	0.0042
wł.	wł.	wł.	-8.76	-15.99	-15.19	-7.88	0.46	0.35	0.28	0.0042
wył.	wł.	wył.	-8.68	-16.68	-15.14	-8.46	0.45	0.35	0.28	0.0045
wł.	wł.	wył.	-8.58	-16.30	-15.23	-8.31	0.44	0.34	0.27	0.0044

Tabela 8.5. Uzyskane wyniki przy użyciu różnych filtrów wbudowanych w kodek VVC, profil RA



Rysunek 8.3. Porównanie efektów poprawy jakości dla architektury DenseNet oraz GAN, sekwencja Johnny ($qp = 47$, klatka I, filtry wyłączone, profil RA), przybliżenie na fragment 256×256 zawierający głowę prezentera

W tabelach 8.7 oraz 8.8 przedstawiono porównanie wyników uzyskanych przez proponowane rozwiązanie z wynikami uzyskanymi przy użyciu filtrów wbudowanych w kodek VVC (SAO, ALF i DB) dla wszystkich ich kombinacji. W obu przypadkach proponowana sieć osiągnęła zdecydowanie najlepsze wyniki BD-BR dla wszystkich kanałów.

W tabelach 8.9 oraz 8.10 zaprezentowano porównanie poprawy jakości w stosunku do wyłączonych wszystkich filtrów, dla obu profili kodowania. Wskazuję one, że sieć jest w stanie uzyskać najlepsze efekty wizualne, kiedy używa się jej razem z filtrem DB.

8. Uzyskane wyniki

filtry			BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
SAO	ALF	DB	Y	U	V		Y	U	V	
wył.	wył.	wył.	-16.20	-30.98	-29.55	-17.80	0.86	0.59	0.54	0.0086
wył.	wył.	wł.	-15.33	-29.71	-28.78	-15.65	0.82	0.56	0.53	0.0074
wł.	wł.	wył.	-12.18	-24.84	-23.88	-13.19	0.66	0.46	0.40	0.0062
wył.	wł.	wył.	-12.12	-24.54	-23.10	-13.18	0.66	0.46	0.39	0.0062
wł.	wł.	wł.	-11.87	-25.08	-24.42	-12.61	0.66	0.47	0.42	0.0058
wył.	wł.	wł.	-11.84	-24.85	-23.89	-12.57	0.66	0.47	0.41	0.0058
wł.	wył.	wył.	-11.61	-24.98	-25.27	-13.62	0.58	0.42	0.38	0.0064
wł.	wył.	wł.	-10.63	-23.54	-24.15	-11.65	0.53	0.38	0.36	0.0053

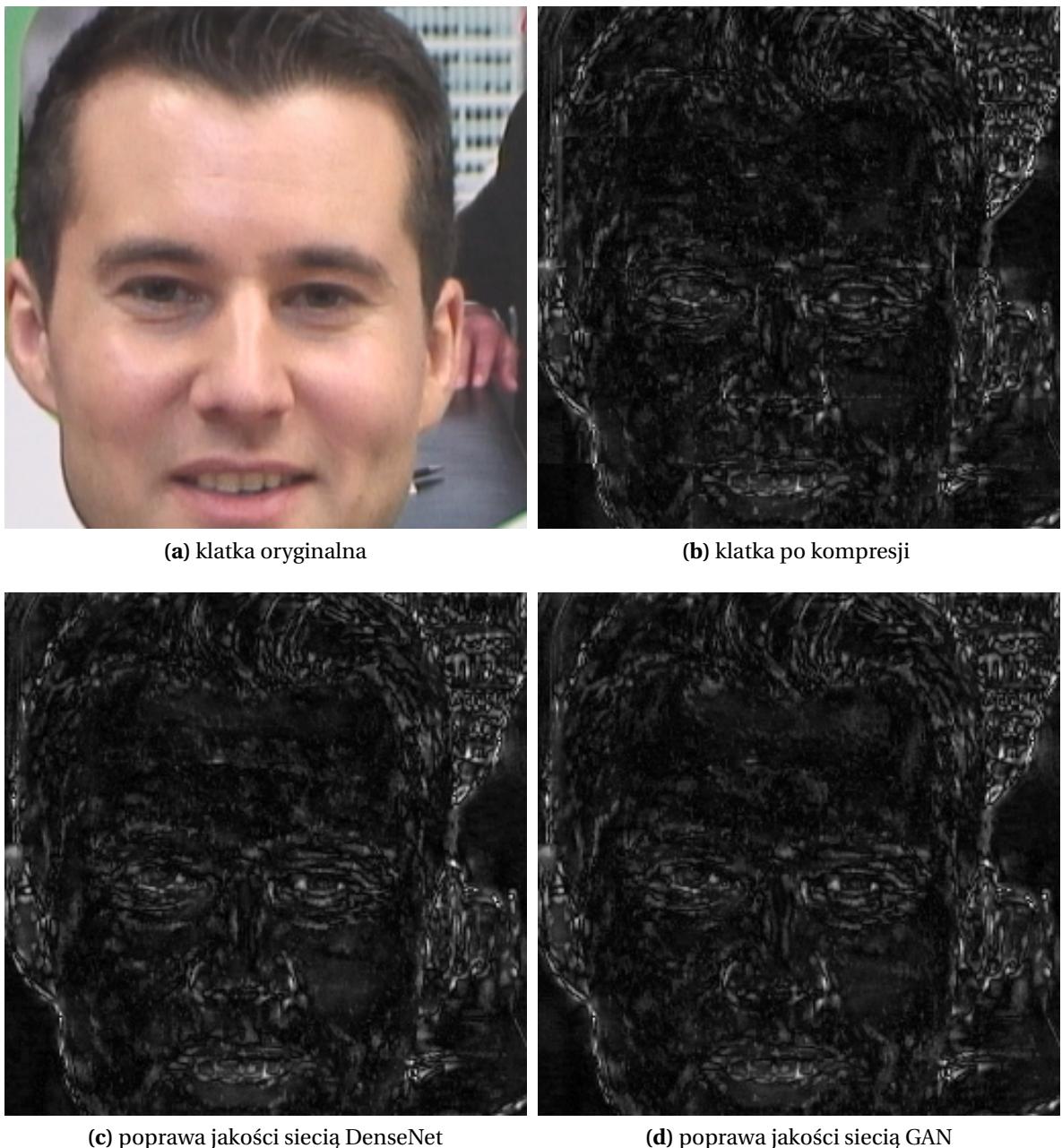
Tabela 8.6. Uzyskane wyniki przy użyciu różnych filtrów wbudowanych w kodek VVC, profil AI

filtry			BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
SAO	ALF	DB	Y	U	V		Y	U	V	
wył.	wł.	wł.	-3.13	-4.77	-4.90	-3.93	0.13	0.13	0.12	0.0027
wł.	wł.	wł.	-3.07	-4.53	-4.39	-3.84	0.13	0.12	0.11	0.0026
wył.	wył.	wł.	-2.67	-2.91	-3.80	-3.17	0.11	0.07	0.09	0.0021
wł.	wył.	wł.	-2.48	-2.63	-3.47	-2.65	0.10	0.06	0.08	0.0018
wł.	wł.	wył.	-1.81	-2.77	-2.25	-2.26	0.07	0.09	0.07	0.0014
wył.	wł.	wył.	-1.66	-2.84	-2.35	-2.21	0.07	0.09	0.07	0.0014
wył.	wył.	wył.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0000
wł.	wył.	wył.	0.07	0.37	0.18	0.26	-0.00	-0.00	0.00	-0.0001
proponowana sieć			-9.94	-19.20	-17.80	-10.97	0.50	0.39	0.33	0.0059

Tabela 8.7. Porównanie poprawy jakości przy użyciu filtrów wbudowanych w kodek VVC oraz proponowanej sieci, profil RA

filtry			BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
SAO	ALF	DB	Y	U	V		Y	U	V	
wł.	wł.	wł.	-2.78	-5.98	-7.16	-3.78	0.14	0.16	0.20	0.0022
wył.	wł.	wł.	-2.67	-6.08	-7.25	-3.77	0.14	0.16	0.21	0.0022
wł.	wł.	wył.	-2.62	-4.40	-5.09	-3.24	0.14	0.15	0.17	0.0019
wył.	wł.	wył.	-2.41	-4.49	-5.16	-3.23	0.12	0.15	0.17	0.0019
wł.	wył.	wł.	-1.41	-4.12	-5.16	-2.08	0.07	0.09	0.13	0.0012
wył.	wył.	wł.	-1.27	-4.21	-5.23	-2.13	0.06	0.10	0.13	0.0012
wł.	wył.	wył.	-0.29	0.11	0.08	0.08	0.02	-0.00	-0.00	-0.0000
wył.	wył.	wył.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0000
proponowana sieć			-16.20	-30.98	-29.55	-17.80	0.86	0.59	0.54	0.0086

Tabela 8.8. Porównanie poprawy jakości przy użyciu filtrów wbudowanych w kodek VVC oraz proponowanej sieci, profil AI



Rysunek 8.4. Porównanie efektów poprawy jakości dla architektury DenseNet oraz GAN, sekwencja Johnny ($qp = 47$, klatka I, filtry wyłączone, profil RA), przybliżenie na fragment 256×256 zawierający głowę prezentera

Dla profilu AI proponowane rozwiązanie świetnie radzi sobie w przypadku wyłączenia wszystkich filtrów.

8.3. Porównanie uzyskanych wyników na tle literatury

W tabeli 8.11 przedstawiono porównanie wyników uzyskanych przez proponowane rozwiązanie dla profilu RA przy wszystkich filtrach włączonych, z wynikami dwóch naj-

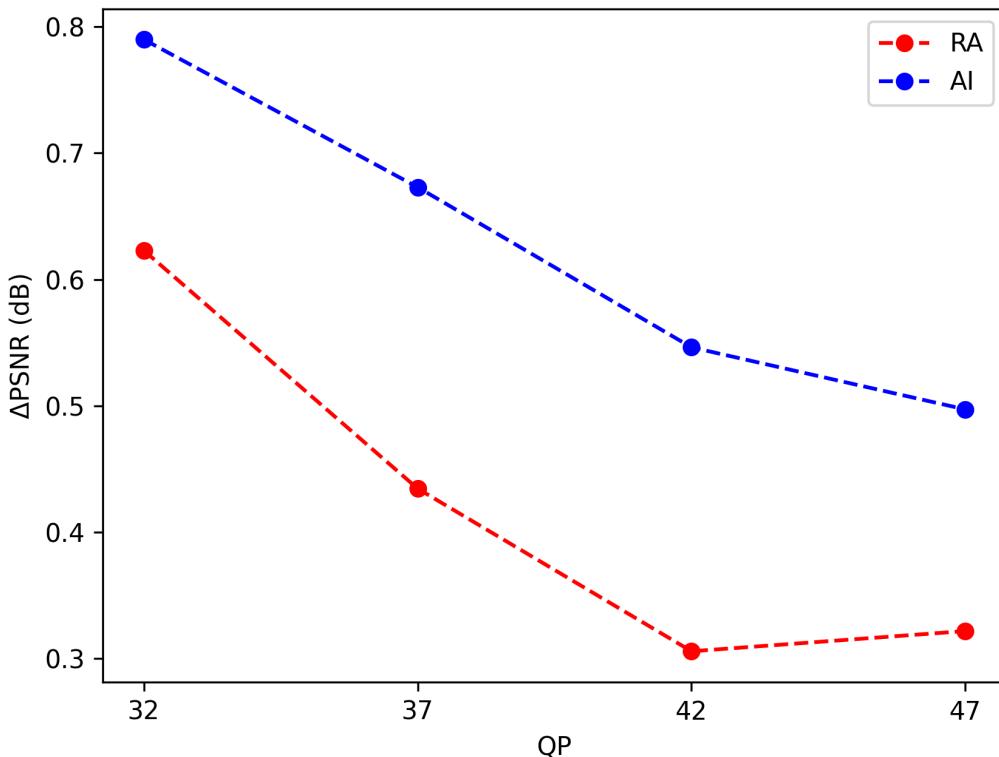
8. Uzyskane wyniki

filtry			BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
SAO	ALF	DB	Y	U	V		Y	U	V	
wył.	wył.	wł.	-12.13	-20.82	-20.69	-12.21	0.59	0.44	0.40	0.0069
wył.	wł.	wł.	-11.97	-21.12	-20.22	-11.83	0.60	0.49	0.42	0.0069
wł.	wł.	wł.	-11.83	-20.52	-19.57	-11.72	0.59	0.48	0.39	0.0068
wł.	wył.	wł.	-11.35	-19.24	-19.71	-11.29	0.55	0.41	0.37	0.0063
wł.	wł.	wył.	-10.39	-19.07	-17.48	-10.58	0.52	0.43	0.34	0.0059
wył.	wł.	wył.	-10.34	-19.52	-17.50	-10.66	0.51	0.44	0.35	0.0059
wył.	wył.	wył.	-9.94	-19.20	-17.80	-10.97	0.50	0.39	0.33	0.0059
wł.	wył.	wył.	-9.17	-17.40	-16.88	-10.09	0.46	0.35	0.30	0.0054

Tabela 8.9. Porównanie poprawy jakości w stosunku do wyłączeniowych wszystkich filtrów, profil RA

filtry			BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
SAO	ALF	DB	Y	U	V		Y	U	V	
wył.	wył.	wł.	-16.60	-33.92	-34.01	-17.77	0.88	0.66	0.66	0.0086
wył.	wył.	wył.	-16.20	-30.98	-29.55	-17.80	0.86	0.59	0.54	0.0086
wł.	wł.	wył.	-14.81	-29.24	-28.97	-16.43	0.80	0.60	0.57	0.0081
wł.	wł.	wł.	-14.66	-31.07	-31.58	-16.39	0.80	0.63	0.62	0.0080
wył.	wł.	wył.	-14.54	-29.04	-28.26	-16.41	0.78	0.60	0.57	0.0081
wył.	wł.	wł.	-14.50	-30.93	-31.13	-16.35	0.79	0.63	0.62	0.0080
wł.	wył.	wł.	-12.04	-27.66	-29.31	-13.73	0.60	0.48	0.49	0.0065
wł.	wył.	wył.	-11.90	-24.87	-25.20	-13.54	0.60	0.42	0.38	0.0064

Tabela 8.10. Porównanie poprawy jakości w stosunku do wyłączeniowych wszystkich filtrów, profil AI



Rysunek 8.5. Wykres wzmacniania PSNR w zależności od QP dla obu profili kodowania

lepszych metod z literatury, tj. z pracy [27] oraz VVCNN [22]. Wytrenowana w ramach tej pracy sieć pozwoliła uzyskać lepszą poprawę jakości, niż najlepsze istniejące metody.

Należy jednak podkreślić, że porównanie to może nie odzwierciedlać pełnego obrazu ze względu na różnice w zbiorach testowych używanych dla trzech metod. Oba rozwiązania z literatury podają wyniki dla JVET common test conditions (ang. *ogólne warunki testowe*) [39], które określają nie tylko konfigurację kodeka, ale również zakresy QP (22, 27, 32 i 37) i testowe treści wideo. Różnice w danych, charakterystyce i różnorodności wideo w poszczególnych zbiorach mogą znacząco wpływać na wyniki, co utrudnia bezpośrednie porównanie efektywności obu metod. Niemniej jednak, uzyskane wyniki są obiecujące i wskazują na potencjalną zdolność proponowanego rozwiązania do konkurowania z obecnie najlepszymi metodami w dziedzinie.

rozwiązanie	BD-BR Y	BD-BR U	BD-BR V
proponowane	-8.76	-15.99	-15.19
najlepsze [27]	-7.40	-7.50	-10.01
VVCNN [22]	-4.54	-13.00	-14.86

Tabela 8.11. Porównanie wyników do najlepszych rozwiązań z literatury: VVCNN [22] i najlepszego [27], profil RA, wszystkie filtry włączone

8. Uzyskane wyniki

Choć porównanie bezpośrednie może być obarczone pewnymi ograniczeniami, obiecujące wyniki stanowią solidną podstawę do dalszego badania i rozwoju proponowanej metody. Biorąc pod uwagę, że dla przedstawionego w tej pracy rozwiązania zaobserwowano odwrotnie proporcjonalne wzmacnienie PSNR do parametru QP, najlepsze wyniki osiągnięto dla QP 32, można by liczyć na jeszcze lepsze rezultaty dla zakresów QP określonych w CTC.

8.4. Wpływ dodatkowych informacji na wejściu sieci na wyniki

W celu zbadania wpływu dodatkowych informacji na wejściu sieci na wyniki, przeprowadzono dalsze eksperymenty z wykorzystaniem sieci DenseNet z wielowarstwowymi blokami gęstymi, trenowanej w metodologii GAN. Wytrenowano sieć wyłączając wszystkie informacje na temat kodowania, a następnie z samym parametrem QP.

W tabelach 8.12 oraz 8.13 przedstawiono wyniki uzyskane dla profilu RA oraz AI. Potwierdzają one, że przekazanie do sieci kontekstu dotyczącego QP kompresji ma znaczący wpływ na zdolność sieci do poprawy jakości. Zastosowanie pozostałych parametrów również powoduje poprawę wyników, choć wzrost metryk nie jest aż tak duży, jak w przypadku samego parametru QP.

Architektura	BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
	Y	U	V		Y	U	V	
Pełne informacje	-9.04	-17.10	-16.11	-8.94	0.46	0.36	0.29	0.0048
Sam parametr QP	-8.49	-12.91	-7.75	-8.83	0.45	0.26	0.18	0.0046
Brak dodatkowych informacji	-7.20	-7.87	-5.45	-7.24	0.35	0.13	0.10	0.0038

Tabela 8.12. Porównanie wyników bez wykorzystania dodatkowych informacji na wejściu sieci, profil RA

Architektura	BD-BR (PSNR)			BD-BR (SSIM)	BD-PSNR			BD-SSIM
	Y	U	V		Y	U	V	
Pełne informacje	-12.72	-26.06	-25.38	-13.78	0.68	0.48	0.43	0.0065
Sam parametr QP	-11.93	-19.75	-12.27	-12.77	0.65	0.34	0.24	0.0060
Brak dodatkowych informacji	-8.73	-2.08	8.03	-10.87	0.43	0.09	0.05	0.0050

Tabela 8.13. Porównanie wyników bez wykorzystania dodatkowych informacji na wejściu sieci, profil AI

9. Wnioski

Pomimo stosowania sieci o stosunkowo niewielkiej liczbie parametrów oraz ograniczonej głębokości, wyniki uzyskane dzięki zastosowanym technikom są bardzo obiecujące, co pokazuje, że skuteczność modelu nie zawsze jest proporcjonalna do jego złożoności. W szczególności, wykorzystanie dodatkowych informacji, takich jak parametry kompresji i typ ramki, znaczco przyczyniło się do poprawy wyników. Te informacje dostarczają sieci cennych wskazówek dotyczących charakterystyki i jakości źródłowego obrazu wideo oraz specyfiki artefaktów wprowadzonych przez proces kompresji, co pozwala na bardziej celowane i efektywne działanie sieci.

Dodatkowo, zastosowanie treningu w metodologii Generative Adversarial Networks (GAN) okazało się być kluczowym krokiem w kierunku uzyskania lepszej jakości obrazu. Sieci GAN, poprzez swoją naturę konkurencyjną między generatorem a dyskryminatorem, są w stanie wygenerować szczególnie realistyczne obrazy, minimalizując jednocześnie ryzyko nadmiernego wygładzania, czy utraty istotnych detali. W kontekście poprawy jakości wideo po dekompresji, użycie GAN pozwala nie tylko na redukcję artefaktów, ale także na przywrócenie i uwydatnienie detali obrazu, co jest szczególnie istotne w przypadku treści o wysokim stopniu szczegółowości.

Choć stosowane sieci są stosunkowo proste, to dzięki strategicznemu wykorzystaniu dodatkowych informacji kontekstowych i zaawansowanym technikom treningowym, takim jak GAN, możliwe jest osiągnięcie znaczących popraw w jakości wideo po dekompresji, otwierając nowe perspektywy dla efektywnego przetwarzania i optymalizacji wideo w różnorodnych zastosowaniach.

9.1. Możliwości zastosowania w praktyce

Przetwarzanie wideo w czasie rzeczywistym wymaga bardzo szybkich predykcji, co może być trudne do osiągnięcia z użyciem sieci neuronowych, szczególnie przy wysokiej rozdzielczości wideo. Proponowane rozwiązanie, pomimo małej liczby parametrów sieci neuronowej, nie nadaje się do poprawy jakości w czasie rzeczywistym, nawet na nowoczesnym sprzęcie - przetwarzanie 3 sekundowej sekwencji wideo na komputerze z kartą graficzną Nvidia GTX 1080 Ti zajmuje około 3 minut.

Dodatkowo, wdrożenie i zintegrowanie zaawansowanych modeli sieci neuronowych z oprogramowaniem i sprzętem urządzeń końcowych może być wyzwaniem z uwagi na różnorodność platform i ograniczenia sprzętowe.

Sieci neuronowe są trenowane na określonym zestawie danych, co może ograniczać ich zdolność do generalizacji i efektywnej pracy na różnorodnych danych wejściowych, które nie były częścią danych treningowych. Modele sieci neuronowych mogą też czasami tworzyć niepożądane artefakty lub zbytnio wygładzać obrazy, tracąc przy tym ważne szczegóły, zwłaszcza gdy model jest nadmiernie dopasowany do danych treningowych.

9.2. Ograniczenia i kierunki dalszych badań

Rozwiązań opisanych w tej pracy mają pewne ograniczenia, które mogą być przedmiotem dalszych badań. W szczególności, istnieje wiele obszarów, w których można kontynuować badania nad poprawą jakości wideo.

9.2.1. Zbadanie wyników w warunkach określonych w JVET CTC

Dużym ograniczeniem tej pracy było wykorzystanie stosunkowo małego zbioru danych testowych, różniącego się od określonego w JVET CTC [39]. Taki wybór wynikał z głównego celu pracy, którym była chęć zbadania wpływu filtrów kodeka VVC na wyniki oraz z ograniczeń sprzętowych. W dalszych badaniach należało skupić się na analizie działania proponowanej sieci na standardowych danych, określonych w common test conditions, aby móc rzetельnie porównać osiągniętą poprawę jakości po kompresji z rozwiązaniami z literatury.

9.2.2. Wykorzystanie zależności czasowych

Wykorzystanie zależności czasowych w celu poprawy jakości wideo może znacząco poprawić jakość rekonstrukcji. W przypadku architektur badanych w ramach tej pracy można np. przetwarzać sekwencje klatek zamiast pojedynczych klatek, co pozwoliłoby na wykorzystanie zależności czasowych w celu poprawy jakości wideo. Innym podejściem mogłoby być wykorzystanie informacji o trybach predykcji i wektorach ruchów w przypadku ramek inter przy profilu RA.

9.2.3. Wykorzystanie innych informacji

Informacje, takie jak mapa podziału CTU, mogłyby znacznie poprawić jakość rekonstrukcji, ponieważ zawierają one informacje o strukturze obrazu, które mogą być wykorzystane do polepszenia jakości wideo.

9.2.4. Zaawansowane techniki uczenia głębokiego

Eksploracja głębszych i bardziej zaawansowanych modeli sieci neuronowych może prowadzić do znaczącej poprawy jakości rekonstrukcji wideo. Ulepszone metody mogą efektywniej radzić sobie z artefaktami kompresji, zapewniając wyższą jakość wizualną, co jest szczególnie ważne w branżach, gdzie jakość obrazu ma kluczowe znaczenie, jak medycyna czy bezpieczeństwo.

Sieci neuronowe typu transformer są stosunkowo nową techniką, która może być wykorzystana do poprawy jakości wideo. W szczególności, sieci te są w stanie efektywnie radzić sobie z długimi zależnościami czasowymi, co może przyczynić się do znaczącej poprawy jakości wideo.

9.2.5. Optymalizacja dla różnych rodzajów treści wideo

Dostosowywanie metod poprawy do specyficznych typów treści wideo może zapewnić znacznie lepsze wyniki. Na przykład, techniki optymalizowane pod kątem dynamicznych

scen sportowych mogą różnić się od tych stosowanych w treściach filmowych, co umożliwia precyzyjniejsze i bardziej efektywne poprawianie jakości w różnych kontekstach.

9.2.6. Badanie efektów percepcyjnych

Skupienie się na percepcyjnej jakości obrazu może doprowadzić do opracowania metod, które są bardziej dostosowane do ludzkiego postrzegania wizualnego. Może to poprawić subiektywne wrażenia użytkowników końcowych, nawet jeśli obiektywne metryki, takie jak PSNR czy SSIM, nie wskazują znaczącej różnicy.

9.2.7. Optymalizacja pod kątem wykorzystania w czasie rzeczywistym

Rozwój metod umożliwiających poprawę jakości w czasie rzeczywistym może znacząco wpływać na aplikacje, takie jak streaming wideo na żywo, gry wideo, czy telekonferencje, oferując użytkownikom wyższą jakość obrazu bez opóźnień. W szczególności, wykorzystanie technik kompresji sieci neuronowych w celu zmniejszenia wymagań dotyczących mocy obliczeniowej, może znacząco przyczynić się do rozwoju technologii poprawy jakości wideo.

9.2.8. Zastosowania w specjalistycznych dziedzinach

Badanie zastosowań technik poprawy jakości wideo w specjalistycznych dziedzinach, takich jak telemedycyna, może przyczynić się do znaczącej poprawy jakości usług i efektywności diagnozy, gdzie wysoka jakość wideo jest niezbędna.

Podsumowując, dalsze badania w tych obszarach mogą znacząco przyczynić się do rozwoju technologii poprawy jakości wideo, oferując szeroki zakres korzyści od poprawy doświadczeń użytkowników końcowych, przez zwiększenie efektywności branżowych aplikacji, po rozwój nowych, innowacyjnych rozwiązań w dziedzinie przetwarzania wizualnego.

10. Podsumowanie

W niniejszej pracy magisterskiej podjęto wyzwanie zbadania i wykorzystania potencjału sieci neuronowych w kontekście poprawy jakości wideo po kompresji. W erze cyfrowej, gdzie jakość przekazu ma kluczowe znaczenie, przedstawione badania stanowią istotny wkład w dziedzinę przetwarzania i kompresji danych. Przez analizę i porównanie różnych architektur sieci neuronowych, w tym ResNet i DenseNet oraz wykorzystanie metodologii GAN, praca rzuca światło na możliwości związane z poprawą wizualną materiałów wideo po ich kompresji.

Szczególną wartość dodaną pracy stanowi wykorzystanie dodatkowych informacji o profilu kodowania, parametrze QP, stosowanych filtrach i typie ramki, co pozwoliło na znaczące usprawnienie procesu poprawy jakości. Wprowadzenie tych informacji, jako dodatkowych wejść dla modeli sieci neuronowych, zwiększyło ich zdolność do precyzyjnej i efektywnej korekty artefaktów. Wyniki uzyskane dzięki tym metodom, szczególnie w kanale luminancji (Y), wskazują na wysoki potencjał tego typu podejścia dla rozwiązania problemu poprawy jakości wideo po kompresji.

Chociaż wyniki przedstawione w niniejszej pracy są obiecujące, należy zauważyć, że obecne rozwiązanie ma charakter uproszczony. Wykorzystanie jedynie 64 pierwszych klatek wszystkich sekwencji ze zbioru danych stanowi ograniczenie, które wpływa na ogólną generalizację i skalowalność proponowanych metod. Takie podejście, choć cenne w fazie początkowej, nie obejmuje pełnej złożoności i różnorodności scenariuszy, które mogą wystąpić w rzeczywistych warunkach transmisji wideo.

Podsumowując, choć prezentowane rozwiązanie pozwala na znaczną poprawę jakości wideo po kompresji kodekiem VVC, rozległe możliwości dalszego rozwoju i udoskonalania podkreślają otwarty i ewolucyjny charakter badań. Ich kontynuacja może znaczco przekształcić sposób, w jaki doświadczamy cyfrowego wideo.

Bibliografia

- [1] "Xiph.org Video Test Media". dostęp zdalny 28.12.2023. (), adr.: <https://media.xiph.org/video/derf/>.
- [2] J. V. E. T. (JVET), *VTM reference software for VVC*, Gitlab, 2020. adr.: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM.
- [3] I. T. Union, spraw. tech., czer. 2021. adr.: <https://www.itu.int/rec/T-REC-H.266>.
- [4] I. T. Union, spraw. tech., czer. 2021. adr.: <https://www.itu.int/rec/T-REC-H.264>.
- [5] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm i T. Wiegand, "High Efficiency Video Coding (HEVC) Text Specification Draft 9", paź. 2019.
- [6] "Coded representation of immersive media, part 3: Versatile video coding", International Organization for Standardization, Geneva, CH, Standard, grud. 2022.
- [7] *Brazilian SBTVD forum selects V-nova LCEVC for Brazil's upcoming TV 3.0*, dostęp zdalny 28.12.2023, sty. 2022. adr.: <https://www.digitalmediaworld.tv/broadcast/4118-brazilian-sbtvd-forum-selects-v-nova-lcevc-for-brazil-s-upcoming-tv-3-0>.
- [8] dostęp zdalny 28.12.2023, lut. 2022. adr.: <https://www.avcaesar.com/news/1781/dvb-adds-the-vvc-h266-codec-to-its-video-standards-for-8k>.
- [9] A. Norkin, C.-M. Fu, Y.-W. Huang i S. Lei, "In-Loop Filters in HEVC", eng, w *High Efficiency Video Coding (HEVC)*, ser. Integrated Circuits and Systems, Cham: Springer International Publishing, 2014, s. 171–208, ISBN: 9783319068947.
- [10] A. Vakili i J.-C. Grégoire, "QoE Management in a Video Conferencing Application", *Computer Networks*, t. 57, s. 1726–1738, maj 2013. DOI: 10.1016/j.comnet.2013.03.002.
- [11] K. He, X. Zhang, S. Ren i J. Sun, "Deep Residual Learning for Image Recognition", eng, 2015.
- [12] G. Huang, Z. Liu i K. Q. Weinberger, "Densely Connected Convolutional Networks", *CoRR*, t. abs/1608.06993, 2016. arXiv: 1608.06993. adr.: <http://arxiv.org/abs/1608.06993>.
- [13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza i in., *Generative Adversarial Networks*, 2014. arXiv: 1406.2661 [stat.ML].
- [14] D. Saxena i J. Cao, "Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions", eng, *ACM computing surveys*, t. 54, nr. 3, s. 1–42, 2021, ISSN: 0360-0300.
- [15] Z. Zhao, Q. Sun, H. Yang, H. Qiao, Z. Wang i D. O. Wu, "Compression artifacts reduction by improved generative adversarial networks", eng, *EURASIP journal on image and video processing*, t. 2019, nr. 1, s. 1–7, 2019, ISSN: 1687-5281.
- [16] X. Song, J. Yao, L. Zhou i in., "A practical convolutional neural network as loop filter for intra frame", 2018. arXiv: 1805.06121 [cs.MM].

10. Bibliografia

- [17] R. Yang, M. Xu, T. Liu, Z. Wang i Z. Guan, "Enhancing Quality for HEVC Compressed Videos", *IEEE Transactions on Circuits and Systems for Video Technology*, t. 29, nr. 7, s. 2039–2054, lip. 2019, ISSN: 1558-2205. DOI: 10.1109/tcsvt.2018.2867568. adr.: <http://dx.doi.org/10.1109/TCSVT.2018.2867568>.
- [18] D. Ma, F. Zhang i D. R. Bull, "CVEGAN: A Perceptually-inspired GAN for Compressed Video Enhancement", 2020. arXiv: 2011.09190 [eess.IV].
- [19] D. Danier, C. Feng, F. Zhang i D. Bull, *Enhancing VVC with Deep Learning based Multi-Frame Post-Processing*, 2022. arXiv: 2205.09458 [eess.IV].
- [20] *YouTube Media Algorithms team, YouTube User Generated Content*, dostęp zdalny 28.12.2023. adr.: <https://media.withyoutube.com/>.
- [21] H. Zhang, C. Jung, D. Zou i M. Li, "WCDANN: A Lightweight CNN Post-Processing Filter for VVC-Based Video Compression", *IEEE Access*, t. 11, s. 83 400–83 413, 2023. DOI: 10.1109/ACCESS.2023.3301145.
- [22] T. Das, K. Choi i J. Choi, "High Quality Video Frames From VVC: A Deep Neural Network Approach", *IEEE Access*, t. 11, s. 54 254–54 264, 2023. DOI: 10.1109/ACCESS.2023.3281975.
- [23] C. Jia, S. Wang, X. Zhang, S. Wang i S. Ma, "Spatial-temporal residue network based in-loop filter for video coding", w *2017 IEEE Visual Communications and Image Processing (VCIP)*, 2017, s. 1–4. DOI: 10.1109/VCIP.2017.8305149.
- [24] J. Wang, X. Deng, M. Xu, C. Chen i Y. Song, *Multi-level Wavelet-based Generative Adversarial Network for Perceptual Quality Enhancement of Compressed Video*, 2020. arXiv: 2008.00499 [eess.IV].
- [25] J. Kang, S. Kim i K. M. Lee, "Multi-modal/multi-scale convolutional neural network based in-loop filter design for next generation video codec", w *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, s. 26–30. DOI: 10.1109/ICIP.2017.8296236.
- [26] X. Meng, X. Deng, S. Zhu i B. Zeng, "BSTN: An Effective Framework for Compressed Video Quality Enhancement", w *2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2020, s. 320–325. DOI: 10.1109/MIPR49039.2020.00072.
- [27] F. Nasiri, W. Hamidouche, L. Morin, N. Dhollande i G. Cocherel, "A CNN-Based Prediction-Aware Quality Enhancement Framework for VVC", *IEEE Open Journal of Signal Processing*, t. 2, s. 466–483, 2021. DOI: 10.1109/OJSP.2021.3092598.
- [28] O. Tange, "GNU Parallel - The Command-Line Power Tool", ;*login: The USENIX Magazine*, t. 36, nr. 1, s. 42–47, lut. 2011. adr.: <http://www.gnu.org/s/parallel>.
- [29] K. He, X. Zhang, S. Ren i J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015. arXiv: 1502.01852 [cs.CV].
- [30] L. Biewald, *Experiment Tracking with Weights and Biases*, Software available from wandb.com, 2020. adr.: <https://www.wandb.com/>.

- [31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever i R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting”, *J. Mach. Learn. Res.*, t. 15, nr. 1, s. 1929–1958, sty. 2014, ISSN: 1532-4435.
- [32] S. Ioffe i C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, *CoRR*, t. abs/1502.03167, 2015. arXiv: 1502.03167. adr.: <http://arxiv.org/abs/1502.03167>.
- [33] A. Paszke, S. Gross, S. Chintala i in., “Automatic differentiation in PyTorch”, w *NIPS-W*, 2017.
- [34] W. Falcon i The PyTorch Lightning team, *PyTorch Lightning*, ver. 1.4, mar. 2019. DOI: 10.5281/zenodo.3828935. adr.: <https://github.com/Lightning-AI/lightning>.
- [35] O. Yadan, *Hydra - A framework for elegantly configuring complex applications*, GitHub, 2019. adr.: <https://github.com/facebookresearch/hydra>.
- [36] P. S. Inc., *Pydantic*, GitHub. adr.: <https://github.com/pydantic/pydantic/>.
- [37] S. Tomar, “Converting video formats with FFmpeg”, *Linux Journal*, t. 2006, nr. 146, s. 10, 2006.
- [38] W. McKinney, “Data Structures for Statistical Computing in Python”, w *Proceedings of the 9th Python in Science Conference*, S. van der Walt i J. Millman, red., 2010, s. 56–61. DOI: 10.25080/Majora-92bf1922-00a.
- [39] J. Boyce, K. Suehring, X. Li i V. Seregin, *JVET-J1010: JVET common test conditions and software reference configurations*, lip. 2018.

Wykaz symboli i skrótów

EiTI – Wydział Elektroniki i Technik Informacyjnych

PW – Politechnika Warszawska

CNN – ang. *Convolutional Neural Network*

ResNet – ang. *Residual Network*

DenseNet – ang. *Densely Connected Convolutional Network*

GAN – ang. *Generative Adversarial Network*

VVC – ang. *Versatile Video Coding*

PSNR – ang. *Peak Signal-to-Noise Ratio*

SSIM – ang. *Structural Similarity*

BD-BR – Bjøntegaard Delta Bit Rate

BD-PSNR – Bjøntegaard Delta Peak Signal-to-Noise Ratio

Spis rysunków

2.1	Porównanie różnych rodzajów ramek, źródło: Wikipedia	13
2.2	Efekt blokowy przy QP=50	20
2.3	Ringing artifact	20
2.4	Wpływ parametru QP na jakość obrazu oraz przepływność	21
3.1	Diagram struktury warstwy konwolucyjnej	24
3.2	Diagram struktury warstwy ResNet	25
3.3	Diagram struktury warstwy DenseNet	25
3.4	Architektura sieci ARGAN[15]	26
4.1	Przykładowe klatki sekwencji kontrolnych	30
4.2	Metodologia normalizacji danych poprzez podział na fragmenty o stałym rozmiarze	35
5.1	Diagram wspólnej części architektur sieci użytych do poprawy jakości	38
6.1	Problem z odwzorowaniem kolorów przy poprawie jakości przez sieć	49
6.2	Przebieg funkcji straty podczas treningu sieci DenseNet poprawiającej jakość	50
6.3	Przebieg funkcji straty podczas treningu sieci GAN dla pierwszego stabilnego treningu	51
6.4	Przebieg funkcji straty po zastosowaniu selektywnego treningu	52
6.5	Przebieg funkcji straty po zastosowaniu wszystkich usprawnień w treningu sieci GAN	54
7.1	Diagram struktury pakietu projektu	67

8.1 Porównanie efektów poprawy jakości dla różnych architektur sieci dla sekwencji Johnny ($qp = 47$, filtry wyłączone, profil RA), przybliżenie na fragment $256x256$ zawierający głowę prezentera	80
8.2 Porównanie maski różnic pomiędzy oryginałem, ramką po dekompresji i poprawie jakości badanymi architekturami sieci dla sekwencji Johnny ($qp = 47$, filtry wyłączone, profil AI), przybliżenie na fragment $256x256$ zawierający głowę prezentera	81
8.3 Porównanie efektów poprawy jakości dla architektury DenseNet oraz GAN, sekwencja Johnny ($qp = 47$, klatka I, filtry wyłączone, profil RA), przybliżenie na fragment $256x256$ zawierający głowę prezentera	83
8.4 Porównanie efektów poprawy jakości dla architektury DenseNet oraz GAN, sekwencja Johnny ($qp = 47$, klatka I, filtry wyłączone, profil RA), przybliżenie na fragment $256x256$ zawierający głowę prezentera	85
8.5 Wykres wzmacnienia PSNR w zależności od QP dla obu profili kodowania	87

Spis tabel

5.1 Liczba filtrów oraz rozmiar jądra splotu na warstwach sieci ResNet	39
5.2 Architektura proponowanej sieci DenseNet	40
5.3 Liczba filtrów oraz rozmiar jądra splotu na warstwach sieci konwolucyjnej	41
5.4 Architektura sieci DenseNet-121	42
6.1 Porównanie średnich wartości PSNR i SSIM składowych YUV danych testowych po kompresji kodekiem H.266/VVC	49
8.1 Porównanie wyników w zależności od architektury sieci dla profilu RA	79
8.2 Porównanie wyników w zależności od architektury sieci dla profilu AI	79
8.3 Wyniki uzyskane dla sieci wytrenowanej w metodologii GAN, profil RA	82
8.4 Wyniki uzyskane dla sieci wytrenowanej w metodologii GAN, profil AI	82
8.5 Uzyskane wyniki przy użyciu różnych filtrów wbudowanych w kodęk VVC, profil RA	82
8.6 Uzyskane wyniki przy użyciu różnych filtrów wbudowanych w kodęk VVC, profil AI	84
8.7 Porównanie poprawy jakości przy użyciu filtrów wbudowanych w kodęk VVC oraz proponowanej sieci, profil RA	84
8.8 Porównanie poprawy jakości przy użyciu filtrów wbudowanych w kodęk VVC oraz proponowanej sieci, profil AI	84
8.9 Porównanie poprawy jakości w stosunku do wyłączonych wszystkich filtrów, profil RA	86

8.10 Porównanie poprawy jakości w stosunku do wyłączonych wszystkich filtrów, profil AI	86
8.11 Porównanie wyników do najlepszych rozwiązań z literatury: VVCNN [22] i najlepszego [27], profil RA, wszystkie filtry włączone	87
8.12 Porównanie wyników bez wykorzystania dodatkowych informacji na wejściu sieci, profil RA	88
8.13 Porównanie wyników bez wykorzystania dodatkowych informacji na wejściu sieci, profil AI	88