SVEUČILIŠTE U ZAGREBU

GRAFIČKI FAKULTET

Marko Butula

Hibridni pristupi generativne suparničke super-rezolucije: arhitektonska načela, optimizacija i primjena Real-ESRGAN dubokih neuronskih mreža u digitalnoj vizualizaciji

Zagreb, 2025

Ovaj rad izrađen je na Grafičkom fakultetu na Katedri za tiskarske procese pod vodstvom izv. prof. dr. sc. Diane Bratić i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2024./2025.

POPIS I OBJAŠNJENJE KRATICA KORIŠTENIH U RADU

- AI - Artificial Intelligence (umjetna inteligencija)
- AV1 - AOMedia Video 1 (video kodek)
- BGR - Blue Green Red (redoslijed kanala u slici)
- BGRA - Blue Green Red Alpha (redoslijed kanala u slici s alpha kanalom)
- CNN - Convolutional Neural Network (konvolucijska neuronska mreža)
- CPU - Central Processing Unit (središnja procesorska jedinica)
- DIV2K - Diverse 2K resolution dataset (dataset za trening super-rezolucijskih modela)
- ESRGAN - Enhanced Super-Resolution Generative Adversarial Network (poboljšana generativna suparnička mreža za super-rezoluciju)
- FFHQ - Flickr-Faces-HQ Dataset (dataset visokokvalitetnih lica)
- GAN - Generative Adversarial Network (generativna suparnička mreža)
- GFPGAN - Generative Facial Prior GAN (prethodni generativni facijalni GAN)
- GPU - Graphics Processing Unit (grafička procesorska jedinica)
- HR - High Resolution (visoka rezolucija)
- JPEG - Joint Photographic Experts Group (format kompresije slike)
- LPIPS - Learned Perceptual Image Patch Similarity (metrika mjerenja kvalitete slike)
- LR - Low Resolution (niska rezolucija)
- NIQE - Natural Image Quality Evaluator (metrika mjerenja kvalitete slike)
- ONNX - Open Neural Network Exchange (format modela za prijenos i izvođenje)
- PCV - Pixel Convolutional Video (video obrađen konvolucijskim mrežama na razini piksela)
- PReLU - Parametric Rectified Linear Unit (aktivacijska funkcija neuronske mreže)
- PSNR - Peak Signal-to-Noise Ratio (metrika mjerenja kvalitete slike)
- PyTorch - Python biblioteka za duboko učenje
- ReLU - Rectified Linear Unit (aktivacijska funkcija)
- RRDB - Residual-in-Residual Dense Block (tip sloja mreže u Real-ESRGAN-u)
- SR - Super-Resolution (super-rezolucija)
- SSIM - Structural Similarity Index (metrika mjerenja kvalitete slike)
- URL - Uniform Resource Locator (jedinstveni lokator resursa)
- VGG - Visual Geometry Group (arhitektura neuronske mreže)
- VRAM - Video Random Access Memory (memorija GPU-a)

SADRŽAJ

1. UVOD

Doprinos umjetne inteligencije tijekom posljednjeg desetljeća uvelike je oblikovao područje računalnog vida i digitalne obrade slike i videa, dovodeći do značajnih promjena u mogućnostima rekonstrukcije, poboljšanja i automatizacije vizualnih sadržaja. Primjena naprednih dubokih konvolucijskih i generativnih neuronskih mreža omogućila je metode nadogradnje rezolucije, uklanjanja šuma, restauracije povijesnih materijala te sinteze vizualnih podataka koje daleko nadmašuju tradicionalne pristupe temeljem interpolacije i klasičnih filtracijskih algoritama.

U kontekstu super-rezolucija postupaka umjetna inteligencija ne samo da povećava dimenzije slike nego i generira vjerodostojne visokofrekvencijske detalje, čime se postiže znatno viša razina perceptivne kvalitete. Ove sposobnosti posebno dolaze do izražaja kod degradiranih ili niskokvalitetnih vizualnih podataka, gdje su konvencionalne metode ograničene, a algoritmi temeljeni na dubokom učenju rekonstruiraju izgubljene informacije i generiraju uvjerljive vizualne prikaze koji unapređuju kvalitetu digitalnih sadržaja u domenama od multimedije i medicine do sigurnosti i umjetnosti.

Real-ESRGAN, razvijen od strane Xinntaoa i suradnika, predstavlja jedan od najnaprednijih algoritama za nadogradnju slika (super-rezolucija) koji koristi generativne suparničke mreže (eng. *GAN* ili *Generative Adversarial Network*) [1]. Ovaj model nastavlja na tradiciju ESRGAN-a (eng. *Enhanced Super-Resolution Generative Adversarial Network* tj. poboljšana generativna suparnička mreža super-rezolucije), ali unosi značajna poboljšanja u realističnu rekonstrukciju slika, posebno u slučajevima s visokim stupnjem degradacije. Real-ESRGAN je posebno dizajniran za rješavanje problema stvarnih slika, koje često sadrže šum, artefakte kompresije (npr. JPEG) i druge nesavršenosti koje otežavaju tradicionalne metode nadogradnje [2, 3].

Područje istraživanja obrade slike unutar računalnog vida posljednjih je desetljeća obilježeno intenzivnim razvojem metoda za nadogradnju slike, koje imaju ključnu ulogu u medicinskoj dijagnostici, restauraciji vizualne građe, nadzornim sustavima i digitalnoj multimediji.

U kontekstu digitalne obrade slika, nadogradnja slike (super-rezolucija) ima ključnu ulogu u brojnim primjenama, poput medicinske image analize, nadzornih sustava, restauracije starih fotografija i poboljšanja kvalitete video sadržaja. Tradicionalne metode, poput interpolacije (npr. bilinearna ili bikubična), često daju mutne rezultate bez dovoljno detalja. S druge strane, dubinski modeli, posebno oni temeljeni na GAN-ovima, omogućuju generiranje oštrijih i

realističnijih slika. ESRGAN je već postavio nove standarde u ovoj oblasti, ali Real-ESRGAN dalje unapređuje tu tehnologiju fokusirajući se na realistične scenarije degradacije [4, 5,].
Problem koji se pojavljuje u praksi jest da većina modela super-rezolucije, uključujući ESRGAN, pokazuje smanjenu učinkovitost pri radu sa stvarnim slikama koje sadrže kompleksne degradacije: šum, zamućenje, kompresijske artefakte i nelinearne distorzije. Takve degradacije često nisu dovoljno dobro modelirane u fazi treniranja, zbog čega rezultati na realnim podacima odstupaju od onih postignutih na sintetičkim skupovima [6]. Stoga ključni doprinos Real-ESRGAN-a leži u njegovoj sposobnosti da generalizira na različite vrste degradacija, uključujući one koje nisu eksplicitno modelirane u skupu podataka za obuku [7].

Za razliku od prethodnih metoda koje su se oslanjale na sintetičke podatke s jednostavnim modelima degradacije (npr. Gaussov šum ili blur), Real-ESRGAN koristi složenije procese degradacije koji bolje oponašaju stvarne uvjete. Ovo uključuje kombinaciju bluriranja, šuma, kompresijskih artefakata i periodičkih šumova, što omogućuje modelu da se nosi s raznolikim izazovima u stvarnim slikama. Arhitektura Real-ESRGAN-a temelji se na ESRGAN-u, ali uvodi nekoliko ključnih modifikacija. Prvo, koristi veći i dublji generator s residualnim blokovima i poboljšanim mehanizmima za prijenos značajki. Drugo, uvodi novi diskriminator (U-Net diskriminator) koji bolje procjenjuje lokalne i globalne karakteristike slike. Treće, poboljšani su gubitci (loss functions) kako bi se postigla bolja teksturna detaljnost i manje artefakata [8]. Posebno je važan tzv. perceptual loss ili perceptivni gubitak, koji se oslanja na VGG (eng. *Visual Geometry Group*) mrežu za očuvanje visokofrekventnih detalja [9, 10].

Real-ESRGAN je evaluiran na različitim standardnim skupovima podataka (npr. RealSR, DIV2K) i pokazao je superiorne rezultate u usporedbi s drugim state-of-the-art metodama. Osim toga, model je praktično primjenjiv zahvaljujući svojoj sposobnosti da radi s različitim razinama degradacije bez potrebe za prilagodbom. Ovo ga čini posebno korisnim u scenarijima gdje su ulazne slike nepoznate kvalitete, kao što su arhivske fotografije ili snimke s lošim uvjetima osvjetljenja [11].

Cilj ovoga rada je detaljno analizirati arhitektonska rješenja Real-ESRGAN modela s posebnim naglaskom na U-Net diskriminator sa spektralnom normalizacijom i RRDB (eng. *Residual-in-Residual Dense Block*) generator, njegov proces treniranja i evaluacije, te pokazati njegove prednosti i ograničenja u odnosu na prethodne metode. Specifični ciljevi uključuju: (1) analizu implementacijskih detalja ključnih komponenti, (2) evaluaciju performansi modela na različitim tipovima slika, i (3) identifikaciju ograničenja i potencijalnih poboljšanja. Ispunjenjem ovih ciljeva, rad će osim znanstvene dimenzije imati i značajnu aplikativnu

vrijednost, doprinoseći praktičnoj primjeni Real-ESRGAN modela u područjima digitalne restauracije, video nadogradnje i računalnog vida, čime se omogućuje unaprjeđenje kvalitete vizualnih sadržaja u realnim uvjetima.

Radna hipoteza polazi od pretpostavke da Real-ESRGAN može ostvariti višu razinu vizualne vjernosti i perceptivne kvalitete u uvjetima realnih degradacija u odnosu na tradicionalne i ranije GAN pristupe odnosno da kombinacija naprednih arhitektonskih inovacija (U-Net diskriminator, spektralna normalizacija, složeni modeli degradacije) u Real-ESRGAN-u omogućuje superiornu kvalitetu super-rezolucije u usporedbi s tradicionalnim metodama, posebno na stvarnim slikama s kompleksnim degradacijama.
U tu svrhu će se detaljno analizirati arhitektura Real-ESRGAN-a, njegove glavne inovacije u odnosu na prethodne modele, te metodologija obuke i evaluacije, a također će se objasniti njegove prednosti i ograničenja, kao i potencijalne smjerove daljnjeg razvoja. Konačno, u praktičnom dijelu rada istaknut će se praktične primjene Real-ESRGAN-a u područjima digitalne restauracije, video nadogradnje i računalnog vida.

Metodologija istraživanja uključuje analizu implementacije ključnih komponenti kroz Python kod, usporedbu rezultata na test slikama, te evaluaciju ograničenja modela. Rad je organiziran u osam poglavlja koja pokrivaju teorijske osnove, arhitektonsku analizu, implementacijske detalje, praktičnu primjenu i evaluaciju rezultata.

2. Real-ESRGAN: ARHITEKTURA I FUNKCIONALNOST

Real-ESRGAN (eng. *Real-Enhanced Super-Resolution Generative Adversarial Network*) je iznimno važan napredak u području računalnog vida i obrade slike, posebno u zadatku super-rezolucije, odnosno povećanja kvalitete i detalja digitalnih fotografija i videa. Ovaj model dubokog učenja razvijen je u okviru generativnih suparničkih mreža (eng. *GAN* ili *Generative Adversarial Network*), s ciljem da poboljša rezoluciju slika i nadmaši dotadašnje najpoznatije pristupe, poput ESRGAN-a, s posebnim naglaskom na rad sa stvarnim, često snažno degradiranim slikama koji predstavljaju poseban izazov zbog nepredvidivih oblika oštećenja i gubitka podataka [2, 4, 9].

U središtu koncepta super-rezolucije nalazi se potreba da se od niskorezolucijskih slika rekonstruiraju njihove visoko-rezolucijske varijante, uz minimalno gubljenje informacija, odnosno da finalna slika što vjernije odražava originalan sadržaj. Klasična super-rezolucija u obradi slike tradicionalno se borila s izazovima poput zamućenja, šuma, kompresijskih artefakata, gubitka kontrasta i detalja uslijed digitalnog povećanja. Raniji modeli, pa tako i ESRGAN, trenirani su uglavnom na sintetički degradiranim slikama uz dobro poznate metode poput bicubic downsamplinga, što ih je činilo manje otpornima na raznolikost stvarnih oštećenja koja se pojavljuju u praksi [5, 7].

Stvarne slike, bilo iz mobilnih uređaja, starih skenera, arhivskih filmova ili video nadzora, često su podložne kompleksnim, višestrukim degradacijama. Upravo zato je motivacija iza stvaranja Real-ESRGAN-a bila omogućiti univerzalnost i korak dalje prema pouzdanoj primjeni u stvarnim situacijama, gdje sintetičko simuliranje oštećenja nije dovoljno za modeliranje realnog svijeta [3]. Real-ESRGAN je s te strane donio revoluciju pristupom te poboljšao mogućnosti restauracije, povećanja rezolucije i povrata detalja na slikama koje dolaze iz svakodnevnog života, povijesnih arhiva, medicinske dijagnostike, sigurnosnih sustava i mnogih drugih aplikacija. Tehnička platforma Real-ESRGAN-a nadograđuje temeljnu arhitekturu ESRGAN-a dodajući nekoliko ključnih inovacija i dorada [9, 10]. Srce modela je generator izgrađen na RRDB principu (eng. *RRDB* ili *Residual-in-Residual Dense Blocks*). Ovi blokovi, koji predstavljaju evoluciju klasičnih konvolucijskih neuronskih mreža (eng. *CNN* ili *Convolutional Neural Network*), omogućuju dubinsko izdvajanje značajki iz slike, čime model bolje razumije sadržaj slike, teksture i minuciozne detalje. Zahvaljujući RRDB-u, Real-ESRGAN uspješno rekonstruira bogate teksture i fine elemente, čime konačna slika izgleda prirodnije i kvalitetnije [5, 9, 12, 13, 14].

Na strani diskriminatora, za razliku od standardnih rješenja, Real-ESRGAN primjenjuje U-Net diskriminator sa spektralnom normalizacijom. U-Net arhitektura omogućava višeslojnu procjenu slike i ne samo da razlikuje generirane od autentičnih slika, već pomaže i stabilizaciji procesa treniranja i daje modelu mogućnost procjene kvalitete na razini lokalnog i globalnog konteksta slike. Spektralna normalizacija dodatno doprinosi stabilnosti treniranja modela, smanjuje mogućnost stvaranja artefakata i povećava postojanost rezultata [14]. Proces treniranja Real-ESRGAN-a odvija se na velikim skupovima podatka, poput DIV2K, Flickr2K i OutdoorSceneTraining koji pokrivaju raznolike vrste slika. Model najprije uči na sintetički degradiranim slikama, što mu daje početnu sposobnost prepoznavanja i restauracije slike, dok se u drugoj fazi trenira na stvarnim niskokvalitetnim slikama uz "weak supervision" pristup [15]. Ova dvostupanjska strategija omogućuje bolju generalizaciju modela i podiže njegovu sposobnost restauracije kompleksno oštećenih stvarnih slika [1].

Jedna od ključnih inovacija u Real-ESRGAN-u je modeliranje degradacije [16]. Za razliku od bicubic degradacije, koja je vrlo jednostavna i ne odražava realne uvjete, Real-ESRGAN primjenjuje višestupanjske, stohastičke modele degradacije. Model tijekom treninga simulira raznovrsne procese: zamućenja, šuma, kompresijske artefakte, promjene veličine i druge realne uzroke gubitka kvalitete vizuala. U tom kontekstu koristi i sintetičke filtere, poput sinc filtra, čime još preciznije simulira proces degradacije kroz koji slike prolaze u stvarnosti [3, 17]. Takvo modeliranje omogućuje Real-ESRGAN-u da se uspješno nosi s raznim vrstama oštećenja i u stvarnim uvjetima generira slike izvanredno visoke kvalitete, s bogatijim detaljima i teksturama. Na algoritamskoj razini, Real-ESRGAN kombinira više različitih funkcija gubitka kako bi postigao konačne rezultate. L1 ili L2 loss (gubitak) mjeri pikselnu razliku između izvorne i restaurirane slike, perceptual loss koristi predtreniranu VGG mrežu za procjenu sličnosti na razini značajki i vodi model prema obnovi tekstura koje ljudsko oko percipira kao prirodne, dok GAN loss procjenjuje sposobnost diskriminatora da razlikuje restaurirane slike od stvarnih. Generator tako konstantno pokušava "prevariti" diskriminator, stvarajući što realističnije slike, a rezultat tog procesa su slike iznimne vizualne vjernosti [2, 9, 14, 18].

Real-ESRGAN je izuzetno fleksibilan u funkcionalnosti i uporabljivosti. Podržava povećanje rezolucije za razne faktore skaliranja, najčešće × 2 ili × 4, ali nudi mogućnost i za bilo koju vrijednost skaliranja, ovisno o potrebama korisnika. Osim statičnih slika, uz dodatne modifikacije podržava i video – može povećavati rezoluciju svakog framea pojedinačno, ali i održavati konzistenciju između frameova putem temporal consistency modula. Posebno je

važna podrška za poboljšanje lica zahvaljujući integraciji s GFPGAN (eng. *Generative Facial Prior GAN*) modelom, što ga čini idealnim za restauraciju portreta, obiteljskih fotografija i slika s ljudskim subjektima. Model podržava slike različitih ekstenzija, alpha kanale, 16-bitnu dubinu, kao i crno-bijele slike, čime dodatno proširuje područje primjene [3, 10].

Primjene Real-ESRGAN-a su iznimno raznolike i od velikog društvenog značaja. U digitalnoj fotografiji koristi se za restauraciju starih ili loših slika, povratak detalja na izblijedjelim vizualima i jasnije prikazivanje finih tekstura. U industriji video igara postao je neizostavan alat pri remasteriranju klasičnih naslova, gdje skenira i dovodi originalne spriteove i teksture na novu rezolucijsku razinu bez gubitka kvalitete. Restauracija umjetničkih djela, digitalizacija povijesnih dokumenata i slika, kao i znanstvene analize u medicini, posebno radiologiji, predstavljaju klasične primjere gdje Real-ESRGAN omogućuje poboljšanje slika i lakšu detekciju kritičnih detalja. U forenzici i nadzoru često poboljšava snimke s kamera niske kvalitete, doprinosi prepoznavanju osoba i objekata, te pomaže pri analizi pojedinosti koje bi inače bile neprepoznatljive [2, 3, 5].

Usporedba s prethodnim modelima otkriva koliko je Real-ESRGAN unaprijedio područje. ESRGAN je bio značajna prekretnica u teksturalnoj restauraciji, ali je limitiran segmentom sintetičkih degradacija. Real-ESRGAN uvodi robusniji trening, napredne algoritme za simulaciju realnih degradacija, jaču ocjenu kvalitete slike putem U-Net diskriminatora, adaptira se na različite uvjete i nudi poboljšanje širokog spektra stvarnih slika [19]. U svim testiranjima nadmašuje ESRGAN po sposobnosti restauracije tekstura te smanjenju artefakata kao što su ringing, overshoot, šum i zamućenje. Evaluacija i kvaliteta modela provodi se kombinacijom subjektivne vizualne usporedbe slika te korištenjem metrika poput NIQE (eng. *Natural Image Quality Evaluator*). Unatoč tome što kvantitativne metrike nisu uvijek savršene u korelaciji s ljudskom percepcijom, Real-ESRGAN se pokazao superiornim na čitavom nizu testnih skupova podataka, posebno na slikama s nepoznatim ili kompleksnim degradacijama [7, 9].

Po pitanju jednostavnosti korištenja, Real-ESRGAN je distribuiran kao open-source projekt putem GitHub repozitorija, uz jasne upute za implementaciju i korištenje. Model je moguće pokrenuti na desktopu, u Google Colab okruženju, ili inkorporirati unutar vlastitih Python projekata [14]. Za rad je potrebno instalirati pakete poput PyTorcha, a svi modeli dostupni su bez licence, podržavaju Windows, Linux i macOS operativne sustave [20, 21, 22]. Automatizirani batch procesi podržavaju obradu velikih količina podataka, što omogućava primjenu u profesionalnim i high-throughput okruženjima. Tipičan workflow pri korištenju

Real-ESRGAN modela uključuje sljedeće korake: učitavanje slike ili video zapisa, konfiguraciju parametara poput faktora skaliranja i aktivacije modula za poboljšanje lica, zatim obradu i generiranje slike te završnu vizualnu inspekciju rezultata. Korisnici mogu koristiti naredbeni redak (eng. *CLI* ili *Command Line Interface*) ili grafičko korisničko sučelje (eng. *GUI* ili *Graphical User Interface*) za jednostavnije upravljanje, a batch obrada i podrška za različite formate olakšavaju korištenje u profesionalnim, ali i amaterskim uvjetima [2, 3, 5, 10].

Unatoč izvanrednim sposobnostima, Real-ESRGAN ima i svoja ograničenja. Vrlo oštećene ili ekstremno degradirane slike, kod kojih je originalna informacija nepovratno izgubljena, predstavljaju izazov koji ni ovaj model ne može u potpunosti riješiti. Kod slika sa specifičnim artefaktima, aliasingom ili velikim gubitkom podataka, model može "halucinirati" detalje, te se stoga preporučuje oprez kod profesionalnih aplikacija poput medicine ili forenzike [3].

3. U-NET DISKRIMINATOR SA SPEKTRALNOM NORMALIZACIJOM

U prethodnom poglavlju prikazana je teorijska uloga U-Net diskriminatora i spektralne normalizacije u okviru Real-ESRGAN arhitekture. Tamo je naglasak bio na konceptualnim značajkama i razlozima njihove primjene, dok se u ovom poglavlju daje detaljan pregled praktične implementacije modela kroz programski kod. Analizirat će se originalna implementacija U-Net diskriminatora iz Real-ESRGAN repozitorija [8, 9] dostupnog na GitHub platformi, s fokusom na ključne komponente datoteke *discriminator.py* kako bi se razjasnila praktična implementacija teorijskih koncepata opisanih ranije. Prikazuje se način na koji je arhitektura definirana u Python kodu, uz objašnjenja pojedinih funkcionalnih blokova i njihovog doprinosa cjelokupnom procesu diskriminacije stvarnih i generiranih slika.

U-Net diskriminator sa spektralnom normalizacijom u Real-ESRGAN arhitekturi koristi se za razlikovanje stvarnih od generiranih slika te za evaluaciju njihove vjerodostojnosti na lokalnoj i globalnoj razini. Time se osigurava da treniranje modela ne rezultira samo vizualno privlačnim, već i strukturalno uvjerljivim rekonstrukcijama. Analiza koda omogućuje dublje razumijevanje specifičnih arhitektonskih odluka, kao što su dimenzije kernela, koraci konvolucije, strategije normalizacije i način transformacije ulaznog tensora kroz mrežu. Posebna pažnja posvećena je funkciji skip veza koje očuvaju prostorne informacije, kao i odabiru parametara koji optimiziraju performanse mreže.

U sljedećim sekcijama detaljno će se razmotriti svaki ključni blok U-Net diskriminatora: inicijalizacija parametara, encoder za smanjenje dimenzija, decoder za rekonstrukciju, implementacija skip veza te finalni klasifikacijski sloj koji proizvodi procjenu vjerodostojnosti slike. Takvim pristupom omogućuje se razumijevanje kako se teorijski koncepti spektralne normalizacije i encoder-decoder arhitekture pretvaraju u funkcionalne PyTorch module, te kako praktična implementacija doprinosi kvaliteti i stabilnosti Real-ESRGAN modela.

Eksperimentalni dio rada izveden je na Apple Mac Mini M4 računalnoj platformi koja je korištena za izradu programskog koda i praktičnih primjera. Sustav uključuje 10-jezgreni CPU (4 jezgre visokih performansi i 6 energetski učinkovitih jezgri), 10-jezgreni GPU s hardverski ubrzanim ray tracingom, 16-jezgreni Neural Engine te memorijsku propusnost od 120 GB/s. Radna memorija obuhvaća 16 GB unified RAM-a, dok je za pohranu podataka korišten SSD kapaciteta 512 GB. Uređaj posjeduje Media Engine s hardverskom podrškom za kodiranje i dekodiranje H.264, HEVC, ProRes i ProRes RAW formata, uključujući i AV1 dekodiranje. Operacijski sustav macOS Sequoia, u kombinaciji s napajanjem od 155 W, osigurava stabilan

rad, optimalnu podršku za napredne AI funkcije, paralelnu obradu i visokopropusnu komunikaciju memorije. Navedene računalne performanse omogućile su brzu obradu velikih skupova slika te uspješno pokretanje i testiranje naprednih deep learning modela.

Za provedbu eksperimentalnih implementacija korišten je programski jezik Python, uz skup specijaliziranih biblioteka koje su omogućile razvoj, obradu podataka i vizualizaciju rezultata:

- NumPy: temeljna biblioteka za numeričke operacije i rad s višedimenzionalnim poljima.
- PyTorch (torch): primarni okvir za implementaciju dubokih neuronskih mreža, treniranje i evaluaciju Real-ESRGAN modela.
- OpenCV (cv2): učitavanje, konverzija i osnovna obrada slika.
- Pillow (PIL): spremanje i manipulacija slikama u različitim formatima.
- Matplotlib: vizualizacija rezultata, usporedbe prije i poslije obrade te prikaz evaluacijskih metrika.
- Real-ESRGAN: izvorna implementacija i unaprijed trenirani modeli za povećanje rezolucije.
- TorchVision: priprema i evaluacija podatkovnih skupova.
- Tqdm: prikaz progress barova tijekom iterativnih postupaka i batch evaluacija.

Integracija ovih biblioteka omogućila je efikasnu implementaciju složenih neuronskih arhitektura, automatiziranu obradu i organizaciju eksperimentalnih podataka te standardiziranu vizualnu evaluaciju rezultata.

3.1. Tehnička arhitektura U-Net-a: dimenzije i transformacije

U-Net diskriminator u Real-ESRGAN implementaciji koristi simetričnu encoder-decoder arhitekturu s precizno definiranima prostornim transformacijama. Transformacije prostornih dimenzija kroz mrežu podliježu matematičkim pravilima konvolucijskih operacija. Dimenzije izlaza pojedinog konvolucijskog sloja izračunavaju se prema standardnoj formuli za 2D konvoluciju:

$$H_{out} = \left\lfloor \frac{H_{in}+2P-K}{S} \right\rfloor + 1 \quad (1)$$

$$W_{out} = \left\lfloor \frac{W_{in}+2P-K}{S} \right\rfloor + 1 \quad (2)$$

gdje su K = *kernel_size*, S = *stride*, P = *padding*, a $\lfloor \cdot \rfloor$ označava *floor* funkciju. H_{out} i W_{out} označavaju visinu i širinu izlaznog feature mapa za sloj. Ova formula omogućuje precizno

predviđanje dimenzija izlaza svakog konvolucijskog sloja u mreži, što je ključno za očuvanje prostorne koherencije i pravilno funkcioniranje skip veza u U-Net arhitekturi.

Za standardnu RGB sliku dimenzija $H \times W \times 3$ encoder faza postupno smanjuje prostorne dimenzije dok broj kanala feature mapa proporcionalno raste. Primjena konvolucijskih formula kroz arhitekturu daje sljedeće transformacije (Tablica 1):

Tablica 1. Transformacija dimenzija kroz encoder U-Net diskriminatora

Sloj	Ulazne dimenzije	Izlazne dimenzije	Opis
Conv0	(H, W, 3)	(H, W, 64)	Početna transformacija ulazne slike u feature prostor
Conv1	(H, W, 64)	(H/2, W/2, 128)	Prva redukcija prostorne dimenzije uz udvostručenje kanala
Conv2	(H/2, W/2, 128)	(H/4, W/4, 256)	Druga redukcija dimenzija i udvostručenje kanala
Conv3	(H/4, W/4, 256)	(H/8, W/8, 512)	Treća redukcija do bottleneck-a

Ova progresivna redukcija prostornih dimenzija omogućuje mreži ekstrakciju hijerarhijskih značajki, od lokalnih tekstura na višim rezolucijama do globalnih strukturnih informacija na nižim rezolucijama.

Decoder faza koristi interpolaciju ili konvolucijske transponirane slojeve za postupno vraćanje prostorne dimenzije, dok broj kanala proporcionalno opada. Kroz kombinaciju naučenih značajki i skip veza moguće je rekonstruirati detaljne prostorne informacije, zadržavajući lokalnu oštrinu i globalnu strukturu slike.

Ključna prednost ovakve arhitekture leži u sposobnosti simultane analize lokalnih i globalnih karakteristika, što je kritično za preciznu diskriminaciju između generiranih i autentičnih slika u GAN treniranju. Zbog toga U-Net diskriminator s implementiranom spektralnom normalizacijom u Real-ESRGAN modelu osigurava da generirane slike nisu samo vizualno uvjerljive, već i strukturalno konzistentne.

3.2. Inicijalizacija

U fazi inicijalizacije definira se osnovna struktura U-Net diskriminatora te parametri koji određuju način obrade ulaznih podataka. Ključni parametri uključuju broj ulaznih kanala (*num_in_ch*), broj početnih filtara (*num_feat*) i postojanje skip veza (*skip_connection*), koji zajedno određuju fleksibilnost modela i njegovu sposobnost prilagodbe različitim vrstama ulaznih slika.

Posebno važna komponenta je spektralna normalizacija (*spectral_norm*), primijenjena na konvolucijske slojeve radi stabilnijeg treniranja GAN arhitektura. Spektralna normalizacija

regularizira težine slojeva tako da njihova Lipschitz konstanta bude ograničena, što sprječava eksplozivne gradijente i omogućuje stabilniji trening. Matematički se računa najveća singularna vrijednost matrice težina W i dijeli sve težine tom vrijednošću:

$$W_{SN} = \frac{W}{\sigma(W)} \tag{3}$$

gdje je σ(W) najveća singularna vrijednost matrice W.

U Real-ESRGAN implementaciji, spektralna normalizacija primjenjuje se selektivno:

- Na sve downsample slojeve (conv1–conv3) za stabilno mapiranje iz visokih u niske dimenzije.
- Na upsample slojeve (conv4–conv8) za održavanje konzistentnosti tijekom rekonstrukcije.
- Ne primjenjuje se na prvi (conv0) i zadnji sloj (conv9) jer prvi sloj treba fleksibilnost za ulaznu transformaciju, a zadnji za finalni output.

PyTorch *spectral_norm* koristi power iteration metodu za efikasno računanje spektralne norme tijekom forward pass-a, čime se osigurava kontinuirana regularizacija bez značajnog povećanja računalnog opterećenja (slika 1).

```
def __init__(self, num_in_ch, num_feat=64, skip_connection=True):
    super().__init__()
    self.skip_connection = skip_connection
    norm = spectral_norm  # Spektralna normalizacija za stabilnost GAN-a
```

Slika 1. Inicijalizacija U-Net diskriminatora s primjenom spektralne normalizacije

Izvor: prikaz autora

Pojedini parametri U-Net diskriminatora definiraju njegovu osnovnu funkcionalnost. Broj ulaznih kanala (*num_in_ch*) obično je postavljen na 3 za RGB slike, no može se prilagoditi i drugim vrstama podataka, poput jednokanalnih medicinskih slika, čime se mreža prilagođava formatu ulaza. Broj osnovnih filtara (*num_feat*), standardno postavljen na 64, određuje širinu mreže i utječe na količinu značajki koju mreža izvlači u početnim slojevima; povećanjem tog broja mreža postaje osjetljivija na složenije strukture, ali se povećava i računalna složenost. Opcija preskočnih veza (*skip_connection*) definira hoće li se aktivirati veze između odgovarajućih slojeva u fazama smanjenja i povećanja rezolucije, što poboljšava očuvanje detalja i prijenos informacija bez gubitka konteksta. Spektralna normalizacija (*spectral_norm*)

primjenjuje se na konvolucijske slojeve kako bi se spriječio nekontrolirani rast težina, održavajući stabilnu dinamiku između diskriminatora i generatora u GAN arhitekturi. Zajedno, ovi parametri čine temeljnu strukturu U-Net diskriminatora i postavljaju okvir za njegovu funkciju razlikovanja stvarnih i generiranih slika.

3.3. Smanjenje rezolucije (Downsample blok)

Downsample blok U-Net diskriminatora sastoji se od tri konvolucijska sloja koji postupno smanjuju dimenzije slike i istovremeno povećavaju broj kanala feature mapa. Na taj način mreža izdvaja hijerarhijske značajke, od lokalnih detalja do globalnih struktura, što je ključno za razlikovanje stvarnih i generiranih slika. Svaki sloj prati odgovarajuća aktivacija (ReLU, PReLU ili LeakyReLU), a po potrebi se primjenjuje spektralna normalizacija radi stabilnijeg treniranja GAN arhitekture.

Primjena standardne formule za 2D konvoluciju određuje izlazne dimenzije slojeva, dok Downsample blok implementira koncept enkodera predstavljen u prethodnim sekcijama. Spektralna normalizacija ograničava maksimalnu singularnu vrijednost matrica težina konvolucijskih slojeva, sprječavajući eksplozivne gradijente i prekomjerno jačanje diskriminatora. Time se održava stabilan GAN proces, smanjujući rizik da diskriminator nadvlada generator (slika 2).

```
self.conv0 = nn.Conv2d(num_in_ch, num_feat, kernel_size=3, stride=1, padding=1)
self.conv1 = norm(nn.Conv2d(num_feat, num_feat*2, 4, 2, 1, bias=False))  # 1/2 rezolucije
self.conv2 = norm(nn.Conv2d(num_feat*2, num_feat*4, 4, 2, 1, bias=False))  # 1/4 rezolucije
self.conv3 = norm(nn.Conv2d(num_feat*4, num_feat*8, 4, 2, 1, bias=False))  # 1/8 rezolucije
```

Slika 2. Downsample blok U-Net diskriminatora s progresivnim smanjenjem dimenzija i udvostručenjem kanala

Izvor: prikaz autora

Tablica 2 prikazuje dimenzije izlaza za tipičnu RGB ulaznu sliku dimenzija 512 × 512 te broj kanala feature mapa nakon svakog sloja.

Tablica 2. Dimenzije izlaza i broj kanala feature mapa po slojevima SRVGGNetCompact mreže za ulaznu RGB sliku 512 × 512

Sloj	Kernel	Stride	Padding	Dimenzije izlaza (H×W)	Broj kanala
Conv0	3	1	1	512×512	64
Conv1	4	2	1	256×256	128
Conv2	4	2	1	128×128	256
Conv3	4	2	1	64×64	512

Parametri inicijalizacije *num_in_ch*, *num_feat*, *skip_connection* i *spectral_norm* određuju način na koji mreža procesuira ulazne podatke. Broj ulaznih kanala (*num_in_ch*) najčešće je 3 za RGB slike, ali može se prilagoditi i za jednokanalne ulazne podatke poput medicinskih slika. Funkcija *num_feat* definira širinu mreže i broj značajki koje se izvlače u početnim slojevima. Opcija *skip_connection* aktivira preskočne veze koje omogućuju očuvanje detalja i prijenos informacija kroz mrežu. Spektralna normalizacija (*spectral_norm*) primjenjuje se na konvolucijske slojeve radi stabilnog treniranja GAN arhitekture.

Downsample blok transformira ulaznu sliku u visokoapstraktne značajke koje pružaju diskriminatoru reprezentativne informacije za preciznu evaluaciju složenih struktura i detalja. Sljedeći korak u U-Net arhitekturi predstavlja upsample blok, koji vraća prostorne dimenzije slike prema originalnoj rezoluciji i rekonstruira detalje kroz kombinaciju naučenih značajki.
Svaki od slojeva conv1 do conv3 koristi spektralnu normalizaciju, koja je ključna za stabilnost treniranja GAN arhitekture. Spektralna normalizacija ograničava maksimalnu singularnu vrijednost matrica težina konvolucijskih slojeva, sprječavajući eksplozivne gradijente i prekomjerno jačanje diskriminatora. Time se smanjuje rizik da diskriminator „nadvlada“ generator, održavajući uravnotežen GAN proces.
Downsample blok time transformira ulaznu sliku iz niskorazinske prostorne reprezentacije u visokoapstraktne značajke viših dimenzija, pružajući diskriminatoru izražene i reprezentativne informacije za preciznu evaluaciju složenih struktura i detalja na slici.

Sljedeći korak u U-Net diskriminatoru predstavlja upsample blok, koji ima zadatak vratiti prostorne dimenzije slike prema originalnoj rezoluciji. Ovaj dio mreže koristi interpolaciju i konvolucijske slojeve s ciljem rekonstrukcije detalja i kombiniranja naučenih značajki kroz proces povećanja rezolucije.

3.4. Povećanje rezolucije (Upsample blok)

Nakon faze smanjenja rezolucije potrebno je postupno vratiti dimenzije slike na izvorni format. Na taj se način omogućuje da diskriminator evaluira generiranu sliku ne samo u apstraktnom prostoru značajki, nego i u dimenzijama koje su usporedive s izvornim podacima. Rekonstrukcija prostornih dimenzija provodi se u okviru upsample bloka koji kombinira interpolaciju i konvolucijske slojeve (slika 3).

```
self.conv4 = norm(nn.Conv2d(num_feat*8, num_feat*4, 3, 1, 1, bias=False))  # 512 → 256
self.conv5 = norm(nn.Conv2d(num_feat*4, num_feat*2, 3, 1, 1, bias=False))  # 256 → 128
self.conv6 = norm(nn.Conv2d(num_feat*2, num_feat, 3, 1, 1, bias=False))    # 128 → 64
```

Slika 3. Upsample blok U-Net diskriminatora

Izvor: prikaz autora

U prvoj fazi, sloj conv4 smanjuje broj kanala s 512 na 256, pri čemu prostorne dimenzije slike ostaju nepromijenjene unutar same konvolucijske operacije. Povećanje rezolucije provodi se interpolacijom koja prethodi konvoluciji. U drugom koraku, sloj conv5 dodatno smanjuje broj kanala na 128, pritom čuvajući reprezentacije značajki dobivene u prethodnim slojevima. Ovaj sloj sudjeluje u daljnjem rekonstruiranju slike prema višoj razini detalja. Treći sloj, conv6, vraća broj kanala na početnih 64 i tako usklađuje strukturu izlaznog tenzora s ulaznim parametrima mreže, čime se omogućuje integracija s preskočnim vezama.

Interpolacija povećava prostorne dimenzije slike obično za faktor 2 u svakoj dimenziji, što se može izraziti kao:

$$H_{interp} = 2 \times H_{in} \tag{4}$$

$$W_{interp} = 2 \times W_{in} \tag{5}$$

Nakon povećanja dimenzija interpolacijom, slijedi primjena konvolucijskih slojeva sa zadanim parametrima *kernel_size* = 3, *stride* = 1 i *padding* = 1. Kao što je već prikazano u potpoglavlju 3.1, izlazne dimenzije u tom slučaju ostaju jednake ulaznima. Time se osigurava da se prostorne dimenzije zadrže, dok se istodobno smanjuje broj kanala i rekonstruira slika prema izvornom formatu. Na primjeru ulaznog tenzora dimenzija 64 × 64, nakon interpolacije dimenzije rastu na 128 × 128, dok nakon konvolucije kroz sloj conv4 ostaju 128 × 128. Ovaj se postupak ponavlja u slojevima conv5 i conv6, čime se dimenzije slike postupno vraćaju na izvorni format.

Povećanje dimenzija odvija se dakle interpolacijom prije svake konvolucijske operacije, što predstavlja matematički precizan postupak udvostručenja prostorne visine i širine ulaznog tenzora. Na taj se način omogućuje glatko vraćanje slike na izvornu rezoluciju bez naglih prijelaza koji bi mogli uzrokovati artefakte poput blokova ili zubatosti. Kao i u slučaju downsample bloka, spektralna normalizacija se primjenjuje na svaki konvolucijski sloj unutar upsample bloka. Ova vrsta regularizacije doprinosi stabilnosti treniranja GAN-a i sprječava

dominaciju diskriminatora nad generatorom, čime se osigurava uravnotežena i konvergentna dinamika učenja.

U konačnici, upsample blok omogućuje da se visokoapstraktne značajke iz prethodnih slojeva mreže rekonstruiraju i prenesu u dimenzije usporedive s ulaznim slikama, što diskriminatoru pruža mogućnost precizne procjene vjerodostojnosti generiranog sadržaja.

3.5. Preskočne veze (Skip connections)

Skip veze predstavljaju ključnu komponentu U-Net arhitekture jer omogućuju prijenos informacija između slojeva u fazi smanjenja i povećanja rezolucije. Na taj način mreža zadržava fine detalje koji bi se inače izgubili tijekom višestrukih transformacija slike. U Real-ESRGAN diskriminatoru skip veze se aktiviraju samo ako je parametar *skip_connection* = True čime dodaju se informacije iz odgovarajućeg downsample sloja (slika 4).

```
if self.skip_connection:
    x4 = x4 + x2  # Dodaj odgovarajući donji sloj
```

Slika 4. Implementacija skip konekcija za očuvanje prostornih detalja kroz U-Net arhitekturu

Izvor: prikaz autora

Uloga skip veza može se sagledati kroz nekoliko aspekata. One povezuju odgovarajuće slojeve tako da se informacije iz downsample bloka, npr. × 2, prenose u odgovarajući upsample sloj, npr. × 4. Na taj način mreža kombinira niskorazinske značajke koje nose detalje s visokorazinskim značajkama koje opisuju globalnu strukturu slike. Preskočne veze osiguravaju očuvanje finih detalja koji bi tijekom procesa smanjenja rezolucije bili izgubljeni i time omogućuju vjerodostojniju rekonstrukciju. Osim toga, one olakšavaju propagaciju gradijenata kroz mrežu, čime se smanjuje rizik od nestajanja gradijenata i osigurava stabilnije i učinkovitije učenje. Parametar *skip_connection* omogućuje fleksibilno uključivanje ili isključivanje ove funkcionalnosti, što modelu daje mogućnost prilagodbe različitim scenarijima treniranja i evaluacije. Uvođenjem skip veza diskriminator postaje osjetljiviji na suptilne razlike između stvarnih i generiranih slika, što rezultira preciznijom i robusnijom procjenom.

3.6. Finalni blok: dodatne konvolucije i izlazni sloj

Nakon što su podaci prošli kroz downsample blokove, upsample blokove i skip veze, slijedi završna obrada u finalnom bloku čija je svrha proizvesti izlaznu procjenu vjerodostojnosti ulazne slike. Ovaj blok obuhvaća nekoliko dodatnih konvolucijskih slojeva te jedan završni

izlazni sloj. Slojevi conv7 i conv8 predstavljaju dodatne konvolucije koje zadržavaju dimenzije slike, ali omogućuju nelinearnu transformaciju značajki. Njihova uloga je dodatno prilagoditi reprezentacije za klasifikaciju i povećati sposobnost mreže da uoči složenije obrasce i ukloni redundantne informacije. Na oba sloja primijenjena je spektralna normalizacija koja osigurava stabilnost treniranja i smanjuje mogućnost pojave artefakata u završnom izlazu. Posljednji sloj, conv9, proizvodi jedan izlazni kanal koji odgovara procjeni vjerodostojnosti slike. Vrijednosti ovog kanala nalaze se u rasponu između 0 i 1, pri čemu rezultati bliži 1 označavaju veću vjerojatnost da je ulazna slika stvarna, dok vrijednosti bliže 0 upućuju na generirani uzorak (slika 5).

```
self.conv7 = norm(nn.Conv2d(num_feat, num_feat, 3, 1, 1, bias=False))  # Dodatna obrada

self.conv8 = norm(nn.Conv2d(num_feat, num_feat, 3, 1, 1, bias=False))  # Još obrade

self.conv9 = nn.Conv2d(num_feat, 1, 3, 1, 1)  # Konačni izlaz (1 kanal)
```

Slika 5. Finalni klasifikacijski blok U-Net diskriminatora s dodatnim konvolucijama i izlaznim slojem

Izvor: prikaz autora

Finalni blok tako dovršava proces evaluacije i omogućuje diskriminatoru donošenje binarne odluke o autentičnosti ulaznih podataka. U cijelosti, U-Net diskriminator prvo smanjuje rezoluciju ulazne slike kroz niz downsample blokova, zatim putem upsample blokova i skip veza rekonstruira izvorne dimenzije, dok se u završnom bloku primjenom dodatnih konvolucija i spektralne normalizacije dobiva konačna odluka o vjerodostojnosti ulaznog uzorka.

4. OPTIMIZIRANA IMPLEMENTACIJA Real-ESRGAN MODELA ZA REKONSTRUKCIJU VISOKO-REZOLUCIJSKIH SLIKA

Implementacija Real-ESRGAN-a u praksi zahtijeva dodatne optimizacije kako bi se osigurala učinkovitost u radu s vrlo velikim slikama i ograničenim hardverskim resursima. U tu svrhu razvijen je modul utilis.py, čija je zadaća omogućiti pouzdano i memorijski učinkovito izvođenje procesa nadogradnje rezolucije. Ključni element ovog modula je klasa RealESRGANer, koja omogućuje primjenu super-rezolucije na velike ulazne slike bez kompromisa u pogledu performansi ili kvalitete. Najvažnije optimizacije uključuju podjelu slike na manje segmente (tiling), podršku za različite formate uključujući transparentne kanale, mogućnost rada s 16-bitnom preciznošću radi ubrzanja te automatski odabir između CPU (eng. *Central Processing Unit* ili središnja procesorska jedinica**)** i GPU (eng. *Graphics Processing Unit* ili grafička procesorska jedinica) uređaja. Na taj način postiže se balans između kvalitete rekonstrukcije i računalne efikasnosti, čime se omogućuje fleksibilna i pouzdana primjena u zahtjevnim realnim scenarijima obrade slika.

4.1. Inicijalizacija klase i parametara

Prvi korak u optimiziranoj implementaciji Real-ESRGAN-a odnosi se na inicijalizaciju klase RealESRGANer, koja predstavlja ključno sučelje za korištenje modela. U ovoj fazi definiraju se osnovni parametri rada, uključujući faktor povećanja rezolucije, putanju do unaprijed treniranog modela, veličinu područja obrade (tile), te opcije vezane uz memorijsku efikasnost, preciznost izračuna i odabir između CPU ili GPU uređaja. Takva fleksibilnost omogućuje prilagodbu specifičnim uvjetima obrade, primjerice veličini ulazne slike, dostupnim hardverskim resursima i zahtjevima korisnika, čime se optimizira balans između brzine, memorijske zahtjevnosti i kvalitete rezultata. Sljedeći kod prikazuje osnovnu funkciju konstruktora klase s naglaskom na najvažnije parametre (slika 6).

```
1. def __init__(self, scale, model_path, dni_weight=None, model=None, tile=0, tile_pad=10,
   pre_pad=10, half=False, device=None, gpu_id=None):
```

Slika 6. Inicijalizacija klase RealESRGANer s definiranim ulaznim parametrima za kontrolu super-rezolucije

Izvor: prikaz autora

Parametri funkcije init imaju jasno definirane uloge u procesu inicijalizacije klase RealESRGANer. Parametar *scale* određuje faktor povećanja rezolucije, primjerice × 2 ili × 4,

što izravno utječe na konačnu veličinu izlazne slike, ali i na zahtjeve za memorijom i računanjem. Parametar *model_path* specificira putanju do unaprijed treniranog modela, pri čemu se može koristiti lokalna datoteka ili URL (eng. *Uniform Resource Locator* ili jedinstveni lokator resursa) za preuzimanje. Parametar *tile* određuje veličinu područja u pikselima na koja će se slika podijeliti radi obrade, pri čemu vrijednost 0 označava obradu cijele slike odjednom. Manji tile omogućuje obradu vrlo velikih slika u uvjetima ograničene memorije, iako može usporiti postupak. Parametar *tile_pad* definira veličinu dodatnog rubnog područja oko svakog tila kako bi prijelazi između segmenata bili glatki i bez artefakata, dok *pre_pad* određuje broj piksela kojima se slike dodatno proširuju prije obrade s ciljem očuvanja rubnih detalja. Parametar *half* predstavlja logičku opciju kojom se definira koristi li se 16-bitna preciznost proračuna, što značajno ubrzava proces uz minimalan gubitak kvalitete rezultata. Parametar *device* specificira uređaj na kojem će se model izvršavati, bilo CPU ili GPU, a ako nije izričito naveden odabir se određuje automatski. Konačno, parametar *gpu_id* omogućuje preciznu odredbu koji će se od dostupnih GPU uređaja koristiti za izvršavanje modela.

Sljedeći primjer prikazuje konkretno stvaranje objekta RealESRGANer s definiranim ključnim parametrima (slika 7).

```
upscaler = RealESRGANer(scale=4, model_path='models/RRDB_ESRGAN_x4.pth', tile=512)
```

Slika 7. Inicijalizacija objekta RealESRGANer s faktorom skaliranja × 4 i definiranom putanjom do modela

Izvor: prikaz autora

Ovaj primjer inicijalizira instancu RealESRGANer za × 4 povećanje rezolucije s modelom spremljenim lokalno, koristi tile-ove veličine 512 piksela s dodatnim paddingom, omogućuje 16-bitnu preciznost za bržu obradu i koristi prvi GPU uređaj.

4.2. Postavljanje uređaja (GPU/CPU)

Ključni korak u optimiziranoj implementaciji Real-ESRGAN-a je definiranje računalnog uređaja za izvođenje modela. Budući da duboke neuronske mreže zahtijevaju intenzivne izračune, GPU-ovi značajno ubrzavaju proces u usporedbi s CPU-om. Implementacija automatski detektira dostupne GPU resurse i preferira njihov odabir, dok se u slučaju njihove nedostupnosti ili neadekvatnosti model prebacuje na CPU, čime se osigurava maksimalna kompatibilnost. Mogućnost specificiranja GPU identifikatora dodatno omogućuje kontrolu nad korištenjem resursa u sustavima s više GPU jedinica (slika 8).

```
self.device = torch.device(f'cuda:{gpu_id}' if torch.cuda.is_available() else 'cpu')
```

Slika 8. Automatsko određivanje uređaja za izvršavanje modela Real-ESRGAN

Izvor: prikaz autora

Ova linija koda osigurava da se model uvijek može pokrenuti neovisno o dostupnosti specijaliziranog hardvera. Ako je GPU dostupan, koristi se grafički procesor određen zadanim ID-om, dok se u suprotnom obrada izvršava na CPU-u. Određivanje uređaja odvija se automatski prema dostupnosti resursa: sustav detektira CUDA-kompatibilni GPU i, ako je prisutan, pokreće model na tom uređaju čime se značajno ubrzava proces super-rezolucije, osobito kod visoko-rezolucijskih slika. U sustavima s više GPU jedinica moguće je precizno definirati koji GPU će se koristiti pomoću parametra *gpu_id*, čime se omogućuje kontrola nad raspodjelom resursa. Ako GPU nije dostupan, izvršavanje se automatski prebacuje na CPU, što i dalje osigurava funkcionalnost modela, iako uz sporiju obradu. Na taj način Real-ESRGAN postiže visoku fleksibilnost i prilagodljivost različitim računalnim okruženjima.

4.3. Učitavanje modela

Nakon inicijalizacije klase i definiranja uređaja na kojem će se model pokretati, sljedeći ključni korak je učitavanje prethodno treniranih težina. Bez njih proces super-rezolucije ne bi bio moguć jer bi model morao prolaziti kroz dugotrajan postupak učenja na velikim skupovima podataka. Implementacija u okviru RealESRGANer klase omogućuje jednostavno i fleksibilno učitavanje modela, bilo da se nalazi lokalno ili da se preuzima s udaljenog repozitorija. U ovoj fazi uspostavlja se veza između programskog koda i prethodno treniranih težina, što omogućuje daljnju nadogradnju i obradu slika. Provjerava se je li putanja do modela URL adresa, što omogućuje automatsko preuzimanje s interneta ukoliko model nije lokalno dostupan, primjenom funkcije *load_file_from_url*. Nakon preuzimanja, model se učitava u PyTorch format pomoću funkcije *torch.load*, s mapiranjem na CPU uređaj radi osiguravanja kompatibilnosti i fleksibilnosti u različitim okruženjima, od lokalnih računala do poslužitelja sa specifičnim hardverskim konfiguracijama (slika 9).

```
if model_path.startswith('https://'):
    model_path = load_file_from_url(model_path, model_dir='weights')
loadnet = torch.load(model_path, map_location='cpu')
```

Slika 9. Učitavanje prethodno treniranog modela Real-ESRGAN iz lokalnog ili mrežnog izvora

Izvor: prikaz autora

Proces učitavanja modela odvija se u nekoliko koraka. Prvo se provjerava izvor modela; ako *model_path* predstavlja URL, skripta automatski preuzima datoteku s mrežnog repozitorija i sprema je u lokalni direktorij *weights*, čime se izbjegava ručno preuzimanje. Ako je model već pohranjen lokalno, učitava se iz specificirane putanje pomoću funkcije *torch.load*, pri čemu parametar *map_location='cpu'* osigurava inicijalno učitavanje na CPU, nakon čega se prema potrebi težine premještaju na GPU. Nakon uspješnog učitavanja, model je spreman za obradu slika i, u kombinaciji s prethodno definiranim parametrima i odabranim uređajem, omogućuje potpunu inicijalizaciju sustava Real-ESRGAN.

4.4. Pred-obrada slike

Prije obrade modelom, slika se priprema u format prihvatljiv za PyTorch. Ulazna slika se konvertira iz NumPy polja u PyTorch tensor uz promjenu redoslijeda dimenzija s (visina, širina, kanali) u (kanali, visina, širina), a batch dimenzija se dodaje pomoću *unsqueeze(0)*, čime se dobiva oblik (1, *C*, *H*, *W*). Pripremljeni tensor se zatim premješta na odabrani uređaj (CPU ili GPU), osiguravajući da podaci imaju odgovarajuću strukturu i lokaciju za pravilno izvođenje modela Real-ESRGAN (slika 10).

```
img = torch.from_numpy(np.transpose(img, (2, 0, 1))).float()

self.img = img.unsqueeze(0).to(self.device)
```

Slika 10. Pred-obrada ulazne slike: konverzija u tensor, dodavanje batch dimenzije i premještanje na uređaj

Izvor: prikaz autora

Proces pred-obrade ulazne slike uključuje nekoliko faza. Prvo se slika, inicijalno učitana kao NumPy polje dimenzija (*H*, *W*, *C*), transponira u oblik (*C*, *H*, *W*) kako bi odgovarala PyTorch konvenciji rada s kanalima i konvertira u tip float radi numeričke preciznosti. Zatim se dodaje batch dimenzija pomoću funkcije *unsqueeze(0)*, čime slika poprima oblik (1, *C*, *H*, *W*) i zadovoljava standardne zahtjeve za ulaz u neuronske mreže. Nakon transformacija, pripremljeni tensor se premješta na prethodno definirani uređaj (CPU ili GPU), osiguravajući optimiziranu i konzistentnu daljnju obradu.

4.5. Obrada po dijelovima (Tiling)

Za obradu visoko-rezolucijskih slika koje premašuju memorijske kapacitete uređaja koristi se metoda tiling, pri kojoj se slika dijeli na manje, preklapajuće kvadratne segmente (tile-ove).

Svaki tile se zasebno obrađuje modelom, pri čemu dvostruka petlja prolazi kroz sve tile-ove po vertikalnoj i horizontalnoj osi, a rezultati se pažljivo spajaju u izlazni tensor, rekonstruirajući kompletnu nadograđenu sliku. Ovaj pristup omogućuje paralelnu i memorijski efikasnu obradu velikih slika, čime se postiže optimalan balans između performansi i kvalitete rezultata, čak i kod ograničenih hardverskih resursa (slika 11).

```
for y in range(tiles_y):
    for x in range(tiles_x):
        input_tile = self.img[:, :, y_start:y_end, x_start:x_end]
        output_tile = self.model(input_tile)
        self.output[:, :, y_out_start:y_out_end, x_out_start:x_out_end] = output_tile
```

Slika 11. Obrada slike po tile-ovima u Real-ESRGAN implementaciji

Izvor: prikaz autora

Proces obrade po dijelovima (tiling) u Real-ESRGAN implementaciji odvija se kroz tri ključna koraka. Prvo, ulazna slika se dijeli na manje kvadratne segmente, tzv. tile-ove, čime se omogućuje obrada vrlo velikih slika unutar memorijskih ograničenja. Zatim se svaki tile obrađuje zasebno pomoću modela, čime se postiže paralelna i učinkovita nadogradnja rezolucije bez preopterećenja hardvera. Na kraju, rezultati svih tile-ova pažljivo se spajaju u konačni izlazni tensor, a dodatni padding osigurava glatke prijelaze i očuvanje rubnih detalja, eliminirajući artefakte i osiguravajući kontinuitet slike. Ovaj pristup omogućuje skalabilnu, memorijski efikasnu i visoko kvalitetnu super-rezoluciju slika, primjenjivu u stvarnim scenarijima poput restauracije arhivskih fotografija, digitalizacije i pripreme vizualnih materijala za tiskane i digitalne medije. Za lakšu organizaciju rezultata preporuča se generirati nazive izlaznih slika na temelju originalnog imena datoteke uz dodatak oznake nadograđene rezolucije, primjerice *_upscaled_4×*.

4.6. Obrada alpha kanala

Kod slika koje osim standardnih RGB kanala sadrže i alpha kanal (RGBA format), potrebno je osigurati dodatnu obradu transparentnosti. Alpha kanal predstavlja informacije o prozirnosti pojedinih piksela i ključan je u slučajevima gdje se slike koriste u slojevitim kompozicijama, digitalnoj grafici ili vizualnim efektima. Ako se alpha kanal ne obradi pravilno, rezultat super-rezolucije može dovesti do neželjenih artefakata na rubovima prozirnih područja (slika 12).

```
if img_mode == 'RGBA':
    alpha = img[:, :, 3]
    output_img = cv2.cvtColor(output_img, cv2.COLOR_BGR2BGRA)
    output_img[:, :, 3] = output_alpha
```

Slika 12. Obrada alpha kanala i očuvanje prozirnosti u RGBA slikama

Izvor: prikaz autora

Obrada alpha kanala u Real-ESRGAN implementaciji uključuje nekoliko ključnih faza. Prvo se, ako je slika u RGBA formatu, četvrti kanal odvaja kao posebna matrica koja nosi podatke o prozirnosti piksela. Zatim se rezultat super-rezolucije, inicijalno u BGR formatu, konvertira u BGRA format kako bi se omogućilo dodavanje alpha kanala. Na kraju se izdvojeni alpha sloj spaja s obrađenom RGB slikom, čime se zadržava izvorna transparentnost dok se simultano povećava rezolucija vizualnog sadržaja. Ovaj postupak osigurava da Real-ESRGAN može pouzdano obrađivati slike s prozirnošću, što je ključno u slojevitim kompozicijama, digitalnoj grafici i vizualnim efektima.

4.7. Post-obrada

Nakon što neuronska mreža generira rezultat super-rezolucije, dobiveni podaci se i dalje nalaze u obliku PyTorch tenzora, s dodatnim dimenzijama i u formatu kanala koji nije izravno upotrebljiv za prikaz ili spremanje. Post-obrada je stoga ključan korak kojim se izlazna slika vraća u standardni format te priprema za vizualizaciju ili pohranu. Ovaj proces obuhvaća uklanjanje viška dimenzija, skaliranje vrijednosti u dopušteni raspon i transformaciju u *numpy* polje s odgovarajućim rasporedom kanala (slika 13).

```
output_img = output_img.data.squeeze().float().cpu().clamp_(0, 1).numpy()
output_img = np.transpose(output_img[[2, 1, 0], :, :], (1, 2, 0))
```

Slika 13. Post-obrada izlazne slike i transformacija u standardni RGB format

Izvor: prikaz autora

Post-obrada rezultata u Real-ESRGAN implementaciji sastoji se od nekoliko faza. Prvo se funkcijom *squeeze()* uklanjaju nepotrebne dimenzije tenzora nastale tijekom obrade, čime se podaci vraćaju u standardni oblik slike. Zatim se rezultat pretvara u tip float, premješta s GPU-a na CPU, a funkcija *clamp_(0,1)* osigurava da svi pikseli ostanu unutar raspona od 0 do 1, sprječavajući numeričke anomalije. Dobiveni podaci konvertiraju se u *numpy* format,

omogućujući daljnju obradu pomoću biblioteka poput OpenCV-a, nakon čega se transponiranjem kanala (*np.transpose*) prilagođava redoslijed iz BGR u standardni RGB format za ispravan prikaz. Ovim postupkom izlazna slika postaje spremna za vizualizaciju, daljnju analizu ili spremanje u datoteku.

4.8. Paralelno učitavanje (PrefetchReader)

Kako bi se optimizirao tijek obrade i spriječila uska grla u performansama, implementiran je modul za paralelno učitavanje slika. Glavna ideja je da se slike učitavaju u memoriju u pozadini, dok se prethodne već obrađuju. Time se izbjegava nepotrebno čekanje na I/O operacije i maksimalno koristi procesorsko ili grafičko vrijeme. Ovaj pristup je posebno koristan kod obrade velikih skupova podataka, gdje bi serijsko učitavanje značajno usporilo cijeli proces (slika 14).

```
def run(self):
    for img_path in self.img_list:
        img = cv2.imread(img_path, cv2.IMREAD_UNCHANGED)
        self.que.put(img)
```

Slika 14. Paralelno učitavanje slika pomoću PrefetchReader modula

Izvor: prikaz autora

Implementacija Real-ESRGAN-a uključuje paralelno učitavanje i obradu slika kako bi se maksimizirala učinkovitost sustava. Funkcija *run()* prolazi kroz listu putanja slika i učitava svaku sliku pomoću OpenCV funkcije *cv2.imread*, pohranjujući ih u red čekanja (queue) iz kojeg glavni proces preuzima podatke čim je spreman za obradu. Takav pristup omogućuje istovremeno učitavanje i obradu, čime se smanjuje ukupno trajanje postupka i povećava protočnost sustava. Ovaj dizajn, u skladu s principima višedretvenog programiranja, osigurava da GPU kontinuirano obrađuje podatke dok CPU paralelno priprema nove ulazne slike, čime se optimizira iskorištenost resursa i minimizira vrijeme čekanja.

4.9. Paralelno spremanje (IOConsumer)

Nakon što se slike obrade i transformiraju pomoću Real-ESRGAN modela, potrebno ih je pohraniti na disk. Kako bi se izbjegla blokada glavnog toka obrade zbog I/O operacija, implementirano je paralelno spremanje rezultata. Ovaj pristup omogućava da se dok se jedne

slike obrađuju, već završene odmah šalju u zasebnu dretvu za spremanje, čime se značajno ubrzava ukupni tijek rada (slika 15).

```
def run(self):
    while True:
        output = self._queue.get()
        cv2.imwrite(output['save_path'], output['output'])
```

Slika 15. Paralelno spremanje obrađenih slika pomoću IOConsumer modula

Izvor: prikaz autora

Implementacija Real-ESRGAN-a omogućuje paralelno spremanje obrađenih slika radi povećanja učinkovitosti sustava. Obrađene slike pohranjuju se u red čekanja (queue) iz kojeg ih modul IOConsumer preuzima čim postanu dostupne, dok se zapis na disk odvija u neovisnoj paralelnoj dretvi. Takav pristup uklanja usko grlo uzrokovano sporim I/O operacijama, povećava protočnost sustava i omogućuje kontinuiranu obradu velikih količina podataka bez zastoja. Time se optimizira iskorištenje GPU i CPU resursa za obradu, dok se rezultati istovremeno i neovisno arhiviraju, čineći Real-ESRGAN skalabilnim i pogodnim za profesionalna okruženja s velikim kolekcijama vizualnog materijala.

4.10. Primjer cjelovite upotrebe Real-ESRGAN modela

Ovdje se jasno i sažeto prikazuje kompletan, praktični workflow, od inicijalizacije klase, preko učitavanja slike i poboljšanja kvalitete, do spremanja rezultata.

Dodavanje ključnih karakteristika kao sažeti pregled ispod koda dodatno objektivno ističe prednosti implementacije.

Taj segment služi kao jasan most između tehničkih detalja i primjene, što korisnicima i čitateljima olakšava razumijevanje načina korištenja modela (slika 16).

```
# Inicijalizacija
upscaler = RealESRGANer(scale=4, model_path='models/RealESRGAN_x4.pth', tile=512)

# Učitavanje slike
img = cv2.imread('input.jpg', cv2.IMREAD_UNCHANGED)
```

```
# Poboljšanje kvalitete
output, _ = upscaler.enhance(img, outscale=4)


# Spremanje rezultata
cv2.imwrite('output.jpg', output)
```

Slika 16. Rezultat poboljšanja kvalitete slike pomoću RealESRGAN×4

Izvor: prikaz autora

Real-ESRGAN implementacija karakterizira visoka memorijska efikasnost, omogućena metodom tiling koja omogućuje obradu vrlo velikih slika koje inače ne stanu u memoriju ili bi zahtijevale prevelike resurse. Sustav je izuzetno fleksibilan, podržava različite ulazne formate slika uključujući RGB, RGBA (slike s alpha kanalom) i grayscale, čime je primjenjiv u širokom rasponu scenarija. Optimizacija izvedbe postiže se korištenjem half-precision (16-bitne) numeričke preciznosti te paralelnom obradom, što ubrzava cijeli proces i omogućuje efikasnu obradu u realnom vremenu ili batch režimu. Kvaliteta rezultata osigurava se integracijom naprednih modela za super-rezoluciju, čime se postiže očuvanje detalja i minimalni artefakti u konačnoj slici.

5. VGG-STIL SUPER-REZOLUCIJSKA ARHITEKTURA

U ovom poglavlju opisuje se VGG-stil (eng. *Visual Geometry Group*) arhitektura za super-rezoluciju slika, implementirana u modulu *srvgg_arch.py*. Arhitektura koristi duboke konvolucijske mreže sastavljene od sekvenci VGG blokova optimiziranih za povećanje rezolucije slike, balansirajući između složenosti modela i izvedbenih zahtjeva. Modularna definicija mreže omogućuje jednostavnu prilagodbu broja slojeva, broja značajki i faktora povećanja rezolucije, čime se postiže fleksibilnost u implementaciji i efikasnost u primjeni.

5.1. Definicija klase i inicijalizacija

Klasa SRVGGNetCompact definira kompaktni VGG-stil model duboke neuronske mreže dizajniran za zadatke super-rezolucije slika. Kroz inicijalizaciju klase određuju se ključni parametri, uključujući broj ulaznih i izlaznih kanala (obično 3 za RGB slike), broj konvolucijskih slojeva te faktor skaliranja rezolucije, npr. × 4. Zadane vrijednosti konstruktora omogućuju direktnu primjenu modela na RGB slike, dok se aktivacijska funkcija (*act_type*) može prilagoditi različitim varijantama optimizacije učenja. Ova konfigurabilnost čini arhitekturu prikladnom za širok spektar aplikacija u super-rezoluciji, istovremeno osiguravajući fleksibilnost i efikasnost izvedbe (slika 17).

```
@ARCH_REGISTRY.register()  # Registrira model u globalni registar arhitektura (koristi se u BasicSR)
class SRVGGNetCompact(nn.Module):  # Definira PyTorch neuronsku mrežu
    def __init__(
        self,
        num_in_ch=3,      # Ulazni kanali (3 za RGB)
        num_out_ch=3,     # Izlazni kanali (3 za RGB)
        num_feat=64,      # Broj značajki u skrivenim slojevima
        num_conv=16,      # Broj konvolucijskih slojeva u tijelu mreže
        upscale=4,        # Faktor povećanja rezolucije (npr. 4x super-rezolucija)
        act_type='prelu'  # Tip aktivacijske funkcije ('relu', 'prelu', 'leakyrelu')
    ):
```

```
super(SRVGGNetCompact, self).__init__()  # Inicijalizira nadklasu (nn.Module)
```

Slika 17. Struktura i inicijalizacija klase SRVGGNetCompact za VGG-stil super-rezolucijsku mrežu

Izvor: prikaz autora

Klasa SRVGGNetCompact implementira kompaktnu VGG-stil super-rezolucijsku mrežu optimiziranu za RGB slike. Konstruktor klase omogućuje detaljnu konfiguraciju mreže, uključujući ulazno-izlazne dimenzije, broj značajki u skrivenim slojevima, broj konvolucijskih slojeva i faktor povećanja rezolucije.

Parametri klase imaju sljedeću funkcionalnost:

- num_in_ch: definira broj ulaznih kanala, što odgovara dubini ulazne slike; za standardne RGB slike koristi se vrijednost 3,
- num_out_ch: određuje broj kanala izlazne slike, obično također 3 za RGB,
- num_feat: postavlja broj značajki u svakom konvolucijskom sloju skrivenog tijela mreže, što utječe na kapacitet mreže za učenje reprezentacija slike,
- num_conv: definira broj konvolucijskih slojeva u tijelu mreže; veći broj slojeva omogućuje detaljnije učenje tekstura i struktura slike, ali povećava zahtjeve za memorijom i vrijeme treniranja,
- upscale: određuje faktor povećanja rezolucije, što je ključni parametar za super-rezolucijske aplikacije,
- act_type: specificira tip aktivacijske funkcije koja se koristi nakon svakog konvolucijskog sloja; prelu je zadani izbor zbog svoje sposobnosti prilagodbe negativnih vrijednosti i stabilnog treniranja.

Konstruktor ove klase također registrira mrežu u globalni registar arhitektura putem dekoratora @ARCH_REGISTRY.register(), što omogućuje jednostavno pozivanje i korištenje mreže u okviru BasicSR biblioteke. Zadane vrijednosti parametara omogućuju treniranje mreže za standardne super-rezolucijske zadatke nad RGB slikama s faktorom povećanja × 4, što je uobičajeni scenarij u primjeni super-rezolucijskih tehnologija na fotografije i video.

5.2. Pohrana parametara i inicijalizacija tijela mreže

U ovoj fazi konstruktor klase SRVGGNetCompact pohranjuje konfiguracijske parametre mreže, uključujući broj ulaznih i izlaznih kanala, broj značajki u skrivenim slojevima, broj konvolucijskih slojeva i faktor skaliranja rezolucije. Istovremeno inicijalizira tijelo mreže koristeći *nn.ModuleList*, što predstavlja spremište za konvolucijske slojeve. Korištenje

ModuleList objekta omogućuje fleksibilno upravljanje slojevima mreže i osigurava pravilnu registraciju slojeva za automatizirano računanje gradijenata u PyTorchu. Time se postiže transparentnost i efikasnost u fazi treniranja i izvođenja modela, što je ključno za pouzdanu super-rezolucijsku obradu slika (slika 18).

```
self.num_in_ch = num_in_ch

self.num_out_ch = num_out_ch

self.num_feat = num_feat

self.num_conv = num_conv

self.upscale = upscale

self.act_type = act_type



self.body = nn.ModuleList()  # Lista za pohranu svih slojeva (fleksibilnija od Sequential)
```

Slika 18. Pohrana slojeva tijela VGG-style SR mreže

Izvor: prikaz autora

Parametri pohranjeni u objektu omogućuju jednostavan pristup konfiguraciji mreže tijekom treniranja i evaluacije. ModuleList se koristi za pohranu slojeva jer omogućuje dinamičko dodavanje ili izmjenu slojeva, a istovremeno pravilno registrira sve module u PyTorchovom autograd sustavu, osiguravajući automatsko izračunavanje gradijenata.

Ova struktura omogućuje fleksibilno definiranje tijela mreže s proizvoljnim brojem slojeva i značajki, što je ključno za eksperimentiranje s različitim arhitekturama i kapacitetima VGG-stil super-rezolucijskih mreža.

5.3. Prva konvolucija i aktivacija

Prvi korak u tijelu mreže SRVGGNetCompact pretvara ulaznu RGB sliku s tri kanala u prostor značajki s većim brojem kanala (*num_feat*), obično 64-kanalni feature map, čime se omogućuje izdvajanje lokalnih obrazaca i tekstura. Nakon konvolucije primjenjuje se aktivacijska funkcija koja uvodi nelinearnost u model, što mreži omogućuje modeliranje složenih i ne-linearnih odnosa unutar podataka. Podržane su različite vrste aktivacija, uključujući ReLU, PReLU (parametrizirani ReLU koji uči nagib za negativne vrijednosti) i LeakyReLU (koji dozvoljava mali nagib za negativne vrijednosti), čime se povećava fleksibilnost i prilagodljivost mreže zadatku super-rezolucije (slika 19).

```
# Prva konvolucija: Preslikava ulaz (RGB) u prostor značajki
    self.body.append(nn.Conv2d(num_in_ch, num_feat, 3, 1, 1))  # Conv2d(ulaz, izlaz, jezgra, korak, dopuna)

    # Prva aktivacija: Uvodi nelinearnost
    if act_type == 'relu':
        activation = nn.ReLU(inplace=True)  # ReLU: max(0, x)
    elif act_type == 'prelu':
        activation = nn.PReLU(num_feat)     # PReLU: uči nagib za negativne vrijednosti
    elif act_type == 'leakyrelu':
        activation = nn.LeakyReLU(0.1, inplace=True)  # LeakyReLU: mali nagib (0.1) za negativne
    self.body.append(activation)
```

Slika 19. Prvi konvolucijski sloj i aktivacija VGG-style SR mreže

Izvor: prikaz autora

Prvi konvolucijski sloj Conv2d preslikava ulaznu sliku s tri kanala (RGB) u prostor značajki definiran brojem kanala *num_feat*, npr. 64. Jezgra konvolucije veličine 3 × 3 s korakom 1 i paddingom 1 zadržava prostorne dimenzije slike, omogućujući mreži učenje lokalnih obrazaca. Sljedeća aktivacija uvodi nelinearnost u mrežu, što je ključno za modeliranje složenih uzoraka i tekstura. Tip aktivacije je konfigurabilan: ReLU primjenjuje funkciju *max(0,x)*, jednostavnu i učinkovitu aktivaciju; PReLU uči nagib za negativne vrijednosti, čime se poboljšavaju performanse kod kompleksnih uzoraka; LeakyReLU uvodi mali nagib od 0,1 za negativne vrijednosti kako bi se izbjeglo "mrtvo" stanje neurona. Ovaj korak postavlja temelje za daljnje konvolucijske slojeve mreže, omogućujući učinkovito učenje visokodimenzionalnih reprezentacija.

5.4. Konvolucijski blokovi u tijelu mreže

U tijelu mreže SRVGGNetCompact definira se niz uzastopnih konvolucijskih blokova, svaki sastavljen od konvolucijskog sloja s jezgrama 3 × 3 i odgovarajuće aktivacije. Ovi blokovi se ponavljaju onoliko puta koliko je definirano parametrom *num_conv* (zadano 16 slojeva). Sve

konvolucije koriste korak 1 i padding 1, čime se očuvavaju prostorne dimenzije značajki kroz sve slojeve, dok broj kanala značajki ostaje konstantan (*num_feat = 64*). Takav niz slojeva omogućuje duboko i postupno ekstraktiranje značajki iz ulazne slike, što je ključno za postizanje visoke kvalitete super-rezolucije (slika 20).

```
# Ponavlja Conv + Aktivacija `num_conv` puta (zadano: 16 slojeva)
    for _ in range(num_conv):
        self.body.append(nn.Conv2d(num_feat, num_feat, 3, 1, 1))  # Ista konvolucija kao gore
        if act_type == 'relu':
            activation = nn.ReLU(inplace=True)
        elif act_type == 'prelu':
            activation = nn.PReLU(num_feat)
        elif act_type == 'leakyrelu':
            activation = nn.LeakyReLU(0.1, inplace=True)
        self.body.append(activation)
```

Slika 20. Konvolucijski blokovi tijela VGG-style SR mreže

Izvor: prikaz autora

Ovi ponovljeni blokovi imaju nekoliko ključnih značajki:

- 3 × 3 jezgre konvolucije: Standardni VGG-stil omogućuje učinkovito učenje lokalnih obrazaca i tekstura,
- očuvanje prostorne rezolucije: korak 1 i padding 1 osiguravaju da dimenzije značajki ostanu iste kao ulazne, što je važno za super-rezolucijske zadatke,
- fiksni broj kanala: svaki sloj koristi num_feat značajki (npr. 64), čime se postiže kompaktan dizajn bez gubitka kapaciteta učenja,
- nelinearnost: aktivacijski sloj (ReLU, PReLU ili LeakyReLU) uvodi nelinearnost u mrežu, omogućujući modeliranje složenih tekstura i obrazaca.

Ovaj segment tijela mreže predstavlja jezgru VGG-style super-rezolucijske arhitekture, gdje ponovljeni Conv + Aktivacija blokovi postupno poboljšavaju reprezentaciju značajki i pripremaju podatke za završne slojeve mreže.

5.5. Završna konvolucija i PixelShuffle

Završni slojevi mreže SRVGGNetCompact pripremaju značajke za generiranje izlazne slike u visokoj rezoluciji. Konvolucijski sloj proširuje broj kanala na vrijednost potrebnu za prostorno povećanje rezolucije, što znači da broj izlaznih kanala postaje jednak broju kanala izlazne slike pomnoženom s kvadratom faktora skaliranja (npr. × 4 povećanje: $3 \times 4^2 = 48$). Nakon toga, PixelShuffle sloj reorganizira te proširene kanale u prostorne blokove, efektivno rekonstruirajući sliku povećane razlučivosti. Ova funkcija "raspakira" zgusnute značajke u prostorne koordinate, omogućujući povećanje dimenzija ulazne slike bez gubitka informacija i precizno nadopunjavanje detalja u konačnoj slici (slika 21).

```
# Završna konvolucija: Proširuje kanale za PixelShuffle
    self.body.append(nn.Conv2d(num_feat, num_out_ch * (upscale ** 2), 3, 1, 1))


    # PixelShuffle: Preuređuje kanale u prostorno povećanje (npr. 4x)
    self.upsampler = nn.PixelShuffle(upscale)
```

Slika 21. Završna konvolucija i PixelShuffle u VGG-style SR mreži

Izvor: prikaz autora

Završna Conv2d operacija povećava broj kanala na *num_out_ch*×*upscale*², što za × 4 super-rezoluciju i RGB slike daje 48 kanala. Ti kanali se zatim preuređuju pomoću PixelShuffle sloja u prostorne koordinate, efektivno skalirajući sliku. Svaki piksel niske rezolucije raspakira se u blok od 4 × 4 piksela visoke rezolucije, zadržavajući detalje i teksture iz prethodnih konvolucijskih slojeva, što omogućuje preciznu rekonstrukciju i očuvanje informacija.

5.6. Prolaz unaprijed (ključni koraci)

Metoda forward u SRVGGNetCompact modelu definira ključni protok podataka tijekom inferencije. Ulazna niskorezolucijska slika (LR) prvo prolazi kroz sve slojeve tijela mreže, niz konvolucijskih slojeva i aktivacija, gdje se ekstraktiraju dubinske značajke važne za rekonstrukciju visoke rezolucije (HR). Zatim se značajke prostorno povećavaju pomoću PixelShuffle sloja, transformirajući ih u prostor HR slike. Na kraju se primjenjuje residualno učenje dodavanjem osnovno povećane LR slike, dobivene nearest-neighbor interpolacijom, čime mreža uči samo detalje i korekcije potrebne za kvalitetnu super-rezoluciju (slika 22).

```
def forward(self, x):
    out = x  # Počinje s LR ulazom (npr. 32x32 RGB)

    # Prolazi kroz sve slojeve tijela (konvolucije + aktivacije)
    for layer in self.body:
        out = layer(out)  # out oblik: [B, num_feat, H, W]

    # Povećava rezoluciju pomoću PixelShuffle (npr. 32x32 -> 128x128)
    out = self.upsampler(out)  # out oblik: [B, 3, H*4, W*4]

    # Dodaje LR sliku povećanu nearest-neighbor metodom (residualno učenje)
    base = F.interpolate(x, scale_factor=self.upscale, mode='nearest')
    out += base  # Dodaje osnovnu povećanu sliku naučenim detaljima

    return out  # Konačni HR izlaz (npr. 128x128 RGB)
```

Slika 22. Prolaz unaprijed u SRVGGNetCompact

Izvor: prikaz autora

Prolaz unaprijed u SRVGGNetCompact uključuje tri ključna koraka. Prvo, LR ulaz prolazi kroz sve konvolucijske slojeve tijela mreže, što omogućuje ekstrakciju dubinskih značajki i detalja slike. Zatim PixelShuffle sloj prostorno reorganizira višekanalni izlaz u HR prostor, efektivno transformirajući značajke u piksele visoke rezolucije. Konačno, residualno učenje omogućuje mreži da nauči samo razliku između stvarne HR slike i jednostavno povećane LR slike, pri čemu dodavanje osnovne interpolirane LR slike olakšava treniranje i poboljšava stabilnost učenja. Ovaj kombinirani pristup omogućuje preciznu rekonstrukciju visokorezolucijskih slika, balansirajući dubinsko učenje, efikasno skaliranje i stabilnost mreže.

5.7. Vizualizacija ključnih operacija i koraka

U ovom poglavlju prikazuje se vizualni pregled ključnih operacija i tijeka podataka unutar SRVGGNetCompact mreže. Posebna pažnja posvećena je dimenzijama tenzora prije i nakon

svake operacije, kao i funkcionalnoj ulozi svakog sloja ili modula. Sljedeća tablica pruža sažet prikaz glavnih koraka obrade slike kroz mrežu, olakšavajući razumijevanje strukture i dinamike modela (tablica 3).

Tablica 3. Ključne operacije i dimenzije tenzora kroz SRVGGNetCompact mrežu

Operacija	Ulazni oblik	Izlazni oblik	Funkcija
Conv2d(3, 64, 3)	[1, 3, 32, 32]	[1, 64, 32, 32]	Ekstrahira početne značajke iz LR ulaza
16× Conv2d(64, 64, 3)	[1, 64, 32, 32]	[1, 64, 32, 32]	Postupno poboljšava značajke slike
Conv2d(64, 48, 3)	[1, 64, 32, 32]	[1, 48, 32, 32]	Priprema značajke za PixelShuffle
PixelShuffle(4)	[1, 48, 32, 32]	[1, 3, 128, 128]	Povećava rezoluciju 4x
+ nearest_interp	[1, 3, 32, 32] → [1, 3, 128, 128]	[1, 3, 128, 128]	Dodaje residualnu bazu, olakšavajući učenje detalja

Ova tablica vizualizira tijek podataka kroz mrežu, prikazujući ključne transformacije dimenzija i funkcionalnost slojeva, uključujući konvolucije, PixelShuffle i residualno učenje za generiranje visokorezolucijskog izlaza.

Za usporedbu s postojećim super-rezolucijskim modelima korišten je Real-ESRGAN (xinntao). Postupci pokretanja na macOS i Windows sustavima uključuju korištenje portable izvršnih datoteka ili Python skripti s Anacondom/CUDA. Osnovni parametri uključuju izbor modela, faktor povećanja i opciju za automatsko poboljšanje lica (tablica 4).

Tablica 4. Sažeti koraci za korištenje Real-ESRGAN modela (macOS / Windows)

Operativni sustav	Metoda	Opis
macOS	Portable	Preuzimanje „Real-ESRGAN-ncnn-vulkan“ s GitHub Releases, raspakiranje i pokretanje u Terminalu; pojedinačne slike ili folder
macOS	Python	Instalacija Anaconde, kloniranje repozitorija i pokretanje skripte za inferenciju nad folderom slika; veća fleksibilnost i integracija
Windows	Portable (.exe)	Preuzimanje .zip arhive, raspakiranje, slike u istu mapu kao .exe, pokretanje u CMD/PowerShellu
Windows	Python + CUDA	Instalacija Anaconde i PyTorcha s CUDA podrškom, kreiranje virtualnog okruženja, instalacija paketa, pokretanje skripti; GPU akceleracija
Općenito	Dodatni savjeti	Video: razbiti na frameove i ponovno spojiti; fotografije: realesrgan-x4plus; anime: realesrgan-x4plus-anime; --face_enhance za lica; sve naredbe u direktoriju s izvršnom datotekom

6. REZULTATI

Ovo poglavlje prikazuje učinke primjene Real-ESRGAN modela na različite vrste ulaznih slika. Svaka slika prikazana je u dva stupca: originalna i obrađena verzija, kako bi se jasno istaknuli rezultati povećanja rezolucije i poboljšanja detalja.
Cjelokupan postupak primjene Real-ESERGAN modela dostupan je na YouTube kanalu: https://www.youtube.com/watch?v=Ra4DiNXys2w

Slike prikazuju efekt primjene Real-ESRGAN modela na različite vrste ulaznih slika. Svaka slika je prikazana kao originalna slika i slika nakon AI poboljšanja.

Slika 23. Primjer 1 (originalna slika lijevo i AI poboljšana slika desno)
Izvor: prikaz autora

Slika 23 prikazuje osnovni učinak Real-ESRGAN-a na standardnu fotografiju. Nakon obrade vidljivo je poboljšanje oštrine, izraženiji detalji i jasniji teksturalni elementi, bez pojave vidljivih artefakata. Primjer jasno pokazuje sposobnost Real-ESRGAN-a za nadogradnju kvalitete slike u stvarnim uvjetima, potvrđujući njegovu učinkovitost i u praktičnim scenarijima poput video tutoriala.

Slika 24. Primjer 2 (originalna slika lijevo i AI poboljšana slika desno)

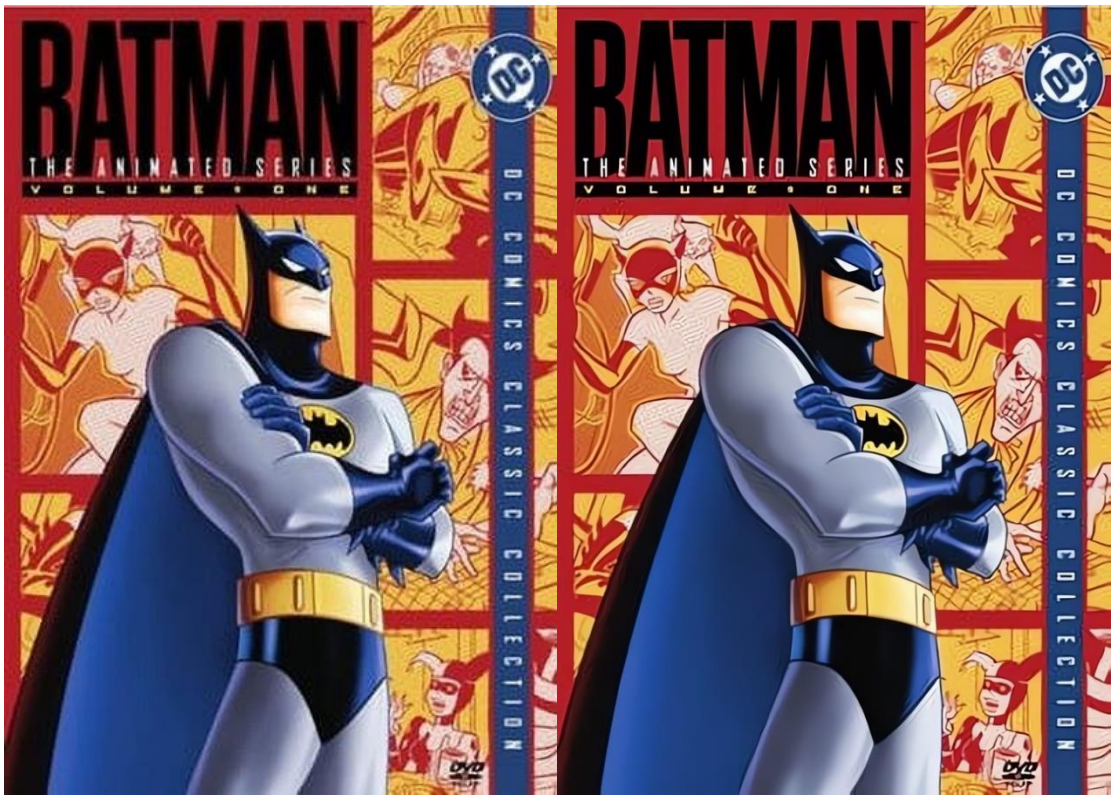
Izvor: prikaz autora

Slika 24 prikazuje učinak Real-ESRGAN-a na ilustraciju ili fotografiju s finim detaljima. Nakon obrade rubovi su jasno definirani, teksture i detalji su vizualno izraženiji, a šum i artefakti značajno smanjeni, čime se poboljšava ukupna vizualna kvaliteta slike i potvrđuje učinkovitost modela na raznovrsnim scenama.

Slika 25. Primjer 3 (originalna slika lijevo i AI poboljšana slika desno)

Izvor: prikaz autora

Slika 25 prikazuje učinak Real-ESRGAN-a na sliku s kombinacijom teksture i boja. Model učinkovito povećava rezoluciju i naglašava detalje, zadržavajući prirodan izgled i vizualnu koherenciju. U područjima s tekstom mogu se pojaviti manja odstupanja u čitljivosti zbog kompleksnosti tipografije, no ukupna kvaliteta slike je značajno poboljšana.



Slika 26. Primjer 4 (originalna slika lijevo i AI poboljšana slika desno)

Izvor: prikaz autora

Slika 26 prikazuje ograničenja Real-ESRGAN-a kod slika s tekstualnim elementima. Model često ne zadržava oštrinu i čitljivost slova, posebno kod malih fontova ili niske rezolucije, pri čemu tipografski elementi mogu postati deformirani jer mreža nije trenirana za preciznu reprodukciju geometrije teksta.

Jedan od čestih problema pri korištenju Real-ESRGAN-a javlja se kod slika koje sadrže tekst. Iako model vrlo učinkovito poboljšava vizualne detalje i strukturu prirodnih scena, tekstualni elementi, osobito mali ili niske rezolucije, često gube jasnoću i oštrinu te mogu postati deformirani ili nečitki. To proizlazi iz činjenice da je mreža prvenstveno trenirana za reprodukciju prirodnih detalja, a ne za očuvanje preciznih geometrijskih oblika karakterističnih za tipografiju. U praksi to znači da kod vizualnog materijala koji kombinira slike i tekst, poput postera ili skeniranih dokumenata, Real-ESRGAN može smanjiti čitljivost, što ograničava njegovu primjenu u takvim scenarijima.

7. USPOREDBA S PSNR, SSIM, NIQE i LPIPS METRIKAMA

Ovo poglavlje prikazuje kvantitativnu i subjektivnu evaluaciju Real-ESRGAN modela. Analiza obuhvaća usporedbu s prethodnim modelima ESRGAN i AESRGAN na standardnim datasetima, uz dodatni osvrt na praktične izazove batch i video obrade. Primjena više metrika omogućuje holistički uvid u performanse modela s aspekta numeričke preciznosti i perceptualne kvalitete.

7.1. Opis korištenih metrika

Za kvantitativnu ocjenu kvalitete rekonstruiranih slika korištene su četiri vodeće metrike: PSNR (eng. *Peak Signal-to-Noise Ratio*), SSIM (eng. *Structural Similarity Index*), NIQE (eng. *Natural Image Quality Evaluator*) i LPIPS (eng. *Learned Perceptual Image Patch Similarity*) [19, 23, 24]. One pružaju komplementarne uvide, od čiste numeričke preciznosti do perceptualne vizualne kvalitete. Tablica 5 prikazuje osnovne karakteristike svake metrike i način interpretacije rezultata.

Tablica 5. Opis korištenih metrika za kvantitativnu evaluaciju (PSNR, SSIM, NIQE, LPIPS)

Metrika	Opis	Interpretacija
PSNR (Peak Signal-to-Noise Ratio)	Mjeri razliku između originalne i obrađene slike na osnovi piksela	Veće vrijednosti = bliže originalu
SSIM (Structural Similarity Index)	Procjenjuje sličnost u strukturi, kontrastu i svjetlini	Veće vrijednosti = vizualno sličnije originalu
NIQE (Natural Image Quality Evaluator)	No-reference metrika koja ocjenjuje prirodnost slike	Manje vrijednosti = prirodnija slika
LPIPS (Learned Perceptual Image Patch Similarity)	Mjeri perceptualnu sličnost temeljem dubokih značajki	Manje vrijednosti = perceptivno bliže originalu

Ove metrike omogućuju kombinaciju objektivnog numeričkog mjerenja i perceptualne evaluacije, što je ključno za cjelovitu procjenu kvalitete super-rezolucijskih slika. PSNR pruža numeričku vjernost piksel-po-piksel, SSIM bolje korelira s ljudskom percepcijom, NIQE ocjenjuje prirodnost bez referentne slike, a LPIPS prati perceptualnu sličnost s ljudskim dojmom.

7.2. Kvantitativna usporedba modela

U tablici 6 prikazani su prosječni rezultati Real-ESRGAN-a u usporedbi s ESRGAN i AESRGAN modelima na različitim standardnim datasetima. Tablica omogućuje jednostavno prepoznavanje prednosti svakog modela i njihove sposobnosti generalizacije.

Tablica 6. Kvantitativna usporedba SR modela na standardnim datasetima (PSNR, SSIM, NIQE, LPIPS, vizualni učinak)

Model	Dataset	PSNR (dB)	SSIM	NIQE	LPIPS	Vizualni učinak
ESRGAN	FFHQ	24,14	0,72	4,21	0,35	Blago zamućeni detalji, artefakti oko rubova, fine teksture djelomično izgubljene.
Real-ESRGAN	FFHQ	24,97	0,76	3,85	0,28	Jasniji detalji lica (oči, kosa), glatko uklanjanje artefakata, prirodniji izgled.
AESRGAN	FFHQ	26,42	0,80	3,62	0,25	Najveća jasnoća i detaljnost, minimalni artefakti, superiorna percepcijska kvaliteta.
ESRGAN	Set5	28,31	0,78	4,05	-	Djelomično izoštravanje, teksture prirodnog okruženja ponekad izgubljene.
Real-ESRGAN	Set5	29,05	0,81	3,79	-	Bolja reprodukcija detalja, ravnomjernije boje, prirodniji kontrast.
AESRGAN	Set5	29,88	0,83	3,55	-	Najpreciznija struktura, finiji detalji i kontrast, vizualno najbliže originalu.
ESRGAN	Set14	27,89	0,76	4,12	-	Oštri rubovi ponekad degradirani, artefakti vidljivi u teksturama.
Real-ESRGAN	Set14	28,54	0,79	3,88	-	Poboljšana oštrina, manje artefakata, prirodnija tekstura i boje.
AESRGAN	Set14	29,35	0,82	3,63	-	Superiorna detaljnost i vizualna koherencija, gotovo bez vidljivih artefakata.

Rezultati pokazuju da Real-ESRGAN dosljedno poboljšava percepcijsku kvalitetu u odnosu na klasični ESRGAN, dok AESRGAN postiže najbolje numeričke rezultate. Ovi podaci potvrđuju da Real-ESRGAN balansira između očuvanja detalja i uklanjanja artefakata, dok AESRGAN naglašava maksimalnu jasnoću i preciznost.

7.3. Evaluacija na stvarnim i specijalnim datasetima

Nakon analize standardnih datasetova, evaluacija je provedena i na stvarnim i specijalnim datasetovima koji uključuju lica, prirodne i urbane scene, kao i medicinske i satelitske slike. Tablica 7 sažima ključne karakteristike svakog dataseta i rezultat Real-ESRGAN modela s obzirom na očuvanje detalja, smanjenje artefakata i poboljšanje vizualnog dojma.

Tablica 7. Evaluacija Real-ESRGAN modela na stvarnim i specijalnim datasetima

Dataset	Karakteristike	Rezultat Real-ESRGAN
FFHQ	Lica, fin detalj (oči, kosa, koža)	Zadržava strukturu, smanjuje artefakte, poboljšava subjektivni dojam.
Set5 / Set14	Prirodne i urban scene	Bolja reprodukcija detalja, ravnomjernije boje i kontrast.
BSD100	Generalne slike visoke varijabilnosti	Poboljšava rubove i teksture, minimalno degradira detalje.
Medicinski / satelitski setovi	Specijalni obrasci i visoka rezolucija	Superiorna percepcijska kvaliteta, očuvanje finih uzoraka.

Evaluacija na stvarnim datasetima pokazuje tendenciju Real-ESRGAN-a za blago zaglađivanje finih tekstura, ali ujedno očuvanje strukture i smanjenje artefakata u kritičnim regijama. Subjektivne ocjene (MOS) potvrđuju da model poboljšava vizualni dojam lica i složenih scena, dok batch obrada osigurava visoku učinkovitost s prosječnim vremenom obrade od 0,08 sekundi po slici.

7.4. Izazovi i rješenja u batch obradi

Kod batch obrade velikih skupova slika pojavljuju se specifični izazovi vezani uz upravljanje memorijom, paralelizaciju i skalabilnost. Tablica 8 daje pregled najčešćih problema i tehničkih rješenja koja omogućuju pouzdanu i učinkovitu batch obradu s Real-ESRGAN modelom.

Tablica 8. Izazovi i rješenja u batch obradi velikih skupova slika

Izazov	Uzrok	Rješenje
Ograničen VRAM / I/O bottleneck	Velike batch veličine, podatkovni transferi	Dinamično određivanje batch size, asinkrono čitanje/pisanje, lazy loading.
Paralelna obrada na slabijim GPU-ima	Prekomjerno opterećenje GPU memorije	GPU scheduling, CUDA streamovi, ograničavanje simultanih procesa.
Distribuirana obrada	Višestruki serveri ili cloud	Task queue sustavi (Celery, Redis) za kontrolu dohvaćanja i spremanja rezultata.

Kod velikih batchova (> 1000 slika), problemi s upravljanjem memorijom i I/O uskim grlima značajno utječu na performanse. Tehničke strategije, uključujući dinamičko određivanje veličine batcha, asinkrono čitanje/pisanje i lazy loading, te paralelnu kontrolu GPU resursa, omogućuju stabilnu i skalabilnu batch obradu.

7.5. Specifičnosti rada s video i nestandardnim formatima

Video upscaling ima dodatnu kompleksnost jer obrada frameova mora održati prostornu i vremensku konzistenciju kako bi se smanjio “flicker” i artefakti između susjednih frameova.

Tipičan workflow uključuje dekodiranje videa na pojedinačne frameove (npr. ffmpeg), batch obradu svakog framea zasebno s Real-ESRGAN-om, te ponovno enkodiranje u video format [10]. Problemi mogu nastati zbog limitiranog GPU memorijskog kapaciteta pri obradi visoke rezolucije i velike količine frameova, kao i zbog produženog vremena procesiranja kod videa iznad 30 s, što odgovara više od 900 frameova za 30 FPS [25].

Još jedan izazov predstavljaju specijalni formati slike i videa (HEIC, RAW, nestandardni video kodeksi). Real-ESRGAN po defaultu podržava najčešće formate (PNG, JPG, MP4, AVI), ali za naprednije primjene integracija s konverterima (ImageMagick, ffmpeg, OpenCV) omogućuje efikasno pretvaranje inputa prije batch upscalinga [26, 27].

Za streaming aplikacije potrebno je višestruko optimizirati pipeline, uključujući model quantization, deployment kao ONNX ili TensorRT, kako bi upscaling mogao ići u polu-realnom vremenu, pri čemu treba balansirati između kvalitete i brzine, uz potencijalnu redukciju veličine modela ili primjenu pruninga [25, 26].

Batch obrada velikih kolekcija slika ili frameova donosi dodatne izazove u upravljanju memorijom, skalabilnosti i paralelizaciji obrade. Real-ESRGAN podržava batch processing kroz moderne deep learning frameworke (PyTorch, TensorFlow), ali kod velikih batchova (>1000 slika) može doći do problema s VRAM-om i I/O bottleneckovima [28]. Rješenja uključuju dinamično određivanje veličine batcha sukladno dostupnoj memoriji, asinkrono čitanje i pisanje podataka te “lazy loading” mehanizme.

Paralelna obrada više instanci na istom GPU-u može dovesti do zagušenja resursa, posebno na slabijim karticama. Preporučuje se korištenje GPU scheduling-a, npr. NVIDIA CUDA streamova, ili ograničavanje simultanih procesa. Za distribuciju zadataka na više servera ili cloud okruženja, korisno je primijeniti task queue sustave (Celery, Redis Queue) kako bi se kontroliralo dohvaćanje i spremanje rezultata [28].

Primjena Real-ESRGAN modela na video i nestandardne formate pokazuje da pravilna kombinacija optimizacija i prilagodbe workflowa omogućuje održavanje visoke vizualne kvalitete i prostorne konzistencije frameova. Unatoč povećanim zahtjevima za memorijom i procesorskom snagom, integracija batch obrade, model quantization i specijaliziranih konverzijskih alata osigurava pouzdanu izvedbu u stvarnim scenarijima. Ove strategije rezultiraju stabilnim i predvidljivim rezultatima, omogućujući upotrebu Real-ESRGAN-a u različitim okruženjima, od klasičnih slikovnih datasetova do video produkcije i specijalnih aplikacija.

8. ZAKLJUČAK

Primjena modela Real-ESRGAN pokazala je da suvremene metode dubokog učenja predstavljaju značajan iskorak u odnosu na tradicionalne tehnike povećanja rezolucije slike. Zahvaljujući arhitekturi koja kombinira RRDB generator i U-Net diskriminator sa spektralnom normalizacijom, omogućena je rekonstrukcija vizualnog sadržaja visoke perceptivne vjernosti, s naglašenim detaljima i prirodnijim teksturama. Testiranja provedena na različitim tipovima sadržaja potvrdila su da model pouzdano uklanja šum i artefakte te vraća uvjerljive detalje, čime se ističe njegova svestranost u obradi portreta, pejzaža, povijesnih i oštećenih fotografija, tehničkih ilustracija i multimedijskih materijala. Takva prilagodljivost čini Real-ESRGAN posebno korisnim u područjima poput digitalne restauracije, grafičke i multimedijske produkcije, očuvanja kulturne baštine, pa čak i u domenama znanstvene i tehničke vizualizacije.

Unatoč prednostima, rezultati također ukazuju na određena ograničenja. Model ne rekonstruira sliku matematičkom točnošću nego generira detalje prema uzorcima naučenima tijekom treniranja, što može dovesti do tzv. “haluciniranih” struktura koje ne postoje u izvornom zapisu. Iako to u većini slučajeva poboljšava estetsku kvalitetu slike, u kontekstu forenzike, medicinske dijagnostike ili sigurnosnih aplikacija može biti problematično. Dodatno, visoka računalna zahtjevnost i dulje vrijeme obrade predstavljaju prepreku široj i realno-vremenskoj primjeni, što upućuje na potrebu optimizacije arhitekture i razvoja učinkovitijih varijanti prilagođenih mobilnim i ugrađenim sustavima.

Real-ESRGAN se stoga pokazuje kao slojevit alat koji je sposoban unaprijediti estetske standarde, rasteretiti kreativne procese i povećati komunikacijsku vrijednost vizualnih sadržaja, ali istodobno otvara pitanja vjerodostojnosti i odgovorne uporabe. Njegova vrijednost leži u spoju tehničke naprednosti i perceptivne uvjerljivosti, što ga čini ključnim doprinosom razvoju grafičkog inženjerstva, vizualnih komunikacija i računalnog vida. Zaključno, Real-ESRGAN ne samo da podiže razinu kvalitete digitalne vizualne obrade nego i pokazuje smjer u kojem će se razvijati budući standardi u obradi slike, uz nužnost daljnjeg istraživanja i etički promišljene primjene.

LITERATURA

[1] Jozdani, S., Chen, D., Pouliot, D., Johnson, B. A. (2022). A review and meta-analysis of Generative Adversarial Networks and their applications in remote sensing. *International Journal of Applied Earth Observation and Geoinformation*, 108(1), 102734, str. 1-22.

[2] Joshi, D., Jana, A., Lone, H., Taru, V., Thorat, S. (2025). Image and Video Upscaling Using Real-ESRGAN. *Journal Publications of Internationl Research for Engineering and Management (JOIREM)*, 5(4), str. 1-5.

[3] Bharambe, S., Sable, S., Chavan, Y., Rai, A., Mahajan, A. (2025). A Survey on Image Quality Enhancement using Real-ESRGAN: A Comprehensive Review of Datasets and Approaches. *International Journal of Innovative Research in Technology (IJIRT)*, 11(11), 176026, str. 6436-6441.

[4] Tian, C., Zhang, X., Zhu, Q., Zhang, B., Chun-Wei Lin, J. (2024). Generative Adversarial Networks for Image Super-Resolution: A Survey. *Electrical Engineering and Systems Science*, 1(1), str. 1-31.

[5] Kumari, M. (2021). An Overview on Deep Learning in Image Super-Resolution for Advanced Machine Vision System. *Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC 2021)*. 6.-7.8.2021., Bangalore, India.

[6] Aggarwal, A., Mittal, M., Battineni, G. (2021). Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, 1(1), 100004, str. 1-9.

[7] Zhao, J., Hou, X., Pan, M., Zhang, H. (2022). Attention-based generative adversarial network in medical imaging: A narrative review. *Computers in Biology and Medicine*, 149(2), 105948.

[8] Muhammad, W., Aramvith, S., Onoye, T. (2024). CANS: Combined Attention Network for Single Image Super-Resolution. *IEEE Access*, 12(1), str. 167498-167517.

[9] Neves, M. C., Filgueiras, J., Kokkinogenis, Z., Silva, M. C. F., Campos J. B. L. M., Reis, L. P. (2024). Enhancing experimental image quality in two-phase bubbly systems with super-resolution using generative adversarial networks. *International Journal of Multiphase Flow*, 180(1), 104952, str. 1-14.

[10] Kędzierski, J., Rusiecki, A. (2024). Deep Learning Approach to Improve Image Super-Resolution with Aliasing. *Procedia Computer Sciences*, 246(1), str. 1250-1259.

[11] Alotaibi, A., Ahmed, M. (2025). Neural Architecture Search for Generative Adversarial Networks: A Comprehensive Review and Critical Analysis. *Applied Sciences*, 15(7), 3623, str. 1-31.

[12] Li, S., Gao, L., Zhang, B., Liu, Z., Zhang, X., Guan, L., Zheng, J. (2025). Research on Super-Resolution Reconstruction of Coarse Aggregate Particle Images for Earth–Rock Dam Construction Based on Real-ESRGAN. *Sensors*, 25(13), 4084, str. 1-31.

[13] Mistry, A., Shetty, S., Nagar, K., Avhad, R., Gajare, J. (2025). Image and Video Upscaling Using Real-ESRGAN. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 5(10), str. 547-554.

[14] Wang, X., Xie, L, Dong, C., Shan, Y. (2021). Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 11.-17.10.2021., Montreal, BC, Kanada.

[15] Zhao, Z., Gao, Y., Min, B., Miao, X., Hu, J., Liu, Y., Pharksuwan, K. (2025). PConvSRGAN: Real-world super-resolution reconstruction with pure convolutional networks. *Computer Vision and Image Understanding*, 260(1), 104465, str. 1-11.

[16] Sreelakshmy, I. J., Kovoor, Binsu C. (2022). Generative Inpainting of High-resolution Images: Redefined with Real-ESRGAN. *International Journal on Artificial Intelligence Tools*, 31(5), 2250035.

[17] Hu, Y., Zhou, Y., Ma, H., Yang, H., Wang, H., Zhang, S., Li, X. (2025). Wavelet-based dual discriminator GAN for image super-resolution. *Knowledge-Based Systems*, 317(1), 113383.

[18] Zhu, Z., Lei, Y., Qin, Y, Zhu, C., Zhu, Y. (2023). IRE: Improved Image Super-Resolution Based on Real-ESRGAN. *IEEE Access*, 11(1), str. 45334-45348.

[19] Senthil, A., Tomlinson, L. (2025). Enhancing Super-Resolution Models: A Comparative Analysis of ESRGAN, Real-ESRGAN, and AESRGAN. *Proceedings of the 7th International Conference on Image Processing and Machine Vision (IPMV 2025)*. 10.-12.1.2025., Hong Kong, Kina.

[20] Real-ESRGAN: models, datasets, tests and scripts for Python. (2025). Dostupno na: https://huggingface.co/spaces/akhaliq/RealESRGAN/commit/810c8ea8b0684f9a069421062d4d8cd0ffdfc6f3?cv=1 (Pristupljeno 22.7.2025.)

[21] Real-ESRGAN: Project Template for Python. (2025.) Dostupno na: https://github.com/xinntao/Real-ESRGAN (Pristupljeno 22.7.2025.)

[22] Real-ESRGAN Windows. (2025). Dostupno na: https://github.com/bycloudai/Real-ESRGAN-Windows (Pristupljeno 22.7.2025.)

[23] Andriadi, K., Zarlis, M., Heryadi, Y., Chowanda, A. (2024). Evaluation of REAL-ESRGAN Using Different Types of Image Degradation. *Proceedings of the 2024 International Conference on ICT for Smart Society (ICISS)*, 4.-5.9.2024., Yogyakarta, Indonezija.

[24] Guo, M., Zhang, Z., Liu, H., Huang, Y. (2022). NDSRGAN: A Novel Dense Generative Adversarial Network for Real Aerial Imagery Super-Resolution Reconstruction. *Remote Sensing*, 14(7), 1574, str. 1-23.

[25] Hu, T., Chen, Z., Wang, M., Hou, X., Lu, X., Pan, Y., Li, J. (2024). Global sparse attention network for remote sensing image super-resolution. *Knowledge-Based Systems*, 304(01), 112448.

[26] Bellili, S., Ouamane, A., Chouchane, A., Himeur, Y., Atalla, S., Mansoor, W., Bourennane, S., Bensaali, F. (2025). Tensor-driven face recognition: Integrating super-resolution and multilinear subspace learning for low-resolution images. *Information Fusion*, 120(1), 103075.

[27] López-Tapia, S., Lucas, A., Molina, R., Katsaggelos, A. K. (2020). A single video super-resolution GAN for multiple downsampling operators based on pseudo-inverse image formation models. *Digital Signal Processing*, 104(1), 102801.

[28] Song, J., Yi, H., Xu, W., Li, X., Li, B., Liu, Y. (2023). ESRGAN-DP: Enhanced super-resolution generative adversarial network with adaptive dual perceptual loss. *Heliyon*, 9(4), e15134, str. 1-14.

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Marko Butula: Hibridni pristupi generativne suparničke super-rezolucije: arhitektonska načela, optimizacija i primjena Real-ESRGAN dubokih neuronskih mreža u digitalnoj vizualizaciji

U ovom radu istražena je primjena dubokih generativnih neuronskih mreža u području nadogradnje rezolucije digitalnih slika, s naglaskom na arhitekturu i funkcionalnost modela Real-ESRGAN. Model je razvijen kao nadogradnja ESRGAN-a kroz složenije modele degradacije, poboljšani diskriminator i modificirane funkcije gubitka, a cilj rada bio je analizirati njegova arhitektonska rješenja, implementaciju i praktičnu primjenu u rekonstrukciji degradiranih slika.

Teorijski okvir obuhvatio je pregled metoda super-rezolucije, ograničenja tradicionalnih pristupa i prednosti GAN arhitektura. Posebno je razmotren U-Net diskriminator sa spektralnom normalizacijom za istodobnu analizu lokalnih i globalnih značajki te generator temeljen na RRDB blokovima koji omogućuje rekonstrukciju finih detalja.

Metodološki, Real-ESRGAN je implementiran u Pythonu korištenjem biblioteka PyTorch, NumPy, OpenCV, Pillow i Matplotlib, dok je testiranje provedeno na Apple Mac Mini platformi s M4 čipom. U radu su prikazani kodni isječci i ključna implementacijska rješenja, a cjeloviti kod priložen je u dodatku.

Rezultati pokazuju da Real-ESRGAN nadmašuje prethodne modele u rekonstrukciji slika s degradacijama poput šuma, zamućenja i kompresijskih artefakata. Vizualne usporedbe i metrike poput NIQE potvrđuju bolju očuvanost detalja i perceptivnu kvalitetu. Model je pokazao potencijal u restauraciji arhivskih fotografija, medicinskoj obradi, poboljšanju video sadržaja i forenzici.

Zaključno, Real-ESRGAN predstavlja značajan napredak u području računalnog vida, uz ograničenja vezana uz moguće halucinacije detalja i visoke računalne zahtjeve. Buduća istraživanja trebala bi se usmjeriti na optimizaciju modela i prilagodbu specifičnim domenama radi povećanja robusnosti i praktične primjenjivosti.

Ključne riječi: Real-ESRGAN, nadogradnja slika, super-rezolucija, generativne suparničke mreže, računalni vid.

ABSTRACT

Marko Butula: Generative Adverbial Super-Resolution Hybrid Approaches: Architectural Principles, Optimization and Application of Real-ESRGAN Deep Neural Networks in Digital Visualization

This paper investigates the application of deep generative neural networks in the field of digital image super-resolution, with a focus on the architecture and functionality of the Real-ESRGAN model. Developed as an upgrade of ESRGAN, the model introduces more complex degradation models, an improved discriminator, and modified loss functions. The aim of the study was to analyze the architectural design of Real-ESRGAN, its implementation, and its practical use in reconstructing degraded images.

The theoretical framework included an overview of existing super-resolution methods, the limitations of traditional approaches, and the advantages of GAN-based architectures. Particular attention was given to the U-Net discriminator with spectral normalization, which enables simultaneous analysis of local and global image features, and to the generator based on RRDB blocks, designed to reconstruct fine textures and details.

Methodologically, Real-ESRGAN was implemented in Python using the PyTorch, NumPy, OpenCV, Pillow, and Matplotlib libraries, while testing was performed on an Apple Mac Mini with an M4 chip. The paper presents code snippets and key implementation details, with the complete code provided in the appendix.

The results demonstrate that Real-ESRGAN outperforms previous models in reconstructing images affected by noise, blur, and compression artifacts. Visual comparisons and metrics such as NIQE confirm its superior ability to preserve details and enhance perceptual quality. The model has shown potential in archival photo restoration, medical imaging, video enhancement, and forensic applications.

In conclusion, Real-ESRGAN represents a significant advancement in computer vision, although it remains limited by the risk of detail hallucination and high computational demands. Future research should focus on optimizing the model and adapting it to domain-specific data in order to further increase its robustness and applicability.

Keywords: Real-ESRGAN, image enhancement, super-resolution, generative adversarial networks, computer vision.

PRILOG

PROGRAMSKI KOD

DISCRIMINATOR_ARCH.py

```
from basicsr.utils.registry import ARCH_REGISTRY
from torch import nn as nn
from torch.nn import functional as F
from torch.nn.utils import spectral_norm


@ARCH_REGISTRY.register()
class UNetDiscriminatorSN(nn.Module):
    """Defines a U-Net discriminator with spectral normalization (SN)

    It is used in Real-ESRGAN: Training Real-World Blind Super-
Resolution with Pure Synthetic Data.

    Arg:
        num_in_ch (int): Channel number of inputs. Default: 3.
        num_feat (int): Channel number of base intermediate features.
Default: 64.
        skip_connection (bool): Whether to use skip connections between
U-Net. Default: True.
    """

    def __init__(self, num_in_ch, num_feat=64, skip_connection=True):
        super(UNetDiscriminatorSN, self).__init__()
        self.skip_connection = skip_connection
        norm = spectral_norm
        # the first convolution
        self.conv0 = nn.Conv2d(num_in_ch, num_feat, kernel_size=3,
stride=1, padding=1)
        # downsample
        self.conv1 = norm(nn.Conv2d(num_feat, num_feat * 2, 4, 2, 1,
bias=False))
        self.conv2 = norm(nn.Conv2d(num_feat * 2, num_feat * 4, 4, 2,
1, bias=False))
        self.conv3 = norm(nn.Conv2d(num_feat * 4, num_feat * 8, 4, 2,
1, bias=False))
        # upsample
        self.conv4 = norm(nn.Conv2d(num_feat * 8, num_feat * 4, 3, 1,
1, bias=False))
        self.conv5 = norm(nn.Conv2d(num_feat * 4, num_feat * 2, 3, 1,
1, bias=False))
        self.conv6 = norm(nn.Conv2d(num_feat * 2, num_feat, 3, 1, 1,
bias=False))
        # extra convolutions
        self.conv7 = norm(nn.Conv2d(num_feat, num_feat, 3, 1, 1,
bias=False))
        self.conv8 = norm(nn.Conv2d(num_feat, num_feat, 3, 1, 1,
bias=False))
        self.conv9 = nn.Conv2d(num_feat, 1, 3, 1, 1)


    def forward(self, x):
```

```
        # downsample
        x0 = F.leaky_relu(self.conv0(x), negative_slope=0.2,
inplace=True)
        x1 = F.leaky_relu(self.conv1(x0), negative_slope=0.2,
inplace=True)
        x2 = F.leaky_relu(self.conv2(x1), negative_slope=0.2,
inplace=True)
        x3 = F.leaky_relu(self.conv3(x2), negative_slope=0.2,
inplace=True)

        # upsample
        x3 = F.interpolate(x3, scale_factor=2, mode='bilinear',
align_corners=False)
        x4 = F.leaky_relu(self.conv4(x3), negative_slope=0.2,
inplace=True)

        if self.skip_connection:
            x4 = x4 + x2
        x4 = F.interpolate(x4, scale_factor=2, mode='bilinear',
align_corners=False)
        x5 = F.leaky_relu(self.conv5(x4), negative_slope=0.2,
inplace=True)

        if self.skip_connection:
            x5 = x5 + x1
        x5 = F.interpolate(x5, scale_factor=2, mode='bilinear',
align_corners=False)
        x6 = F.leaky_relu(self.conv6(x5), negative_slope=0.2,
inplace=True)

        if self.skip_connection:
            x6 = x6 + x0

        # extra convolutions
        out = F.leaky_relu(self.conv7(x6), negative_slope=0.2,
inplace=True)
        out = F.leaky_relu(self.conv8(out), negative_slope=0.2,
inplace=True)
        out = self.conv9(out)

        return out
```

UTILS.py

```
import cv2
import math
import numpy as np
import os
import queue
import threading
import torch
from basicsr.utils.download_util import load_file_from_url
from torch.nn import functional as F

ROOT_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

```
class RealESRGANer():
    """A helper class for upsampling images with RealESRGAN.

    Args:
        scale (int): Upsampling scale factor used in the networks. It
is usually 2 or 4.
        model_path (str): The path to the pretrained model. It can be
urls (will first download it automatically).
        model (nn.Module): The defined network. Default: None.
        tile (int): As too large images result in the out of GPU memory
issue, so this tile option will first crop
            input images into tiles, and then process each of them.
Finally, they will be merged into one image.
            0 denotes for do not use tile. Default: 0.
        tile_pad (int): The pad size for each tile, to remove border
artifacts. Default: 10.
        pre_pad (int): Pad the input images to avoid border artifacts.
Default: 10.
        half (float): Whether to use half precision during inference.
Default: False.
    """

    def __init__(self,
                 scale,
                 model_path,
                 dni_weight=None,
                 model=None,
                 tile=0,
                 tile_pad=10,
                 pre_pad=10,
                 half=False,
                 device=None,
                 gpu_id=None):
        self.scale = scale
        self.tile_size = tile
        self.tile_pad = tile_pad
        self.pre_pad = pre_pad
        self.mod_scale = None
        self.half = half

        # initialize model
        if gpu_id:
            self.device = torch.device(
                f'cuda:{gpu_id}' if torch.cuda.is_available() else
'cpu') if device is None else device
        else:
            self.device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu') if device is None else device

        if isinstance(model_path, list):
            # dni
            assert len(model_path) == len(dni_weight), 'model_path and
dni_weight should have the save length.'
            loadnet = self.dni(model_path[0], model_path[1],
dni_weight)
        else:
            # if the model_path starts with https, it will first
download models to the folder: weights
```

```
            if model_path.startswith('https://'):
                model_path = load_file_from_url(
                    url=model_path, model_dir=os.path.join(ROOT_DIR,
'weights'), progress=True, file_name=None)
            loadnet = torch.load(model_path,
map_location=torch.device('cpu'))

        # prefer to use params_ema
        if 'params_ema' in loadnet:
            keyname = 'params_ema'
        else:
            keyname = 'params'
        model.load_state_dict(loadnet[keyname], strict=True)

        model.eval()
        self.model = model.to(self.device)
        if self.half:
            self.model = self.model.half()

    def dni(self, net_a, net_b, dni_weight, key='params', loc='cpu'):
        """Deep network interpolation.

        ``Paper: Deep Network Interpolation for Continuous Imagery
Effect Transition``
        """
        net_a = torch.load(net_a, map_location=torch.device(loc))
        net_b = torch.load(net_b, map_location=torch.device(loc))
        for k, v_a in net_a[key].items():
            net_a[key][k] = dni_weight[0] * v_a + dni_weight[1] *
net_b[key][k]
        return net_a

    def pre_process(self, img):
        """Pre-process, such as pre-pad and mod pad, so that the images
can be divisible
        """
        img = torch.from_numpy(np.transpose(img, (2, 0, 1))).float()
        self.img = img.unsqueeze(0).to(self.device)
        if self.half:
            self.img = self.img.half()

        # pre_pad
        if self.pre_pad != 0:
            self.img = F.pad(self.img, (0, self.pre_pad, 0,
self.pre_pad), 'reflect')
        # mod pad for divisible borders
        if self.scale == 2:
            self.mod_scale = 2
        elif self.scale == 1:
            self.mod_scale = 4
        if self.mod_scale is not None:
            self.mod_pad_h, self.mod_pad_w = 0, 0
            _, _, h, w = self.img.size()
            if (h % self.mod_scale != 0):
                self.mod_pad_h = (self.mod_scale - h % self.mod_scale)
            if (w % self.mod_scale != 0):
                self.mod_pad_w = (self.mod_scale - w % self.mod_scale)
```

```
            self.img = F.pad(self.img, (0, self.mod_pad_w, 0,
self.mod_pad_h), 'reflect')

    def process(self):
        # model inference
        self.output = self.model(self.img)

    def tile_process(self):
        """It will first crop input images to tiles, and then process
each tile.
        Finally, all the processed tiles are merged into one images.

        Modified from: https://github.com/ata4/esrgan-launcher
        """
        batch, channel, height, width = self.img.shape
        output_height = height * self.scale
        output_width = width * self.scale
        output_shape = (batch, channel, output_height, output_width)

        # start with black image
        self.output = self.img.new_zeros(output_shape)
        tiles_x = math.ceil(width / self.tile_size)
        tiles_y = math.ceil(height / self.tile_size)

        # loop over all tiles
        for y in range(tiles_y):
            for x in range(tiles_x):
                # extract tile from input image
                ofs_x = x * self.tile_size
                ofs_y = y * self.tile_size
                # input tile area on total image
                input_start_x = ofs_x
                input_end_x = min(ofs_x + self.tile_size, width)
                input_start_y = ofs_y
                input_end_y = min(ofs_y + self.tile_size, height)

                # input tile area on total image with padding
                input_start_x_pad = max(input_start_x - self.tile_pad,
0)
                input_end_x_pad = min(input_end_x + self.tile_pad,
width)
                input_start_y_pad = max(input_start_y - self.tile_pad,
0)
                input_end_y_pad = min(input_end_y + self.tile_pad,
height)

                # input tile dimensions
                input_tile_width = input_end_x - input_start_x
                input_tile_height = input_end_y - input_start_y
                tile_idx = y * tiles_x + x + 1
                input_tile = self.img[:, :,
input_start_y_pad:input_end_y_pad, input_start_x_pad:input_end_x_pad]

                # upscale tile
                try:
                    with torch.no_grad():
                        output_tile = self.model(input_tile)
                except RuntimeError as error:
```

```
                    print('Error', error)
                print(f'\tTile {tile_idx}/{tiles_x * tiles_y}')

                # output tile area on total image
                output_start_x = input_start_x * self.scale
                output_end_x = input_end_x * self.scale
                output_start_y = input_start_y * self.scale
                output_end_y = input_end_y * self.scale

                # output tile area without padding
                output_start_x_tile = (input_start_x -
input_start_x_pad) * self.scale
                output_end_x_tile = output_start_x_tile +
input_tile_width * self.scale
                output_start_y_tile = (input_start_y -
input_start_y_pad) * self.scale
                output_end_y_tile = output_start_y_tile +
input_tile_height * self.scale

                # put tile into output image
                self.output[:, :, output_start_y:output_end_y,
                            output_start_x:output_end_x] =
output_tile[:, :, output_start_y_tile:output_end_y_tile,

output_start_x_tile:output_end_x_tile]

    def post_process(self):
        # remove extra pad
        if self.mod_scale is not None:
            _, _, h, w = self.output.size()
            self.output = self.output[:, :, 0:h - self.mod_pad_h *
self.scale, 0:w - self.mod_pad_w * self.scale]
        # remove prepad
        if self.pre_pad != 0:
            _, _, h, w = self.output.size()
            self.output = self.output[:, :, 0:h - self.pre_pad *
self.scale, 0:w - self.pre_pad * self.scale]
        return self.output

    @torch.no_grad()
    def enhance(self, img, outscale=None,
alpha_upsampler='realesrgan'):
        h_input, w_input = img.shape[0:2]
        # img: numpy
        img = img.astype(np.float32)
        if np.max(img) > 256:  # 16-bit image
            max_range = 65535
            print('\tInput is a 16-bit image')
        else:
            max_range = 255
        img = img / max_range
        if len(img.shape) == 2:  # gray image
            img_mode = 'L'
            img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
        elif img.shape[2] == 4:  # RGBA image with alpha channel
            img_mode = 'RGBA'
            alpha = img[:, :, 3]
            img = img[:, :, 0:3]
```

```
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            if alpha_upsampler == 'realesrgan':
                alpha = cv2.cvtColor(alpha, cv2.COLOR_GRAY2RGB)
        else:
            img_mode = 'RGB'
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # ------------------- process image (without the alpha channel)
------------------- #
        self.pre_process(img)
        if self.tile_size > 0:
            self.tile_process()
        else:
            self.process()
        output_img = self.post_process()
        output_img = output_img.data.squeeze().float().cpu().clamp_(0,
1).numpy()
        output_img = np.transpose(output_img[[2, 1, 0], :, :], (1, 2,
0))
        if img_mode == 'L':
            output_img = cv2.cvtColor(output_img, cv2.COLOR_BGR2GRAY)

        # ------------------- process the alpha channel if necessary --
----------------- #
        if img_mode == 'RGBA':
            if alpha_upsampler == 'realesrgan':
                self.pre_process(alpha)
                if self.tile_size > 0:
                    self.tile_process()
                else:
                    self.process()
                output_alpha = self.post_process()
                output_alpha =
output_alpha.data.squeeze().float().cpu().clamp_(0, 1).numpy()
                output_alpha = np.transpose(output_alpha[[2, 1, 0], :,
:], (1, 2, 0))
                output_alpha = cv2.cvtColor(output_alpha,
cv2.COLOR_BGR2GRAY)
            else:  # use the cv2 resize for alpha channel
                h, w = alpha.shape[0:2]
                output_alpha = cv2.resize(alpha, (w * self.scale, h *
self.scale), interpolation=cv2.INTER_LINEAR)

            # merge the alpha channel
            output_img = cv2.cvtColor(output_img, cv2.COLOR_BGR2BGRA)
            output_img[:, :, 3] = output_alpha

        # ------------------------------ return ----------------------
------- #
        if max_range == 65535:  # 16-bit image
            output = (output_img * 65535.0).round().astype(np.uint16)
        else:
            output = (output_img * 255.0).round().astype(np.uint8)

        if outscale is not None and outscale != float(self.scale):
            output = cv2.resize(
                output, (
                    int(w_input * outscale),
```

```
                    int(h_input * outscale),
                ), interpolation=cv2.INTER_LANCZOS4)

        return output, img_mode


class PrefetchReader(threading.Thread):
    """Prefetch images.

    Args:
        img_list (list[str]): A image list of image paths to be read.
        num_prefetch_queue (int): Number of prefetch queue.
    """

    def __init__(self, img_list, num_prefetch_queue):
        super().__init__()
        self.que = queue.Queue(num_prefetch_queue)
        self.img_list = img_list

    def run(self):
        for img_path in self.img_list:
            img = cv2.imread(img_path, cv2.IMREAD_UNCHANGED)
            self.que.put(img)

        self.que.put(None)

    def __next__(self):
        next_item = self.que.get()
        if next_item is None:
            raise StopIteration
        return next_item

    def __iter__(self):
        return self


class IOConsumer(threading.Thread):

    def __init__(self, opt, que, qid):
        super().__init__()
        self._queue = que
        self.qid = qid
        self.opt = opt

    def run(self):
        while True:
            msg = self._queue.get()
            if isinstance(msg, str) and msg == 'quit':
                break

            output = msg['output']
            save_path = msg['save_path']
            cv2.imwrite(save_path, output)
        print(f'IO worker {self.qid} is done.')
```

SRVGG_ARCH.py

```
from basicsr.utils.registry import ARCH_REGISTRY
from torch import nn as nn
from torch.nn import functional as F


@ARCH_REGISTRY.register()
class SRVGGNetCompact(nn.Module):
    """A compact VGG-style network structure for super-resolution.

    It is a compact network structure, which performs upsampling in the
last layer and no convolution is
    conducted on the HR feature space.

    Args:
        num_in_ch (int): Channel number of inputs. Default: 3.
        num_out_ch (int): Channel number of outputs. Default: 3.
        num_feat (int): Channel number of intermediate features.
Default: 64.
        num_conv (int): Number of convolution layers in the body
network. Default: 16.
        upscale (int): Upsampling factor. Default: 4.
        act_type (str): Activation type, options: 'relu', 'prelu',
'leakyrelu'. Default: prelu.
    """

    def __init__(self, num_in_ch=3, num_out_ch=3, num_feat=64,
num_conv=16, upscale=4, act_type='prelu'):
        super(SRVGGNetCompact, self).__init__()
        self.num_in_ch = num_in_ch
        self.num_out_ch = num_out_ch
        self.num_feat = num_feat
        self.num_conv = num_conv
        self.upscale = upscale
        self.act_type = act_type

        self.body = nn.ModuleList()
        # the first conv
        self.body.append(nn.Conv2d(num_in_ch, num_feat, 3, 1, 1))
        # the first activation
        if act_type == 'relu':
            activation = nn.ReLU(inplace=True)
        elif act_type == 'prelu':
            activation = nn.PReLU(num_parameters=num_feat)
        elif act_type == 'leakyrelu':
            activation = nn.LeakyReLU(negative_slope=0.1, inplace=True)
        self.body.append(activation)

        # the body structure
        for _ in range(num_conv):
            self.body.append(nn.Conv2d(num_feat, num_feat, 3, 1, 1))
            # activation
            if act_type == 'relu':
                activation = nn.ReLU(inplace=True)
            elif act_type == 'prelu':
                activation = nn.PReLU(num_parameters=num_feat)
            elif act_type == 'leakyrelu':
```

```
                activation = nn.LeakyReLU(negative_slope=0.1,
inplace=True)
            self.body.append(activation)

        # the last conv
        self.body.append(nn.Conv2d(num_feat, num_out_ch * upscale *
upscale, 3, 1, 1))
        # upsample
        self.upsampler = nn.PixelShuffle(upscale)

    def forward(self, x):
        out = x
        for i in range(0, len(self.body)):
            out = self.body[i](out)

        out = self.upsampler(out)
        # add the nearest upsampled image, so that the network learns
the residual
        base = F.interpolate(x, scale_factor=self.upscale,
mode='nearest')
        out += base
        return out
```

ŽIVOTOPIS

Marko Butula rođen je 1. travnja 2001. godine u Zagrebu. Osnovno obrazovanje stekao je u Osnovnoj školi Jordanovac, a srednjoškolsko u Školi za grafiku, dizajn i medijsku produkciju, smjer Web Design. Od ranih dana pokazivao je interes za dizajn i tehnologiju, što ga je potaknulo na razvoj vještina u području grafike i programiranja. Autor je završnog rada pod nazivom "Povijesni razvoj programa Adobe Photoshop". Trenutno je student pete godine diplomskog studija na Grafičkom fakultetu Sveučilišta u Zagrebu, smjer Tehničko-tehnološki, gdje usavršava znanja u programiranju, dizajnu i primjeni suvremenih tehnologija.

Kao autor CarRentalApp aplikacije za iOS, dodatno je proširio svoje praktične vještine u mobilnom razvoju, pokazujući sposobnost kreiranja funkcionalnih i suvremenih mobilnih rješenja u programskom jeziku Swift. U okviru freelance grafičkog dizajna radio je za različite klijente, među kojima su istaknuti Asmir Begović, Porsche Digital i drugi, što potvrđuje njegovo iskustvo u kreativnom sektoru.

Za svoj sportski angažman osvojio je Dekanovu nagradu za 2. mjesto postignuto u veslačkoj sekciji Fakulteta. Ima praktično iskustvo kao database administrator i freelance grafički dizajner, s portfeljem radova u digitalnom i grafičkom dizajnu. Motiviran je daljnjim profesionalnim razvojem u mobilnom razvoju i dizajnu u kontekstu umjetne inteligencije.