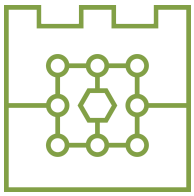
Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji



Sylwester Wieczorek

Numer albumu: 140268

Zastosowanie sieci typu KAN w zagadnieniu kompresji obrazu

Application of Kolmogorov-Arnold Networks in Image Compression

Praca ~~licencjacka/inżynierska~~/magisterska
na kierunku INFORMATYKA
specjalność DATA SCIENCE

Praca wykonana pod kierunkiem:
dr inż. Dominika Cywicka
dr Ilona Urbaniak

Kraków 2025

Spis treści

1. Zagadnienia kompresji z wykorzystaniem modeli neuronowych

Kompresja obrazów cyfrowych polega na zmianie reprezentacji obrazu do formy zajmującej mniej przestrzeni dyskowej, przy zachowaniu lub możliwie najmniejszej stracie na „jakości". Opracowano wiele algorytmów kompresji wykorzystujących własności statystyczne obrazu - zarówno bezstratnych, zachowujących pełnię informacji, jak i stratnych, które tracą część danych w zamian za wyższy stopień kompresji. W ostatnich latach coraz większą rolę odgrywają modele uczenia maszynowego, w szczególności sieci neuronowe, które potrafią uczyć się skutecznej kompresji danych na podstawie zbiorów treningowych. Autoenkodery (głębokie sieci neuronowe uczące się odwzorowania identycznościowego przez wąskie warstwy ukryte) są stosowane w tematyce kompresji obrazu - sieć uczy się kodować obraz wejściowy do wektora o niższym wymiarze, z którego następnie dekoder odtwarza obraz wyjściowy [1].

Kompresja przy pomocy modeli neuronowych zyskały na popularności, ponieważ w wielu przypadkach zaczęły przewyższać klasyczne metody oparte na algorytmach. Prace naukowe wskazują, że współczesne modele kompresji obrazu oparte na głębokich sieciach (ang. neural image compression, NIC) potrafią uzyskać lepszą efektywność (stosunek jakości do stopnia kompresji) niż klasyczne kodeki takie jak JPEG czy JPEG2000 [2], [3]. Zaprezentowany w 2017 pierwszy autoenkoder kompresujący obrazy end-to-end, który dla niezależnego zestawu testowego uzyskał wyższą jakość od JPEG przy porównywalnym rozmiarze pliku [3]. Od tamtej pory rozwinięto wiele usprawnień m.in. wariacyjne autoenkodery, sieci uwzględniające modelowanie entropii czy transformery - dzięki czemu obecnie sieci neuronowe stanowią obiecującą alternatywę dla tradycyjnych metod w kompresji obrazów [2].

Jednym z nowszych podejść w dziedzinie sieci neuronowych są modele typu KAN (ang. Kolmogorov-Arnold Networks), inspirowane twierdzeniem Kołmogorowa-Arnolda o superpozycji funkcji [4]. Niniejsza praca skupia się na zbadaniu, na ile modele KAN mogą zostać wykorzystane do efektywnej kompresji obrazów. W dalszych częściach przedstawione zostaną zarówno teoretyczne podstawy kompresji i modeli KAN, jak i wyniki eksperymentów porównujących autoenkodery oparte na KAN z klasycznym autoenkoderem konwolucyjnym (CNN).

1.1. Wykorzystanie modelu KAN w praktyce

Model KAN jest zupełnie nowym podejściem, zaproponowanym w literaturze dopiero w 2024 roku [5]. Został zainspirowany twierdzeniem Kołmogorowa o reprezentacji każdej funkcji wielowymiarowej za pomocą funkcji jednowymiarowych (szczegóły w rozdziale 3). W modelu KAN zamiast standardowych wag neuronowych funkcje kształtu (ang. activation functions) zmienia się w czasie trwania treningu i to one odpowiadają za „ewolucje" modelu. W praktyce funkcje te są parametryzowane np. za pomocą splinów (funkcji sklejanych) o pewnym stopniu. Dzięki temu KAN może aproksymować złożone nieliniowe zależności z użyciem mniejszej liczby neuronów niż tradycyjna sieć wielowarstwowa MLP (ang. Multilayer Perceptron), w której nieliniowość jest narzucona z góry (np. funkcją simgoid, ReLU) i identyczna dla wszystkich neuronów. Stosunkowo niewielkie sieci KAN mogą osiągać dokładność porównywalną lub lepszą od znacznie większych klasycznych MLP. KAN swoje zastosowanie może znaleść również m.in. w zagadnieniach aproksymacji funkcji matematycznych i rozwiązywania równań różniczkowych np. jako narzędzie do odkrywania praw fizycznych na podstawie danych eksperymentalnych [5].

Pomimo, że KAN jest architekturą stosunkowo nową, szybko zwrócono na nią uwagę. Opublikowano otwartoźródłową (ang. OpenSource) implementację o nazwie pyKAN, która w krótkim czasie zyskała dużą popularność [6] (popularność oceniam na podstawie liczby gwiazdek na oficjalnej implementacji zamieszczonej na GitHub). Dostępność kodu ułatwia eksperymentowanie z KAN w różnych zadaniach. W praktyce KAN był już testowany m.in. w modelowaniu szeregów czasowych (powstały czasowe odmiany T-KAN [7]), w sieciach grafowych (Graph KAN [8]). Wszystko to wskazuje, że KAN ma potencjał do szerokiego wykorzystania. Niniejsza praca stanowi jedno z pierwszych badań użycia KAN do kompresji obrazów, dziedziny, w której dotąd dominowały klasyczne sieci konwolucyjne oraz transformatory.

1.2. Cel i zakres pracy

Celem pracy jest zaprojektowanie, implementacja oraz ocena działania autoenkodera kompresującego obrazy z wykorzystaniem architektury KAN w tym:

a) Napisanie własnej implementacji modelu KAN pozwalającą na zmianę rodzajów splinów.
b) Opracowanie autoenkoder opartego o warstwy KAN zdolnego do kompresji obrazów.
c) Przystosowanie autoenkoder dla implementacji FastKAN.
d) Opracowanie autoenkodera opartego o CNN który zachowa identyczny stopień kompresji.
e) Porównanie jego skuteczność z autoenkoderem konwolucyjnym (CNN) pod względem stopnia kompresji oraz jakości odtworzenia obrazów.

Zakres pracy obejmuje przegląd literatury i podstaw teoretycznych związanych z kompresją obrazów i modelami KAN (rozdziały 2-3), opis zaproponowanych modeli autoenkoderów opartych na KAN (rozdział 4), szczegółowe omówienie środowiska implementacji i metodologii eksperymentów (rozdział 5) oraz prezentację wyników eksperymentalnych - porównanie KAN vs CNN w kompresji obrazów (rozdziały 6-7). Całość kończy podsumowanie uzyskanych wyników wraz z wnioskami praktycznymi i propozycjami dalszych kierunków badań (rozdział 8).

W eksperymentach wykorzystano zarówno autorską implementację pozwalającą na zmianę rodzaju funkcji bazowych (spline) na krawędziach modelu KAN, aby ocenić ich wpływ na dokładność i wydajność kompresji oraz implementację FastKAN zaproponowaną przez Ziyao Li, w której zamiast funkcji sklejanych zastosowano radialne funkcje bazowe Gaussa, co pozwala przyspieszyć propagację w przód względem oryginalnego KAN [9].

2. Podstawy teoretyczne

2.1. Reprezentacja obrazu cyfrowego

Obraz cyfrowy to dwuwymiarowa siatka pikseli którym przypisane są wartości numeryczne reprezentujące jasność (dla obrazów w skali szarości) lub kolor (dla obrazów barwnych). Obrazy barwne zazwyczaj zapisuje się za pomocą trzech składowych kolorów (np. model RGB - czerwony, zielony, niebieski), gdzie każdy piksel jest wektorem tych trzech wartości. Często stosuje się też przestrzenie barw pochodne, np. YCbCr używaną w kompresji JPEG, gdzie separuje się składową luminancji od chrominancji. Fundamentalnymi parametrami obrazu cyfrowego są rozdzielczość (liczba pikseli w poziomie i pionie) oraz głębią koloru określająca, ile bitów przypada na reprezentację jednego piksela. Przykładowo, obraz o rozdzielczości 1920×1080 i 24-bitowej głębi koloru (po 8 bitów na każdą składową RGB) zawiera ponad 2 miliony pikseli, a każdy piksel może przyjmować $2^8 = 256$ odcieni na kanał (czyli ok. $16,7$ miliona kombinacji kolorów). Na podstawie tych danych można obliczyć, że taki nieskompresowany obraz zajmuje ok. 6,2 miliona bajtów (6 MB).

W praktyce obrazy często przechowuje się w formatach skompresowanych, aby zmniejszyć ich rozmiar. Niekompresowana reprezentacja jest wykorzystywana głównie w określonych zastosowaniach (np. profesjonalna fotogria, obrazy RTG). Najczęściej dokonuje się kompresji wykorzystując fakt, że piksele w obrazie nie są niezależne - występuje znaczna redundancja przestrzenna (sąsiednie piksele są zwykle podobne) oraz ludzki wzrok mniej czuły jest na pewne detale. Kompresja stara się usunąć te redundantne informacje lub przechowywać je w bardziej zwartej formie.

2.2. Kompresja obrazu

Kompresja danych polega na zamianie oryginalnego strumienia bitów na inny, krótszy - tak aby wynikowy zbiór danych można było zdekodować do formy zbliżonej lub identycznej z oryginałem. Rozróżniamy dwa główne rodzaje kompresji danych: bezstratną i stratną.

Stopień kompresji określa się zazwyczaj poprzez współczynnik kompresji - stosunek liczby bitów danych oryginalnych do liczby bitów danych skompresowanych. Przykładowo, współczynnik 1:10 oznacza, że obraz po kompresji zajmuje 10 razy mniej miejsca niż przed kompresją. W kompresji bezstratnej współczynnik ten jest ograniczony redundancją danych, natomiast w stratnej można go zwiększać kosztem pogarszania jakości. W praktyce dąży się do znalezienia kompromisu między wysokim stopniem kompresji a akceptowalną jakością wyniku.

2.2.1. Kompresja bezstratna

Kompresja bezstratna opiera się na znalezieniu takiej reprezentacji danych, która usuwa nadmiarowość, ale pozwala dokładnie odtworzyć oryginał. Jedną ze znanych technik to kodowanie entropijne. Wykorzystuje fakt, że niektóre wartości pojawiają się częściej niż inne. Przykładem jest kodowanie Huffmana, przypisujące częstym symbolom krótkie kody bitowe, a rzadkim dłuższe. Inna metoda to kodowanie arytmetyczne, które osiąga bliską optymalnej długość kodu dla danego rozkładu symboli. Kodowanie entropijne stanowi często ostatni etap większości algorytmów kompresji (zarówno bezstratnych, jak i stratnych).

Inna forma kompresji bezstratnej to kodowanie słownikowe polega na zastępowaniu powtarzających się sekwencji znaków odniesieniami do ich pierwszego wystąpienia. Algorytmy z rodziny LZ (Lempel-Ziv), takie jak LZ77 i LZ78, tworzą słownik napotkanych ciągów danych i wstawiają odwołania (np. para offset, długość) do wcześniejszego wystąpienia zamiast powtarzać cały ciąg. Format ZIP, a także PNG, używają udoskonalonej odmiany LZ77, łączącej słownik LZ z kodowaniem Huffmana dla optymalizacji bitów [10].

Przed zastosowaniem kodowania entropijnego można przekształcić dane, by zwiększyć ich „kompresowalność”. Przykładem jest wspomniane filtrowanie w PNG, gdzie każdy piksel zastępuje się różnicą względem pewnej kombinacji sąsiadów (np. różnica w stosunku do piksela z lewej, powyżej, czy średniej z nich). Po takim zabiegu dane zawierają wiele powtarzalnych wzorców (np. ciągi zer po odjęciu tła), które łatwiej zakodować. W obrazach często stosuje się też kodeki bezstratne wykorzystujące transformaty - np. tryb bezstratny JPEG2000 używa całkowitoliczbowej transformaty falkowej, którą można odwrócić bez błędu.

W kontekście obrazów, kompresja bezstratna znajduje zastosowanie tam, gdzie wymagana jest pełna wierność - np. w obrazowaniu medycznym, archiwizacji dzieł sztuki czy w surowych zdjęciach (RAW) wykorzystywanych do dalszej obróbki. Jednak ze względu na ograniczone możliwości zmniejszenia rozmiaru, formaty bezstratne nie sprawdzają się tam, gdzie priorytetem jest maksymalne zmniejszenie rozmiaru pliku (np. w transmisji wideo, stronach internetowych itp.). W takich przypadkach sięga się po kompresję stratną.

2.2.2. Kompresja stratna

Kompresja stratna usuwa część informacji oryginalnego sygnału, zazwyczaj tę najmniej zauważalną dla odbiorcy, aby drastycznie zmniejszyć ilość danych. Do popularnych technik należą:

Zamiast bezpośrednio kompresować wartości pikseli, dokonuje się transformacji do innej dziedziny, w której ujawnia się struktura sygnału. Przykładowo, JPEG dzieli obraz na bloki 8×8 pikseli i stosuje dyskretną transformację cosinusową (DCT) na każdym bloku. Powstałe współczynniki DCT reprezentują sygnał w dziedzinie częstotliwości przestrzennych, pierwsze odpowiadają składowym niskoczęstotliwościowym (równe obszary), a kolejne, coraz wyższym częstotliwościom (drobnym szczegółom). Alternatywą jest transformata falkowa (JPEG2000 dzieli obraz na pasma częstotliwości na różnych skalach) lub transformaty Fourierowskie, Principal Component Analysis (PCA) itp.

Zaokrąglenie lub wyzerowanie części współczynników transformaty, redukując precyzję informacji tzw. kwantyzacja. W JPEG stosuje się macierz kwantyzacji 8×8: każdy z 64 współczynników DCT dzieli się przez odpowiadającą mu wartość kwantyzacji i zaokrągla (w praktyce, współczynniki wysokoczęstotliwościowe dzielone są przez większe liczby, przez co wiele z nich staje się zerem). Powoduje to utratę drobnych szczegółów, ale przynosi ogromne korzyści — długie ciągi zer można potem zakodować bardzo efektywnie. Stopień kwantyzacji kontroluje jakość kompresji - im silniejsza kwantyzacja (większe dzielniki), tym mniejszy plik, ale więcej informacji ginie i obraz jest bardziej zniekształcony.

Na końcu wynik kwantyzacji poddaje się bezstratnemu upakowaniu bitów, tak jak opisano w sekcji 2.2.1. JPEG używa uproszczonego algorytmu Huffmana lub arytmetycznego do zakodowania długości serii zer i wartości niezerowych. Nowsze standardy (HEVC dla obrazów statycznych - H.265/HEIC) mogą wykorzystywać bardziej zaawansowane metody modelowania i kodowania entropijnego kontekstowo zależnego. [11]

Istotnym elementem kompresji stratnej obrazów jest model barw i subsampling. To jak postrzegamy kolort cechuje wieksza rozdzielczość dla jasności niż dla barwy - dlatego formaty takie jak JPEG najpierw przekształcają RGB na YCbCr (jasność + dwie składowe różnicowe koloru), po czym zapisują sygnały w niższej rozdzielczości (np. 2×2 bloki pikseli dzielą jedną wartość - tzw. subsampling 4:2:0). Na tym kroku tracimy część informacji, niew jest to jednak bardzo rzucające się w oczy, a pozwala zaoszczędzić miejsce na dysku.

Podsumowując, kompresja stratna jest efektywnym sposobem redukcji rozmiaru obrazów, lecz generuje artefakty kompresji, rozmycie detali, utrata ostrości czy pasmowanie jednolitych obszarów.

2.2.3. Metryki jakości

Ocena jakości skompresowanego obrazu jest istotnym wyzwaniem badawczym, które wymaga wykorzystania odpowiednich mierników do określenia, w jakim stopniu kompresja zmienia obraz w stosunku do oryginału. Stosuje się zarówno metryki obiektywne (matematyczne porównania piksel po pikselu), jak i subiektywne (badania z udziałem obserwatorów lub metryki starające to symulować). W niniejszej pracy do oceny wyników kompresji użyto głównie standardowych metryk pełnoreferencyjnych, porównujących obraz wynikowy z oryginalnym:

MSE (Mean Squared Error) - średni błąd kwadratowy. Definiowany jako średnia arytmetyczna kwadratów różnic jasności pikseli oryginalnych X i zdekompresowanych $\hat{X}$:

$$\mathrm{MSE} = \frac{1}{\mathrm{NM}} \sum_{x=1}^{N} \sum_{y=1}^{M} \left[X_{x,y} - \hat{X}_{x,y} \right]^2,$$

gdzie obraz ma rozmiar $N \times M$. Im mniejsze MSE, tym mniejsza ogólna różnica między obrazami.

PSNR (Peak Signal-to-Noise Ratio) - powszechnie stosowana metryka oparta na MSE, wyrażana w skali logarytmicznej. Definiowana jest jako:

$$\mathrm{PSNR} = 10 \log_{10} \frac{\mathrm{MAX}^2}{\mathrm{MSE}},$$

gdzie MAX to maksymalna możliwa wartość piksela (np. 255 dla obrazów 8-bitowych). PSNR określa stosunek mocy sygnału (idealnego obrazu) do mocy szumu (błędu). Wyższy PSNR oznacza lepszą jakość — typowo dla kompresji stratnej obrazów 8-bitowych wartości PSNR mieszczą się w przedziale 30-50 dB (im bardziej skompresowany i zniekształcony obraz, tym niższy PSNR) [12]. Na przykład akceptowalna jakość dla zdjęć to zwykle powyżej 30 dB, bardzo dobra powyżej 40 dB, jednak zależy to od zawartości obrazu i wrażliwości obserwatora.

SSIM (Structural Similarity Index) - bardziej złożona metryka zaproponowana w 2004, która lepiej niż MSE/PSNR odzwierciedla to jak oceniamy jakość kompresji [13]. SSIM porównuje lokalne struktury jasności między obrazem oryginalnym a przetworzonym, biorąc pod uwagę także kontrast i luminancje. Wartość SSIM wynosi 1 dla obrazów identycznych i spada, gdy pojawiają się różnice strukturalne. SSIM oblicza się na małych oknach (np. 8×8), porównując średnie, wariancje i kowariancje jasności między obrazami. Jeśli zdekompresowany obraz ma zachowaną strukturę (np. krawędzie, tekstury w tych samych miejscach) nawet przy drobnych różnicach pikseli, SSIM będzie wysoki, podczas gdy MSE/PSNR mogłyby znacząco spaść. Metryka SSIM szczególnie dobrze wychwytuje artefakty i rozmycie.

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)x(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Gdzie μ_x, μ_y to średnie (jasność) w lokalnym oknie dla obrazów x oraz y, σ_x^2, σ_y^2 to wariancje (kontrast) w tym oknie. σ_{xy} kowariancja (podobieństwo struktury) między x oraz y natomiast C_1, C_2 to stałe stabilizujące (zapobiegają dzieleniu przez małe liczby). $C_1 = (K_1, L)^2$,$C_2 = (K_2, L)^2$ typowo $K_1 = 0.01; K_2 = 0.03$ gdzie L to zakres dynamiki obrazu np. 255 dla obrazów 8 bitowych [14].

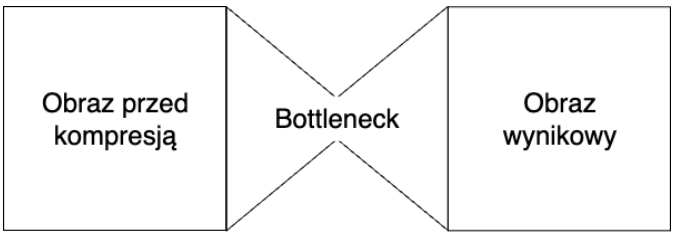
2.3. Wykorzystanie sieci neuronowych w kompresji obrazów

Sieci neuronowe od roku 2010 zrewolucjonizowały wiele dziedzin przetwarzania obrazu od klasyfikacji, przez segmentację, po generowanie obrazów. Również w kompresji znalazły swoje zastosowanie. Zasadnicza idea polega na tym, by pozwolić sieci nauczyć się optymalnej reprezentacji danych zamiast polegać wyłącznie na ręcznie zaprojektowanych algorytmach (jak DCT, kodowanie Huffmana).

Pierwsze prace nad uczeniem maszynowym w kompresji obrazów skupiały się na autoenkoderach - sieci neuronowej z warstwą ukrytą o mniejszym wymiarze niż wejście, trenowanej do rekonstrukcji danych wejściowych na wyjściu. Taka sieć, jeśli uda się ją skutecznie wytrenować, automatycznie wyodrębnia w warstwie ukrytej skompresowaną postać danych (tzw. embedding lub kod). W latach 80. i 90. testowano płytkie autoenkodery liniowe i nieli-

niowe, jednak dopiero rozwój głębokiego uczenia (deep learning) i wzrost mocy obliczeniowej umożliwiły uzyskanie jakości przewyższającej klasyczne metody.

Współczesne podejścia do kompresji z użyciem sieci wykorzystują zazwyczaj głębokie autoenkodery konwolucyjne. Architektura typowego systemu kompresji opartego na uczeniu głębokim wygląda następująco:



Rys. 2.1: Prosty schemat kompresji obrazu dla autoenkodera (źródło własne).

Obraz najpierw trafia do enkodera, sieć przetwarza obraz i produkuje zbiór kodów latentnych (wąskie gardło, bottleneck) - czyli ciąg liczb o znacznie mniejszym rozmiarze niż oryginalny obraz. Np. dla obrazu 256×256, enkoder może wygenerować 64 mapy cech o rozdzielczości 32×32, to już 4-krotna redukcja wymiarów w każdej osi czyli 16-krotna kompresja liczby pixeli. enkoder uczy się wyodrębniać z obrazu istotne cechy i odwzorowywać je w zagęszczonej formie.

W środku systemu, aby uzyskać rzeczywistą kompresję, ciąg wyjściowy enkodera musi zostać zakodowany jako ciąg bitów. Zwykle stosuje się tu operację kwantyzacji neuronowego kodu do skończonej liczby poziomów (np. zaokrąglenie wartości do ustalonej siatki). Ponieważ kwantyzacja jest niefunkcjonalna (nie ma gradientu), wiele prac stosuje surrogaty - np. dodawanie szumu podczas treningu lub metodę straight-through estimator, by móc trenować sieć w pełnym zakresie mimo obecności kwantyzacji. Po kwantyzacji powstają symbole, które poddaje się kodowaniu entropijnemu (np. metodą arytmetyczną) korzystając z modelu probabilistycznego dla tych symboli. Wytrenowane modele często potrafią lepiej dopasować rozkład prawdopodobieństwa niż klasyczne statyczne modele, co skutkuje mniejszą liczbą bitów. W tej pracy jednak skupiamy się na samych właściwościach autoenkodera - etap binarnego kodowania nie będzie zaimplementowany, a ocena skupi się na jakości rekonstrukcji i rozmiarze wewnętrznej reprezentacji [15].

Druga część sieci (dekoder), odwracająca działanie enkodera, ciagu liczb w bottleneck stara się odtworzyć oryginalny obraz. Dekoder zwykle ma strukturę „odwrotną” do enkodera (np. wykorzystuje warstwy transponowanej konwolucji deconv lub upsamplingu). Celem treningu jest minimalizacja różnicy między obrazem wyjściowym a wejściowym według pewnej funkcji kosztu (najczęściej MSE, percepcyjnej lub kombinacji). Po udanym treningu dekoder generuje obraz bardzo zbliżony do oryginalnego, korzystając wyłącznie z informacji zawartej w skompresowanym kodzie.

Trenowanie takiego systemu end-to-end odbywa się zazwyczaj metodą stochastycznego spadku gradientu (SGD) z algorytmami typu optymalizacyjnymi (przykładem takiego algorytmu jest Adam). Wymaga to dużej liczby przykładowych obrazów treningowych. Często korzysta się z publicznych zbiorów: np. Kodak PhotoCD (24 zdjęcia) do testów lub duży zbiór ImageNet do trenowania. Pojawiają się także dedykowane zbiory treningowe do kompresji (np. Workshop CLIC - Challenge on Learned Image Compression - udostępnia zestawy obrazów) [16].

Kompresja oparata na sieciach neuronowych ma wiele zalet np. model może dostosować się do statystyk obrazów lepiej niż uniwersalny algorytm - np. może nauczyć się specyficznych wzorców (tekstur, krawędzi) występujących często w obrazach naturalnych. W rezultacie osiąga lepszy stosunek między jakością a rozmiarem niż stałe algorytmy (co potwierdzono m.in. w pracach, gdzie autoenkodery przewyższyły JPEG/JPEG2000 [3]). Ponadto sieci mogą optymalizować dowolną wybraną metrykę (np. bezpośrednio maximować MS-SSIM zamiast minimalizować MSE). Inną zaletą jest możliwość uczenia kompresji kontekstowej - np. ważniejsze fragmenty obrazu (twarz, tekst) mogą być zakodowane dokładniej niż tło, jeśli taka zależność wynika z danych i funkcji kosztu.

Modele uczenia maszynowego nie zawsze jednak są najlepszy rozwiązaniem, wymagają sporej mocy obliczeniowej (zarówno do trenowania, jak i do enkodowania/dekodowania - co utrudnia ich użycie na urządzeniach mobilnych). Ważna jest generalizacja - model wytrenowany na pewnym zbiorze musi radzić sobie z obrazami spoza niego. Zauważono, że sieci mogą przeuczyć się do statystyk treningowych i tracić efektywność na innych obrazach, stąd pojawiły się badania nad poprawą uogólnienia i odporności na zmiany danych [2]. Ponadto, w praktycznych zastosowaniach ważna jest kontrola nad stopniem kompresji, klasyczne kodeki pozwalają płynnie regulować jakość/bitrat, dla modeli uczonych jest to zagadnienie bardziej skomplikowane.

Mimo tych wyzwań, kierunek rozwoju jest obiecujący. W roku 2023 po raz pierwszy zaprezentowano prototypowe wdrożenia sieciowego kodeka (np. Google przedstawił algorytm kompresji o nazwie Lyra dla audio i podobne próby trwają dla obrazów).

Podsumowując, sieci neuronowe, a w szczególności autoenkodery, stały się ważną gałęzią badań nad kompresją obrazów. Celem pracy jest sprawdzenie, czy Kolmogorov-Arnold Networks, będące nowatorską architekturą sieci neuronowej, mogą konkurować z klasycznymi autoenkoderami konwolucyjnymi w zadaniu kompresji obrazu. Zanim jednak przejdziemy do samych modeli KAN i eksperymentów, omówimy koncepcję tych sieci oraz ich właściwości w kontekście znanych architektur neuronowych.

3. Modele typu KAN (Kolmogorov-Arnold Networks)

3.1. Idea modelu KAN

Idea modelu KAN wywodzi się z fundamentalnego rezultatu analizy matematycznej znanego jako twierdzenie Kołmogorowa-Arnolda o superpozycji. Twierdzenie to (udowodnione przez A.N. Kołmogorowa w 1957 r., rozwinięte przez jego ucznia V. Arnol'da) mówi nam, że każdą ciągłą funkcję wielu zmiennych można dokładnie przedstawić jako złożenie funkcji jednowymiarowych. Mówiąc ściślej, dla dowolnej ciągłej funkcji $f : [0,1]^n \to \mathbb{R}$, istnieje reprezentacja postaci:

$$f(x_1, ..., x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \varphi_{(q,p)}(x_p) \right)$$

gdzie $\varphi_{q,p} : [0,1] \to \mathbb{R}$ oraz $\Phi_q : \mathbb{R} \to \mathbb{R}$ są pewnymi ciągłymi funkcjami jednowymiarowymi. Innymi słowy, funkcję n-wymiarową da się złożyć z funkcji zależnych tylko od jednej zmiennej (oraz operacji dodawania tych składowych) [17]. Oryginalna konstrukcja KAN nie była praktyczna obliczeniowo. Niemniej zainspirowała późniejsze prace nad sieciami neuronowymi. Można zauważyć, że powyższa formuła przypomina sieć dwuwarstwową: najpierw n zmiennych x_p jest przekształcanych przez funkcje $\varphi_{q,p}$ (warstwa wejściowa), potem sumowane (sumowanie to operacja na neuronach), a na koniec przekształcone przez funkcje Φ_q otrzymujemy wynik i sumę końcową. Mamy tu ściśle dwie warstwy (ukrytą o rozmiarze $N = 2n + 1$ i wyjściową, sumującą wyniki). Warto zaznaczyć, że oryginalne twierdzenie wymaga $2n + 1$ funkcji zewnętrznych Φ_q - później Lorentz pokazał, że wystarczy nawet jedna funkcja Φ sumująca odpowiednio zagnieżdżone φ (kosztem zwiększenia złożoności wewnętrznej) [18], ale ogólna idea pozostaje: wielowymiarowość można "schować" poprzez dodawanie jednowymiarowych składowych.

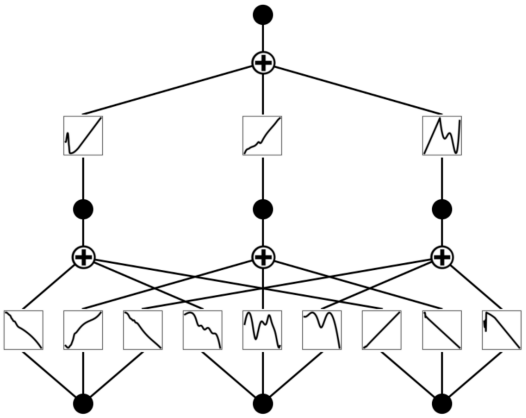
Model KAN wprowadza ten koncept na grunt sieci neuronowych. Bazuje on na zaobserwowaniu, że standardowa sieć MLP również jest pewną aproksymacją dowolnej funkcji, wg. klasycznego twierdzenia o uniwersalnej aproksymacji, sieć z jedną warstwą ukrytą o dostatecznej liczbie neuronów sigmoidalnych potrafi aproksymować dowolną ciągłą funkcję na

kompaktowym zbiorze. Dlaczego więc potrzebujemy KAN? Otóż, choć MLP ma uniwersalną zdolność aproksymacyjną, narzucona architektura (liniowe kombinacje wejść + proste funkcje aktywacji) może wymagać bardzo wielu neuronów, by uzyskać skomplikowaną funkcję.

Model KAN zakłada architekturę, w której każda krawędź w sieci realizuje pewną funkcję jednowymiarową $\varphi(x)$ z parametrami uczonymi z danych, natomiast każdy neuron sumuje wyniki funkcji ze wszystkich krawędzi wchodzących. Innymi słowy, w KAN to, co w zwykłej sieci jest stałą wagą mnożoną przez wejście, zostaje zastąpione funkcją: zamiast $\boldsymbol{w} \cdot \boldsymbol{x}$ mamy $\boldsymbol{\varphi}(\boldsymbol{x})$, gdzie kształt funkcji φ jest regulowany za pomocą punktów kontrolnych (parametrów). Zazwyczaj funkcje te na krawędziach są parametryzowane w sposób elastyczny, w oryginalnej pracy zaproponowano wykorzystanie do tego celu B-splinów stopnia 3 (cubic B--spline) [5]. B-spline to funkcja składana używana często w interpolacji, jej kształt wyznaczają węzły (punkty kontrolne), a pomiędzy nimi funkcja jest wielomianem określonego stopnia. Wybór B-splinów wynika z ich własności aproksymacyjnych - przez odpowiednie ustawienie węzłów można przybliżyć dowolną funkcję ciągłą z zadaną dokładnością, a jednocześnie mają one lokalny charakter (zmiana parametru wpływa tylko na ograniczony zakres argumentu, co ułatwia trenowanie). W implementacji KAN zakłada się pewną początkową siatkę punktów dla każdej funkcji krawędzi (np. 5 przedziałów równoodległych w zakresie wejściowym, co daje 6 węzłów do optymalizacji przy B-spline stopnia 3). Parametrami zmienijącymi się w trakcie uczenia stają się wartości funkcji w tych węzłach - czyli w praktyce wektor kilkunastu liczb na każdą krawędź zamiast pojedynczej wagi.

Podsumowując, KAN to sieć neuronowa, która uczy się kształtu funkcji aktywacji dla każdej wejściowej zmiennej do neuronu, zamiast uczyć się tylko ich liniowej kombinacji. Jest to istotne odejście od tradycyjnego modelu perceptronu, w którym wagi są liczbowe, a funkcja nieliniowości pozostaje niezmienna (np. ReLU, sigmoida). W KAN nieliniowość jest specyficzna dla każdego wejścia danego neuronu i zmienia się podczas treningu.

3.2. Architektura i właściwości modeli KAN



Rys. 3.1: Schemat architektury Kolmogorov-Arnold Network (KAN) na przykładzie prostej sieci (źródło własne).

W modelu KAN każdy neuron sumuje sygnały od wszystkich neuronów z warstwy poprzedniej. Sygnały te nie są mnożone przez stałe wagi, lecz najpierw przechodzą przez funkcje (białe bloki na rysunku) specyficzną dla każdej pary połączenia. Ilustracja pokazuje również warstwę wyjściową, gdzie sumowane są wszystkie sygnały z warstwy ukrytej po przejściu przez kolejne funkcje Φ. Formalnie, pojedyncza warstwa KAN przekształcająca wektor wejściowy $\boldsymbol{x} = [x_1, ..., x_z]$ w wektor wyjściowy $\boldsymbol{y} = [y_1, ..., y_{z'}]$ działa według zależności:

$$y_i = \sum_{j=1}^{z} \varphi_{i,j}(x_j), \text{dla} \quad i = 1, ..., z',$$

gdzie $\varphi_{i,j}$ jest funkcją przypisaną do krawędzi z j-tego wejścia do i-tego neuronu wyjściowego. Wszystkie $\varphi_{i,j}$ są uczone podczas treningu przejmując role wag. Taka warstwa jest odpowiednikiem warstwy w pełni połączonej (dense layer) w klasycznej sieci, z tą różnicą, że w zwykłej warstwie mielibyśmy $y_i = \sigma\left(\sum_j w_{i,j} x_j + b_i\right)$ ze stałą funkcją aktywacji σ (np. ReLU).

Architektura wielowarstwowa KAN powstaje poprzez złożenie kolejnych warstw. Uogólnienie pierwotnej konstrukcji na dowolną liczbę i szerokość warstw pozwala zwiększyć ekspresyjność modelu. Działanie sieci o L warstwach można wówczas zapisać rekurencyjnie w następującej postaci:

$$h^0 = x,$$

$$h^{(l+1)} = \mathbf{\Phi}_l\left(h^{(l)}\right), l = 0, 1, ..., L-1,$$

gdzie Φ_l oznacza przekształcenie przez l-tą warstwę KAN według powyższego wzoru (Φ_l jest macierzą funkcji, rozmiaru $d_{l+1} \times d_l$ jeśli d_l to liczba wyjść warstwy $l-1$, a d_{l+1} - liczba wyjść warstwy l) [17]. Składanie takich warstw tworzy sieć głęboką. Dla porównania, standardowa głęboka sieć MLP z nieliniowością σ między warstwami miałaby:

$$h^{(l+1)} = \sigma\big(W_l \times h^{(l)} + b_l\big),$$

gdzie W_l to macierz wag, a b_l to wektor biasów. W KAN nie ma macierzy wag W_l - jej rolę spełnia „macierz funkcji" Φ_l.

Kilka właściwości modelu KAN wynikających z tej architektury:

Więcej parametrów na połączenie: Podczas gdy w MLP pojedyncze połączenie neuron-neuron niesie 1 parametr (wagę), w KAN to połączenie niesie wiele parametrów opisujących funkcję φ. Przykładowo, jeśli używamy B-splinów stopnia 3 z m odcinkami, to każda funkcja ma $m+3$ parametrów. Typowe ustawienia mogą mieć np. 5 segmentów, co daje 8 parametrów na krawędź. To oznacza, że sieć KAN może mieć znacznie więcej parametrów niż odpowiednik MLP o tej samej topologii.

Lepsza ekspresyjność: Funkcja krawędzi $\varphi(x)$ może realizować dowolne przekształcenie zmiennej, podczas gdy MLP ma tylko skalowanie $\boldsymbol{w} \cdot \boldsymbol{x}$. Na przykładzie: jeśli zależność między x a wyjściem sieci jest silnie nieliniowa i różna w różnych zakresach x, to MLP musiałby kombinacją neuronów to uchwycić - KAN może nauczyć jedną krawędź $\varphi(x)$ tak, że już ta jedna funkcja odwzoruje np. x na x^3 dla małych wartości i na $\log x$ dla dużych (to oczywiście przykład hipotetyczny). Dzięki temu KAN może potencjalnie potrzebować mniej neuronów warstwy ukrytej do aproksymacji skomplikowanej funkcji. Znacznie mniejsze sieci KAN potrafiły osiągnąć porównywalną lub wyższą dokładność niż dużo większe MLP w zadaniach aproksymacji danych [5].

Interpretowalność: Jednym z mocnych argumentów autorów KAN jest łatwość interpretacji wyuczonego modelu [5]. Każdą funkcję $\varphi_{i,j}(x)$ można po treningu wyrysować na wykresie, jest to zależność jednej zmiennej, zrozumiała dla człowieka (np. może okazać się, że funkcja ta przypomina pewną znaną zależność matematyczną). To duża zaleta w porównaniu z konwencjonalnymi sieciami, które działają jak „czarne skrzynki". W KAN można dosłownie zajrzeć do wnętrza i zobaczyć, jakie transformacje nakładane są na poszczególne wejścia. Przykładowo, w zastosowaniu do odkrywania praw fizycznych KAN był w stanie nauczyć się funkcji krawędzi odpowiadających fundamentalnym zależnościom (jak np. prawo kwadratowe) i poprzez inspekcję tych funkcji można te prawidłowości odczytać [5]. Również w naszym kontekście kompresji obrazów pewna interpretowalność może się pojawić np. możemy zobaczyć, jak sieć transformuje poziomy jasności czy koloru przed kompresją. Ponadto, istnieją metody przycinania (pruning) dedykowane KAN, gdzie można usunąć krawędzie o znikomej wadze (tj. funkcje bliskie zerowej) aby uszczuplić model bez utraty jakości [17].

Brak konwolucyjności i lokalności: Warto zauważyć, że podstawowa architektura KAN jest z natury w pełni połączona i nie posiada mechanizmu splotu ani lokalnych receptorów tak jak CNN. To znaczy, że jeśli stosujemy KAN do obrazu, najprostsze podejście to potraktować cały obraz jako wektor wejściowy i łączyć każdy piksel z każdym neuronem ukrytym. W przypadku obrazów o dużej rozdzielczości byłoby to niewyobrażalnie dużo połączeń i parametrów - np. obraz 32×32 (1024 piksele) i tylko 100 neuronów ukrytych dałby $1024 \times 100 = 102400$ funkcji krawędzi do nauczenia (każda z kilkoma parametrami). Można podjąć próbę wprowadzenia lokalności: np. podzielić obraz na mniejsze bloki/patche i dla każdego stosować oddzielną sieć KAN, lub opracować konwolucyjną wersję KAN.

Złożoność obliczeniowa: Z punktu widzenia obliczeń, KAN jest bardziej wymagający od MLP. Każde połączenie wymaga obliczenia funkcji nieliniowej zamiast prostego mnożenia. W oryginalnrj pracy wskazano, że na CPU KAN też bywa wolniejszy od MLP przy tej samej liczbie neuronów, choć bywa dokładniejszy [19]. Pojawiają się jednak optymalizacje, np. wspomniany wariant FastKAN zastępuje funkcje sklejane Gaussowskimi funkcjami bazowymi, które

mogą być obliczane szybciej [9]. Ponadto można zmniejszać liczbę parametrów np. poprzez ograniczenie stopnia spline lub używając aproksymacji ortogonalnych. Reasumując, klasyczne sieci CNN korzystają z gotowych bibliotek tensorycznych które są dobrze zoptymalizowane i przebadane, a sieciach KAN zoptymalizowane implementacje są na etapie eksperymentalnym.

3.3. Porównanie z klasycznymi modelami neuronowymi

Zamiast dokładania warstw czy filtrów można zmienić sposób w jaki modele implementują nieliniowość. Ten rodział ma na celu porównać, jak ta strategia w KAN wypada względem MLP i CNN oraz innych popularnych implementacji MLP.

Tabela 3.1: Prównanie teoretyczne KAN, MLP, CNN

KAN a MLP	W klasycznym MLP nieliniowość (np. ReLU) jest stała, a uczone są jedynie wagi liniowe, w KAN uczone są jednowymiarowe funkcje na krawędziach połączeń, które zastępują stałe wagi. KAN osiąga porównywalną dokładność przy mniejszej liczbie neuronów, kosztem większej liczby parametrów na połączenie i większego kosztu obliczeniowego [5], [17].
KAN a CNN	KAN nie ma wrodzonej lokalności, przez co dla dużych obrazów może być mniej efektywny. Trwają jednak prace nad warstwami konwolucyjnymi z komponentami KAN oraz nad architekturami hybrydowymi (np. U-Net), które potrafią dorównywać lub przewyższać klasyczne CNN przy mniejszej liczbie parametrów. Wyniki empiryczne wskazują, że podejścia te są komplementarne i mogą być łączone [19].
KAN a inne MLP	KAN wpisuje się w szerszy nurt alternatywnych aktywacji/parametryzacji w MLP (np. SIREN z funkcjami sinusoidalnymi). Jego wyróżnikiem jest między innymi interpretowalność. Wyuczone funkcje $\varphi_{i,j}$ można analizować i wizualizować. Pojawiły się też warianty przyspieszone, m.in. FastKAN z radialnymi funkcjami bazowymi (RBF), które zachowują wysoką jakość przy znacznie krótszym czasie wnioskowania [5], [9].

Podsumowując, KAN to alternatywna architektura sieci neuronowej, która w pewnych zadaniach może osiągać lepsze wyniki niż klasyczny MLP, oraz potencjalnie rywalizować z CNN. Jej moc tkwi w elastyczności funkcji aktywacji, a słabość w złożoności obliczeniowej.

3.4. Zalety i ograniczenia w kontekście kompresji obrazów

W kontekście kompresji obrazów za pomocą autoenkoderów, można wymienić następujące zalety modeli KAN:

Wysoka zdolność aproksymacyjna na ograniczonej liczbie neuronów: dzięki uczonym funkcjom na wagach, KAN może potrzebować mniej neuronów warstwy ukrytej, do osiągnięcia wysokiej jakości rekonstrukcji, od autoenkodera MLP. To oznacza potencjalnie bardziej kompaktowy model kompresujący.

Elastyczność funkcji aktywacji: w autoenkoderze KAN zarówno enkoder, jak i dekoder mogą uczyć się nietypowych transformacji dostosowanych do danych. Przykładowo, może nauczyć się, że dla pewnego zakresu intensywności pikseli lepiej je skompresować (np. ciemne cienie potraktować nieliniowo inaczej niż jasne obszary). Tradycyjny autoenkoder z ReLU czy tanh nie ma takiej swobody - musiałby poprzez kombinację neuronów przybliżyć podobny efekt.

Interpretowalność transformacji kompresującej: W standardowych autoenkoderach trudno zrozumieć, co dokładnie robią kolejne warstwy. W autoenkoderze KAN możemy przeanalizować wyuczone funkcje $\varphi_{i,j}$. Może to dać pewien wgląd w to, co sieć uznaje za istotne informacje w obrazie, a co pomija. Przykładowo, jeśli któraś funkcja krawędzi jest niemal liniowa, to znaczy że dana cecha jest przetwarzana niemal liniowo. Jeśli inna jest silnie nieliniowa (np. wygasza małe wartości do zera), to może sygnalizować działanie podobne do progowania (thresholding).

Uniezależnienie od konkretnej funkcji aktywacji: W klasycznych sieciach neuronowych (np. CNN) wybór funkcji aktywacji (ReLU, Leaky ReLU, ELU itp.) istotnie wpływa na przebieg i efektywność uczenia. W modelach KAN ten problem nie występuje, ponieważ kształt funkcji aktywacyjnych dostosowuje się automatycznie w trakcie treningu. Projektant modelu nie musi ręcznie dobierać najlepszej aktywacji. Z drugiej jednak strony, sieć musi samodzielnie „nauczyć się" odpowiednich kształtów od zera, co bywa wyzwaniem przy niewielkiej liczbie danych. Dodatkowo, ponieważ początkowa postać funkcji jest losowa, uzyskiwane wyniki mogą się różnić pomiędzy poszczególnymi próbami uczenia.

Niemniej, modele KAN mają też istotne ograniczenia i wyzwania w zastosowaniu do kompresji obrazów.

Znacząco zwiększa liczba parametrów: Kan posiada znacząco większą liczbe parametrów na połączenie. W autoenkoderze obrazów, gdzie już i tak liczba wag bywa duża, KAN może być trudny do trenowania. Stwierdzono, że podstawowy KAN ma spory narzut czasowy [19]. FastKAN częściowo to adresuje poprzez użycie szybszych funkcji RBF kosztem minimalnej utraty dokładności.

Problemy z optymalizacją i zbieżnością: Większa elastyczność oznacza też większą szanse popadnięcia w minima lokalne czy przeuczenia. Funkcje spline mogą potencjalnie przybrać dziwny kształt, jeśli dane treningowe są niewystarczające, prowadząc do artefaktów w rekonstrukcji.

Zużycie pamięci: Więcej parametrów to też większe zapotrzebowanie pamięciowe zarówno do przechowania modelu, jak i w trakcie treningu. W literaturze wspomina się, że podstawowy KAN wymagał dużo pamięci i mocy obliczeniowej, co zainspirowało różne warianty redukujące parametry [20].

Brak dotychczasowych zastosowań w kompresji: To ograniczenie natury praktycznej. KAN jest tak nowym modelem, że nie ma wypracowanych „dobrych praktyk" jak go użyć do autoenkoderów. Dla CNN istnieje baza zastosowań.

Podsumowując, modele KAN oferują nowy stopień swobody w modelowaniu złożonych zależności, co potencjalnie czyni je potężnym narzędziem do uczenia reprezentacji obrazu. KAN może teoretycznie prowadzić do lepiej dopasowanej kompresji kosztem większej złożoności modelu. W następnych rozdziałach zobaczymy, w jakim stopniu te teoretyczne zalety przekładają się na rzeczywiste wyniki w porównaniu z klasyczną siecią konwolucyjną.

4. Zastosowanie modeli KAN w kompresji obrazu

4.1. Przegląd podejść autoenkoderowych

Kompresja obrazów za pomocą autoenkoderów polega na tym, że sieć neuronowa uczy się odwzorowania $f_\theta : X \to Z$ (enkoder), gdzie X jest przestrzenią obrazów wejściowych, a Z przestrzenią zakodowanych reprezentacji (o niższym wymiarze), oraz odwzorowania $g_\theta : Z \to X$ (dekoder), które rekonstruuje obraz z kodu. Po treningu, obraz x jest kompresowany do $z = f_\theta(x)$, następnie z jest zapisywane (ew. kwantyzowane i zakodowane bitowo), a do odtworzenia obrazu stosuje się $\hat{x} = g_\theta(z)$.

Klasyczne autoenkodery do kompresji obrazów często wykorzystywały warstwy konwolucyjne ze względu na ich efektywność i zdolność do wykrywania lokalnych wzorców w obrazach. Trening takiego autoenkodera odbywa się poprzez minimalizację błędu rekonstrukcji, najczęściej sumy kwadratów błędów (MSE) między x a $\hat{x} = g(f(x))$, gdzie x oznacza obraz oryginalny a $\hat{x}$ obraz skompresowany.

Poza autoenkoderami deterministycznymi, we współczesnych podejściach pojawiają się takie modele jak VAE (wariacyjne autoenkodery). Zamiast jednego „sztywnego" kodu, VAE uczy się rozkładu możliwych kodów dla danego obrazu. $q(z|x)$ oznacza jakie kody z dobrze reprezentują obraz x. Podczas kompresji losowany jest kod z tego rozkładu (w praktyce enkoder podaje średnią i „rozrzut", pozostaje dobrać próbkę). Do uczenia dodaje się warunek, aby kody miały uporządkowany kształt. Temu służy prior $p(z)$, czyli reprezentacja kodów jak najbliższa naturalnej, zwykle prosta gaussowska chmura wokół zera [21].

Innym wartym wspomnienia podejściem jest kompresja progresywna (RNN). Model (np. LSTM) generuje kolejne porcje bitów, które stopniowo redukują błąd rekonstrukcji. Na początku przesyłany jest krótki kod dający prosty zarys obrazu, każda następna iteracja dosyła poprawki (residua). Dzięki temu można uzyskać zmienny bitrate - wcześniejsze przerwanie sekwencji daje mniejszy strumień i gorszą jakość, dalsze kroki zwiększają liczbę bitów i jakość. Taki mechanizm naturalnie adaptuje się do zawartości: proste obrazy potrzebują mniej kroków, złożone - więcej. Ceną jest złożoność treningu (stabilność uczenia długich sekwencji, akumu-

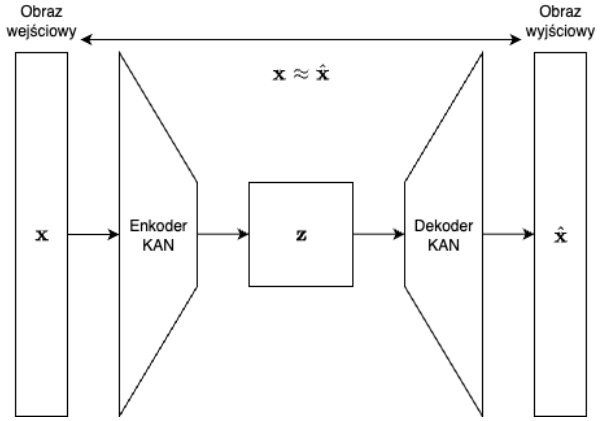
lacja błędów, większy czas inferencji sekwencyjnej) oraz trudniejsze strojenie kompromisu między szybkością a jakością [22].

4.2. Opis autoenkodera opartego na KAN

W ramach pracy zaimplementowano model autoenkodera KAN jako klasyczny schemat enkoder-dekoder. Architektura (rys. 4.1) składa się z:

Enkodera KAN: pojedyncza warstwa KAN, która przekształca spłaszczony obraz x wejściowy (wektor długości N) na wektor kodowy z o wymiarze K ($K \leq N$).

Dekodera KAN: pojedyncza warstwa KAN, która mapuje wektor z (bottleneck) na wektor $\hat{x}$ o długości N. Następnie przekształca na obraz o wymiarach oryginału. Na końcu dekodera stosujemy funkcję sigmoidalną, aby ograniczyć wartości pikseli rekonstruowanego obrazu.



Rys. 4.1: Schemat architektury autoenkodera KAN (źródło własne).

```
class KANAutoenkoder(nn.Module):
    def __init__(...):
        ...
        self.input_dim = H * W
        self.enc = KANLayer(self.input_dim, latent_dim, num_grid=num_grid,
spline_type=spline_type)
        self.dec = nn.Sequential(
            KANLayer(latent_dim, self.input_dim, num_grid=num_grid,
spline_type=spline_type),
            nn.Sigmoid(),
        )
```

Pojedyńcza warstwa KAN oblicza wyjście wyłącznie na podstawie funkcji sklejanej, zgodnie z:

$$y_i = \sum_{j=1}^{N} \varphi_{i,j}(x_j)$$

gdzie $\varphi_{i,j}$ to uczona funkcja neuronu na równomiernej siatce węzłów ($grid \subset [-5, 5]$), liczba węzłów (`num_grid`) wynosi 6. Typ splajnu wybierany jest parametrem `spline_type` (Dostępne opcje to: `linear`, `bezier`, `catmull_rom`, `bspline`). Dla każdego neuronu uczony jest wektor współczynników o długości `num_grid`.

```
class KANLayer(nn.Module):
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        ...
        sp = spline_interpolate(x_e, self.grid, coeff_e,
spline_type=self.spline_type)
        return sp.sum(dim=-1)
```

Taka architektura autoenkodera KAN jest zatem relatywnie płytka, zawiera po jednej warstwie KAN w enkoderze i dekoderze. Mimo to, pojedyncza warstwa KAN może odwzorowywać skomplikowane zalżności.

4.3. Proces trenowania i rekonstruowania obrazów

Trenowanie autoenkodera KAN przebiega analogicznie jak klasycznego autoenkodera, z tą różnicą, że musieliśmy uwzględnić specyfikę warstw KAN. Przygotowanie danych polegało na normalizacji obrazów do zakresu [0,1]. Eksprymenty zostały przeprowadzone na obrazach w skali szarości, jeśli oryginały były kolorowe, konwertowane były do skali szarości przed podaniem do sieci.

Dane podzielono na zbiory treningowy i walidacyjny (testowy) w proporcji 2:8 (trening:walidacja). Model trenowano metodą uczenia nienadzorowanego (unsupervised), wejściem i oczekiwanym wyjściem były te same obrazy, a sieć uczyła się identycznościowego

odwzorowania $x \mapsto \hat{x}$ poprzez bottleneck z. Walidacja polegała na sprawdzaniu średniego błędu rekonstrukcji na obrazach ze zbioru walidacyjnego.

Jako podstawową miarę błędu wykorzystaliśmy błąd średniokwadratowy (MSE) między obrazem oryginalnym x a rekonstrukcją $\hat{x} = g_\theta(f_\theta(x))$. Jako Optymalizator został wybrany algorytm Adam z początkowym współczynnikiem uczenia $\alpha = 10^{-3}$.

Jedynymi parametrami w warstwie KAN są współczynniki funkcji sklejanych. Siatka węzłów `grid` jest deterministyczna ($[-5, 5]$). Współczynniki splinów inicjalizujemy losowo zgodnie z:

$$\text{coeffs} \sim N(0, 0.05^2)$$

w kodzie: `torch.randn(n_out, n_in, num_grid) * 0.05`, czyli ze średnią 0 i odchyleniem standardowym 0.05. Taki start powoduje, że na początku $\varphi_{i,j}$ mają małą amplitudę, więc wyjście enkodera oraz wejście dekodera są bliskie zeru. W toku uczenia współczynniki odchylają się od zera, kształtując nieliniowe $\varphi_{i,j}$ i podnosząc zdolność aproksymacji.

Sieć trenowaliśmy metodą stochastycznego spadku gradientu w porcjach danych (mini-batch). Rozmiar paczki to 64 obrazki. Trenowanie prowadzono przez 10 epok, już po kilku epokach błąd rekonstrukcji stabilizował się. W celu wybrania najlepszego modelu, po każdej epoce obliczano błąd na zbiorze walidacyjnym, zapisywano parametry modelu, jeśli błąd ten był najmniejszy dotychczas. Następnie do dalszego trenowania kontynuowano od bieżącego stanu, finalnie do oceny brano parametry z epoki o najniższym błędzie walidacyjnym.

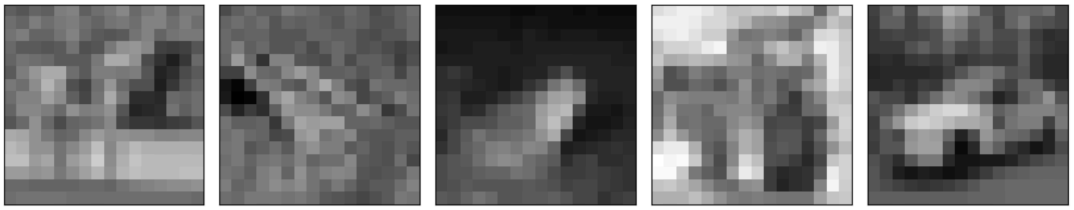
Po zakończeniu treningu autoenkodera można wykorzystać go do kompresji i dekompresji obrazów. Kompresja polega na przepuszczeniu nowego obrazu x (nieobecnego w zbiorze uczącym) przez enkoder f_θ i uzyskaniu wektora kodowego $z = f_\theta(x)$. W implementacji z jest wektorem liczb zmiennoprzecinkowych o wymiarze K. Dekodowanie odbywa się przez podanie z do dekodera g_θ, który generuje obraz $\hat{x} = g_\theta(z)$. Otrzymaną rekonstrukcję porównujemy z oryginałem. Do oceny jakości, poza MSE, używamy PSNR (Peak Signal-to-Noise Ratio) oraz SSIM (Structural Similarity Index), aby uwzględnić zarówno miarę fizyczną błędu jak i percepcyjną (SSIM). Obie te metryki liczymy dla zbiorów testowych po zakończeniu treningu.

5. Środowisko i metodologia

5.1. Opis zbiorów danych

Do treningu i testowania modeli wykorzystano zbiory obrazów oraz zestaw danych syntetycznych, aby sprawdzić zachowanie modeli w różnych warunkach:

CIFAR-10: popularny zbiór fotografii w niskiej rozdzielczości [23]. Oryginalna rozdzielczość CIFAR-10 to 32×32 piksele. W naszych eksperymentach wykorzystaliśmy wersję przeskalowaną do 16×16 pikseli w skali szarości. Przekształcenie do odcieni szarości zrealizowaliśmy za pomocą transformacji luminancji. Na ilustracjach pokazujemy obrazy wielokrotnie powiększone względem oryginału, aby były czytelne na stronie. Takie powiększenie powoduje widoczną pikselizację.



Rys. 5.1: Przykładowe obrazy ze zbioru CIFAR-10.

MNIST: zbiór 70 000 ręcznie pisanych cyfr w odcieniach szarości, znormalizowanych do rozmiaru 28×28 [24]. Użyty jako prostego przypadku testowego: obrazy MNIST mają bardzo niską złożoność (jednobarwne tło, biały odręczny symbol), co stanowi łatwe zadanie dla autoenkodera. Zbiór ten pozwolił zweryfikować, czy model KAN nie ma problemów z podstawowym dopasowaniem danych i dał punkt odniesienia dla minimalnego błędu, jaki można uzyskać przy danym K.

Rys. 5.2: Przykładowe obrazy ze zbioru MINIST.

Shapes: własny zbiór syntetyczny wygenerowany na potrzeby testów. Składa się on z prostych kształtów geometrycznych (kół, kwadratów, trójkątów, linii) rysowanych losowo na czarnym tle. Wygenerowano 1000 obrazków 32×32 z losowo parametryzowanymi figurami (położenie, rozmiar, rotacja). Przykładowe obrazy z tego zbioru to białe figury na czarnym tle. Dane te służyły zbadaniu, czy KAN potrafi efektywnie kompresować obrazy z wyraźnymi strukturami.

Rys. 5.3: Przykładowe obrazy ze zbioru shapes.

Large: aby ocenić modele na bardziej złożonych i zróżnicowanych obrazach, przygotowaliśmy zestaw większych obrazów fotograficznych, przeskalowanych do rozdzielczości 64×64 pikseli w skali szarości. Wykorzystano tutaj zbiór STL-10 [25], zawierający 13 000 kolorowych obrazów 96×96, które dla naszych potrzeb zaskalowano do 64×64. Celem wykorzystania takiego zbioru było sprawdzenie, jak autoenkodery poradzą sobie z zawartością o wysokiej różnorodności (sceny naturalne, tekstury, twarze).

Rys. 5.4: Przykładowe obrazy ze zbioru large.

Reasumując, do trenowania i porównania autoenkoderów używaliśmy głównie CIFAR-10 (16×16) i STL-10 (64×64) jako reprezentantów obrazów naturalnych, MNIST jako prostego przypadku obrazów upraszczonych, oraz zbioru shapes jako przypadku obrazów syntetycznych o ostrych krawędziach. Dzięki temu mogliśmy zaobserwować ewentualne różnice w zachowaniu modeli KAN i CNN w zależności od charakteru danych.

5.2. Środowisko eksperymentalne i implementacja

Wszystkie eksperymenty zaimplementowano w języku Python z wykorzystaniem biblioteki PyTorch (wersja 1.13). Cały kod został uruchomiony na macbook pro z procesorem M2 Pro.

Implementację warstwy KAN oparto na idei klasy KolmogorovArnoldLayer z biblioteki pyKAN [17], jednak przygotowano własną wersję dostosowaną do potrzeb eksperymentu. Zaimplementowany moduł KANLayer, dziedziczący po `nn.Module`, podczas inicjalizacji tworzy komplet parametrów definiujących wszystkie funkcje $\varphi_{i.j}$ w danej warstwie. Parametry te są przechowywane w tensorze `coeffs`, który zawiera wartości funkcji φ w węzłach siatki i jest inicjalizowany małymi, losowymi odchyleniami. Jednocześnie tworzona jest równomierna siatka węzłów `self.grid` w przedziale $[-5,5]$, względem której realizowana jest interpolacja funkcji krawędziowych. Dzięki takiej konstrukcji każda para połączeń (wejście-neuron) ma własną, uczoną w trakcie treningu postać funkcji $\varphi_{i.j}$, a warstwa może elastycznie modelować nieliniowe zależności między sygnałami.

Warto wspomnieć o złożoności obliczeniowej i czasie treningu. Trening pojedynczej konfiguracji modelu (np. KAN oraz CNN na CIFAR-10 przy zadanym K) trwał od kilkunastu minut do kilku godzin, zależnie od modelu. Autoenkoder CNN, oraz Autoenkoder oparty o FastKAN, trenowały się znacznie szybciej. Użycie splotów i mniejsza liczba parametrów sprawiały, że 10 epok na zbiorze CIFAR-10 zajmowało ok. 45min. Własna implementacja autoenkoder KAN była znacznie wolniejszy. 10 epok na CIFAR-10 to ok. 3-4 godziny, głównie z powodu konieczności obliczania splinów składanych dla każdej iteracji.

5.3. Parametryzacja i konfiguracja modelu

Aby zbadać wpływ stopnia kompresji, przeprowadzono eksperymenty dla kilku szerokości bottlenecku, $K \in \{256, 128, 64, 32, 16\}$. Dla obrazów o rozmiarze 32×32 piksele ($N = 1024$) współczynnik kompresji, zdefiniowany jako $\frac{N}{K}$, wynosi od 4 (dla $K = 256$) do 64 (dla $K = 16$). Dla obrazów 64×64 ($N = 4096$) współczynnik ten przyjmuje wartości od 16 do 256. Każdą konfigurację K trenowano i ewaluowano niezależnie, bez treningu wielozadaniowego.

Wszystkie modele trenowano w identycznych warunkach: 10 epok na każdym zbiorze danych; optymalizator Adam z krokiem uczenia 10^{-3} (bez harmonogramu zmian), funkcja kosztu to MSE między obrazem wejściowym a rekonstrukcją, wielkość partii (batch size) równa 64. Po zakończeniu treningu wybierano parametry z epoki o najniższym błędzie walidacyjnym.

Podsumowując, przygotowane środowisko eksperymentalne umożliwiło zbadanie szeregu konfiguracji modeli KAN oraz ich porównanie z klasycznym podejściem konwolucyjnym. W kolejnych rozdziałach przedstawiono wyniki tych porównań, obejmujące metryki ilościowe (MSE, PSNR, SSIM) oraz przykładowe rekonstrukcje obrazów dla wybranych przypadków.

6. Wyniki eksperymentów

Wyniki odnoszą się do trzech rodzin modeli: CNN, KAN (warianty: linear, bézier, catmull, bspline) oraz FastKAN. Dla każdego zbioru danych (dataset): MNIST 28×28, CIFAR-10 16×16, Shapes 32×32, Large 64×64. Trenowano autoenkodery z pięcioma wariantami szerokości bottleneck: (16, 32, 64, 128, 256). Jakość oceniano metrykami MSE (niżej = lepiej), PSNR (wyżej = lepiej) i SSIM (wyżej = lepiej). Czas inferencji (czas potrzebny wytrenowanemu modelowi na wykonanie kompresji) raportowano jako Czas [ms]. „Najlepszy model" w danej kombinacji wybierano według priorytetu: SSIM $\overset{\text{równość}}{\Longrightarrow}$ PSNR $\overset{\text{równość}}{\Longrightarrow}$ MSE, równość oznacza sytuację, w której dwie lub więcej konfiguracje osiągają identyczną wartość danej metryki. W takim przypadku wybór najlepszego modelu jest rozstrzygany kolejną metryką w hierarchii priorytetów. Tabelka przedstawia również rozmiar każdego z modeli za pośrednictwem liczby parametrów (l. param.) które model może dostosowywać w trakcie uczenia.

Tabela 6.1: Najlepiej dopasowane modele

dataset	bottleneck	model	SSIM	PSNR	MSE	l. param.	Czas [ms]
cifar16	16	kan (linear)	0.6887	20.5737	2.6822	49152	0.0283
cifar16	32	kan (linear)	0.8227	22.7102	1.6254	98304	0.1147
cifar16	64	kan (linear)	0.9049	25.2013	0.9101	196608	0.1072
cifar16	128	kan (linear)	0.9474	27.4981	0.5314	393216	0.2017
cifar16	256	kan (linear)	0.9653	29.2584	0.3496	786432	0.3346
large	16	cnn	0.2476	16.2021	112.5016	584977	2.6375
large	32	cnn	0.2709	16.6742	100.5952	718113	2.6351
large	64	cnn	0.2568	16.3439	107.7177	984385	2.6509
large	128	cnn	0.2742	16.728	99.2994	1516929	2.6126
large	256	fastkan	0.2789	17.4895	80.1732	18887440	0.1374
mnist	16	cnn	0.8852	20.5086	8.539	165841	0.4304
mnist	32	cnn	0.9208	22.158	5.8139	217057	0.4299
mnist	64	fastkan	0.9293	24.6805	3.2175	905728	0.0168
mnist	128	fastkan	0.9441	27.3376	1.7049	1809088	0.0158
mnist	256	fastkan	0.943	28.051	1.606	3615808	0.0308
shapes	16	fastkan	0.0882	11.8792	74.9681	298048	0.0141
shapes	32	fastkan	0.1654	14.0272	45.4278	593008	0.0184
shapes	64	fastkan	0.4814	15.8047	28.6465	1182928	0.0211
shapes	128	fastkan	0.581	16.6445	23.3089	2362768	0.0382
shapes	256	fastkan	0.649	17.2391	21.1224	4722448	0.109

Wszystkie wyniki zostały zamieszczone w dodatku A zamieszczonym na końcu pracy.

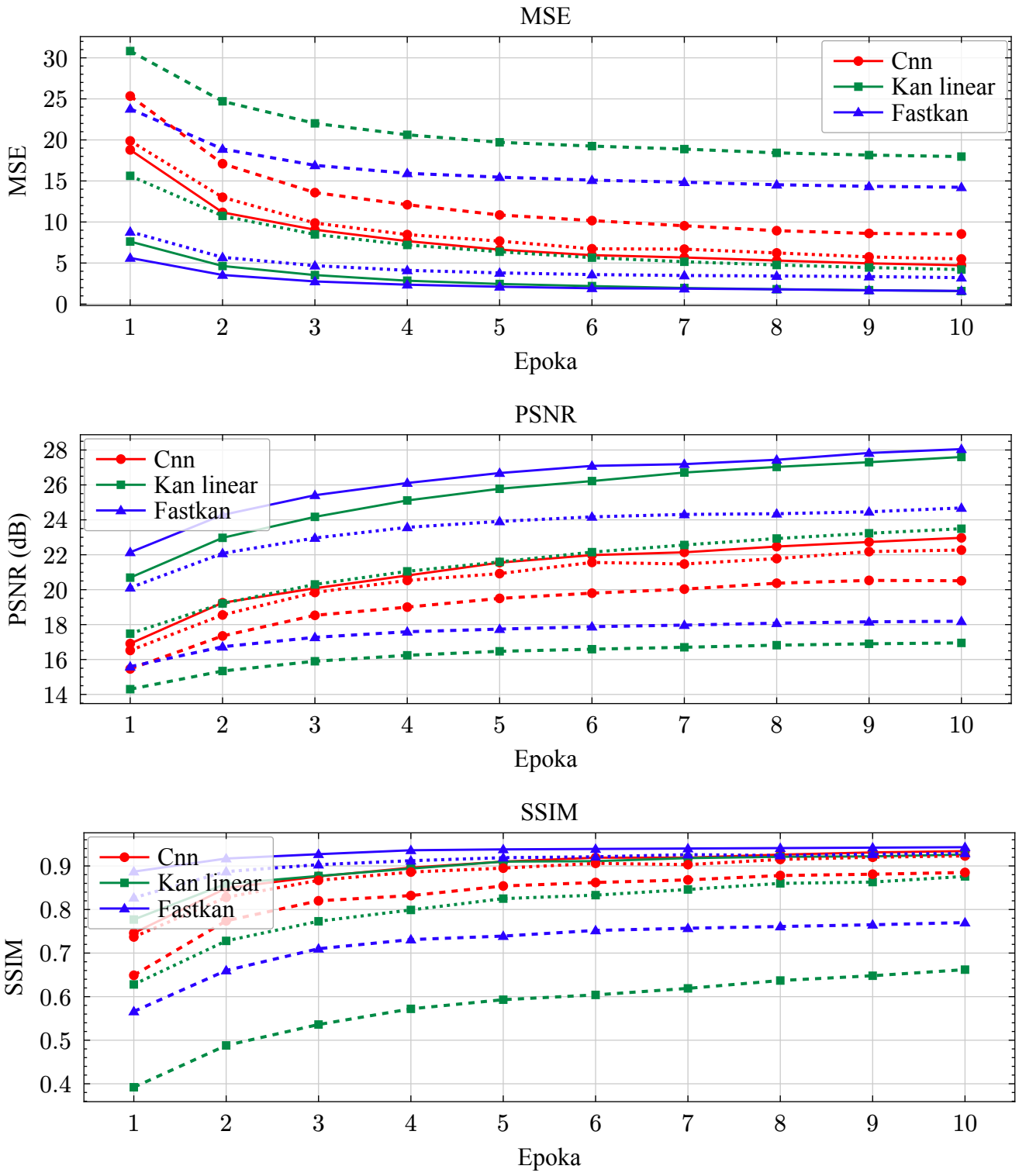
6.1. Analiza ilościowa

Na zbiorze MNIST przy małym bottlenecku ($16 - 32$) przewagę uzyskuje CNN (SSIM 0.885 i 0.921). Po zwiększeniu pojemności reprezentacji wyraźnie wygrywa FastKAN — osiąga SSIM do 0.944 i PSNR do 28.05 dB, jednocześnie znacząco obniżając MSE względem CNN. Wraz ze wzrostem bottlenecku FastKAN efektywniej wykorzystuje dodatkową informację, co przekłada się na zauważalny skok jakości rekonstrukcji.

W całym badanym zakresie bottlenecku 16-256 dla CIFAR-10 najlepsze wyniki uzyskuje KAN (linear), dochodząc do SSIM 0.965 i PSNR 29.26 dB. Wynik ten sugeruje, że elastyczne funkcje jednowymiarowe w KAN szczególnie dobrze dopasowują się do statystyk małych, zróżnicowanych obrazków, co przekłada się na bardzo dokładne rekonstrukcje.

We wszystkich konfiguracjach zbiou Shapes zwycięża FastKAN, a jakość systematycznie rośnie wraz z pojemnością (SSIM od 0.088 do 0.649). Model ten najlepiej zachowuje strukturę prostych figur i ostre krawędzie, podczas gdy CNN ma tendencję do rozmywania konturów, co widać w niskich wartościach SSIM nawet przy większym bottlenecku.

Na zbiorze „Large”, dla bottlenecku 16-128 nieznaczną przewagę posiada CNN (SSIM $\sim 0.25 - 0.27$), jednak przy 256 prym przejmuje FastKAN (SSIM 0.279; PSNR 17.49 dB; wyraźny spadek MSE). Oznacza to, że w przypadku większych i bardziej złożonych obrazów przewaga modeli KAN ujawnia się dopiero przy największej pojemności reprezentacji, podczas gdy dla mniejszych bottlenecków CNN zachowuje minimalnie wyższą zgodność strukturalną.

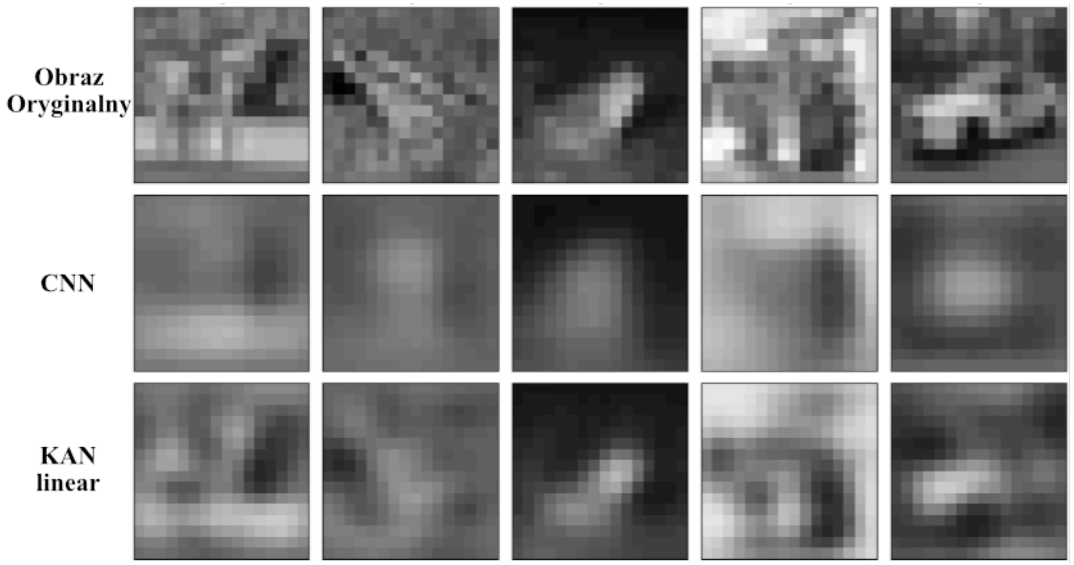


Rys. 6.1: Ewolucja MSE, PSNR oraz SSIM w trakcie uczenia na zbiorze MNIST (źródło własne).

Na wykresach przedstawiono jak zmieniały się metryki w trakcie uczenia modeli. Dla czytelności na wykresach widzimy 3 modele dla 3 rozmiarów bottleneck: 256(linia ciągła), 64(linia kropkowana), 16(linia przerywana). Widać na nich jak silnie stopień kompresji wpływa na jakość kompresji prowadzonej przez modele oparte o KAN.

6.2. Analiza jakościowa

Przy bardzo silnej kompresji (bottleneck $16 - 32$), CNN lepiej zachowuje ogólny „zarys" treści szczególnie widoczne jest to na MNIST, gdzie SSIM pozostaje wysokie. Rekonstrukcje są jednak wyraźnie gładsze i bardziej rozmyte, co z jednej strony sprzyja czytelności konturów przy skrajnie małej pojemności, z drugiej zaś ogranicza dalszą poprawę metryk po zwiększeniu szerokości wąskiego gardła, model ma tendencję do „nasycania się" i nie odtwarza drobnych detali równie dobrze jak pozostałe porównywane modele. KAN (linear) daje natomiast rekonstrukcje ostrzejsze i bogatsze w szczegóły (co potwierdzają wysoki PSNR i niski MSE), ale przy bardzo małym bottlenecku bywa podatny na artefakty. Wraz ze wzrostem pojemności szybko nadrabia i ilościowo przewyższa CNN, lepiej zachowując strukturę i faktury.



Rys. 6.2: Rekonstrukcja CIFAR-10 przed KAN linear oraz CNN dla bottleneck równego 32 (źródło własne).

FastKAN łączy zalety obu podejść: utrzymuje wysoką wierność strukturalną bez typowych dla KAN (linear) artefaktów i bez rozmyć charakterystycznych dla CNN. Dzięki temu dominuje na zbiorach z ostrymi krawędziami (Shapes) oraz, przy większych bottleneckach, okazuje się najlepszy także na pozostałych zbiorach, oferując równocześnie dobre metryki i naturalny odbiór wizualny.

6.3. Złożoność i szybkość

Pod względem kosztu obliczeniowego FastKAN jest w większości przypadków najszybszy, na MNIST osiąga typowo $\sim 0.016 - 0.031$ ms na obraz, a na Shapes $\sim 0.014 - 0.109$ ms, co czyni go szczególnie atrakcyjnym w zastosowaniach czasu rzeczywistego. KAN (linear) na CIFAR-10 również pracował bardzo szybko ($\sim 0.028 - 0.335$ ms), przy czym czas rośnie wraz z szerokością bottlenecku, pozostając jednak w przedziale opóźnień akceptowalnych dla aplikacji interaktywnych. CNN jest zauważalnie wolniejszy na zbiorze Large: $\sim 2.61 - 2.65$ ms na obraz, podczas gdy FastKAN przy bottlenecku 256 potrzebuje jedynie ~ 0.137 ms. Różnica o rząd wielkości na korzyść FastKAN przekłada się na realną przewagę wdrożeniową: niższe opóźnienia oraz większa responsywność systemów wykorzystujących autoenkodery do kompresji i rekonstrukcji obrazów. CNN jednak ma ogromną przewagę czasową nad modelami zaimplementowanymi od zera.

7. Wnioski

Modele oparte na KAN (zwłaszcza FastKAN) osiągają wyższą jakość rekonstrukcji niż klasyczne CNN, o ile szerokość wąskiego gardła jest wystarczająca ($\geq 64 - 128$). Potwierdzają to wartości SSIM i PSNR: na CIFAR-10 KAN linear dochodzi do SSIM = 0,965 i PSNR = 29,26 dB, a na MNIST FastKAN osiąga SSIM = 0,944 i PSNR = 28,05 dB. Dla bardzo małych bottlenecków ($16 - 32$) częściej lepszy bywa CNN, zwłaszcza na prostych danych, ponieważ lepiej zachowuje globalny zarys obiektu (wyższy SSIM). Po zwiększeniu rozmiaru bottleneck'a KAN szybciej podnosi jakość kompresji niż w CNN.

Istotna jest także specyfika danych. Dla obrazów z wyraźnymi krawędziami i prostymi strukturami (Shapes) FastKAN okazał się zdecydowanie najlepszy (przy bottleneck $\text{bottleneck} = 256$ uzyskano SSIM = 0,649), co sugeruje, że elastyczne funkcje jednowymiarowe skutecznie zachowują ostrość konturów. Na bardzo małych obrazkach naturalnych (CIFAR-10) w całym zakresie bottlenecków dominował KAN linear, osiągając niemal bezbłędne rekonstrukcje. W przypadku większych i bardziej złożonych obrazów (Large) przewaga modeli KAN ujawniała się dopiero przy mniejszym stopniu kompresji, dla mniejszych bottlenecków różnice były niewielkie i często na korzyść CNN.

Z punktu widzenia praktycznego FastKAN łączy wysoką jakość z bardzo krótkimi czasami inferencji, często o rząd wielkości krótszymi niż w CNN, co czyni go najpraktyczniejszym wyborem w zastosowaniach autoenkoderowej kompresji. W zestawieniu globalnym FastKAN okazał się najlepszy 9/20 razy, CNN 6/20, a KAN linear 5/20. W konsekwencji, gdy liczą się jednocześnie jakość i szybkość, najlepszą opcją jest FastKAN. Dla bardzo małych, zróżnicowanych obrazów przewagę potrafi mieć KAN linear, przy ekstremalnej kompresji lub bardzo prostych scenariuszach najlepsze wynik zapewniał CNN.

Warianty KAN z bardziej złożonymi krzywymi bazowymi (bézier/catmull/bspline) w takich warunkach jak podczas prowadzonych badań nie przewyższyły KAN linear ani FastKAN i nie zdobyły przewagi w żadnej kombinacji. Najprawdopodobniej wymagają one dłuższego

treningu, silniejszej regularyzacji (np. ograniczeń krzywizny/oscylacji, gładkości) lub lepszej inicjalizacji, by wykorzystać swój potencjał i ustabilizować zbieżność.

Należy pamiętać o ograniczeniach. Badane autoenkodery były płytkie i trenowane krótko (10 epok), co może sprzyjać prostszym wariantom (KAN linear) względem bardziej skomplikowanych splinów. Ponadto nie modelowano kwantyzacji ani entropii, więc oceniliśmy „czystą" rekonstrukcję, a nie rzeczywisty bitrate. Jest to ważny krok w stronę pełnego kodeka opartego na modelach uczenia maszynowego, lecz nie jego pełny substytut.

Ponadto obserwacje odpowiadają wynikom z pracy „Kolmogorov-Arnold Network Autoenkoders", gdzie porównano autoenkodery KAN z konwolucyjnymi autoenkoderami na MNIST, SVHN i CIFAR-10 i wykazano co najmniej konkurencyjną jakość rekonstrukcji. Zastosowano tam jednak bogatszą niż u nas konstrukcję (warstwa KAN + ReLU + warstwa gęsta w enkoderze i decoderze), zestawiając ją z konwolucyjnym autoenkoderami jako punktem odniesienia. Co więcej, wyniki dotyczące FastKAN pokazują, że zastąpienie B-splinów funkcjami radialnymi znacząco przyspiesza obliczenia bez spadku dokładności [26].

7.1. Możliwości dalszego rozwoju

Obiecującym kierunkiem jest rozwój architektury są głębsze autoenkodery KAN, warianty z warstwami konwolucyjnymi lub bloki hybrydowe CNN-KAN, a przy wyższych rozdzielczościach integracja z transformatorami. Taka kombinacja może lepiej wydobywać struktury lokalne, utrzymując wysoką elastyczność funkcji jednowymiarowych KAN.

Istotnym aspektem jest efektywność. Przycinanie połączeń (pruning) w KAN, regularizacja gładkości funkcji krawędziowych, kwantyzacja i destylacja do lżejszych modeli. Te zabiegi, sprzyjają wdrożeniom brzegowym i scenariuszom czasu rzeczywistego.

Bibliografia

[1] A. F. Gad, „Image Compression Using Autoencoders in Keras”, Dostęp: 24 lipiec 2025. [Online]. Dostępne na: https://www.digitalocean.com/community/tutorials/autoencoder--image-compression-keras

[2] K. Lieberman, J. Diffenderfer, C. Godfrey, i B. Kailkhura, „Neural Image Compression: Generalization, Robustness, and Spectral Biases”, w *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. Dostęp: 24 lipiec 2025. [Online]. Dostępne na: https://openreview.net/forum?id=FxRfAIj4s2

[3] J. Ballé, V. Laparra, i E. P. Simoncelli, „End-to-end Optimized Image Compression”, w *ICLR*, 2017. Dostęp: 25 lipiec 2025. [Online]. Dostępne na: https://arxiv.org/abs/1611.01704

[4] A. Kolmogorov, „On the representation of continuous functions of several variables as superpositions of continuous functions of one variable and addition”, 1957.

[5] Z. Liu *i in.*, „KAN: Kolmogorov-Arnold Networks”, 2024, Dostęp: 25 lipiec 2025. [Online]. Dostępne na: https://arxiv.org/abs/2404.19756

[6] Z. Liu i contributors, „pyKAN: Kolmogorov-Arnold Networks (oficjalna implementacja)”. Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://github.com/KindXiaoming/pykan

[7] E. S. Rémi Genet i contributors, „T-KAN: Kolmogorov-Arnold Networks (oficjalna implementacja)”, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://github.com/remigenet/tkan

[8] B. K. Mehrdad Kiamari Mohammad Kiamari, „GKAN: Graph Kolmogorov-Arnold Networks”, 2024, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/abs/2406.06470

[9] Z. Li, „Kolmogorov-Arnold Networks are Radial Basis Function Networks”, 2024, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/abs/2405.06721

[10] C. McAnlis, „How PNG Works”, 2016, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://medium.com/@duhroach/how-png-works-f1174e3cc7b7

[11] G. K. Wallace, „The JPEG Still Picture Compression Standard”, *IEEE Transactions on Consumer Electronics*, 1992, Dostęp: 2 wrzesień 2025. [Online]. Dostępne na: https://ijg.org/files/Wallace.JPEG.pdf

[12] S. Kunwar, „Quality Metrics In Image Compression”, 2021, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://sumn2u.medium.com/quality-metrics-in-image-compression-f6de68df1b8

[13] H. R. S. Zhou Wang Alan C. Bovik i E. P. Simoncelli, „The SSIM Index for Image Quality Assessment”, Dostęp: 8 wrzesień 2025. [Online]. Dostępne na: https://ece.uwaterloo.ca/~z70wang/research/ssim/

[14] I. Urbaniak i R. Pinto, „Quality Assessment of Medical Images”, 2023.

[15] A. C. F. H. Lucas Theis Wenzhe Shi, „Lossy Image Compression with Compressive Autoencoders”, 2017, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/abs/1703.00395

[16] R. T. L. T. J. B. E. A. N. J. F. M. George Toderici Wenzhe Shi, „Workshop and Challenge on Learned Image Compression (CLIC2020)”. Dostęp: 1 sierpień 2025. [Online]. Dostępne na: http://www.compression.cc/

[17] Z. Liu., „Hello, KAN! — Kolmogorov Arnold Network documentation”. Dostęp: 11 sierpień 2025. [Online]. Dostępne na: https://kindxiaoming.github.io/pykan/intro.html

[18] A. Odrzywolek, „Aproksymacja funkcji wielu zmiennych”, 2009, Dostęp: 1 sierpień 2025. [Online]. Dostępne na: https://th.if.uj.edu.pl/~odrzywolek/homepage/presentations/PL/Approx.pdf

[19] I. Drokin, „Kolmogorov-Arnold Convolutions: Design Principles and Empirical Studies”, 2024, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/html/2407.01092v1

[20] X. Z. Zhijie Chen, „LSS-SKAN: Efficient Kolmogorov-Arnold Networks based on Single-Parameterized Function”, 2024, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/pdf/2410.14951

[21] M. W. Diederik P Kingma, „Auto-Encoding Variational Bayes”, 2013, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/abs/1312.6114

[22] S. J. H. D. V. D. M. S. B. M. C. R. S. George Toderici Sean M. O'Malley, „Variable Rate Image Compression with Recurrent Neural Networks”, 2015, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://arxiv.org/abs/1511.06085

[23] A. Krizhevsky, „The CIFAR-10 dataset”, 2009, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://www.cs.toronto.edu/~kriz/cifar.html

[24] „MNIST database”, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://www.mnist.org/

[25] A. Coates, „STL-10 dataset”, Dostęp: 14 sierpień 2025. [Online]. Dostępne na: https://www-cs-faculty.stanford.edu/~acoates/stl10/

[26] E. B. Y.-C. L. Mohammadamin Moradi Shirin Panahi, „Kolmogorov-Arnold Network Autoencoders”, 2024, Dostęp: 2 wrzesień 2025. [Online]. Dostępne na: https://arxiv.org/pdf/2410.02077

Spis obrazów

Spis tabel

Summary

This work explores whether Kolmogorov-Arnold Networks (KAN), neural architecture proposed in 2024 that replaces scalar weights with learned one-dimensional functions can serve as an effective backbone for image compression via autoenkoders, and how they compare to a conventional convolutional autoenkoder (CNN). After outlining the essentials of image compression and the evaluation metrics used (MSE, PSNR, SSIM), the thesis focuses on end-to--end reconstruction quality at fixed latent sizes as a proxy for bitrate. Unlike full neural codecs, explicit quantization and entropy modeling are not included, so results isolate how well different enkoders learn compact, reconstructive representations.

Two families of models were implemented in PyTorch: (I) shallow KAN autoenkoders whose enkoder and decoder are single KAN layers that sum learned per-edge functions $\varphi(x)$ parameterized on a fixed grid (variants: linear, Bézier, Catmull-Rom, B-spline), and (II) a baseline CNN autoenkoder tuned to match the same bottleneck dimensionality K. An additional accelerated variant, FastKAN, substitutes Gaussian radial basis functions for splines to reduce inference cost. All experiments used grayscale inputs and identical training conditions: Adam (1e-3), batch size 64, 10 epochs, with model selection by lowest validation loss on a 20/80 train/validation split. Datasets covered a spectrum of complexity and scale: MNIST (28×28 digits), CIFAR-10 downscaled to 16×16 (small natural images), a synthetic Shapes set (32×32 geometric figures with sharp edges), and a "Large" set (64×64 grayscale from STL-10). Each model was trained independently at $K \in \{16, 32, 64, 128, 256\}$ to probe extreme through moderate compression.

Results show that KAN-based models, especially FastKAN, match or surpass CNNs once the bottleneck is moderately large (typically $K \geq 64 - 128$), whereas CNNs tend to hold a slight advantage at the most severe compression levels ($K \in \{16, 32\}$), where preserving global structure dominates. On MNIST, CNN leads at very small K, but FastKAN becomes best as capacity grows, reaching SSIM 0.944 and PSNR 28.05 dB with lower MSE. On tiny natural images (CIFAR-10 at 16×16), KAN with linear splines is best across all K, up to SSIM 0.965

and PSNR 29.26 dB, suggesting that learned per-edge transforms exploit small-image statistics effectively. On Shapes, FastKAN dominates at every K, improving monotonically with capacity and attaining SSIM 0.649 at $K = 256$, reflecting superior edge preservation compared with CNN's tendency to blur contours. For the Large set (64×64), CNN is marginally better at $K \in \{16, 32, 64, 128\}$, but FastKAN overtakes at $K = 256$ (SSIM 0.279, PSNR 17.49 dB), indicating that KAN's flexibility becomes beneficial given sufficient latent bandwidth.

Efficiency measurements highlight FastKAN's practicality: it achieves the lowest inference latencies (e.g., $\sim 0.016 - 0.031$ ms per MNIST image and $\sim 0.014 - 0.109$ ms on Shapes), and on the Large set at $K = 256$ it is about an order of magnitude faster than CNN (~ 0.137 ms vs. ~ 2.6 ms per image), despite the KAN family's higher per-connection expressivity. Training, however, is heaviest for the custom spline-based KAN (e.g., $\sim 3 - 4$ h for 10 epochs on CIFAR-10) versus ~ 45 min for CNN and FastKAN under identical settings. Counting per-setting wins across datasets and K, FastKAN leads (9/20), followed by CNN (6/20) and KAN-linear (5/20). More complex spline variants (Bézier, Catmull-Rom, B-spline) did not outperform KAN-linear or FastKAN within this training budget, likely needing stronger regularization or longer training to realize their potential.

In sum, KAN-based autoenkoders are a credible, often superior alternative to CNNs for learned image compression when the code is not extremely constrained. FastKAN, in particular, offers a compelling quality-latency trade-off for deployment, while KAN-linear shines on very small natural images. Limitations include shallow architectures, short training runs, and the absence of a rate model, so conclusions pertain to reconstruction at fixed latent sizes rather than true rate-distortion. Future work should explore deeper or hybrid CNN-KAN designs (and transformers at higher resolutions), curvature/growth regularization for spline stability, pruning and quantization for compact models, and full rate-distortion training with learned entropy models to turn KAN autoenkoders into end-to-end neural codecs.

Dodatek A

Wyniki (bottleneck = 256)

Nazwa zbioru	Rozdzielczość	Model	l. parametrów	MSE	PSNR	SSIM	Czas (ms)
mnist	28x28	Cnn	934081	4.7283	22.9721	0.9339	0.4443
mnist	28x28	Kan linear	2408448	1.5829	27.596	0.9256	1.089
mnist	28x28	Kan bezier	2408448	4.6903	22.9067	0.8739	5.241
mnist	28x28	Kan catmull	2408448	2.1863	26.2275	0.916	4.4366
mnist	28x28	Kan bspline	2408448	4.5734	23.0629	0.8763	4.1488
mnist	28x28	fastkan	3615808	1.606	28.051	0.943	0.0308
cifar16	16x16	Cnn	563137	3.1332	19.8543	0.5865	0.1322
cifar16	16x16	Kan linear	786432	0.3496	29.2584	0.9653	0.3346
cifar16	16x16	Kan bezier	786432	1.3797	23.3226	0.8509	1.0933
cifar16	16x16	Kan catmull	786432	0.7274	26.0463	0.9312	1.2598
cifar16	16x16	Kan bspline	786432	1.3666	23.3726	0.8498	1.3722
cifar16	16x16	fastkan	1181200	0.4364	28.2509	0.9541	0.0125
large	64x64	Cnn	2582017	104.5833	16.5244	0.2606	2.686
large	64x64	Kan linear	12582912	84.3818	17.3995	0.2754	9.2223
large	64x64	Kan bezier	12582912	95.5368	16.8378	0.249	22.1951
large	64x64	Kan catmull	12582912	96.6786	16.7788	0.2507	29.8169
large	64x64	Kan bspline	12582912	97.3424	16.7802	0.2437	29.8897
large	64x64	fastkan	18887440	80.1732	17.4895	0.2789	0.1374
shapes	32x32	Cnn	1180801	100.2871	11.4341	0.0963	0.6561
shapes	32x32	Kan linear	3145728	26.3497	16.0314	0.3851	1.4309
shapes	32x32	Kan bezier	3145728	43.8568	14.1722	0.2184	5.5401
shapes	32x32	Kan catmull	3145728	32.616	15.2425	0.289	5.8429
shapes	32x32	Kan bspline	3145728	43.409	14.1969	0.2058	6.6643
shapes	32x32	fastkan	4722448	21.1224	17.2391	0.649	0.109

Wyniki (bottleneck = 128)

Nazwa zbioru	Rozdzielczość	Model	l. parametrów	MSE	PSNR	SSIM	Czas (ms)
mnist	28x28	Cnn	524353	4.6914	23.0984	0.9351	0.4455
mnist	28x28	Kan linear	1204224	2.4249	25.8344	0.9051	0.6231
mnist	28x28	Kan bezier	1204224	6.5101	21.487	0.8415	1.7007
mnist	28x28	Kan catmull	1204224	3.1657	24.6435	0.8883	2.4142
mnist	28x28	Kan bspline	1204224	5.9138	21.9244	0.8508	2.2982
mnist	28x28	fastkan	1809088	1.7049	27.3376	0.9441	0.0158
cifar16	16x16	Cnn	296769	3.0126	20.0347	0.6033	0.1243
cifar16	16x16	Kan linear	393216	0.5314	27.4981	0.9474	0.2017
cifar16	16x16	Kan bezier	393216	1.6438	22.6126	0.8189	0.6276
cifar16	16x16	Kan catmull	393216	0.9479	24.9512	0.9075	0.7611
cifar16	16x16	Kan bspline	393216	1.6048	22.7094	0.8227	0.6843
cifar16	16x16	fastkan	590992	0.573	27.0329	0.9405	0.0139
large	64x64	Cnn	1516929	99.2994	16.728	0.2742	2.6126
large	64x64	Kan linear	6291456	94.2206	16.9217	0.2585	3.9243
large	64x64	Kan bezier	6291456	102.8465	16.5601	0.2399	10.4101
large	64x64	Kan catmull	6291456	104.939	16.4744	0.2385	13.1178
large	64x64	Kan bspline	6291456	105.6544	16.4442	0.2333	12.1936
large	64x64	fastkan	9449872	85.0665	17.2584	0.2636	0.1063
shapes	32x32	Cnn	648193	100.6176	11.4136	0.0891	0.6184
shapes	32x32	Kan linear	1572864	32.4689	15.2293	0.2565	0.7549
shapes	32x32	Kan bezier	1572864	54.0369	13.3789	0.1355	2.7457
shapes	32x32	Kan catmull	1572864	38.6112	14.5883	0.1929	2.7584
shapes	32x32	Kan bspline	1572864	52.5316	13.4608	0.1375	2.8855
shapes	32x32	fastkan	2362768	23.3089	16.6445	0.581	0.0382

Wyniki (bottleneck = 64)

Nazwa zbioru	Rozdzielczość	Model	l. parametrów	MSE	PSNR	SSIM	Czas (ms)
mnist	28x28	Cnn	319489	5.4944	22.2683	0.9231	0.4249
mnist	28x28	Kan linear	602112	4.2041	23.4943	0.8755	0.27
mnist	28x28	Kan bezier	602112	12.8138	18.478	0.7302	0.8353
mnist	28x28	Kan catmull	602112	5.0315	22.6387	0.8479	0.9465
mnist	28x28	Kan bspline	602112	9.1256	19.995	0.7838	1.0574
mnist	28x28	fastkan	905728	3.2175	24.6805	0.9293	0.0168
cifar16	16x16	Cnn	163585	3.2328	19.7406	0.5828	0.1203
cifar16	16x16	Kan linear	196608	0.9101	25.2013	0.9049	0.1072
cifar16	16x16	Kan bezier	196608	2.0036	21.7834	0.7733	0.3126
cifar16	16x16	Kan catmull	196608	1.3113	23.596	0.8674	0.3097
cifar16	16x16	Kan bspline	196608	1.9334	21.9305	0.7796	0.3219
cifar16	16x16	fastkan	295888	1.013	24.5424	0.8945	0.0061
large	64x64	Cnn	984385	107.7177	16.3439	0.2568	2.6509
large	64x64	Kan linear	3145728	106.0055	16.4592	0.2383	1.4112
large	64x64	Kan bezier	3145728	110.0561	16.2826	0.2301	5.943
large	64x64	Kan catmull	3145728	111.7964	16.2262	0.2325	5.4791
large	64x64	Kan bspline	3145728	113.9759	16.14	0.2288	5.3074
large	64x64	fastkan	4731088	91.6402	16.9622	0.2493	0.0709
shapes	32x32	Cnn	381889	103.8265	11.0254	0.0776	0.6626
shapes	32x32	Kan linear	786432	43.5794	14.0759	0.1596	0.4039
shapes	32x32	Kan bezier	786432	74.2494	12.0056	0.0934	1.2803
shapes	32x32	Kan catmull	786432	50.0274	13.5765	0.1381	1.2943
shapes	32x32	Kan bspline	786432	70.2703	12.1349	0.0985	1.4134
shapes	32x32	fastkan	1182928	28.6465	15.8047	0.4814	0.0211

Wyniki (bottleneck = 32)

Nazwa zbioru	Rozdzielczość	Model	l. parametrów	MSE	PSNR	SSIM	Czas (ms)
mnist	28x28	Cnn	217057	5.8139	22.158	0.9208	0.4299
mnist	28x28	Kan linear	301056	8.3915	20.4241	0.7904	0.1568
mnist	28x28	Kan bezier	301056	36.6029	13.7077	0.4118	0.4365
mnist	28x28	Kan catmull	301056	8.9637	20.0765	0.775	0.534
mnist	28x28	Kan bspline	301056	28.0655	14.9041	0.5083	0.4904
mnist	28x28	fastkan	454048	7.1322	21.2322	0.8787	0.0155
cifar16	16x16	Cnn	96993	3.6866	19.1409	0.5302	0.1276
cifar16	16x16	Kan linear	98304	1.6254	22.7102	0.8227	0.1147
cifar16	16x16	Kan bezier	98304	2.5205	20.8223	0.7058	0.1721
cifar16	16x16	Kan catmull	98304	1.7806	22.3186	0.8095	0.1401
cifar16	16x16	Kan bspline	98304	2.3161	21.1817	0.7317	0.1438
cifar16	16x16	fastkan	148336	1.7648	22.2073	0.809	0.0052
large	64x64	Cnn	718113	100.5952	16.6742	0.2709	2.6351
large	64x64	Kan linear	1572864	132.4314	15.5264	0.2113	0.7232
large	64x64	Kan bezier	1572864	117.9022	16.0086	0.2231	3.1341
large	64x64	Kan catmull	1572864	126.4376	15.713	0.2181	2.9117
large	64x64	Kan bspline	1572864	130.8031	15.5762	0.2151	2.886
large	64x64	fastkan	2371696	102.6388	16.5058	0.2308	0.0459
shapes	32x32	Cnn	248737	99.5022	11.4659	0.0835	0.6496
shapes	32x32	Kan linear	393216	62.8713	12.5336	0.1069	0.2063
shapes	32x32	Kan bezier	393216	98.2948	10.7331	0.0574	0.605
shapes	32x32	Kan catmull	393216	72.951	12.0018	0.0915	0.6407
shapes	32x32	Kan bspline	393216	94.9067	10.8295	0.0606	0.6405
shapes	32x32	fastkan	593008	45.4278	14.0272	0.1654	0.0184

Wyniki (bottleneck = 16)

Nazwa zbioru	Rozdzielczość	Model	l. parametrów	MSE	PSNR	SSIM	Czas (ms)
mnist	28x28	Cnn	165841	8.539	20.5086	0.8852	0.4304
mnist	28x28	Kan linear	150528	17.9689	16.9531	0.6617	0.0861
mnist	28x28	Kan bezier	150528	52.5994	11.8941	0.174	0.2371
mnist	28x28	Kan catmull	150528	17.9168	16.9407	0.6425	0.283
mnist	28x28	Kan bspline	150528	41.5222	13.1689	0.3319	0.2316
mnist	28x28	fastkan	228208	14.2298	18.1935	0.7696	0.0157
cifar16	16x16	Cnn	63697	3.6422	19.2561	0.5407	0.1215
cifar16	16x16	Kan linear	49152	2.6822	20.5737	0.6887	0.0283
cifar16	16x16	Kan bezier	49152	3.0854	19.9719	0.6262	0.0813
cifar16	16x16	Kan catmull	49152	2.7673	20.4181	0.6817	0.0667
cifar16	16x16	Kan bspline	49152	2.867	20.2862	0.6633	0.0647
cifar16	16x16	fastkan	74560	2.9404	20.106	0.6597	0.0048
large	64x64	Cnn	584977	112.5016	16.2021	0.2476	2.6375
large	64x64	Kan linear	786432	149.5824	15.0047	0.2042	0.352
large	64x64	Kan bezier	786432	137.6191	15.3557	0.21	1.368
large	64x64	Kan catmull	786432	137.8105	15.3524	0.21	1.3185
large	64x64	Kan bspline	786432	145.792	15.125	0.2064	1.399
large	64x64	fastkan	1192000	116.2522	16.0058	0.2155	0.0511
shapes	32x32	Cnn	182161	99.1781	11.4833	0.0824	0.6299
shapes	32x32	Kan linear	196608	97.2086	10.5298	0.0624	0.1036
shapes	32x32	Kan bezier	196608	116.6507	9.8781	0.0367	0.3981
shapes	32x32	Kan catmull	196608	101.0551	10.4286	0.0555	0.3519
shapes	32x32	Kan bspline	196608	116.4147	9.871	0.0373	0.3488
shapes	32x32	fastkan	298048	74.9681	11.8792	0.0882	0.0141