Potraga za novim česticama pomoću tehnika strojnog učenja

Hadži Veljković, Tin

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Tin Hadži Veljković

Potraga za novim česticama pomoću tehnika strojnog učenja

Diplomski rad

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA; SMJER ISTRAŽIVAČKI

Tin Hadži Veljković

Diplomski rad

Potraga za novim česticama pomoću tehnika strojnog učenja

Voditelj diplomskog rada: Izv. prof. dr. sc. Nikola Poljak

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2021.

Sažetak

U ovom radu prestavljene su dvije metode za detekciju novih čestica na sudarivačima poput LHC-a. Obje metode se zasnivaju na uporabi neuralnih mreža, točnije autoenkodera i U-Net mreže. Podaci korišteni u ovom radu dani su u sklopu natjecanja LHC Olympics 2020. Dobivena su dva skupa podataka: podaci koji ne sadrže nove čestice, odnosno opisuju pozadinu i podaci koji sadrže pozadinu i nepoznat broj novih čestica. Treniranjem autoenkoderskih modela na podacima koji sadrže samo pozadinu pokušali smo naučiti njene bitne značajke. Nadalje, evaluacijom modela na drugim podacima pokušali smo detektirati nove čestice tražeći događaje sa lošom autoenkoderskom rekonstrukcijom. Ako događaji s novim česticama odstupaju od pozadine, onda se očekuje da će ih autoenkoder gore rekonstruirati. Uporabom U-Net mreže pokušali smo naučiti model da oponaša anti-k_T algoritam grupiranja čestica u mlazove te detektirati nove čestice koristeći istu metodologiju kao i za autoenkoderske modele.

Ključne riječi: Fizika izvan standardnog modela, eksperimentalna fizika čestica, strojno učenje, autoenkoderi, U-Net mreža

The search for new particles with the use of machine learning techniques

Abstract

In this work we will present the use of machine learning techniques for detecting new particles. Two different types of neural networks were used, namely autoencoders and U-Net. The data used was provided as a part of the competition LHC Olympics 2020. We were given two datasets: a dataset containing only background events and a dataset containing an unknown number of new particles along with the background events. Autoencoder models were trained using the first dataset with the goal of learning crucial background information via *latent variables*. The new particles are considered as anomalies, and therefore we expect the model to poorly reconstruct the data when evaluated on the events containing new particles. The U-Net model was used to learn the anti-k_T jet clustering algorithm, and anomaly detection was conducted using the same methodology as for the autoencoders.

Keywords: Physics beyond the Standard Model, experimental Particle Physics, machine learning, autoencoders, U-Net

Sadržaj

1 Uvod

Standardni model opisuje poznate elementarne čestice i njihove interakcije. Kao i svaki model, ima svoje prednosti, no postoje i nekonzistentnosti i problemi u njegovoj formulaciji. Iz tog razloga, fizika izvan standardnog modela je aktivno istraživana tema. Fizika izvan standardnog modela sadrži puno teorija koje u svojoj formulaciji sadrže nove čestice te se eksperimentima pokušava utvrditi postojanje istih. Eksperimenti poput sudaranja protona koji se provode na LHC-u (Large Hardon Collider) u CERN-u dosežu luminozitet koji može biti reda veličine 10^{35} cm^{-2} s^{-1}. Čestice proizvedene u takvim sudarima stvaraju pozadinu zbog koje je teško detektirati nove ili rijetke signale te je separacija signala od pozadine kompleksan problem za čije rješavanje postoje mnoge klasične metode. Danas, razvojem novih algoritama i hardware-a, neuralne mreže pokazale su se kao efikasan način za rješavanje problema separacije.

U sklopu natjecanja LHC Olympics [1] dana su dva skupa podataka. Prva skupina ne sadrži nove čestice te služi za opis pozadine, dok druga skupina sadrži i nove čestice i pozadinu. Događaje smo prikazali u obliku slika, stoga su korištene neuralne mreže konvolucijskog tipa koje su prigodnije za analizu ovakvog tipa podataka.

Prvi tip neuralnih mreža koje implementiramo je autoenkoder. Glavna ideja ovakve mreže je da se podaci mapiraju u prostor manjih dimenzija (*latentan prostor*) koristeći funkciju zvanu enkoder te se iz kompresirane reprezentacije dekoderom pokušavaju rekonstruirati originalni podaci. Takvom strukturom mreža je primorana naučiti bitne značajke podataka kako bi ih uspješno rekonstruirala. U ovom radu koristimo autoenkoder za ekstrakciju značajki pozadine te se evaluacijom modela na podacima koji sadrže nove čestice traže anomalije. U svrhu učenja anti-k_T algoritma grupiranja koristi se U-Net mreža koja se trenira na podacima koji sadrže samo pozadinu. Očekuje se da će ovakav model lošije grupirati nove čestice koje ne pripadaju pozadini te tako tražimo anomalije.

U drugom poglavlju predstavljeni su najbitniji koncepti kvantne kromodinamike potrebni za razumijevanje provedene analize. U trećem poglavlju opisan je princip rada detektora čestica na raznim sudarivačima te je detaljno opisan rad CMS detektora. Pošto su mlazovi vrlo čest objekt u analizi sudara, objašnjeni su najbitniji algoritmi grupiranja čestica u mlazove. U četvrtom poglavlju predstavljeni su

osnovni koncepti neuralnih mreža i metoda učenja. Kako su nam podaci u tenzorskom obliku (slike), opisan je rad konvolucijskih neuralnih mreža te je detaljnije opisan rad autoenkodera. U petom poglavlju predstavljena je analiza podataka te su detaljno opisane arhitekture korištenih modela. Konačno, pikazani su rezultati evaluacije različitih autoenkoderskih i U-Net modela u svrhu detekcije novih čestica.

2 Kvantna kromodinamika

Sredinom 20. stoljeća u eksperimentalnoj fizici pojavile su se mnoge inovacije, poput komore na iskre uz pomoć kojih se brzo otkrilo mnogo hadrona. Puno njih imalo je slična svojstva i mase, što je ukazivalo da potencijalno nisu elementarne čestice. Fizičari poput Gell-Manna i Georga Zweiga pokušali su objasniti strukturu grupe hadrona pomoću manjih čestica: kvarkova. 70-ih godina uveden je koncept boje kao izvora jake sile u sklopu kvantne kromodinamike, koji je na kraju postao opće prihvaćeni koncept. U teoriji polja, kvantna kromodinamika (ili skraćeno QCD) je teorija koja opisuje jaku nuklearnu silu između kvarkova i gluona, fundamentalnih čestica koji su konstituenti hadrona. Postoje mnoge sličnosti sa kvantnom elektrodinamikom; ulogu naboja ima boja, a gluoni su prenositelji sile, kao što su fotoni u kvantnoj elektrodinamici. U ovom poglavlju objasnit ćemo nekoliko bitnih koncepata poput baždarnih transformacija, zatočenja boje, asimptotske slobode i hadronizacije.

2.1 SU(3) baždarna invarijantnost

Dok kvantna elektrodinamika počiva na U(1) lokalnoj baždarnoj simetriji, simetrija u kvantnoj kromodinamici jest SU(3). Za danu valnu funkciju $\psi(x)$, odnosno Diracov spinor, SU(3) transformacija lokalne faze može se zapisati kao [2]:

$$\psi(x) \to \psi'(x) = \exp\left[ig_s\,\boldsymbol{\alpha}(x)\cdot\boldsymbol{T}\right]\psi(x), \tag{2.1}$$

pri čemu je g_s faktor proporcionalnosti koji odgovara jačini jake sile. U gornjoj jednadžbi $\boldsymbol{T} = \{T^a\}$ predstavlja osam generatora SU(3) grupe, dok je $\boldsymbol{\alpha}(x) = \{\alpha^a(x)\}$ osam funkcija prostornovremenske koordinate x. Zbog ovoga, valna funkcija $\psi(x)$ mora sadržavati novi stupanj slobode koja možemo prikazati kao vektor od s 3-komponentni vektor. Ovaj novi stupanj slobode nazivamo bojom, pri čemu crvena, plava i zelena označavaju stanja. Dakle, SU(3) transformacija odgovara rotiranju stanja u prostoru boja oko osi koja je različita u svakoj točki prostorvremena, pošto su α^a funckije prostorvremena. Kako bi SU(3) invarijatnost bila zadovoljena, potrebno je u Diracovu jednadžbu uvesti 8 novih polja A^a_μ (svako odgovara jednom od generatora,

dakle a ima 8 indeksa):

$$i\gamma^{\mu}\left[\partial_{\mu} + ig_s A_{\mu}^{a} T^{a}\right]\psi - m\psi = 0 \tag{2.2}$$

Iz Diracove jednadžbe slijedi Lagrangian iz kojeg se mogu izvesti Feynmanova pravila i dobiti amplitude za raznovrsne procese.

2.2 Zatočenje boje

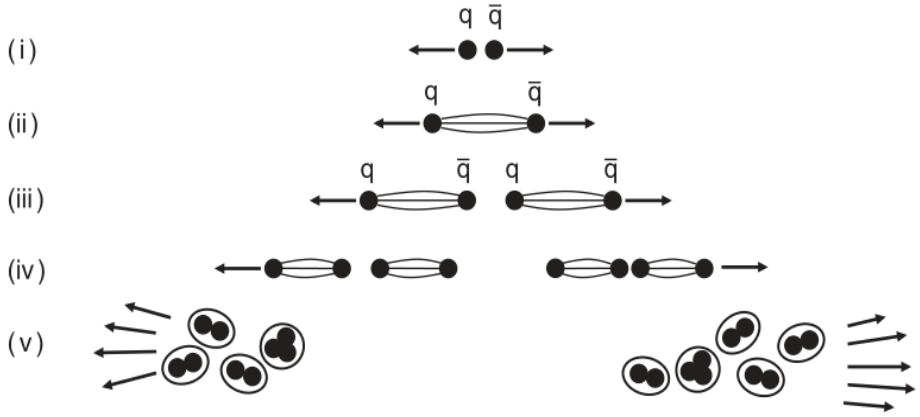
Zatočenje boje je fenomen da čestice s bojom, kvarkovi i gluoni, ne mogu biti izolirani. Točnije, jedino se čestice koje se nalaze u singletnom bezbojnom stanju mogu propagirati kao slobodne čestice. Unatoč tome što ne postoji analitički dokaz za zatočenje boje, svi eksperimenti ukazuju na njenu točnost. Interakcija kvarkova može se gledati kao izmjena virtualnih gluona, a udaljavanjem kvarkova bojno polje zamišljamo kao uski snop koji se sužava. Na većim udaljenostima, gustoća energije u snopu je otprilike konstanta [2], što znači da se potencijal može napisati u sljedećem obliku:

$$V(\boldsymbol{r}) \sim \kappa r, \tag{2.3}$$

pri čemu je r udaljenost između kvarkova. Ovakav oblik potencijala nam govori da je efektivno potrebno uložiti beskonačno energije za separirati kvarkove. Postoje mnogi procesi u kojima su produkti raspada kvarkovi i upravo zbog zatočenja boje, visokoenergetski kvarkovi stvaraju mlazove čestica kroz proces hadronizacije.

2.3 Hadronizacija

U procesima (sudari ili raspadi) koji uključuju čestice sa bojom često kao jedan od produkata nastaju kvark-antikvark parovi. Kako se čestice separiraju, tako u jednom trenutku zbog zatočenja boje postaje energetski povoljnije stvaranje novog kvark-antikvark para. Taj proces se nastavlja sve dok kvarkovi imaju dovoljno energije, a u trenutku kada se energija kvarkova smanji, kao finalni produkt raspada detektiraju se grupirani hadroni. Taj proces se naziva hadronizacija (ili fragementacija) te je shematski prikazan na slici 2.1



Slika 2.1: Shematski prikaz hadronizacije u pet koraka. Uzastopnim stvaranjima kvark-antikvark parova, kao krajnji produkt hadroznizacije nalazimo mezone i barione. [2]

Zanimljivo je da svi kvarkovi hadroniziraju, osim top kvarka. Naime, on se raspada putem slabe sile sa vremenom poluraspada od $\tau = 5 \times 10^{-25}$ s [3], što je kraće od vremenske skale djelovanja jake sile.

2.4 Asimptotska sloboda

Procesi u kvantnoj elektrodinamici mogu se tretirati perturbativno te je račun fizikalnih veličina brzo konvergirao. No, u kvantnoj kromodinamici konstanta vezanja je ovisna o energiji i na niskim energijama pri kojima je konstanta vezanja velika, perturbativni račun nije moguć. Evolucija konstante vezanja u QCD-u dobije se renormalizacijom te ima sljedeći oblik [4]:

$$\alpha(q^2) = \frac{\alpha(\mu^2)}{1 + C\alpha(\mu^2)\ln\left(\frac{q^2}{\mu^2}\right)}, \tag{2.4}$$

pri čemu je C konstanta koja ovisi o broju kvarkova i boja te je $\alpha(\mu^2)$ znana vrijednost konstante vezanja na skali μ. Vidljivo je kako vrijednost konstante vezanja pada s porastom energije; na skali $q > 100$ GeV konstanta vezanja je $\alpha \sim \mathcal{O}(-1)$ što omogućava perturbativni račun na višim energijama. Ovo svojstvo kvantne kromodinamike naziva se asimptotskom slobodom.

3 LHC i eksperimentalna fizika čestica

Većina eksperimenata u području fizike čestica provodi se na velikim sudarivačima čestica. Linearni akceleratori se koriste u eksperimentima gdje želimo promatrati sudar sa statičnom metom, a sinkrotronski, koji su potekli od ciklotrona, kružnog su oblika te sudaraju dva snopa suprotnih smjerova kretanja. Velika prednost sinkrotronskih akceleratora jest što mogu postići puno veće energije u sustavu centra mase u odnosu na linearne akceleratore. Energija u sustavu centra mase $\sqrt{s}$ za n čestica definira se kao:

$$\sqrt{s} = \sqrt{\left(\sum_i^n E_i\right)^2 - \left(\sum_i^n \boldsymbol{p}_i\right)^2} \tag{3.1}$$

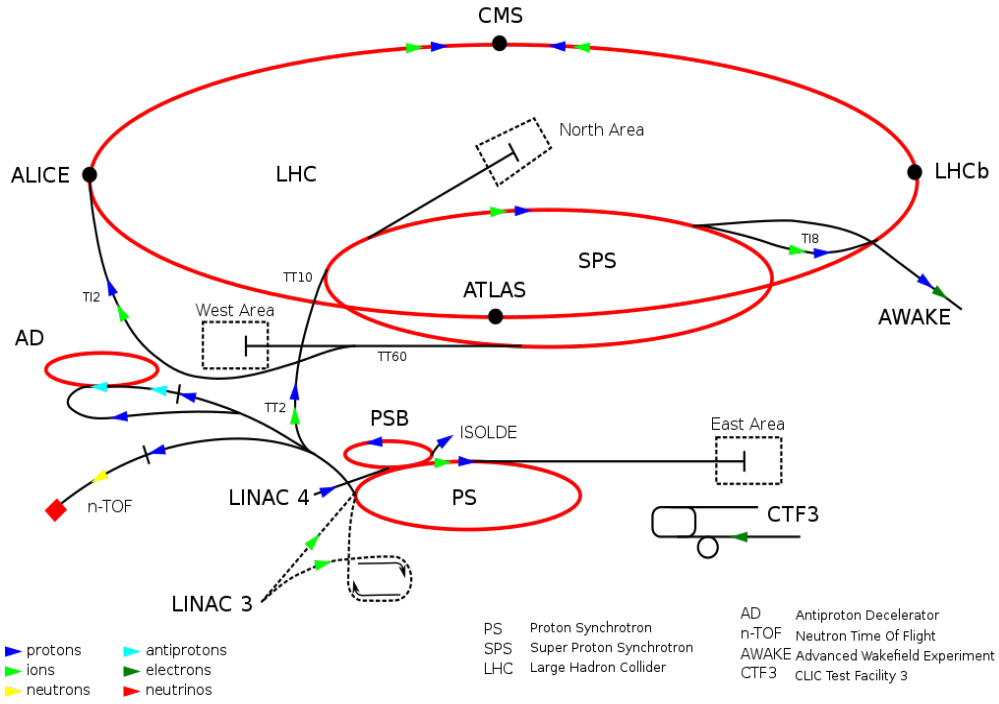
Iz ovoga slijedi, da za slučaj linearnog akceleratora u kojem statičnu metu gađamo protonom energije E, energija u sustavu centra mase aproksimativno iznosi:

$$\sqrt{s} \approx \sqrt{2m_p E}, \tag{3.2}$$

pri čemu je m_p masa protona. Za energiju snopa od $E = 7$ TeV, dostupna energiju za stvaranje novih čestica iznosi svega $\sqrt{s} = 115$ GeV. Dakle, za istu energiju u sustavu centra mase, linearni akcelerator treba otprilike $\mathcal{O}(2)$ puta više energije za stvaranje novih čestica u odnosu na sinkrotronski akcelerator. Iz tog razloga, većina eksperimenata danas ne koristi linearne akceleratore.

3.1 Detekcija čestica na LHC-u

LHC je najveći i najenergetskiji sudarivač čestica na kojem se primarno u eksperimentima promatraju proton-proton sudari, no proučavaju se i sudari težih iona. Mjerenja se tipično vrše nekoliko godina te se nakon toga 2-3 godine detektori popravljaju i unaprijeđuju. Na LHC-u se nalaze 4 veća detektora: ATLAS i CMS (Compact Muon Solenoid) su detektori za opće svrhe, dok su ALICE i LHCb namijenjeni za specijalizirane eksperimente, poput proučavanja kvark-gluon plazme. Shematski prikaz LHC-a prikazan je na slici 3.1



Slika 3.1: Prikaz strukture Large Hadron Collider-a (LHC) te odgovarajućih detektora. Strelicama u boji prikazane su putanje snopova. [5]

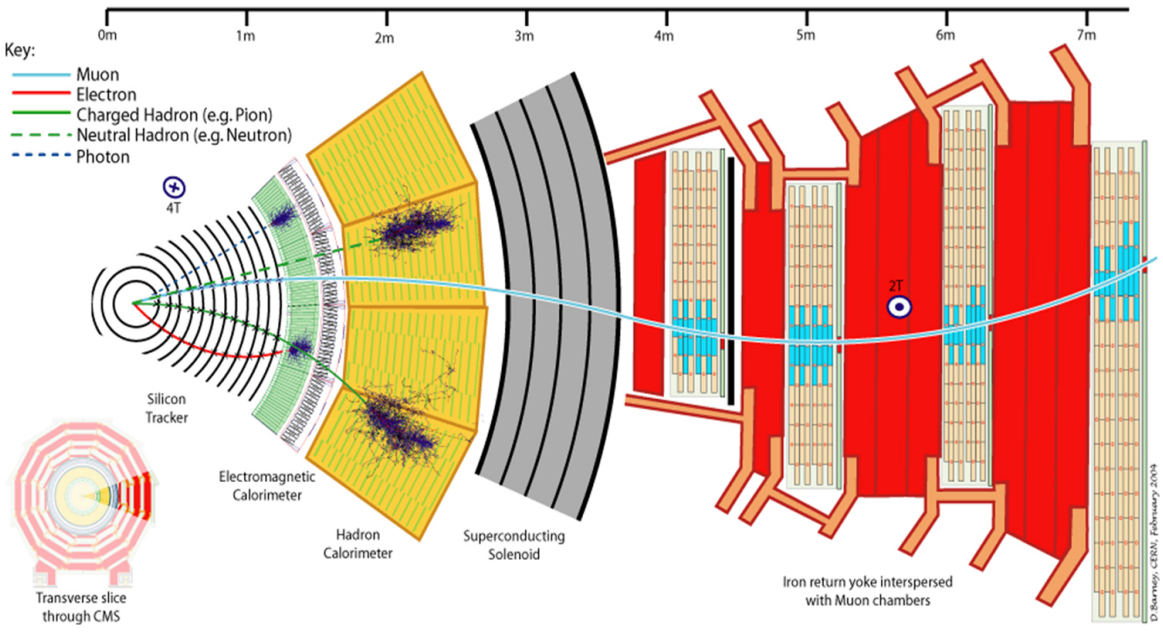
Struktura CMS detektora prikazana je na slici 3.2. Nakon sudaranja snopova, rezultantne čestice inicijalno prolaze kroz tracking detektor (na slici označeno kao silicon tracker). Putanje (nabijenih) čestica kroz detektor zakreće CMS magnet koji ima jačinu ~ 4 T, pomoću čega se može odrediti omjer mase/naboja čestica. Cilj tracking detektora jest da omogući preciznu rekonstrukciju putanja čestica, a preciznost na CMS detektoru je otprilike 10μm. Isto tako je bitno da čestice vrlo malo interagiraju s tracking detektorom, kako im se ne bi previše poremetile inicijalne putanje. Zbog toga se koriste u slojevima postavljeni silikonski detektori koji sadrže piksele u kojima se stvori električni signal prolaskom čestice.

Nakon tracking detektora, čestice prolaze kroz elektromagnetski kalorimetar, skraćeno ECAL. U ovom dijelu detektora čestice prolaze kroz materijale s visokim atomskim brojem Z, konkretno CMS koristi $PbWO_4$, koji je neorganski scintilator. Prolaskom visokoenergetskih nabijenih čestica kroz ECAL, čestice zakočnim zračenjem proizvode fotone, koji stvaraju elektron-pozitron parove. Taj proces se nastavlja dok god čestice imaju dovoljno energije i naziva se elektromagnetski pljusak. Fotoni i elektroni većinu svoje energije deponiraju u ovom dijelu detektora.

U usporedbi sa elektromagnetskim kalorimetrom, hadronski kalorimetar zauzima

puno veći prostor. Razlog tome jest što su udarni presjeci manji, te se nuklearne interakcije odvijaju na većim prostornim skalama. Nabijene hadronske čestice deponirat će dio energije i u ECAL dijelu detektora. Zbog velike mase većinski dio energije najčešće deponiraju u HCAL dijelu detektora. Hadronski kalorimetri često imaju oscilatornu strukturu između absorbera i fluorescentnog scintilatora, no zbog fluktuacija udjela elektromagnetskih pljuskova, preciznost hadronskog kalorimetra je gora u usporedbi s elektromagnetskim kalorimetrom.

Svi do sada nevedeni dijelovi prisutni su u većini modernih detektora, no veliki dio CMS-a posvećen je detekciji miona. Problem detekcije miona jest što mioni deponiraju energiju u ECAL, no zbog mnogo veće mase u odnosu na elektrone, mioni deponiraju samo udio energije. Kako bi se odredile putanje miona te njihovi impulsi, mionski detektor sastoji se od tri glavna dijela: driftne cijevi (mjerenje položaja), katodne komore (identifikacija miona i određivanje korespondencije sa signalom iz trackera) te rezistivne komore (mjerenje impulsa miona). Zbog fizikalne veličine te konstrukcije od više nezavisnih mjernih komora, ovakav sustav je prirodno robustan te može jednostavno filtrirati pozadinske šumove.



Slika 3.2: Shematski prikaz strukture CMS detektora. Volumno najveći dio detektora posvećen je detekciji miona. [6]

3.2 Relativističke kinematičke varijable

Kinematičke varijable koje su pogodne za opis sudara u detektoru poput CMS-a su transverzalni moment i pseudorapiditet. Ako definiramo da je z-os paralelna sa snopom čestica, onda je xy ravnina okomita na snop te transverzalni moment definiramo kao:

$$p_T = \sqrt{p_x^2 + p_y^2} \, . \tag{3.3}$$

Pošto je naš detektor cilindričnog oblika, transverzalni moment odgovara ukupnom momentu okomitom na os simetrije (u ovom slučaju to je z os), odnosno ukupnom momentu koji je usmjeren prema detektoru. Druga relevantna kinematička veličina jest rapiditet:

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \, . \tag{3.4}$$

Korisno svojstvo rapiditeta jest što je razlika dvaju rapiditeta invarijantna na Lorentzove transformacije. U visokoenergetskom režimu, masa je malena u odnosu na energiju, stoga možemo aproksimativno izvrijedniti $p_z \approx E \cos\theta$, pri čemu je θ kut između momenta čestica i osi snopa, odnosno z osi. Koristeći navedenu aproksimaciju dobivamo pseudorapiditet:

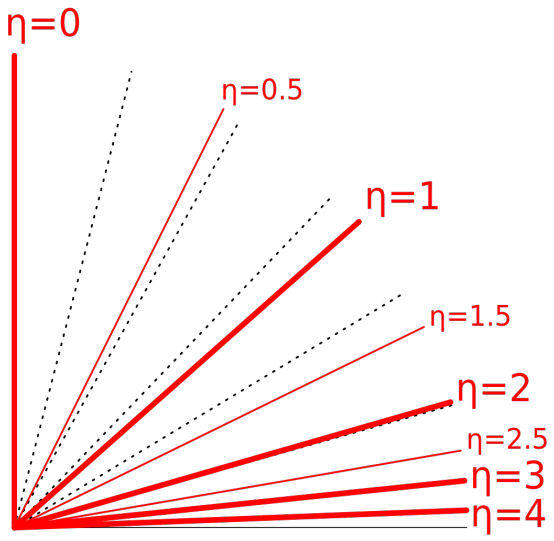
$$\eta = -\ln \left(\tan \frac{\theta}{2} \right) \, . \tag{3.5}$$

Iz gornje definicije vidimo da pseudorapiditet nije ograničena funkcija, no u stvarnom detektoru pseudorapiditet detektiranih čestica ne prelazi 3. Naime, čestice sa većim pseudorapiditetom ne mogu se detektirati pošto je detektor konačnih dimenzija. Na slici 3.3 prikazane su vrijednosti pseudorapiditeta za različite polarne kutove čestica.

Kinematička veličina koja se često koristi kao mjera udaljenosti u detektorima jest ΔR te je definirana kao:

$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2} \, , \tag{3.6}$$

pri čemu je ϕ kut u $x - y$ ravnini mjeren od osi x. Intuitivno, ova udaljenost ima korelaciju sa kutnom udaljenosti između čestica ili mlazova. Uobičajeni prikaz četvetovektora je pomoću baze $(E,\, p_x,\, p_y,\, p_z)$, ali s obzirom da postoji bijekcija između Kartezijevih veličina i gore navedenih varijabli, često se koristi prikaz četverovektora u bazi $(p_T,\, \eta,\, \phi,\, m)$.



Slika 3.3: Vrijednosti pseudorapiditeta za različite putanje čestica. Iscrtkanom linijom prikazani su polarni kutovi u inkrementima od $15°$. [7]

3.3 Mlazovi čestica

Procesima fragmentacije i hadronizacije nastaje usmjereni snop čestica koji nazivamo mlaz. Svojstvo mlazova jest da se sve čestice kreću u sličnom smjeru, kao što je vidljivo u detektorima. Takvo svojstvo je intuitivno, pošto zbroj momenata svih čestica iz mlaza mora odgovarati ukupnom momentu kvarka iz kojeg je nastao mlaz. Pošto je ovakva definicija relativno arbitrarna, postoje različiti algoritmi grupiranja i rekombinacije u mlazove. Dok algoritmi grupiranja određuju kako se skup čestica grupira u mlaz, algoritmi rekombinacije određuju kako se pridružuje četverovektor danom mlazu. U eksperimentima, mlazovi se konstruiraju promatranjem deponirane energije u različitim kalorimetrima i njihovom rekonstrukcijom možemo izvući bitne informacije o partonu čijom hadronizacijom su nastali konstituenti mlaza.

Algoritmi za grupiranje mlazova trebaju zadovoljavati neke uvjete koji garantiraju da je moguće perturbativnim računom dobiti konačne udarne presjeke. Dva najbitnija uvjeta su infracrvena i kolinearna sigurnost: infracrvena sigurnost zahtjeva da algoritam daje isti skup mlazova nakon što se u događaj doda slabo pozadinsko zračenje, a kolinearna sigurnost da konačni skup mlazova bude neovisan o kolinearnom grananju ulaznih čestica. Najčešće korišteni algoritmi su anti-k_T, k_T, Cambridge/Aachen te konusni algoritam.

3.4 Anti-k_T algoritam

Algoritmi grupiranja mlazova su jedni od glavnih alata za analizu podataka iz hadronskih sudara. Prema obliku mlaza, algoritmi se ugrubo mogu razvrstati u regularne, odnosno algoritme koji rezultiraju fiksnijim oblicima mlazova, ili iregularne, koji daju fleksibilnije oblike mlazova. Regularnost, u jednu ruku otežava algoritmu da se prilagodi sukcesivnoj prirodi grananja u QCD procesima, no u drugu ruku fiksniji oblici mlazova omogućavaju raznovrsne eksperimentalne kalibracije i teorijske kalkulacije.

Anti-k_T algoritam zasniva se na sekvencijalnoj rekombinaciji, no dobiveni mlazovi mogu imati općenitiji oblik od mlazova dobivenih algoritmima poput k_T i Cambridge/Aachen. Za početak definirano udaljenost d_{ij} kao udaljenost između čestica i i j te označavamo udaljenost između i-te čestice i snopa kao d_{iB}. Ove udaljenosti definiraju se kao [8]:

$$d_{ij} = \min(p_{Ti}^{2p}, p_{Tj}^{2p}) \frac{\Delta R_{ij}^2}{R^2} \ , \tag{3.7}$$

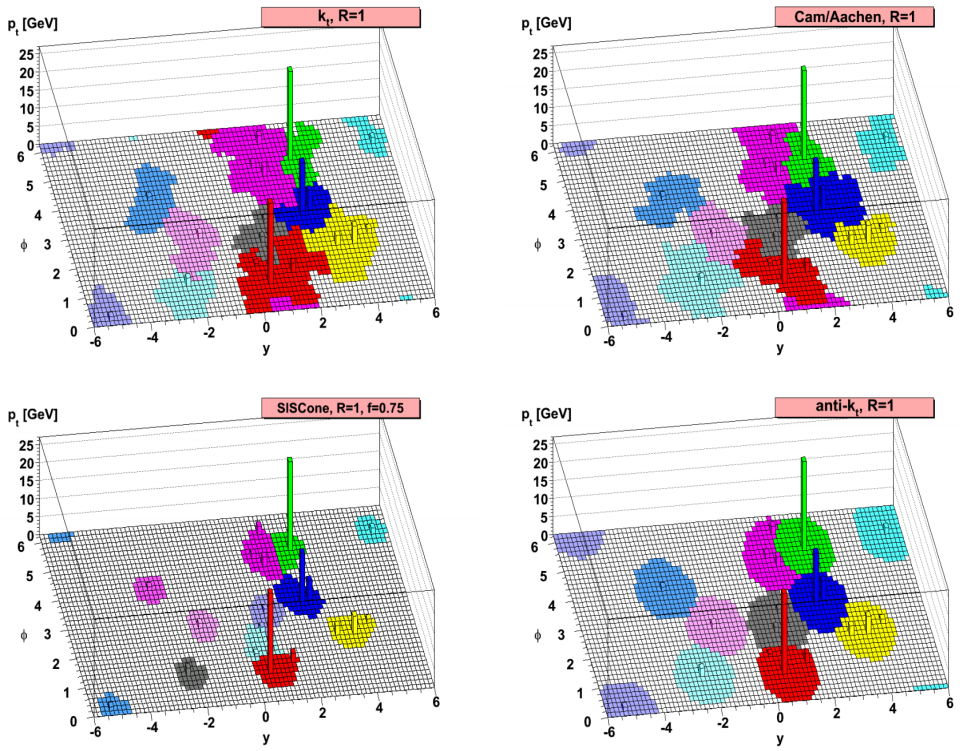
$$d_{iB} = p_{Ti}^{2p} \ , \tag{3.8}$$

pri čemu je ΔR_{ij} mjera udaljenosti definirana pomoću 3.6, p_{Ti} je transverzalni impuls i-te čestice te je R parametar radijusa snopa. Uz ove standardne veličine, parametar p je novitet koji određuje omjere energijske i geometrijske skale.

Sekvencijalno grupiranje čestica u mlaz izvršava se na sljedeći način: tražimo najmanje udaljenosti (neovisno je li to d_{ij} ili d_{iB}) te ako je ta udaljenost d_{ij} (udaljenost između čestica), čestica se dodaje u mlaz, no ako je najmanja udaljenost d_{iB} (udaljenost čestice do snopa), onda se završava grupiranje tog mlaza. Nadalje, sve čestice koje pripadaju toj grupi mičemo iz ukupnog skupa čestica te sa preostalim česticama ponavljamo postupak sve dok nismo iscprili sve čestice.

Za vrijednost parametra $p = 1$, dobiva se poznati k_T algoritam, dok slučaj $p = 0$ odgovara Cambridge/Aachen algoritmu. Poseban slučaj, koji se na prvu čini neintuitivan, jest kada je parametar $p = -1$. Ovaj slučaj naziva se anti-k_T algoritam. Detaljnijim proučavanjem ovog slučaja može se pokazati da niskoenergetske čestice ne modificiraju oblik mlaza, dok visokoenergetske modificiraju. Na slici 3.4 vidimo grupiranje čestica u mlazove za 4 različita algoritma. Prikazani mlazovi dobiveni su tako što su se uzeli jednaki partonski događaji, no dodano je $\sim 10^4$ nasumičnih

čestica niske energije. Vidljivo je kako su k_T i Cambridge/Aachen algoritmi grupirali čestice u mlazove nepravilnih granica, što je neželjeni rezultat ukoliko želimo da nam slabi šum ne utječe na oblik mlaza. Zbog navedenih svojstava, anti-k_T algoritam daje zadovoljavajuće rezultate bez ugrožavanja infracrvene i kolinearne sigurnosti.



Slika 3.4: Rezultati grupiranja čestica u mlazove za 4 različita algoritma. Na osnovni partonski događaj dodano je $\sim 10^4$ niskoenergetskih čestica kao šum. Dok za k_T i Cambridge/Aachen algoritam oblik ovisi o skupu nasumičnih čestica, anti-k_T algoritam rezultira mlazovima istog oblika. [8]

3.5 Simulacija podataka

Mnogi eksperimenti na sudarivačima pokušavaju potvrditi ili opovrgnuti razne teorije koje predviđaju nove čestice. Općenito, kako bi se provjerila usklađenost teorije i eksperimenta, uspoređuju se mjereni rezultati sa očekivanim rezultatima iz simulacija. Najčešće, simulacija podataka izvršava se u tri koraka.

Prvi korak jest Monte-Carlo simulacija. Za računanje matričnih elemenata (odnosno vjerojatnosti) elementarnih procesa koriste se mnogi paketi (poput POWHEG [9]), dok se većinski PYTHIA [10] koristi za modeliranje partonskih pljuskova i fragmenata hadronizacije. Ovaj korak simulira sudare tako što računa sve potencijalne Feynamnove diagrame i rekreira stohastičke rezultate takvih događaja. Ukratko, ovaj

korak stvara sve čestice prije nego što stignu do detektora te bilježi sva njihova svojstva (poput energija, momenata i slično).

Idući korak jest simulacija odaziva detektora na čestice iz prvog koraka. Ovaj korak naravno ovisi o samoj konstrukciji detektora koji promatramo. Isto tako traje puno dulje od ostalih koraka i zahtjeva naviše računanja. Ovakva kompleksnost dolazi od činjenice da svaku česticu iz prethodnog koraka treba propagirati kroz detektor, a čestice deponiraju energiju kroz prolazak te se zbog toga propagacija mora raditi iterativno.

Završni korak jest rekonstrukcija. Do sada, kroz simulaciju znali smo točne informacije o česticama, kao i njihovu deponiranu energiju u detektoru. U ovom koraku koristimo samo signale iz prethodnog koraka (odgovor detektora) te na temelju toga pokušavamo rekonstruirati originalni sudar (čestice, četverovektore, itd.). Bitno je napomenuti da u ovom koraku se ne koriste informacije iz prvog koraka. Isto tako, ovaj korak je identičan kada procesuiramo stvarne podatke sa detektora.

4 Neuralne mreže

Neuralne mreže računalni su sustavi djelomično inspirirani radom mozga. Sastoje se od slojeva neurona koji su međusobno povezani te se aktiviraju ovisno o parametrima i vrijednostima aktivacija neurona iz prijašnjih slojeva. Bitno svojstvo neuralnih mreža jest njihova mogućnost učenja, odnosno prolaskom mnogih primjera podataka kroz mrežu, možemo ju optimizirati da daje željene rezultate. Ciljevi mreže mogu biti regresija, klasifikacija, no i mnogi drugi kompleksniji ciljevi poput estimacije gustoća vjerojatnosti, generiranja primjera i slično.

Neuralne mreže su relativno star koncept; prvi oblici mreža sežu sve do sredine 20. stoljeća. Sedamdesetih godina napravljeni su prvi algoritmi učenja (propagacija pogreške unazad), dok su osamdesetih napravljeni algoritmi koji riječi prikazuju kao vektor značajki te predviđaju iduću riječ u rečenici. Primjenu mreža na kompliciranije slučajeve, poput klasifikacije slika pomoću konvolucijskih neuralnih mreža, ograničavali su tehnički faktori. Unazad 10 godina, razvoj grafičkih kartica (GPU) omogućio je puno veću snagu računanja zbog paraleliziranog računanja koje ovakvi uređaju mogu vršiti. Naime, većina matematičkih operacija u treniranju neuralnih mreža uključuje množenje matrica, traženja njihovih inverza i slično, a grafičke kartice su idealne za ovakve paralelizabilne procese. Danas su neuralne mreže našle primjenu u mnogim sferama pošto služe kao vrlo generalan alat za slične probleme u različitim područjima. Ugrubo, glavna područja istraživanja neuralnih mreža su strojno učenje, obrada prirodnog jezika, računalni vid, robotika te opća umjetna inteligencija. Na primjer, autonomna vozila primjer su kombinacije robotike i računalnog vida.

U ovom poglavlju predstavit ćemo jednostavne neuralne mreže, metode treniranja, konvolucijske neuralne mreže i autoenkodere.

4.1 Jednostavne neuralne mreže

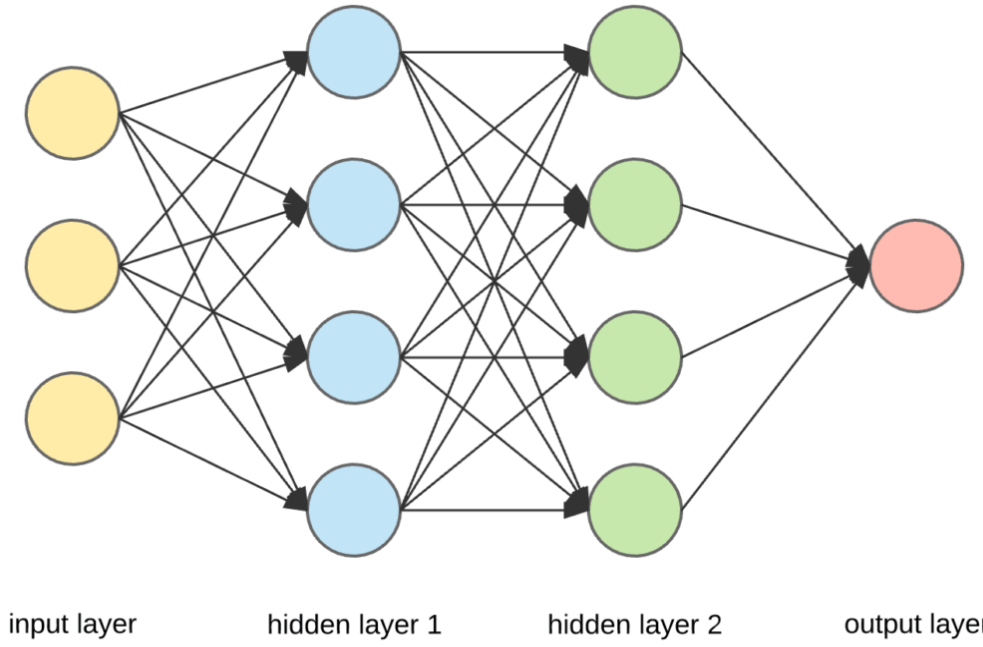
Jednostavne neuralne mreže (odnosno Feedforward Artificial Neural Networks) su najjednostavniji oblik neuralnih mreža koji se sastoji od više slojeva. Svaki sloj ima određen broj neurona, koji su osnovni gradivni blokovi neuralnih mreža. Neuron je zapravo objekt kojem je pridodana vrijednost zvana aktivacija. Stoga sloj možemo prikazati u vektorskom obliku, pri čemu svaki element vektora predstavlja aktivaciju

pojedinog neurona.

Ulazni sloj (prvi sloj mreže) služi kao ulaz naših podataka u mrežu. Vrijednosti aktivacija u idućem sloju, koji može biti ili izlazni sloj ili skriveni sloj, dobiva se na sljedeći način. Nazovimo vektor koji predstavlja neurone iz k-tog sloja $\mathbf{x}^k = \{x_i^k\}$. Tada vrijednost aktivacije j-tog neurona u idućem sloju dobivamo kao:

$$x_j^{k+1} = f\left(\sum_{i=1}^{n} w_i x_i^k + b_j\right). \tag{4.1}$$

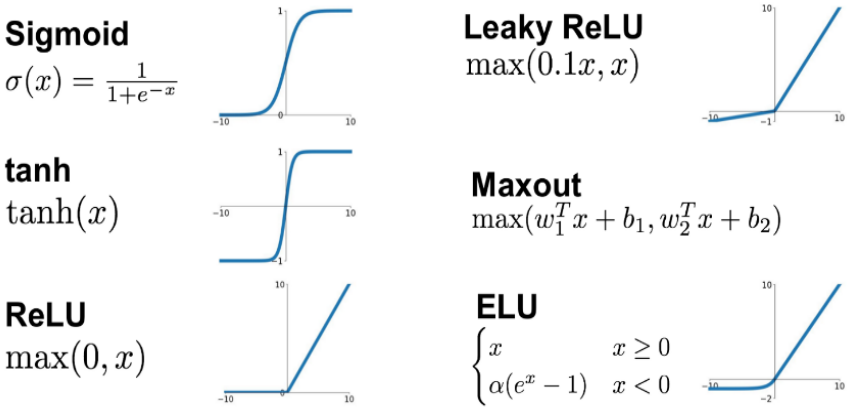
U gornjem izrazu f je aktivacijska funkcija, dok konstante w_i nazivamo težinama, a b_j slobodni konstantni član. Ovakvim postupkom dobivaju se vrijednosti svih neurona u slojevima nakon ulaznog sloja. Shematski prikaz mreže s 2 skrivena sloja prikazan je na slici 4.1.



Slika 4.1: Shematski prikaz neuralne mreže s 2 skrivena sloja. Strelice prikazuju kako su neuroni iz različitih slojeva povezani. [11]

Aktivacijska funkcija f je proizvoljna i povijesno je najčešće bila tangens hiperbolni ili sigmoida. Problem sa ovim aktivacijskim funkcijama jest što imaju plato, odnosno derivacije gotovo iščezavaju za veće apsolutne vrijednosti argumenta funkcije. Iz tog razloga, vrlo česta aktivacijska funkcija jesto ReLU, koja je prikazana na slici 4.2, uz ostale aktivacijske funkcije. Potreba za ovakvim funkcijama dolazi iz potrebe da je model dovoljno ekspresivan; ove funkcije uvode nelinearnosti u model i zbog

toga neuralne mreže mogu dobro opisivati kompleksne povezanosti u podacima.



Slika 4.2: Pregled čestih aktivacijskih funkcija koje se koriste u neuralnim mrežama. [12]

Konstrukcija mreže je relativno arbitrarna. Ono što je fiksno su ulazni i izlazni sloj. Naime, ulazni sloj je određen podacima koje želimo koristiti, a izlazni onime što želimo izračunati. Na primjer, ako želimo predvidjeti cijenu kuće na temelju broja kvadratnih metara, broja soba i godini izgradnje, onda će nam ulazni sloj imati 3 neurona, dok će nam izlazni sloj imati jedan neuron, koji predstavlja cijenu kuće. Svi slojevi između su skriveni te su njihove dimenzije i broj slobodan odabir. No, treba paziti; odabir arhitekture nam određuje i kompleksnost modela i zbog toga ekstremne arhitekture nisu dobre. Prejednostavna arhitektura neće biti dovoljno kompleksna za dobro modeliranje podataka, dok će prekompleksna arhitektura biti sklona učenju napamet.

4.2 Treniranje neuralnih mreža

Optimizacija neuralne mreže podrazumijeva odabir težina **w** i slobodnih konstanti **b** takvih da mreža daje željene rezultate, odnosno rezultate koji najmanje odstupaju od stvarnih vrijednosti. Kao mjeru odstupanja koristimo funkciju koju nazivamo funkcijom gubitka. Na primjer, ako pokušavamo predvidjeti cijenu kuće, tada bismo intuitivno kao mjeru odstupanja koristili apsolutnu razliku stvarne i predviđene cijene, ili pak kvadratno odstupanje tih cijena. Sam odabir funkcije gubitka je proizvoljan i ovisi o vrsti problema. Za klasifikacijske probleme, možemo koristiti unakrsnu entropiju, dok za probleme regresijskog tipa (primjer predviđanja cijene kuće), često

koristimo ME (srednje apsolutno odstupanje) ili MSE (srednje kvadratno odstupanje). Za N primjera podataka ove funkcije gubitaka imaju oblik:

$$\text{ME}(\mathbf{w}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i(\mathbf{w}, \mathbf{b}; \mathbf{x}_i) - y_i| \ , \tag{4.2}$$

$$\text{MSE}(\mathbf{w}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i(\mathbf{w}, \mathbf{b}; \mathbf{x}_i) - y_i)^2 \ , \tag{4.3}$$

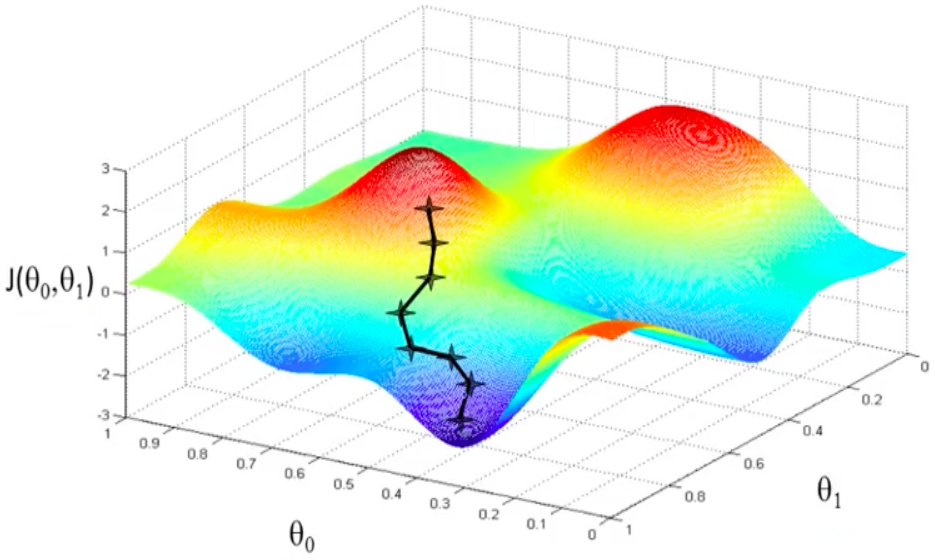
pri čemu je $\hat{y}_i(\mathbf{w}, \mathbf{b}; \mathbf{x}_i)$ vrijednost predviđena mrežom za ulazne podatke $\mathbf{x}_i$, a y_i je stvarna vrijednost. Funkcije gubitka inherentno ovise o težinama mreže **w** i **b** pošto vrijednosti koje predviđa mreža ovise o parametrima mreže. Zbog te ovisnosti, možemo tražiti minimum funkcije gubitka diferencijalnim računom. No u stvarnosti, nemoguće je analitički pronaći minimum, stoga koristimo iterativni optimizacijski algoritam koji se naziva gradijentni spust [13]. Ideja ovakve optimizacije jest da nađemo gradijent funkcije gubitka te iterativnim koracima ažuriramo težine mreže **w** i **b** u obratnom smjeru gradijenta. Matematički, ako skupinu težina mreže nazovemo $\mathbf{a} \equiv (\mathbf{w}, \mathbf{b})$, tada nove težine $\mathbf{a}'$ nakon jednog koraka gradijentnog spusta možemo zapisati kao:

$$\mathbf{a}' = \mathbf{a} - \gamma \vec{\nabla} L(\mathbf{a}) \, , \tag{4.4}$$

pri čemu je γ stopa učenja koja određuje koliko se u svakom koraku pomičemo duž gradijenta. Standardne vrijednosti za stopu učenja su $\gamma \sim 10^{-4}$, no to isto ovisi o dubini mreže, strukturi mreže i tipu problema. Prikaz gradijentnog spusta u 8 iteracija prikazan je na slici 4.3.

Podaci koje koristimo za treniranje mreže često su veliki i nemoguće je istovremeno evaluirati gradijent funkcije gubitka za sve podatke. Iz tog razloga koriste se algoritmi optimizacije poput stohastičkog gradijentnog spusta (eng. Stohastic gradient descent), u kojem se vrši spust na temelju malog podskupa podataka. Takav pristup će često sporije konvergirati, pošto mali podskup podataka ne daje egzaktni gradijent, već će imati "cik-cak" uzorak. Algoritam optimizacije koji se pokazao uspješnim za širok spektar problema jest ADAM (Adaptive Moment Estimation) [14], koji kombinira najbolja svojstva AdaGrad [15] i RMSprop [16] algoritama. Kao metoda računanja gradijenata, koristi se metoda unazadne propagacije. U srži, takva metoda jest samo primjena ulančanog pravila deriviranja, u našem slučaju za funk-

ciju gubitka.



Slika 4.3: Prikaz gradijentnog spusta za funckiju gubitka $J(\theta_0, \theta_1)$ koja ovisi o dva parametra. Crna linija predstavlja ”putanju” težina koristeći gradijentni spust u cilju ostvarivanja lokalnog ekstrema funkcije gubitka.

4.3 Konvolucijske neuralne mreže

Konvolucijske neuralne mreže (eng. CNN) su klasa neuralnih mreža koje se često koriste za analizu slika. Da bismo vidjeli njihova korisna svojstva za analizu objekata koji imaju prostornu povezanost, prvo definiramo konvoluciju funkcija $f(x)$ i $g(x)$ kao:

$$(f * g)(x) \equiv \int_{-\infty}^{\infty} f(x-y)g(y)\,\mathrm{d}y\,. \tag{4.5}$$

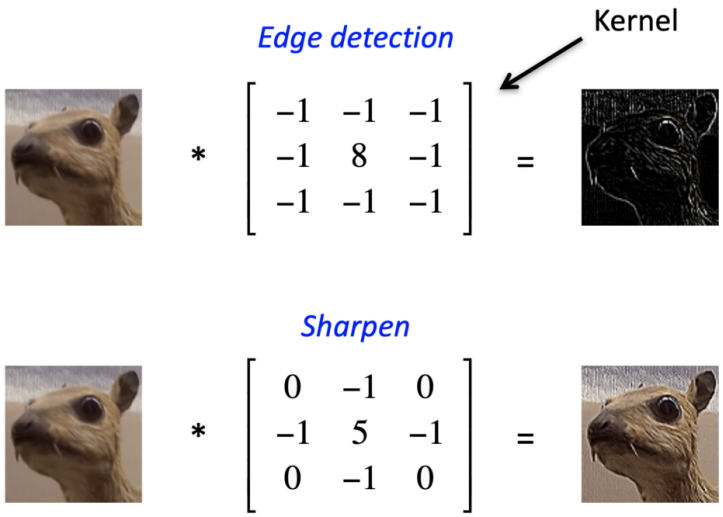
Gornja relacija vrijedi za dvije funkcije jedne varijable, no slike su diskretne funkcije dvije varijable. Ako koordinatu horizontalnog piksela nazovemo i, dok koordinatu vertikalnog j, onda vrijednost piksela na danim koordinatama možemo pisati kao $b[i, j]$. Treba napomenuti da slike u boji imaju još dodatnu dimenziju koja se naziva kanalna dimenzija i slike koje inače promatramo imaju crveni, zeleni i plavi kanal.

Diskretna konvolucija dvaju slika (s jednim kanalom boje) b i k ima sljedeći oblik:

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} b[m,n] \cdot k[i-m, j-n]\,. \tag{4.6}$$

Granice u sumi idu u beskonačnost, dok slike imaju konačan broj piksela, no slike možemo gledati kao da imaju vrijednosti nula na svim ostalim koordinatama (eng. *zero-padding*). Kada bismo u praksi radili konvoluciju dvaju slika, vršili bismo operaciju sve dok ne bismo došli do *zero-padded* dijelova obaju slika.

U procesuiranju slika koriste se konvolucije slika sa matricama koje nazivamo filteri (eng. *kernel*). Fiksni filteri u konvoluciji sa slikama daju određene efekte poput zamućivanja, istančavanja rubova i slično, kao što je prikazano na slici 4.4. Većinom su takve matrice veličina 3×3 ili 5×5, no moguće su i druge dimenzije. Ovakvu konvoluciju možemo zamisliti kao da filter "kližemo" preko slike te na svakoj poziciji dobijemo jednu vrijednost piksela nove slike.



Slika 4.4: Primjena filtera za detekciju rubova i za izoštravanje. Navedeni filteri su prirodno normalizirani, no inače se u cilju očuvanja ukupne vrijednosti piksela matrice normaliziraju tako da se podijele sa sumom svojih matričnih elemenata. [17]

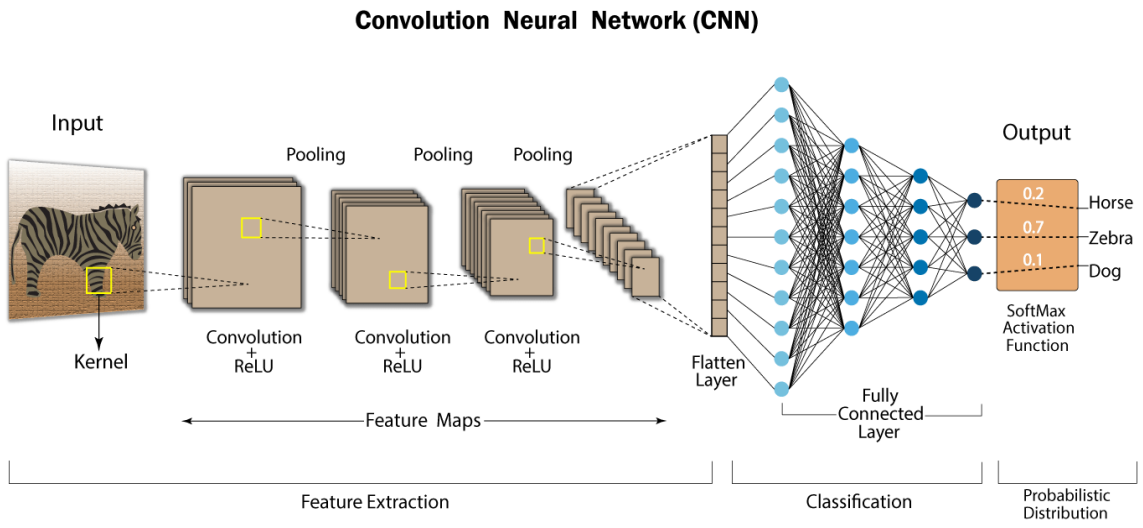
U konvolucijskim neuralnim mrežama koriste se konvolucijski slojevi koji se sastoje od skupa filtera. Za razliku od jednostavnih neuralnih mreža, parametri konvolucijskih slojeva su elementi matrica koji čine filter. Dakle, konvolucijske mreže će učiti filtere koji optimizraju našu mrežu. Sami konvolucijski slojevi sastoji se od filtera, a za svaki sloj biraju se sljedeći glavni parametri filtera:

- Veličina filtera - ovim parametrom biramo dimezije filtera u trenutnom sloju,

- Razmak (eng. *Stride*) - odabir za koliko piksela se pomiče filter prilikom svakog koraka operacije konvolucije. U jednadžbi 4.6 pretpostavljen je razmak 1. Na primjer, da se radi o razmaku 2, zamijenili bismo $n \to 2n$ i $m \to 2m$. Isto tako je bitno primijetiti kako razmak utječe na dimenziju izlazne slike. Ako imamo razmak 2, izlazna slika imat će duplo manje dimenzije od ulazne slike.

- Nadopuna (eng. *Padding*) - odabir hoćemo li u ulaznu sliku uključiti dodatne rubove čija svrha je kontrolirati dimenzije izlazne slike.

Konvolucijske neuralne mreže ne moraju nužno imati samo konvolucijske slojeve. Dapače, većina mreža koristi kombinaciju konvolucijskih slojeva i jednostavnih slojeva. Vrlo često se koriste i slojevi udruživanja (eng. *Pooling layers*) koji smanjuju dimenzije slike tako što grupe lokalnih piksela grupiraju u jednu vrijednost. Česti primjeri grupiranja su koristeći odabir maksimalnog piksela (eng. *Max pooling*) ili odabir srednje vrijednosti piksela (eng. *Average pooling*).



Slika 4.5: Neuralna mreža koja se sastoji od kombinacije konvolucijskih slojeva i jednostavnih slojeva. Ulaz u mrežu je slika, a izlaz su vjerojatnosti pripadnosti slike jednoj od tri navedene kategorije (konj, zebra, pas). [18]

Primjer mreže koja klasificira slike u 3 kategorije (konj, zebra, pas) nalazi se na slici 4.5. Mreža se sastoji od više vrsta slojeva; početni dio mreže sastoji se od 3 konvolucijska sloja sa ReLU aktivacijskim funkcijama koji popraćeni slojevima udruživanja. Nakon tog bloka, slika se izravnjava (eng. *Flatten*), odnosno pretvara iz

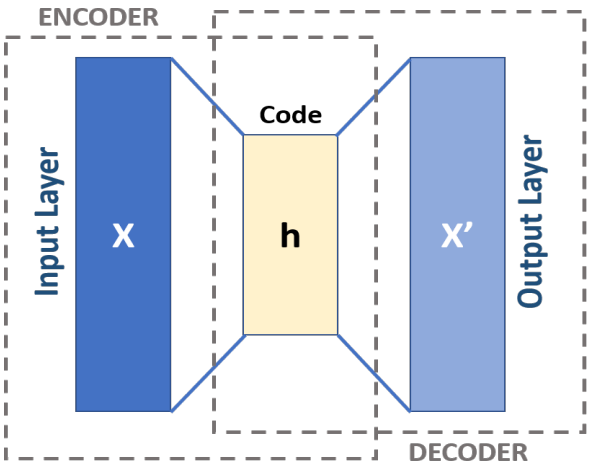
dimenzija (H, W, C) (height, width, channel), u dimenzije $(H \times W \times C)$ (prostorne dimenzije i kanalnu dimenziju sužavamo u jednu dimenziju). Nadalje, izlaz tog bloka služi kao ulaz u jednostavnu neuralnu mrežu s 2 skrivena sloja. Konačni sloj predviđa vjerojatnosti pripadnosti originalne slike danim kategorijama.

4.4 Autoenkoderi

Autoenkoder je vrsta neuralne mreže koja je trenirana tako da pokuša kopirati ulazne podatke na izlazu mreže. Ovakav tip mreže možemo gledati kao da se sastoji od dva dijela: $\mathbf{h}$ (enkoder dio) i $\mathbf{d}$ (dekoder dio). Ako imamo ulazni vektor $\mathbf{x}$, onda je enkodirana verzija podataka dana sa $\mathbf{y} = \mathbf{h}(\mathbf{x})$, dok je dekodirana verzija dana sa $\mathbf{x'} = \mathbf{d}(\mathbf{y})$. Ako autoenkoder radi savršeno, onda imamo identitet, odnosno:

$$\mathbf{x'} = \mathbf{x} \Longrightarrow \mathbf{d}(\mathbf{h}(\mathbf{x})) = \mathbf{x}\,. \tag{4.7}$$

U stvarnosti, ovo nije realan niti željen rezultat. Autoenkoderi su dizajnirani na način da onemogućuju savršenu rekonstrukciju. Često se restrikcija radi tako što limitiramo dimenziju enkodiranih podataka te zahtijevamo da iz manje dimenzije dekoder što bolje reproducira ulazne podatke. Ovakva struktura prikazana je na slici 4.6. Zbog limitacije da prikažemo podatke koristeći manje dimenzija, autoenkoderi su primorani naučiti bitna svojstva i značajke samih podataka. Ovakav pristup je zapravo nenadzirano učenje, pošto su ciljani izlazni podaci jednaki ulaznim.



Slika 4.6: Shematski prikaz simetrične autoenkoderske mreže u kojoj su enkodirani podaci niže dimenzije od ulaznih. [19]

Većinom autoenkoderi imaju simetričnu strukturu, no to nije nužno. Isto tako, sami enkoder i dekoder mogu imati mješovite strukture sastavljene od različitih vrsta slojeva, poput kombinacija jednostavnih i konvolucijskih slojeva. Prilikom konstrukcije autoenkodera, bitno je dobro odrediti dimenziju latentnog prostora (dimenzija enkodiranih podataka). Prevelika dimenzija (slična dimenziji ulaznih podataka) neće natjerati mrežu da izvuče bitne značajke podataka, dok premala dimenzija neće imati dovoljnu ekspresivnu moć za dobar opis podataka.

Postoje mnoge metode koje pomažu autoenkoderima da što efikasnije i bolje nauče latentnu reprezentaciju podataka. Metoda proređenog autoenkodera (eng. *Sparse autoencoder*) uvodi limitaciju na aktivacije neurona; dozvoljeno je samo malom broju neurona da istovremeno ima velike aktivacije. Ovakav pristup je dobar za klasifikacijske probleme, to jest za probleme u kojima su i sami podaci rijetko raspoređeni. Limitacije na aktivacije neurona uvode se tako što se u funkciju gubitka L dodaje dodatni član:

$$L(\mathbf{x}, \mathbf{x'}) \rightarrow L(\mathbf{x}, \mathbf{x'}) + K(\mathbf{h}) \,, \tag{4.8}$$

pri čemu je **h** definiran kao enkoderska funkcija. Dakle, ovisno o odabiru funkcije K, funkcija gubitka ovisit će o samim aktivacijama u sloju **h**. Jedan od mogućih odabira funkcije K može se dobiti iskorištavanjem Kullback-Leiblerove divergencije (eng. *KL divergence*) [20]. Ova divergencija služi kao mjera koliko se dvije distribucije razlikuju jedna od druge. Definirajmo srednju vrijednost (prosjek evaluiran na m primjera podataka) aktivacija u neurona j:

$$\rho_j = \frac{1}{m} \sum_{i=1}^{m} h_j(x_i) \,, \tag{4.9}$$

pri čemu je $h_j(x_i)$ vrijednost ulaznih varijabli u j-ti neurona. Cilj je da ova vrijednost bude što bliža nuli, kako bi većina neurona u prosjeku bila neaktivna. Koristeći KL divergeniju, dodatni član u funkciji gubitka ima oblik:

$$K(\mathbf{h}) = \sum_{j=1}^{m} \left[\rho \ln \frac{\rho}{\rho_j} + (1-\rho) \ln \frac{1-\rho}{1-\rho_j} \right] \,, \tag{4.10}$$

pri čemu suma ide po svim neuronima u danom skrivenom sloju. Ovakav izraz je zapravo suma KL divergencija između distribucije ρ_j i Bernoullijeve nasumične varijable sa srednjom vrijednosti ρ. Parametrom ρ biramo mjeru rijetkosti koju zahtijevamo.

5 Implementacija

U ovom poglavlju predstavit ćemo svoju metodu za rješavanje problema predstavljenog na LHC Olympics 2020. Ovo natjecanje održano je za znanstvenike u području fizike čestica te je cilj bio razviti metodu za detekciju novih, nepoznatih čestica na LHC-u. Naš pokušaj rješavanja problema sastoji se od dvije glavne metode: konvolucijskih autoenkodera te U-Net mreže. Strukturu podataka, konkretne arhitekture i rezultate prezentirat ćemo u idućim potpoglavljima.

5.1 Struktura podataka i pretprocesiranje

U sklopu natjecanja dana su dva glavna skupa podataka, koji su simulirani za CMS. Prvi skup sastoji se od događaja koji opisuju pozadinu, dok se drugi skup sastoji od iste pozadine i malog broja novih, nepoznatih signala. Obje skupine sadrže 10^6 događaja, a svaki događaj opisuje ~ 150 čestica u obliku vektora (p_T, η, ϕ) koji su definirani u poglavlju 3.2.

Najprije treba pronaći prikladan oblik podataka za ulazak u mrežu. Najjednostavnija ideja bi bila uzeti listu svih čestica i pripadnih kinematičkih varijabli kao ulaz u jednostavni sloj neuralne mreže, no to nije funkcionalno iz više razloga. Prvo, različiti događaju imaju različit broj čestica, stoga bi ulaz u mrežu bio varijabilan. To se može riješiti tako da se veličina ulaznog vektora prilagodi događaju s najviše čestica, a ostatak događaja se nadopuni nulama (eng. *zero-padding*). Iz tehničkog aspekta ovo rješava problem, no težine koje bi naučila mreža bi ovisile o samom poretku čestica, a podaci nisu strukturirani na način da je poredak relevantan. Isto tako, ovakva struktura potencijalno ne bi bila prikladna za neki drugi skup podataka koji sadrži više čestica po događaju od skupa na kojem smo trenirali mrežu.

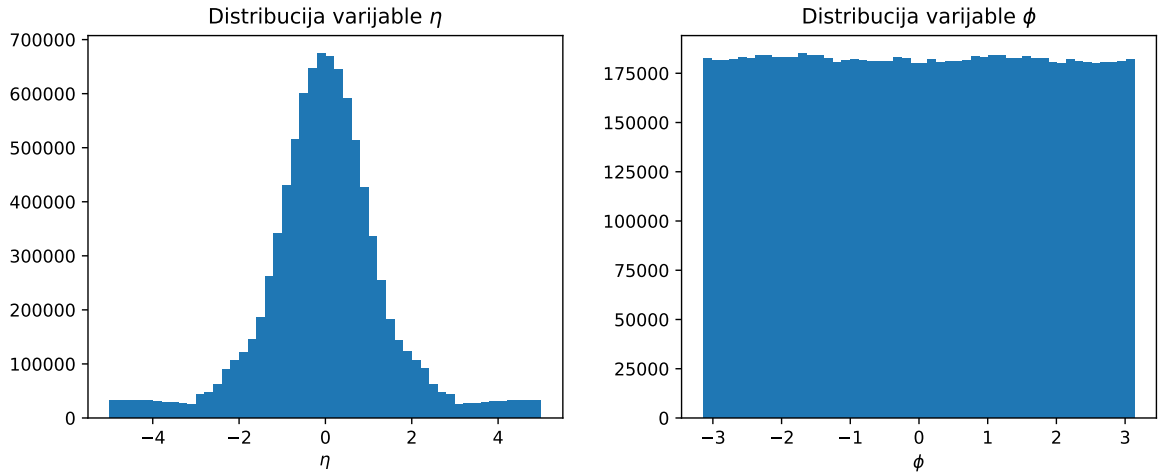
Ovakva problematika motivirala nas je da događaje prikažemo u tenzorskom obliku, odnosno kao sliku. Svaki piksel slike korespondira jednoj čestici iz događaja. Kako kinematičke varijable η i ϕ imaju prostornu interpretaciju, njih smo odabrali kao veličine koje odlučuju o položaju piksela, dok vrijednost veličine p_T određuje vrijednost piksela. Takvu sliku možemo pisati kao $T(x, y)$, pri čemu su x i y koordinate piksela, dok funkcija T opisuje vrijednost transverzalnog momenta na danim koordinatama. Ukoliko više čestica ima istu koordinatu piksela, onda na danom pikselu

zbrajamo njihove vrijednosti. Za sliku s $N \times N$ piksela, koordinate definiramo kao:

$$x \equiv \left[\left(\frac{\eta}{\eta_{\text{max}}} + 1 \right) \times \frac{N}{2} \right] \tag{5.1}$$

$$y \equiv \left[\left(\frac{\phi}{\phi_{\text{max}}} + 1 \right) \times \frac{N}{2} \right] \tag{5.2}$$

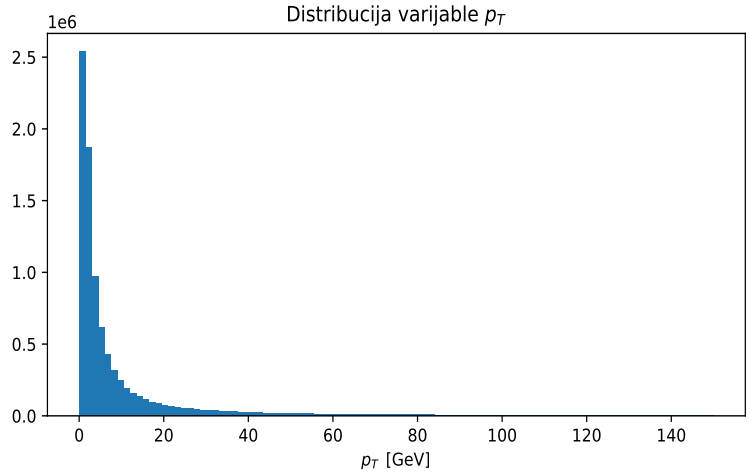
pri čemu $[\ ]$ predstavlja operaciju zaokruživanja. U gornjim jednadžbama, dobivene koordinate su u rangu $[0, N]$. Ovakav zapis ima najviše smisla ako kinematičke varijable imaju vrijednosti u rasponu $[-\eta_{\text{max}}, \eta_{\text{max}}]$ za x koordinatu te $[-\phi_{\text{max}}, \phi_{\text{max}}]$ za y koordinatu. Kad bi neke od prostornih varijabli imale vrlo male standardne devijacije u danom rasponu slike ne bi bile balansirane jer bi pikseli na lokacijama oko očekivane vrijednosti varijable bili puno više zastupljeni. Inspekcijom distribucije kinematičkih varijabli u našim podacima (prikazano na slici 5.1), uočavamo kako je distribucija pseudorapiditeta centrirana oko nule, što je očekivano jer to odgovara točkama detektora koje su najbliže mjestu sudara. Iako je ova distribucija centrirana, standardna devijacija je dovoljno velika da imamo balansirane tenzore. Same vrijednosti pseudorapiditeta ograničene su s $|\eta| < 5$, iz čega zaključujemo da je $\eta_{\text{max}} = 5$. Ovaj podatak je naveden i u opisu problema na stranici LHC Olympics 2020. Limitacija pseudorapiditeta je intuitivna zbog konačne dimenzije detektora, odnosno čestice s $|\eta| > 5$ ne mogu se detektirati.



Slika 5.1: Prikaz distribucija kinematičkih varijabli η i ϕ za skup podataka koji sadrži samo pozadinu. Gotovo identičnu distribuciju prate podaci koji sadrže i pozadinu i signal.

Distribucija kutova ϕ je praktički homogena, što je očekivano zbog osne simetrije problema. Iz gornje distribucije vidljivo je kako je $\phi_{\text{max}} = \pi$. [1].

Inicijalno su generirane slike u kojima je vrijednost piksela određena s p_T, no pomnim proučavanjem distribucije vrijednosti transverzalnog momenta (prikazano na slici 5.2), kao korisnija veličina ispostavila se $\log(p_T + 1)$. Naime, manji broj čestica je visokoenergetsko, stoga su u linearnoj skali takve čestice zasjenjuju većinu niskoenergetskih čestica.

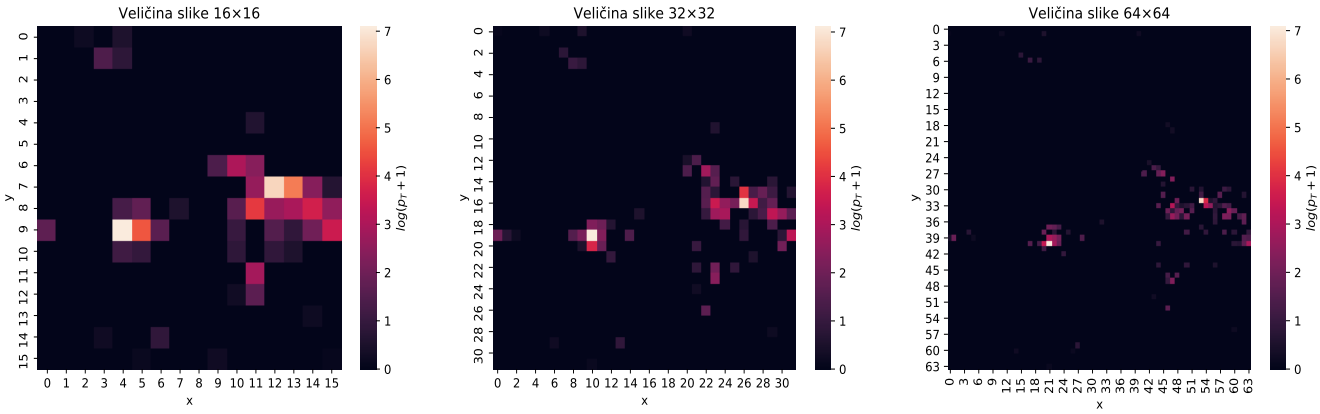


Slika 5.2: Prikaz distribucije transverzalnog momenta za skup podataka koji sadrži samo pozadinu. Kao i kod ostalih varijabli, distribucija je gotovo identična za oba skupa podataka.

Sama veličina generiranih slika je bitna jer je potrebno pronaći balans između dovoljne razlučivosti i prevelike raštrkanosti čestica unutar slike. Zbog konačnog broja piksela postoji intrinzična neodređenost u prostornim koordinatama stoga slike malih dimenzija ne sadrže dovoljno informacija, dok prevelike slike zahtijevaju dublje mreže koje bi potencijalno bile sklone učenju napamet. Prikaz istog događaja za tri različite veličine slike prikazan je na slici 5.3. [2]

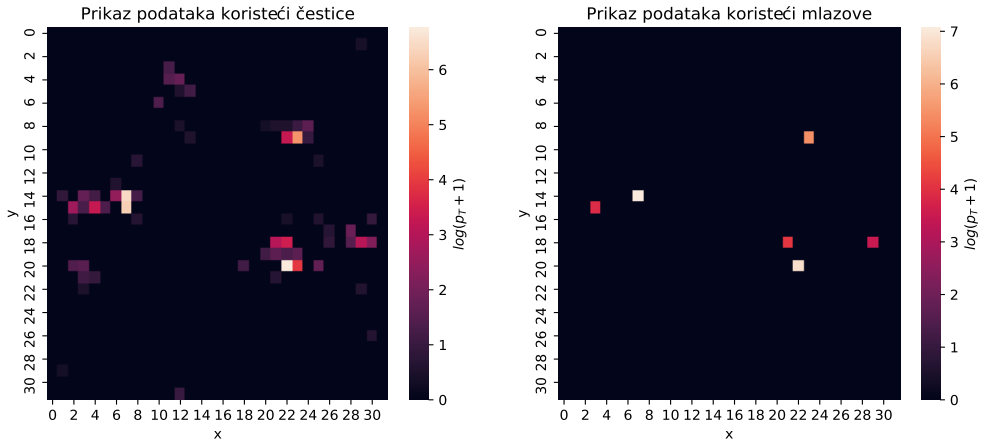
[1] Pošto je ϕ ciklička koordinata, jedina nedoumica bila je vezana uz definiciju kuta, odnosno je li π u rasponu $[0, 2\pi]$ ili $[-\pi, \pi]$

[2] Događaji prikazani na slici 5.3 imaju boju, no to nije zato što je ovaj objekt slika sa 3 kanala boje, već je korišten toplinski prikaz u paketu `seaborn`.



Slika 5.3: Prikaz istog događaja koji sadrži 204 čestica koristeći tri različite rezolucije slika. Povećanjem rezolucije može se uočiti finija struktura reprezentacije događaja, no količina praznog prostora se povećava.

Kao alternativan prikaz događaja koristit ćemo sličan pristup, no umjesto čestica, na slikama ćemo prikazivati mlazove. Procedura za generiranje slika je slična kao i prije, no treba dodati korak u kojem pomoću liste čestica dobijemo listu mlazova. U tu svrhu koristimo paket *PyJet* [21] koji sadrži implementaciju anti-k_T algoritma (opisano u odjeljku 3.4). Za parametar R koristili smo uobičajenu vrijednost 0.4 te je napravljen rez na mlazove čiji transverzalni moment ne zadovoljava $p_T > 20$ GeV.



Slika 5.4: Usporedba prikaza istog događaja koristeći čestice i mlazove dobivene grupiranjem čestica. Algoritam grupiranja je anti-k_T s parametrima $R = 0.4$ i $p_{T_{\text{min}}} = 20$ GeV.

Jedna od prednosti prikazivanja slika koristeći mlazove jest što su filtrirane manje relevantne informacije, poput položaja niskoenergetskih čestica, no takav prikaz u

slikama ostavlja više praznog prostora, odnosno piksela na kojima nema događaja.

Oba skupa podataka dani su u `.h5` formatu te ga je nužno pretvoriti u `NumPy` array format [23] za daljnje procesiranje. Prilikom prvih testiranja modela učitavali smo dođadaje u obliku popisa čestica te smo "*on-the-fly*" stvarali slike. Taj proces je usporavao učenje, stoga smo u fazi pretprocesiranja generirali slike i spremili ih na disk u `NumPy` formatu.

5.2 Opisi modela i učenje mreža

Dvije glavne skupine modela koje ćemo proučavati su konvolucijski autoenkoderi te varijacije U-Net mreže [22]. Naš cilj je detektirati nepoznate čestice (signal) koje se nalaze u drugom skupu podataka (pozadina + signal) te odrediti njenu/njihovu masu.

Osnovna pretpostavka prve metode jest da učenjem autoenkodera na podacima koji sadrže samo pozadinu možemo izvući njene esencijalne značajke te kada bismo evaluirali naš model na skupu podataka koji sadrži nove čestice, rekonstrukcija autoenkodera bila bi loša. U idealnom scenariju, rekonstrukcijska greška događaja koji sadrže nove čestice bit će puno veća u odnosu na rekonstrukcijske greške događaja koji sadrže samo pozadinu. Nažalost, to u većini slučajeva nije slučaj te je potrebna pomna optimizacija same arhitekture autoenkodera.

Druga metoda, odnosno korištenje U-Net arhitekture, motivirana je razvnovrsnom primjenom takvih mreža u problemima segmentacije. Koristeći ovakvu arhitekturu, pokušali smo naučiti mrežu da iz čestičnog prikaza slike pređe u prikaz koristeći mlazove. Drugim riječima, cilj je naučiti mrežu da vrši anti-k_T grupiranje. Potencijalne anomalije bile bi detektirane na isti način kao i u slučaju autoenkodera, odnosno očekujemo da će mreža lošije grupirati u slučaju čestica koje nikada nije vidjela.

Svi modeli napisani su u programskom paketu *PyTorch* [24], koje je razvio Facebook-ov AI Research tim (FAIR). Neke od bitnih značajki ovog paketa su `Autograd` modul koji koristi metodu automatske diferencijacije za efikasnije računanje gradijenata u algoritmima unazadne propagacije i `nn` modul koji omogućava jednostavnu konstrukciju različitih arhitektura. U koraku pretprocesiranja, podaci su bili u formatu `NumPy`, no nužno ih je konvertirati u `torch.Tensor` format kako bi mogli biti prebačeni na GPU. Korištenje grafičke kartice ima ogroman utjecaj u procesu učenja

neuralnih mreža. Naime, GPU uređaji su puno efikasniji u paralelizabilnim procesima, što je većina matematičkih operacija u neuralnim mrežama. Platforma `CUDA` omogućava korištenje GPU uređaja za ovakve kalkulacije. Svi modeli trenirani su koristeći grafičku karticu Nvidia RTX 3070 te verziju `CUDA 11.0`.

5.3 Metoda detekcije anomalija

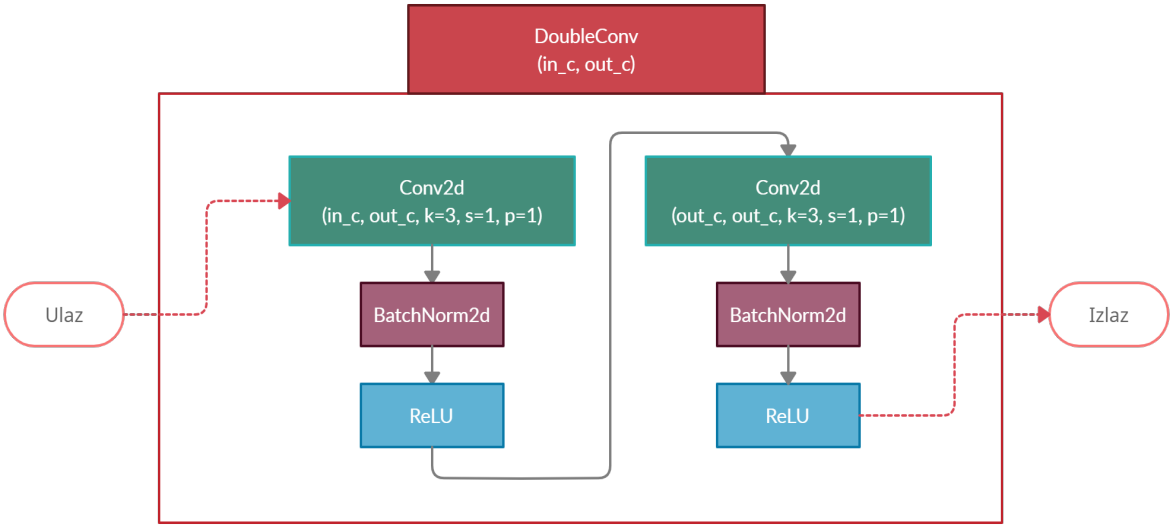
Jednom kada smo istrenirali model na podacima koji sadrže samo pozadinu, idući korak je evaluirati model na podacima koji sadrže pozadinu i signal. Prilikom evaluacije za svaki primjer bilježimo kolika je funkcija gubitka. U autoenkoderskim modelima, funkcija gubitka bit će mjera koliko je loša rekonstrukcija, dok će za U-Net model funkcija gubitka biti mjera netočnosti anti-k_T grupiranja.

Nakon evaluacije, sortiramo događaje prema vrijednosti funkcije gubitka te uzimamo N događaja s najlošijim rezultatom. Za svaki događaj grupiramo čestice koristeći anti-k_T algoritam s parametrom radijusa $R = 0.8$, zbrojimo četverovektore dva najenergetskija mlaza i konačno izračunamo njihovu invarijantnu masu. Ako je zaista mreža lošije rekonstruirala/grupirala događaje koji sadrže signal, onda će distribucija invarijantnih masa imati anomaliju na određenoj masi. O samom odabiru broja N te o preciznijoj definiciji anomalije detaljnije ćemo diskutirati u idućim potpoglavljima.

5.4 Autoenkoderski modeli

Svi autoenkoderski modeli koje smo koristili imaju većinski sličnu strukturu, no razlikuju se u broju slojeva, broju kanala u konvolucijama i slično. Generalna arhitektura modela našeg autoenkoderskog modela prikazana je na slici 5.6. Ulazni tenzor su slike veličine 32×32 sa parametrom `batch_size` $= 64$ ili 128 (algoritam gradijentnog spusta izvršava se na podskupu podataka veličine `batch_size`). Nadalje, podaci ulaze u `Double_Conv` blok (struktura prikazana na slici 5.5), čiji su parametri `in_c` (broj ulaznih kanala) te `out_c` (broj izlaznih kanala). Nakon `Double_Conv` bloka slijedi Max pooling sloj (`MaxPool2d`) u kojem se smanjuju dimenzije slike za faktor 2 (odnosno ukupan broj piksela za faktor 4) koristeći grupiranje odabirom maksimalnog piksela. Ovakav proces se ponavlja (konvolucije popraćene Max pooling slojevima) više puta te finalnu smanjenu reprezentaciju nazivamo enkodirani tenzor. U enkoderu događaju se dva konkurentska efekta; kroz konvolucijske slojeve broj

kanala se povećava (što povećava veličinu reprezentacije), dok Max pooling slojevi smanjuju prostorne dimenzije. Ako želimo imati funkcionalan autoenkoder, onda nam je cilj da enkodirana verzija podataka bude manje veličine od ulazne. Dakle, ako broj piksela kroz Max pooling slojeve pada sa faktorom 4, broj kanala ne smije rasti brže.

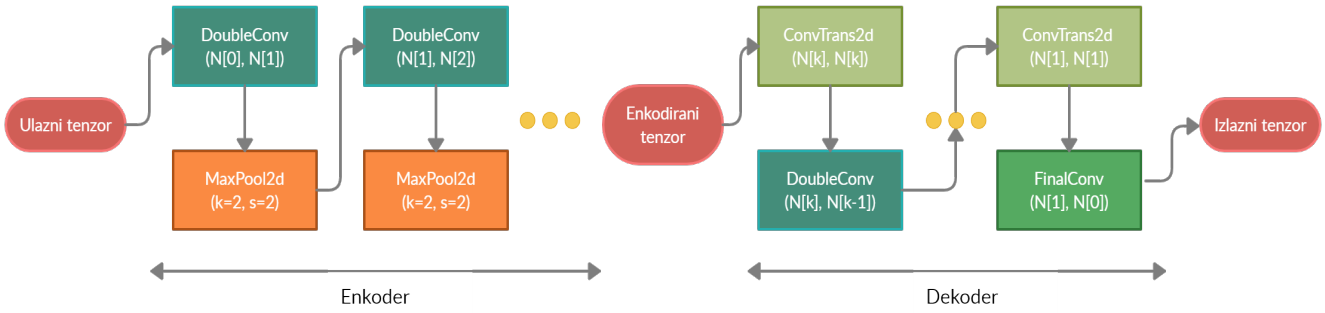


Slika 5.5: Strukturalni prikaz `Double_Conv` bloka. Ulazni podaci dolaze u konvolucijski sloj (`Conv2d`), gdje je ulazni (`in_c`) i izlazni (`out_c`) broj kanala određen samom inicijalizacijom bloka. U prikazu, parametri `k`, `s` i `p` predstavljaju `kernel_size`, `stride` te `padding`. Nadalje, podaci se normaliziraju u batch normalization sloju (`BatchNorm2d`) [25]. Nakon ove operacije slijedi aktivacijska funkcija Rectified Linear Unit (`ReLU`). Ova tri koraka se ponavljaju, no u novom konvolucijskom sloju, broj ulaznih parametara je različit (ovaj put je broj ulaznih i izlaznih parametara jednak `out_c`).

Dekoder je strukturno simetričan enkoderu, no operacije su suprotnog karaktera. Umjesto Max pooling slojeva imamo transponirane konvolucije (eng. *transposed convolution, deconvolution*), pomoću kojih povećavamo rezoluciju slike (korišteni parametri su `kernel_size` $= 2$, `stride` $= 2$). Dok Max pooling slojevi nemaju težine jer su fiksni algoritmi, transponirane konvolucije imaju filtere s težinama slično kao konvolucijski slojevi.[3] Ovakvim simetričnim operacijama postepeno vraćamo originalnu prostornu dimenziju, a paralelno smanjujemo broj kanala. Posljednji sloj u mreži (`FinalConv`) sastoji se od jednostruke konvolucije sa parametrom `kernel_size` $= 1$ te kao izlaz imamo sliku koju nazivamo izlazni tenzor.

[3]Bitno je naglasiti da transponirane konvolucije nisu konvolucije u matematičkom smislu, no zbog sličnosti operacija nazivamo ih tako.

Prilikom inicijalizacije modela unosi se lista `N` koja određuje koliko će biti `Double_Conv` blokova te koliko će svaki imati kanala. Ulazni broj kanala (koji je u našem slučaju 1) je označen kao `N[0]` dok je broj kanala u bottleneck dijelu mreže (enkodirana verzija podataka) jednak `N[k]`[4].



Slika 5.6: Shematski prikaz korištenog konvolucijskog autoenkodera. Kao ulaz u mrežu imamo ulazni tenzor. Uzastopnim prolaskom kroz `DoubleConv` i `MaxPool2d` slojeve dobivamo enkodiranu verziju ulaznog tenzora. Dekoderom, koji je simetričan enkoderu, vraćamo originalne dimenzije slici te kao izlaz iz mreže imamo izlazni tenzor.

Bitan parametar u našem slučaju jest dimenzija enkodiranih podataka. U svim modelima koristili smo ulazne slike dimenzija 32×32, stoga je dimenzija ulaznih podataka dana sa $D_i = 1 \times 32 \times 32$ (faktor 1 dolazi od jednog ulaznog kanala, pošto jedinu "boju" predstavlja vrijednost p_T). Svaki sloj enkodera smanjuje prostornu veličinu za faktor 4, stoga za k slojeva u enkoderu, omjer enkodirane dimenzije D_e i ulazne dimenzije D_i dan je sa:

$$D_{e/i} \equiv \frac{D_e}{D_i} = \frac{\mathrm{N[k]} \times 32 \times 32 \times 4^{-k}}{1 \times 32 \times 32} = \frac{\mathrm{N[k]}}{4^k}\,, \tag{5.3}$$

pri čemu N[k] predstavlja broj kanala enkodiranog tenzora. Ova veličina bit će nam ključna u konstrukciji različitih arhitektura modela. Nakon testiranja preko 50 različitih modela s različitim parametrima poput veličine filtera u konvolucijskim slojevima te modela s jednostavnim slojevima u bottleneck dijelu mreže i slično, ispostavilo se da je ovaj omjer najrelevantniji parametar.

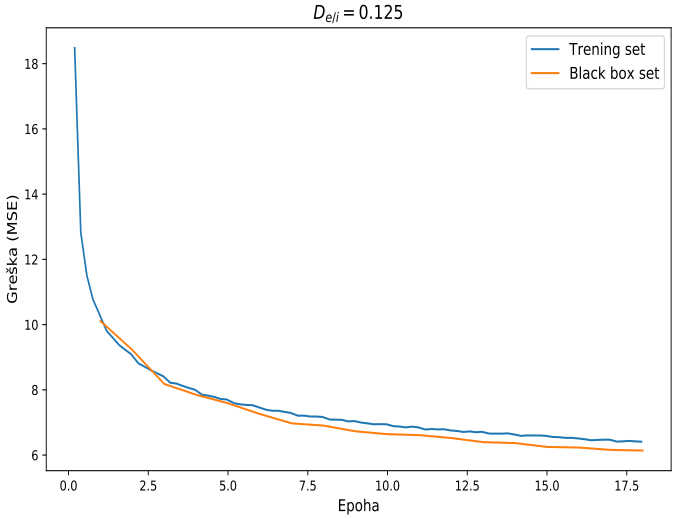
U daljnjim raspravama, radi jednostavnosti, skup podataka koji sadrži samo pozadinu zvat ćemo trening set, dok ćemo skup podataka koji sadrži pozadinu i signal

[4]N je prikazan u formi liste od k elemenata makar ima $k+1$ mogućih indeksa. Razlog tome jest što je ulazni broj kanala N[0] fiksan te se odvojeno unosi, no radi jednostavnosti ovdje je sve prikazano u obliku jedne liste N.

zvati black box set. Prvo ćemo prikazati rezultate koristeći modele sa slikama čestica, a nakon toga ćemo proučiti rezultate dobivene koristeći slike mlazova.

5.4.1 Autoenkoderski čestični model sa omjerom $D_{e/i} = 0.125$

Prilikom učenja ovog modela na trening setu, korištena je veličina `batch_size` $= 128$ te adaptivna stopa učenja `learning_rate` $= 0.0001$ sa stopom raspada 0.9 (nakon svake epohe stopa učenja se smanjuje za faktor 0.9). Model je učen 18 epoha na trening setu koristeći optimizacijski algoritam `ADAM` te je nakon svake epohe napravljena probna evaluacija na manjem podskupu black box podataka (prikazano na slici 5.7), kako bismo mogli uočiti potencijalni overfit, no zanimljivo je da je greška na black box setu nešto niža od greške na trening setu. Vrijeme učenja je otprilike 1 sat, što je razumno ako uzmemo u obzir da se trening set sastoji od 10^6 tenzora veličine 32×32.



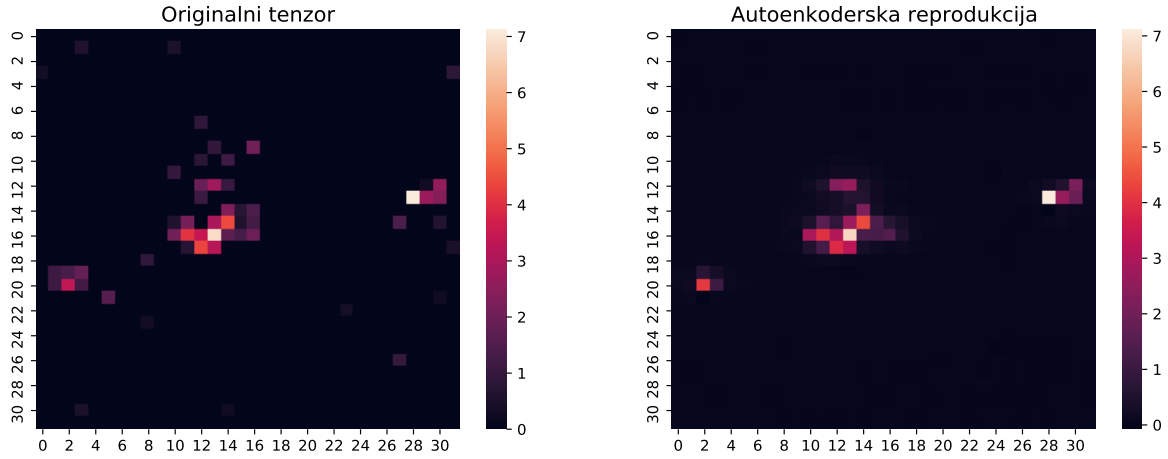
Slika 5.7: Prikaz greške u ovisnosti o epohi za trening set (plava linija) i black box set (narančasta linija).

Lista `N` koja određuje arhitekturu i dubinu mreže u ovom slučaju dana je sa:

$$\texttt{N} = [64, 128, 256, 128, 128]\,. \tag{5.4}$$

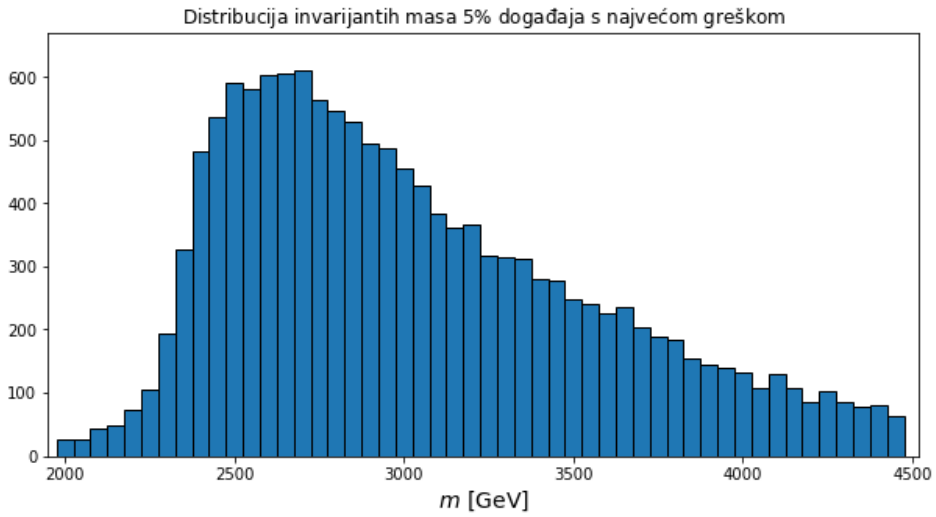
Dakle imat ćemo 5 `DoubleConv` blokova u enkoderu i 5 `DoubleConv` blokova u dekoderu. Ulazna veličina tenzora bit će $1 \times 32 \times 32$, dok će enkodirana veličina biti $128 \times 1 \times 1$ (odavde vidimo da je enkodirana dimenzija 12.5% veličine ulazne dimenzije). Na slici 5.13 je prikazan primjer autoenkoderske rekonstrukcije za jedan događaj. Vidimo kako je autoenkoder dobro rekonstruirao grupiranije, visokoener-

getske čestice, dok udaljene niskoenergetske čestice nisu rekonstruirane. Zbog niskodimenzionalne reprezentacije metode učenja (MSE kao funkcija gubitka), autoenkoder je naučio da je prioritetnije rekonstruirati dominantne čestice.



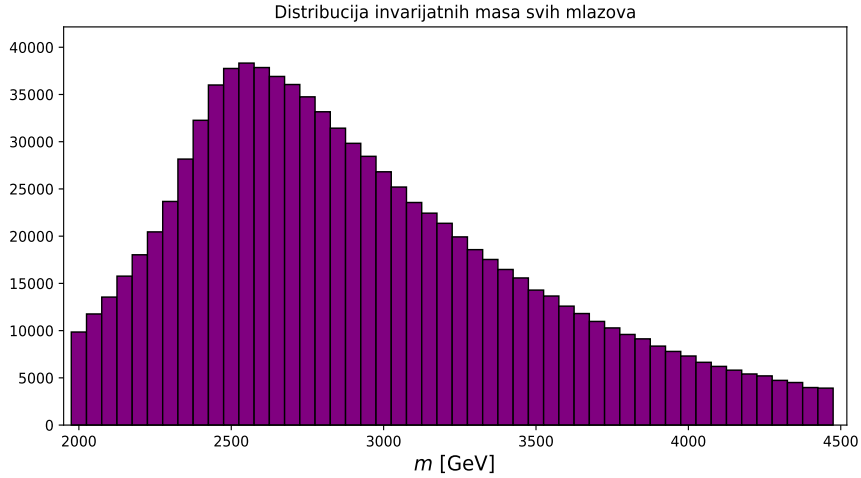
Slika 5.8: Primjer autoenkoderske reprodukcije za autoenkoderski čestični model sa omjerom $D_{e/i} = 0.125$.

Nakon učenja modela, cijeli black box set evaluiramo te spremamo vrijednosti grešaka za svaki događaj. Nadalje, sortiramo događaje po vrijednosti greške i uzimamo 5% događaja sa najvećom greškom. Te događaje grupiramo u mlazove koristeći anti-k_T algoritam te računamo invarijantnu masu 2 najenergetskija mlaza. Zatim promatramo distribuciju tih masa, kao što je prikazano na slici 5.9.



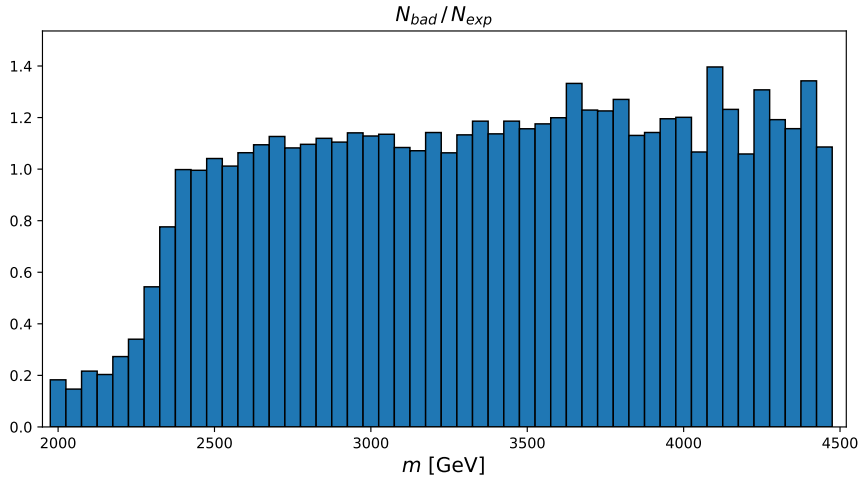
Slika 5.9: Prikaz distribucije invarijantnih masa vodećih mlazova za 5% događaja sa najvećom greškom. Distribucija je dobivena evaluacijom modela $D_{e/i} = 0.125$ na black box setu.

Ovakav prikaz nam nije vrlo koristan, pošto u samim podacima postoji različit broj mlazova na određenim energijama, a nas zanima mjera odstupanja. Motivirani time, izračunali smo distribuciju masa za cijeli black box (invarijantne mase dva najenergetskija mlaza) kao što je prikazano na slici 5.10. U prikazu svih distribucija i računa uzet je interval masa $[2500, 4500]$ GeV pošto je broj mlazova sa energijama izvan tog intervala malen.



Slika 5.10: Distribucija svih invarijantnih masa vodećih mlazova koji se nalaze u black box setu.

Distribuciju gubitaka u ovisnosti o masi dijelimo s distribucijom masa te skaliramo s brojem podataka (5% ukupnog broja događaja) kako bismo dobili relevantan prikaz odstupanja. Ovaj omjer nazivamo N_{bad}/N_{exp} (broj događaja iz distribucije 5% najgore rekonstruiranih tenzora podijeljen sa očekivanim brojem događaja).

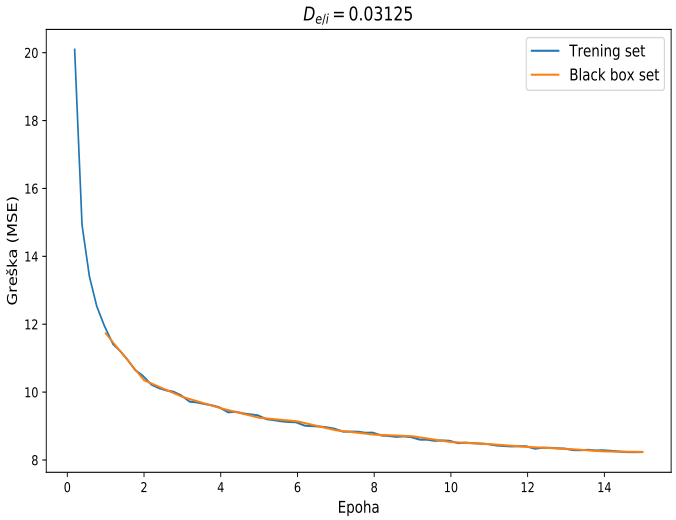


Slika 5.11: Prikaz distribucije omjera broja mlazova koji se nalaze u dijelu distribucije sa najvećom greškom i očekivanog broja mlazova na danoj masi za autoenkoderski čestični model sa omjerom $D_{e/i} = 0.125$.

Kao potencijalne anomalije odlučili smo nazvati događaje koji odstupaju za faktor 2 od očekivane vrijednosti, što u ovom slučaju nije detektirano (najveće odstupanje je za faktor 1.36). Za mase manje od 2500 GeV, uočavamo kako imamo manje događaja od očekivanog. To implicira da se većina takvih događaja dobro rekonstruirala, pošto ih je manje u 5% događaja sa najvećom greškom. Promatrali smo i različite širine stupaca u prikazu N_{bad}/N_{exp} (širine stupaca direktno su povezane sa širinom raspada hipotetske čestice), no nismo uočili anomalije u black box setu te ćemo u idućim modelima postupno smanjivati dimenziju enkodiranih podataka.

5.4.2 Autoenkoderski čestični model sa omjerom $D_{e/i} = 0.03125$

Prilikom učenja ovog modela korištena je ista procedura kao i za treniranje prošlog modela. Jedina razlika jest što smo ovaj model trenirali 15 epoha te je greška ovaj put slična za oba skupa podataka. Prikaz toka greške kroz epohe učenja možemo vidjeti na slici 5.12.



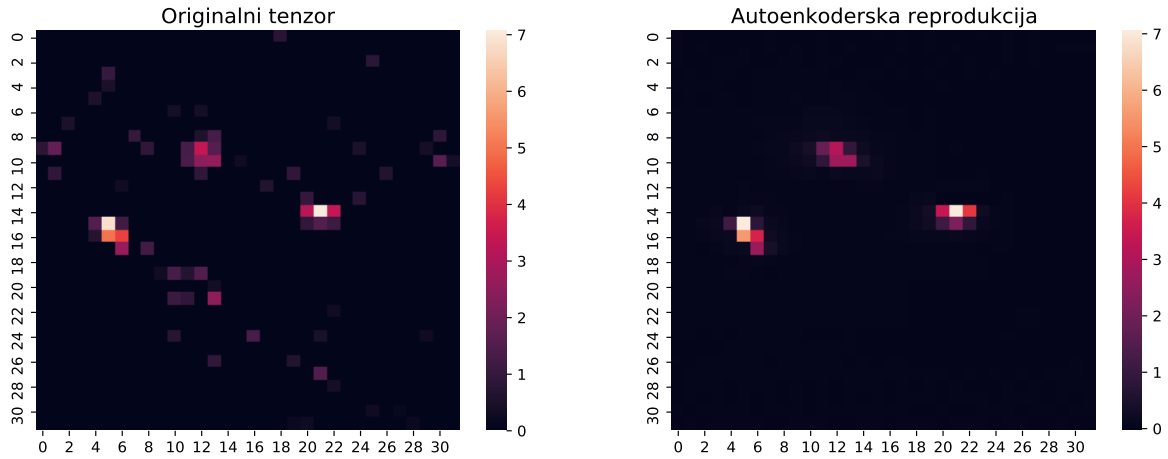
Slika 5.12: Prikaz greške u ovisnosti o epohi za trening set (plava linija) i black box set (narančasta linija).

Lista `N` koja određuje arhitekturu i dubinu mreže u ovom slučaju dana je sa:

$$\texttt{N} = [64, 128, 256, 64, 32]\,. \tag{5.5}$$

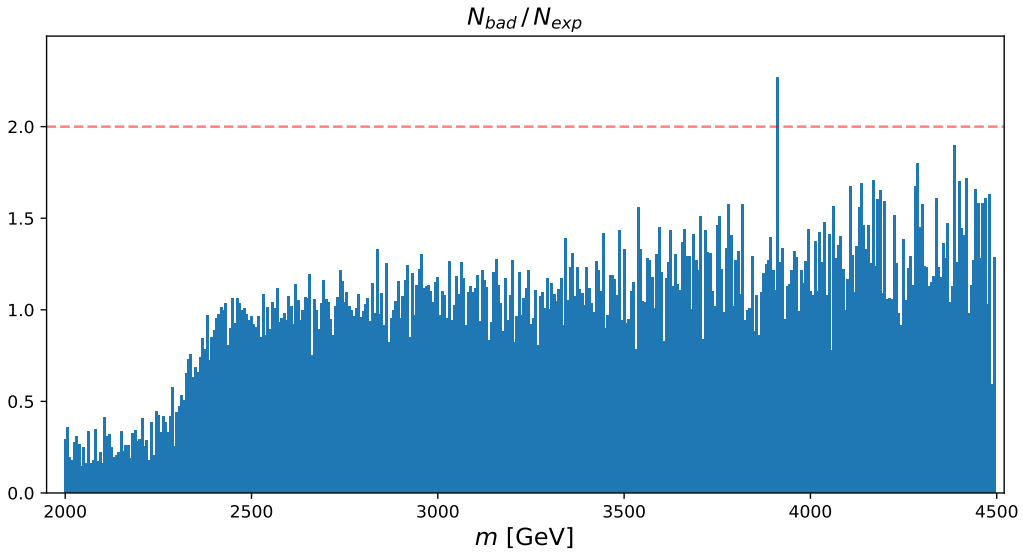
I u ovom modelu ćemo imati 5 `DoubleConv` blokova u enkoderu i 5 `DoubleConv` blokova u dekoderu. Ulazna veličina tenzora bit će $1\times32\times32$, dok će enkodirana veličina biti $32 \times 1 \times 1$ iz čega vidimo da je enkodirana dimenzija 3.125% veličine ulazne di-

menzije. Na slici 5.13 je prikazan primjer autoenkoderske rekonstrukcije za jedan događaj. Kada usporedimo više rekonstrukcija (prikazano u dodatku A), uočavamo kako je ovaj model u prosjeku rekonstruirao manje detalja, što je očekivano s obzirom da je latentni prostor 4 puta manji u odnosu na prošli model sa $D_{e/i} = 0.125$.



Slika 5.13: Primjer autoenkoderske reprodukcije za autoenkoderski čestični model s omjerom $D_{e/i} = 0.03125$.

Nakon evaluacije modela na black box setu, ponovno smo promatrali invarijantnu masu 2 vodeća mlaza koji pripadaju 5% događaja sa najvećom greškom. Povećanjem rezolucije distribucije, uspjeli smo uočiti anomaliju, odnosno masu na kojoj je omjer $N_{bad}/N_{exp} > 2$.



Slika 5.14: Prikaz distribucije omjera N_{bad}/N_{exp} za autoenkoderski čestični model sa omjerom $D_{e/i} = 0.03125$. Horizontalna crvena linija predstavlja granicu koja određuje koje događaje označavamo kao anomalne.

Vrijednost mase i njenog odstupanja na kojoj smo detektirali ovu anomaliju iznose:

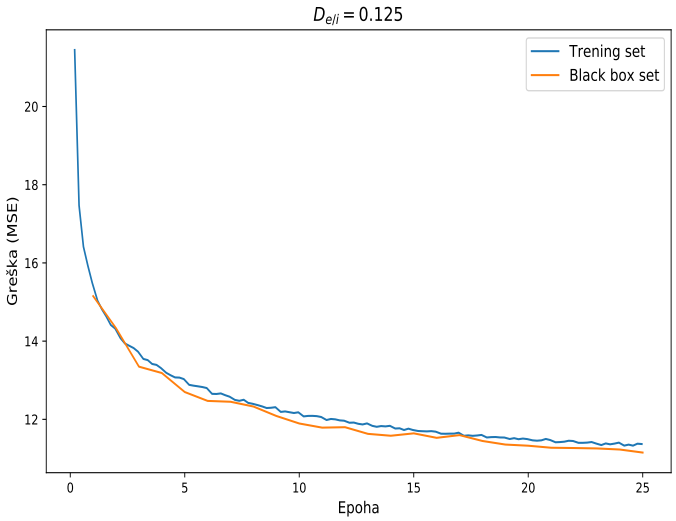
$$m = (3912.5 \pm 5)\ \text{GeV}\,, \tag{5.6}$$

$$N_{bad}/N_{exp} = 2.268\,, \tag{5.7}$$

pri čemu neodređenost mase proizlazi iz širine stupaca, što možemo interpretirati kao veličinu proporcionalnu širini raspada čestice. Možemo uočiti kako čestice na masama blizu 4500 GeV imaju isto visoke vrijednosti odstupanja, no na ovim masama bi trebalo dodatno uzeti u obzir i neodređenosti koje dolaze zbog manjka događaja sa većim vrijednostima inavarijatnih masa. Smanjenje latentnog prostora dovelo je do poboljšanja detekcije anomalnih događaja i zbog toga ćemo probati dodatno smanjiti dimenzije te vidjeti hoće li se rezolucija detekcije anomalija poboljšati.

5.4.3 Autoenkoderski čestični model sa omjerom $D_{e/i} = 0.015625$

Ovaj model učen je na identičan način kao i prošli modeli, uz iznimku da smo model učili 25 epoha zbog sporije konvergencije (prikazano na slici 5.15).

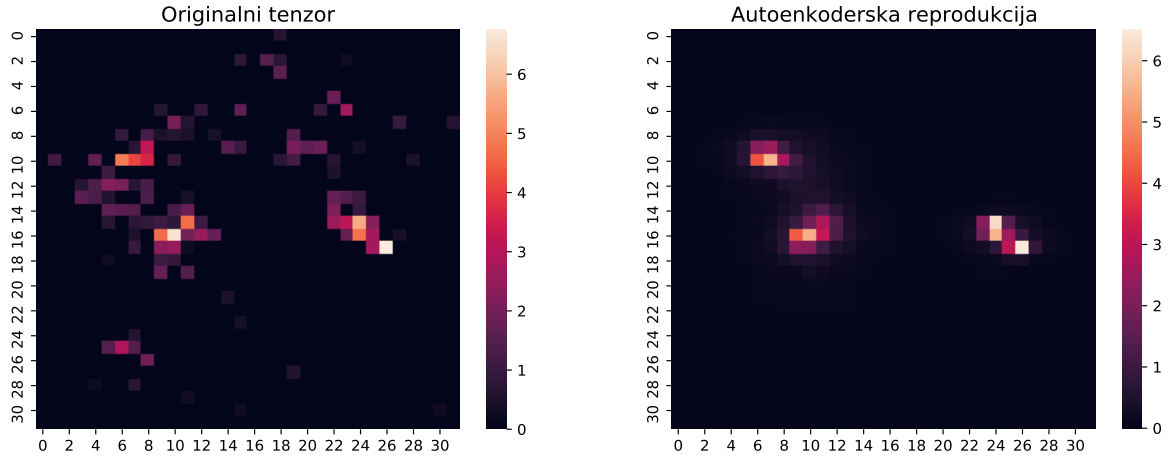


Slika 5.15: Prikaz greške u ovisnosti o epohi za trening set (plava linija) i black box set (narančasta linija).

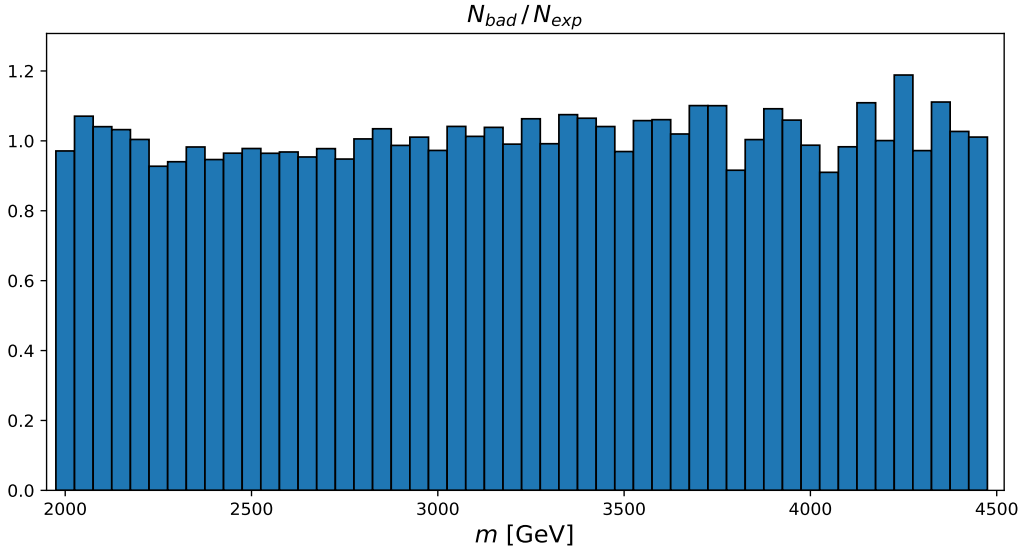
Arhitektura ovog modela ponovno je određena listom `N` koja određuje broj i veličine `Double_Conv` blokova te glasi:

$$\texttt{N} = [64, 128, 256, 64, 16]\,. \tag{5.8}$$

Na slici 5.16 možemo primijetiti kako je autenkoderska rekonstrukcija sve gora, što je posljedica izrazito niske dimenzionalnosti enkodiranog tenzora. Za ovaj model, dimenzija enkodiranog tenzora je svega $\sim 1.6\%$ dimenzije ulaznog tenzora.



Slika 5.16: Primjer autoenkoderske reprodukcije za autoenkoderski čestični model sa omjerom $D_{e/i} = 0.015625$.



Slika 5.17: Prikaz distribucije omjera N_{bad}/N_{exp} za autoenkoderski čestični model s omjerom $D_{e/i} = 0.015625$.

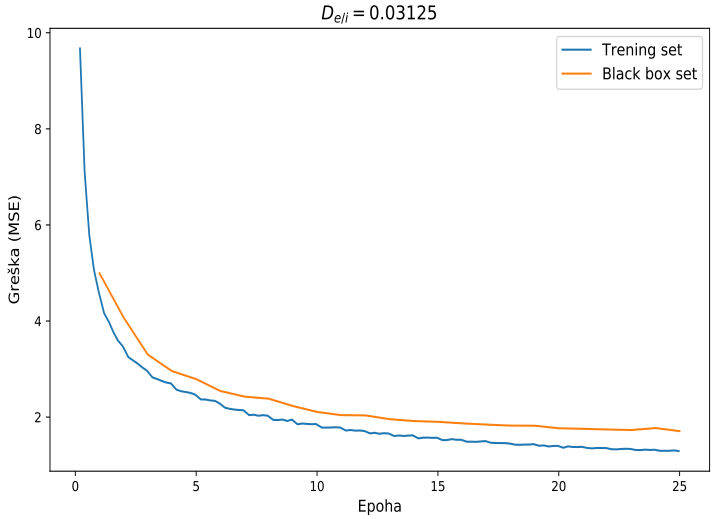
Iz prikaza distribucije omjera N_{bad}/N_{exp} (slika 5.17) vidimo kako ovaj model nije uspio detektirati anomalije. Zanimljivo je da za ovaj model nemamo manjak loše rekonstruiranih događaja na nižim masama. To možemo objasniti nemogućnošću modela da opiše dobro podatke na ikojoj masenoj skali koristeći latentni prostor premalih dimenzija, odnosno model nema preferirani podskup podataka koji je uspio

bolje naučiti od ostalih, kao što je bio slučaj do sada.

U idućem potpoglavlju promotrit ćemo rezultate za model koji je učen na slikama mlazova te usporediti rezultate sa modelima koji su učeni koristeći slike čestica.

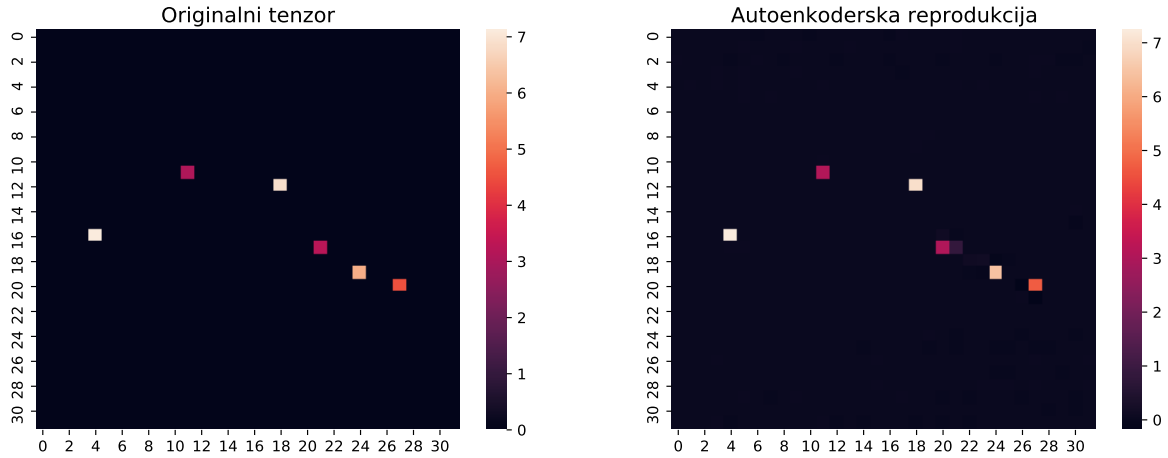
5.4.4 Autoenkoderski mlazni model sa omjerom $D_{e/i} = 0.03125$

Ova vrsta modela koristi istu arhitekturu kao i čestični modeli te je sam proces učenja vrlo sličan. Ovaj model učen je 25 epoha (prikazano na slici 5.18). Za razliku od čestičnih modela, ovdje uočavamo kako je greška na black box setu u prosjeku veća od greške trening seta, što je inače očekivano.

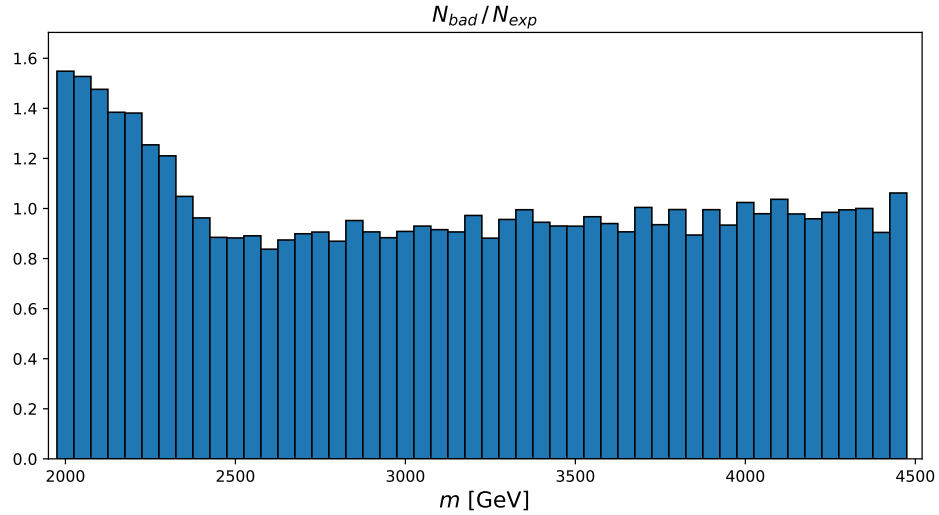


Slika 5.18: Prikaz greške u ovisnosti o epohi za trening set (plava linija) i black box set (narančasta linija).

Arhitektura mreže je identična kao i za čestični model s istim omjerom $D_{e/i}$ te je određena listom 5.5. Na slici 5.19 možemo promotriti autoenkodersku rekonstrukciju mlazova. Možemo primijetiti suptilne razlike u rekonstrukciji na jednom mlazu. Ako promotrimo distribuciju masa događaja s najvećom greškom (prikazano na slici 5.20), zanimljivo je uočiti kako mlazni modeli imaju više loše konstruiranih događaja s manjom masom. Ovaj fenomen može se jednostavno objasniti činjenicom da događaji koji imaju nižu invarijantnu masu imaju u prosjeku više niskoenergetskih mlazova koji su međusobno udaljeni, a takve uzorke nije jednostavno modelirati koristeći autoenkoder manje latentne dimenzije.

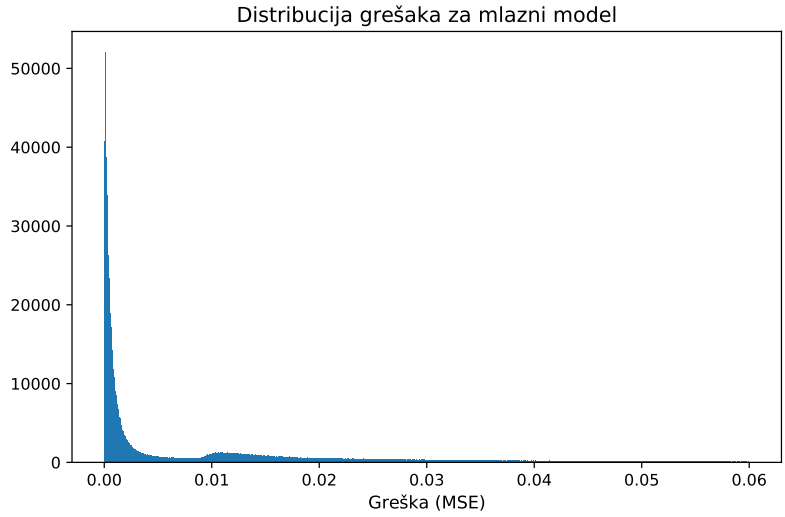


Slika 5.19: Primjer autoenkoderske reprodukcije za autoenkoderski mlazni model sa omjerom $D_{e/i} = 0.03125$.



Slika 5.20: Prikaz distribucije omjera broja mlazova koji se nalaze u dijelu distribucije s najvećom greškom i očekivanog broja mlazova na danoj masi za autoenkoderski mlazni model s omjerom $D_{e/i} = 0.03125$.

Ako su niskoenergetski mlazovi stvarno uzrok ovog fenomena, onda bismo očekivali da će distribucija grešaka biti bimodalna. Najmanje greške očekujemo za viskoenergetske događaje, u kojima imamo malen broj mlazova, a veće greške za niskoenergetske događaje. Ovaj fenomen, makar nije jako izražen, možemo uočiti na slici 5.21. Dio događaja koji ima veće energije je dobro rekonstruiran (malena greška) i porastom greške pada zastupljenost događaja. Nadalje vidimo kako pri većim greškama ponovno imamo veći broj događaja, a upravo tu skupinu čine niskoenergetski događaji sa više mlazova.



Slika 5.21: Prikaz distribucije grešaka za mlazni autoenkoder s omjerom $D_{e/i} = 0.03125$.

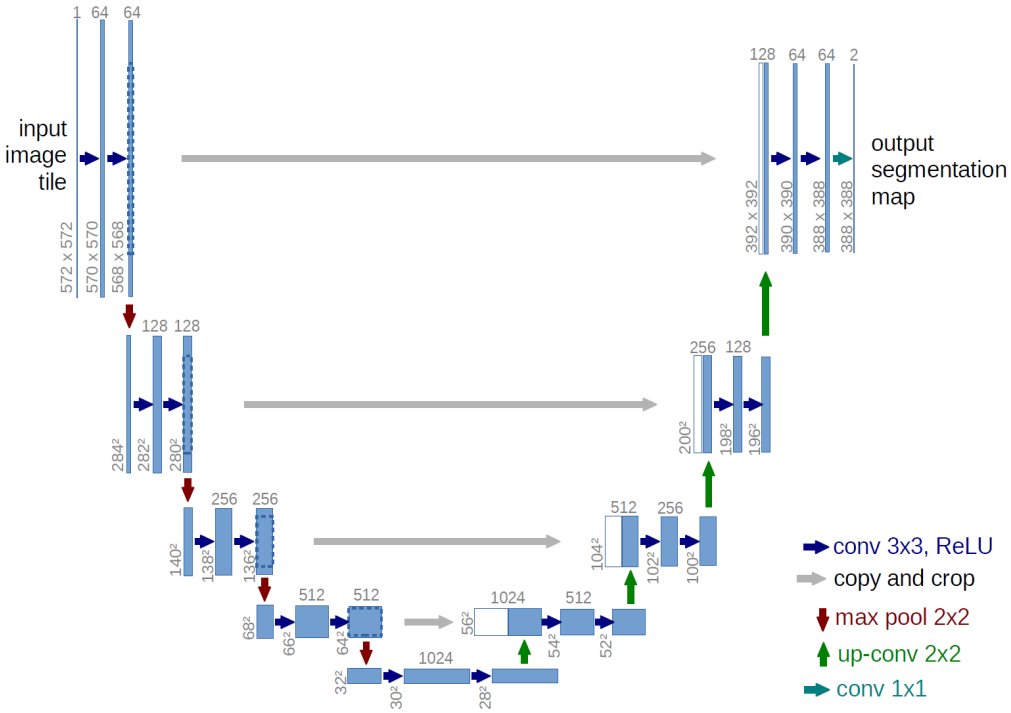
Iz navedenih razloga smatramo kako mlazni prikaz nije dobar za detekciju anomalija, pošto niskoenergetski događaji koji imaju veću grešku zasjenjuju grešku koja dolazi od prisutstva novih signala. Za naš problem, distribucija grešaka morala bi uniformno padati, odnosno imati samo jedan rep distribucije u kojem potencijalno možemo detektirati anomalne događaje. Mlazni autoenkoderi sa većim latentnim prostorom povećavaju navedeni fenomen (izraženija bimodalnost), dok autoenkoderi s manjim latentnim prostorom nemaju dovoljnu ekspresivnu moć za opisivanje podataka te je rekonstrukcijska greška anomalnih događaja reda veličine grešaka rekonstrukcije pozadine.

5.5 U-Net model

Razvojem neuralnih mreža, primjena dubokih konvolucijskih arhitektura postala je česta u medicini, konkretno za segmentaciju CT slika. 2015. godine razvijena je U-Net mreža, koja je našla primjenu u mnogim područjima zbog svoje fleksibilnosti. Iako je originalno mreža osmišljena za svrhe segmentacije (određivanje pripadnosti dijelova slike određenoj klasi), ovakva arhitektura ima visoku učinkovitost i u problemima gdje je potrebno ekstrahirati dijelove slika na ne-binaran način. Primjer takve uporabe jest u audio separaciji gdje se iz spektrograma pjesme odvaja dio spektrograma koji pripada nekom instrumentu. Ovakva adaptivnost modela nas je motivirala da pokušamo kreirati mrežu koja bi učila kako grupirati čestice koristeći anti-k_T algoritam, odnosno, naučila imitirati navedeni algoritam. Zbog toga kao ulaz

u mrežu koristimo slike čestica, dok izlazne slike uspoređujemo sa slikama odgovarajućih mlazova. Ideja za detekciju anomalija ostaje ista te očekujemo da će greška na nepoznatim događajima biti veća.

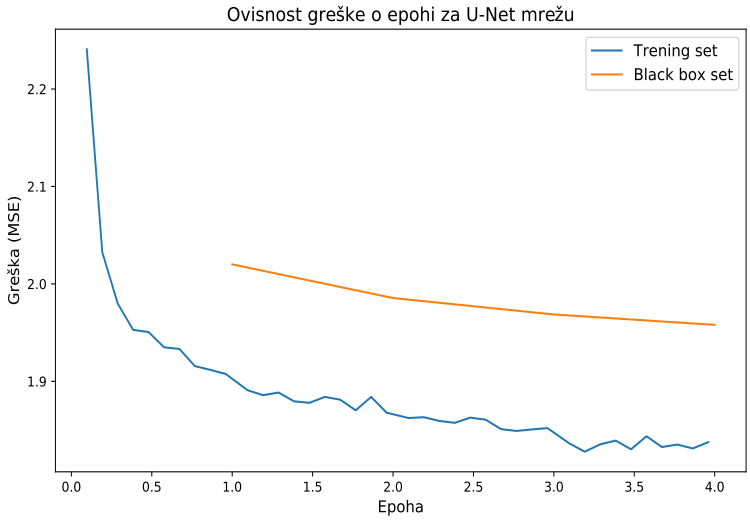
Arhitektura U-Net modela (prikazano na slici 5.22) nalikuje na autoenkoder, no za U-Net nije nužno da dimenzija latentnog prostora bude manja od dimenzije ulaznog vektora. Analogno našem modelu, dvostruke konvolucije su gradivni blokovi mreže (u našem autoenkoderskom modelu smo koristili *padding* da dimenzije tenzora ostanu iste nakon konvolucija). Najveća razlika jest postojanje takozvanih *skip-connection* dijelova (prikazano sivom strelicom) pomoću kojih "lijepimo" dijelove enkodera na simetrično pozicionirane dijelove dekodera duž kanalne dimenzije. Ovakvav mehanizam omogućava dekoderu pristup ranijim značajkama iz slojeva enkodera te omogućava tok gradijenta kroz njih što pomaže procesu učenja.



Slika 5.22: Strukturalni prikaz arhitekture U-Net mreže. Mreža se sastoji od enkodera i dekodera te *skip-connection* veza koje omogućavaju dekoderu da pristupi ranijim značajkama iz enkodera. [22]

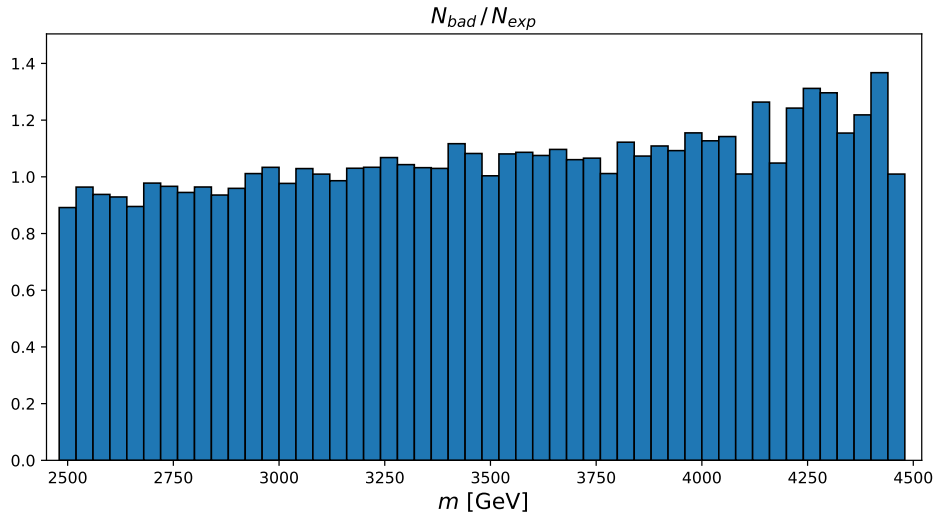
Proces učenja trajao je samo 4 epohe (prikazano na slici 5.23) zbog brze konvergencije. Hiperparametri učenja (optimizacijski algoritam, stopa učenja i sl.) identični su kao za autoenkoderske modele. Lista `N` koja određuje strukturu mreže dana je s:

$$\texttt{N} = [64, 128, 256, 512] \tag{5.9}$$



Slika 5.23: Prikaz greške u ovisnosti o epohi za trening set (plava linija) i black box set (narančasta linija).

U dodatku B prikazano je nekoliko nasumično odabranih događaja te usporedba segmentacije dobivene U-Net mrežom s mlazovima dobivenih koristeći anti-k_T algoritam. Nadalje evaluiramo model na black box setu te promatramo distribuciju omjera broja mlazova koji se nalaze u dijelu distribucije s najvećom greškom i očekivanog broja mlazova (prikazano na slici 5.24). Neovisno o širini stupca histograma u prikazu distribucije, nismo uspjeli uočiti anomalne događaje.



Slika 5.24: Distribucija omjera broja mlazova koji se nalaze u dijelu distribucije s najvećom greškom i očekivanog broja mlazova na danoj masi U-Net model.

U-Net mreža po svojoj konstrukciji nije namijenjena da opisuje ključna svojstva podataka, već da nauči izvršavati operacije poput segmentacije. No, neovisno, zbog

autoenkoderske strukture, postojala je mogućnost da gorom imitacijom anti-k_T grupacije detektiramo potencijalne anomalije unutar podataka.

6 Zaključak

U ovom radu testirane su različite metode strojnog učenja za detekciju anomalija u detektorskoj fizici čestica. Korištena su dva skupa podataka: podaci koji sadrže samo pozadinu i podaci koje sadrže pozadinu i nove čestice. Metoda detekcije koristeći konvolucijske autoenkodere zasniva se na ideji da se mogu naučiti esencijalne značajke pozadine pomoću kojih ćemo razlikovati nove, anomalne događaje, dok se U-Net metoda bazira na učenju anti-k_T grupiranja. Tenzori, koji služe kao ulaz u neuralnu mrežu, generirani su vektorima (η, ϕ, p_T) čestica. Svaki događaj, koji u prosjeku sadrži 150 čestica, predstavljen je jednom slikom.

Prilkom evaluacije čestičnih autoenkoderskih modela, uočeno je da je samo jedan čestični autoenkoderski model uspio detektirati anomaliju s masom $m = 3912.5$ GeV, što je blizu stvarnoj vrijednosti mase anomalne čestice $m_a = 3800$ GeV[5]. Iako je autoenkoder s parametrom $D_{e/i} = 0.03125$ uspio pronaći anomaliju, smatramo kako je ovaj rezultat nestabilan iz više razloga. Prvo, mogućnost detekcije anomalije izrazito je osjetljiva na dimenziju latentnog prostora, stoga ovaj model nije dobar u generalizaciji na druge podatke i drugačije anomalije. Nadalje, samo odstupanje N_{bad}/N_{exp} nije dovoljno veliko da možemo zasigurno tvrditi kako je zaista riječ o anomaliji.

Autoenkoderski modeli koji koriste mlazove nisu uspjeli detektirati anomalije, a njihov neuspjeh se krije u bimodalnosti distribucije grešaka. Rekonstrukcijske greške, koje bi proizašle iz lošeg performansa na evaluaciji anomalnih događaja, zasjenjene su visokim vrijednostima grešaka rekonstrukcije niskoenergetskih mlazova.

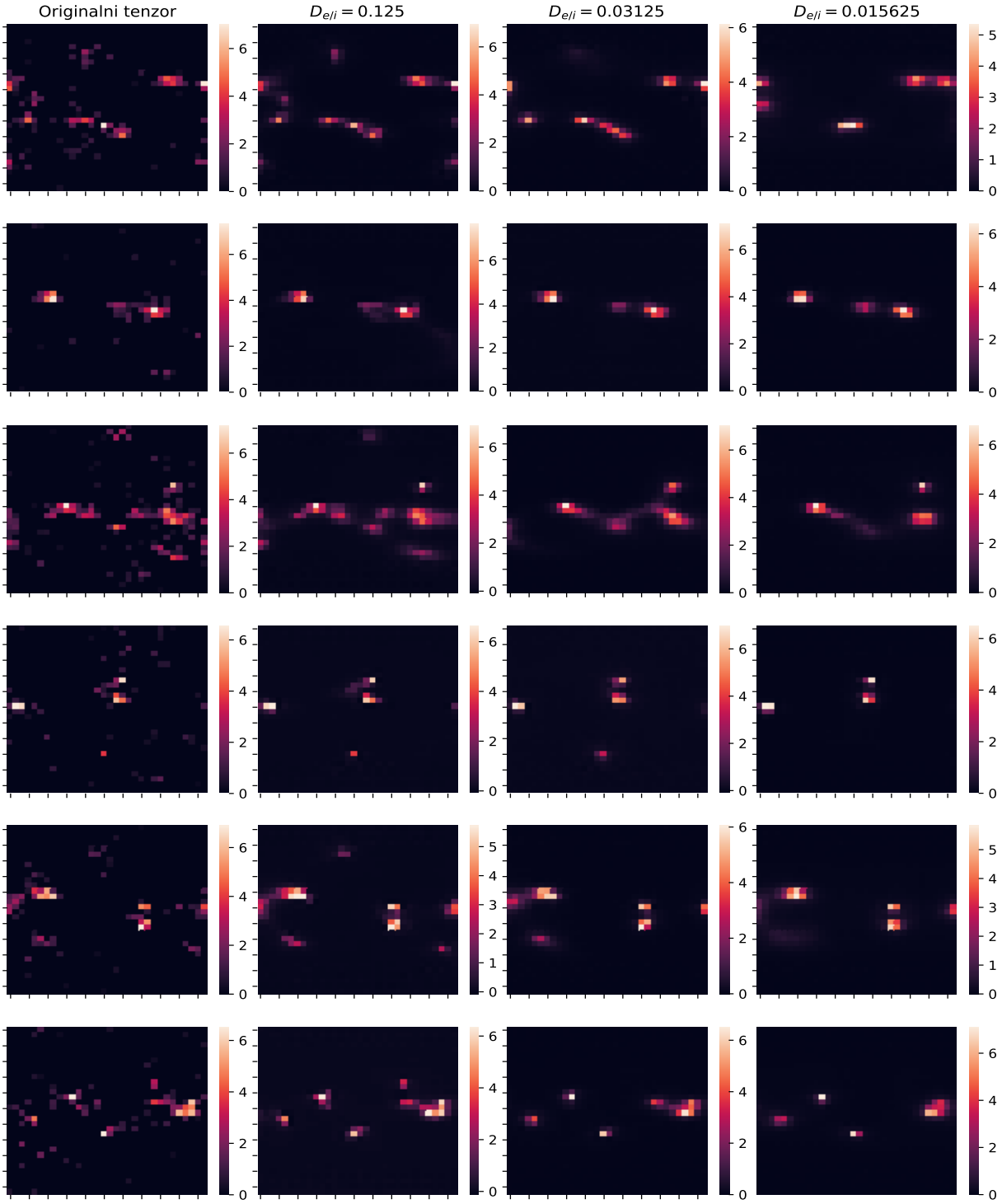
Koristeći U-Net mrežu, pokušali smo naučiti model da imitira anti-k_T algoritam grupiranja. Evaluacijom na black box setu uspostavljeno je da anomalne čestice nisu detektirane promatranjem distribucije grešaka.

Autoenkoderski i U-Net modeli nisu se pokazali kao dobra metoda detekcije novih čestica na ovakvom skupu podataka. No, to ne znači nužno da su autoenkoderi neprikladna metoda za ovakav tip problema, već je vjerojatno problem u reprezentaciji naših podataka. Generirani tenzori određeni su kinematičkim varijablama, koje su dosta primitivne i ne opisuju dobro kompleksnije veličine (na primjer svojstva mlazova). Ovo je čest problem u izradi modela u strojnom učenju; unaprijed nije očito koje veličine koristiti kao značajke.

[5] Stvarna masa anomalne čestice objavljena je na kraju natjecanja LHC Olympics

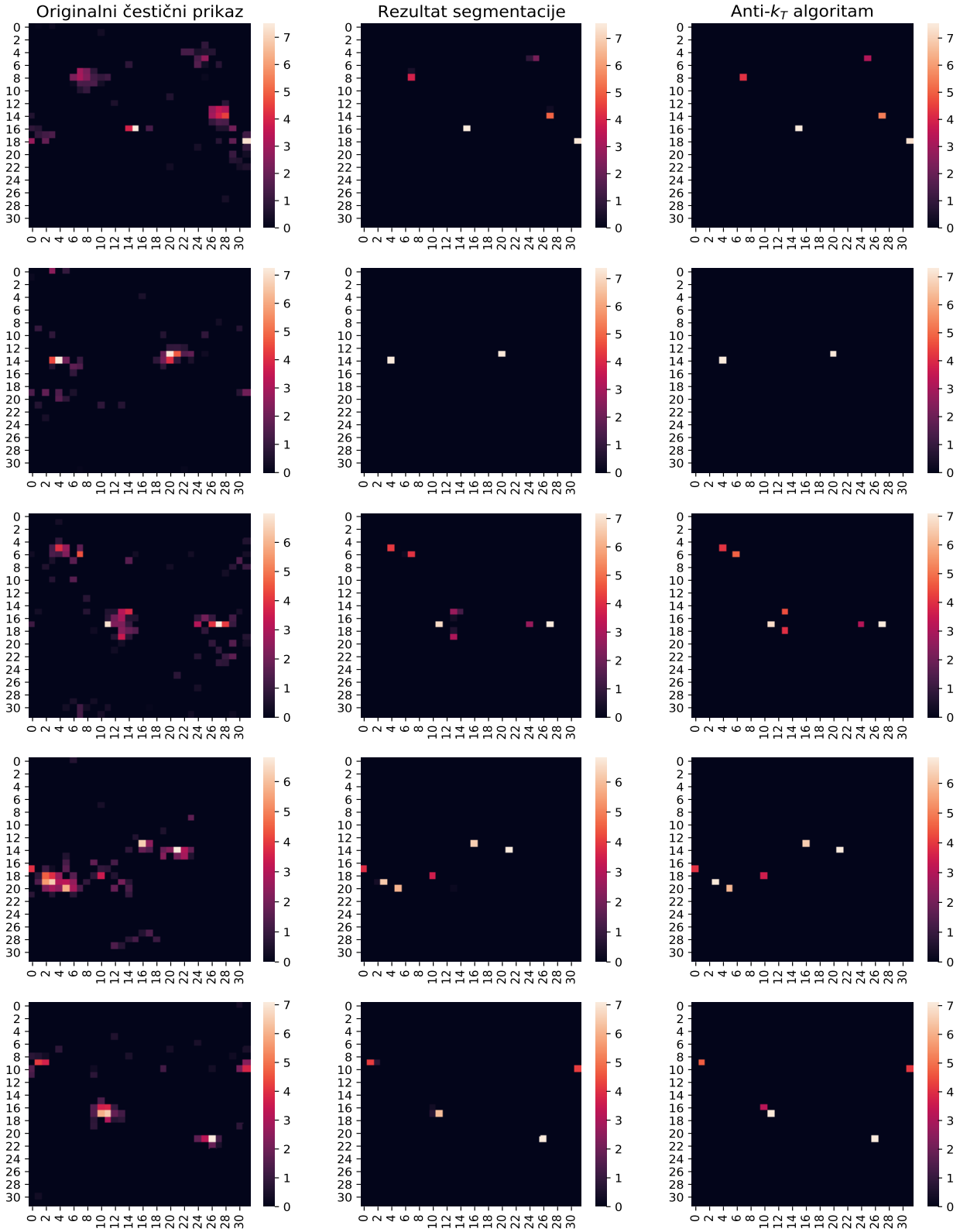
Dodaci

Dodatak A Usporedba autoenkodera



Slika A.1: Prikaz usporedbe autoenkoderskih rekonstrukcija za 3 različita modela korištena u ovom radu.

Dodatak B Primjeri rezultata U-Net mreže



Slika B.2: Prikaz rezultata segmentacije koristeći U-Net mrežu s ciljanim anti-k_T grupiranjem.

Dodatak C Kodovi

C.1 Kod za generiranje tenzora

```
import numpy as np
import pandas as pd
from pyjet import cluster,DTYPE_PTEPM

def Data(data, start, stop):
    a = pd.read_hdf(data, start = start, stop = stop)
    a = a.to_numpy()
    a = np.reshape(a, ((stop-start), 700, 3))
    return a;

def jet_clustering(start, dataset, R=0.4, ptmin=20):
    leadpT=[]
    alljets=[]
    R = R;
    ptmin = ptmin;
    event = Data(dataset,
                 (start),
                 (start+1))
    pseudojets_input = np.zeros(len([x for x in event[0][:, 0] if x > 0]), dtype=DTYPE_PTEPM)
    for j in range(700):
        if (event[0][j][0]>0):
            pseudojets_input[j]['pT'] = event[0][j][0]
            pseudojets_input[j]['eta'] = event[0][j][1]
            pseudojets_input[j]['phi'] = event[0][j][2]
            pass
        pass
    sequence = cluster(pseudojets_input, R=R, p=-1)
    jets = sequence.inclusive_jets(ptmin=ptmin)
    leadpT+= [jets[0].pt]
    alljets += [jets]
    pass
    return jets, alljets

def jet_generator(event, size, mode):
    tensor = np.zeros((size, size))
```

```
    for particle in range (len(event)):
        pt = event[particle].pt
        if(pt>0):
            if(mode=='log'):
                eta = event[particle].eta
                phi = event[particle].phi
                x = int(np.round((eta/5.0+1) * (size)/2))
                y = int(np.round((phi/np.pi+1) * (size)/2))
                tensor[x][y] += pt

    return np.log(tensor+1);

def tensor_generator(event, size, mode):
    tensor = np.zeros((size, size))
    for particle in range (event.shape[0]):
        pt = event[particle][0]
        if(pt>0):
            eta = event[particle][1]
            phi = event[particle][2]
            x = int(np.round((eta/5.0+1) * (size)/2))
            y = int(np.round((phi/np.pi+1) * (size)/2))
            tensor[x][y] += pt
        if(pt==0):
            return np.log(tensor+1);
            break;

def data_to_numpy(path_in, path_out, type)
    data_total = [];
    for n in range(10):
            blackbox = Data(path_in, (n)*100000, (n+1)*100000)
        for event in tqdm(range(len(blackbox))):
            if(type == 'tensor'):
                data_total.append(tensor_generator(blackbox[event],
   size, mode='log'))
            if(type == 'jet')
                data_total.append(jet_generator(blackbox[event], size,
    mode='log'))
    data_total_numpy = np.asarray(data_total)
    np.save(path_out, data_total_numpy )
```

C.2 Kod sa arhitekturama modela

```
import numpy as np
import torch
import torch.nn as nn
import torchvision.transforms.functional as TF


class DoubleConv(nn.Module):
    def __init__(self,in_channels, out_channels):
        super(DoubleConv,self).__init__()
        self.conv=nn.Sequential(nn.Conv2d(in_channels,out_channels,
    kernel_size=3,stride=1,padding=1,bias=False),
                                nn.BatchNorm2d(out_channels),
                                nn.ReLU(),
                                nn.Conv2d(out_channels,out_channels,
    kernel_size=3,stride=1,padding=1,bias=False),
                                nn.BatchNorm2d(out_channels),
                                nn.ReLU())

    def forward(self,x):
        return self.conv(x)



class Autoencoder(nn.Module):
    def __init__(self, mask, features, in_channels=1, out_channels=1):
        super(Autoencoder,self).__init__()
        self.encoder = nn.ModuleList()
        self.decoder = nn.ModuleList()
        self.pool = nn.MaxPool2d(kernel_size=2,stride=2)
        self.mask = mask
        self.in_channels = in_channels
        self.out_channels = out_channels

        for feature in range(len(features)):
            self.encoder.append(DoubleConv(in_channels, features[
    feature]))
            in_channels=features[feature]

```

```
        rev_features =  list(reversed(features))
        for feature in range(len(rev_features)):

            if(feature == len(rev_features)-1):
                self.decoder.append(
              nn.ConvTranspose2d(rev_features[feature], rev_features[
feature], kernel_size=2, stride=2)
                                    )
                self.decoder.append(nn.Conv2d(rev_features[feature],
self.out_channels, kernel_size=1, bias=False))
            else:
                in_channels = rev_features[feature]
                out_channels = rev_features[feature+1]
                self.decoder.append(
                  nn.ConvTranspose2d(in_channels, in_channels,
kernel_size=2, stride=2)
                                    )
                self.decoder.append(DoubleConv(in_channels,
out_channels))


        if(self.mask == True):
            self.final_conv=nn.Sequential(nn.Conv2d(features[0], self.
out_channels, kernel_size=1),
                                          nn.Sigmoid())
        if(self.mask == False):
            self.final_conv=nn.Sequential(nn.Conv2d(features[0], self.
out_channels, kernel_size=1),
                                          nn.ReLU())

    def forward(self,x):

        for encode in self.encoder:
            x=encode(x)
            x=self.pool(x)

        x_encoded = x

        for decode in self.decoder:
            x=decode(x)
```

```
70
71          if(self.mask == True):
72              return x*original;
73          else:
74              return x;
75
76  class UNET(nn.Module):
77      def __init__(self, mask, in_channels=1, out_channels=1, features
    =[16,32,64,128,256,512,1024]):
78          super(UNET,self).__init__()
79          self.downs=nn.ModuleList()
80          self.ups=nn.ModuleList()
81          self.pool=nn.MaxPool2d(kernel_size=2,stride=2)
82          self.mask = mask
83          self.out_channels = out_channels
84
85          for feature in features:
86              self.downs.append(DoubleConv(in_channels,feature))
87              in_channels=feature
88
89          for feature in reversed(features):
90              self.ups.append(
91                nn.ConvTranspose2d(feature*2,feature,kernel_size=2,
    stride=2)
92             )
93              self.ups.append(DoubleConv(feature*2,feature))
94
95          self.bottleneck=DoubleConv(features[-1],features[-1]*2)
96
97          if(self.mask == True):
98              self.final_conv=nn.Sequential(nn.Conv2d(features[0], self.
    out_channels,kernel_size=1),
99                                            nn.Sigmoid())
100         if(self.mask == False):
101             self.final_conv=nn.Sequential(nn.Conv2d(features[0], self.
    out_channels,kernel_size=1),
102                                           nn.ReLU())
103
104     def forward(self,x):
105         skip_connections=[]
```

```
        i=0
        original = x
        for down in self.downs:
            x=down(x)
            skip_connections.append(x)
            i+=1
            x=self.pool(x)

        x=self.bottleneck(x)
        i+=1
        skip_connections=skip_connections[::-1]

        for idx in range(0,len(self.ups),2):
            x=self.ups[idx](x)
            skip_connection=skip_connections[idx//2]
            i+=1
            if x.shape != skip_connection.shape:
                x=TF.resize(x,size=skip_connection.shape[2:])

            concat_skip=torch.cat((skip_connection,x),dim=1)
            x=self.ups[idx+1](concat_skip)
        x=self.final_conv(x)
        if(self.mask == True):
            return x*original;
        else:
            return x;
```

C.3 Kod za trening modela

```
from modules.funkcije import *
from modules.models import *
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import torch
import torchvision
import torch.nn as nn
from torch.optim import lr_scheduler
import torchvision.transforms as transforms
```

```
from tqdm.notebook import tqdm

class Load_Tensor(torch.utils.data.Dataset):
    def __init__(self, path, transform=None, normalize=False):
        self.path = path
        self.transform = transform
        self.tensors = np.load(self.path,  mmap_mode='r+')
        self.data = self.tensors
        self.normalize = normalize
    def __getitem__(self, index):

        tensor = self.tensors[index]
        tensor = torch.from_numpy(tensor)
        size = tensor.shape[0]
        tensor = tensor.view(1, size, size)

        if(self.normalize == True):
            if(torch.sum(tensor)>0):
                tensor = tensor/torch.sum(tensor)
            else:
                tensor = tensor

        if self.transform:
            tensor = self.transform(tensor)

        return tensor

    def __len__(self):
        return len(self.data)

def evaluate(dataloader, model, batch_num, epoch):
    model = model
    model.eval()
    tk0 = tqdm(dataloader, total=int(len(dataloader)))
    semi_running_loss = 0
    with torch.no_grad():
        for i, data in enumerate(tk0):
            inputs, targets = data.view(-1, 1, size, size), data.view
   (-1, 1, size, size)
            inputs, targets = inputs.to(device, dtype=torch.float),
```

```
    targets.to(device, dtype=torch.float)

            outputs = model(inputs)
            criterion = nn.MSELoss().cuda()
            loss =  criterion(outputs, targets)

            semi_running_loss += loss.item() * inputs.size(0)
            if(i>batch_num):
                model.train()
                return [epoch+1, semi_running_loss/batch_num]

def BlackBoxStats(dataset, model, criterion):
    errors = []
    model = model
    model.eval()
    with torch.no_grad():
            for k, testdata in enumerate(tqdm(dataset, 0)):
                testinputs = testdata.view(-1, 1, size, size ).to(
    device, dtype=torch.float)
                testoutputs = model(testinputs)
                testloss = criterion(testoutputs, testinputs)
                errors.append([k, testloss.item()])
            return errors

use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
kwargs = {'num_workers': 4, 'pin_memory': True} if use_cuda else {}

#AUTOENCODER TRAIN#
from models.Architectures.Autoencoder import *
in_channels=1
out_channels=1
mask=False
features=[64,128,256,64,16]
mean, std = 0.1048, 0.5501
normalize = False
size = 32
batch_size = 128

model = Autoencoder(in_channels = in_channels,
```

```
                    out_channels = out_channels,
                    mask = mask,
                    features = features)
model.to(device)
traindata = Load_Tensor('C:/Users/stases/Documents/Diplomski/notebooks
    /data/Tensors/train/'+str(size)+'/tensors.npy',
                                normalize = normalize)
trainloader = torch.utils.data.DataLoader(traindata, batch_size=
    batch_size, shuffle=True)

optimizer = optim.Adam(model.parameters(), lr=0.00015)
scheduler = optim.lr_scheduler.StepLR(optimizer,step_size=1, gamma
    =0.9)

epochs = 25
epoch_losses=[]
running_losses=[]
black_epoch_losses=[]
since = time.time()

for epoch in range(epochs):
    tk0 = tqdm(trainloader, total=int(len(trainloader)))
    semi_running_loss = 0
    running_loss = 0.0
    counter = 0
    for i, data in enumerate(tk0):
        inputs, targets = data.view(-1, 1, size, size), data.view(-1,
    1, size, size)
        inputs, targets = inputs.to(device, dtype=torch.float),
    targets.to(device, dtype=torch.float)
        optimizer.zero_grad()
        outputs = model(inputs)
        criterion = nn.MSELoss().cuda()
        loss =  criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
        semi_running_loss += loss.item() * inputs.size(0)
        counter += 1
        tk0.set_postfix(loss=(running_loss / (counter * trainloader.
```

```
    batch_size)))
123         if((i+1)%1500 == 0):
124             running_losses.append([epoch+i/len(trainloader),
    semi_running_loss/1500])
125             semi_running_loss = 0
126 
127     epoch_loss = running_loss / len(trainloader)
128     print('Training Loss: {:.4f}'.format(epoch_loss))
129     print('Learning rate: {:.8f}', optimizer.param_groups[0]['lr'])
130     scheduler.step()
131     epoch_losses.append([epoch, epoch_loss])
132 
133     blackdata = Load_Tensor(black_data_path, normalize = False)
134     blackloader = torch.utils.data.DataLoader(blackdata, batch_size=
    batch_size, shuffle=True)
135     black_epoch_losses.append(evaluate(dataloader = blackloader,
136                                        model = model,
137                                        batch_num = 1500,
138                                        epoch = epoch))
139     print('Black box loss: ', black_epoch_losses)
140 
141 time_elapsed = time.time() - since
142 print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed //
    60, time_elapsed % 60))
143 
144 #LOSS/EPOCH GRAPH#
145 loss_graph = running_losses
146 black_loss_graph = black_epoch_losses
147 x1, y1 = [a[0] for a in loss_graph], [a[1] for a in loss_graph]
148 x2, y2 = [a[0] for a in black_loss_graph], [a[1] for a in
    black_loss_graph]
149 fig, ax = plt.subplots(figsize=(10,6))
150 plt.title('$D_{e/i} = 0.003125$', fontsize=14)
151 plt.xlabel("Epoha", fontsize=12)
152 plt.ylabel("Gre ka (MSE)", fontsize=12)
153 plt.plot(x1,y1)
154 plt.plot(x2,y2)
155 plt.legend(['Trening set', 'Black box set'],  prop={'size': 12})
156 fig.savefig(loss_path, bbox_inches='tight')
157 
```

```
#MODEL BLACK BOX EVALUATION#
out = out
blackdata = Load_Tensor(black_path)
blackloader = torch.utils.data.DataLoader(black_data_path, batch_size
    =1, shuffle=False)
stats = BlackBoxStats(blackloader, model, nn.MSELoss())
np.save(out, stats)
```

C.4 Kod za detekciju anomalija

```
import numpy as np
import matplotlib.pyplot as plt
import torch.nn as nn


def total_jet_mass(alljets):
    mjj=[]
    for k in range(len(alljets)):
        E = alljets[k][0].e+alljets[k][1].e
        px = alljets[k][0].px+alljets[k][1].px
        py = alljets[k][0].py+alljets[k][1].py
        pz = alljets[k][0].pz+alljets[k][1].pz
        mjj+=[(E**2-px**2-py**2-pz**2)**0.5]
        pass
    pass
    mjj = np.asarray(mjj)
    return mjj.flatten();

def indices_to_mass(indices, split):
    mass_dist = []

    if(len(split)!=0):
        for i in range(len(black_indices)):
            start = split[int(black_indices[i][0])]
            mass_dist.append(float(masses[start]))

    if(len(split) == 0):
        for i in range(len(black_indices)):
            mass_dist.append(float(masses[int(black_indices[i][0])]))
```

```
30
31      return mass_dist
32
33  def sorter(array, num, flip):
34      a=array
35      y=np.argsort(a[:,1],kind='mergesort')
36      a=a[y]
37      a_flip = np.flipud(a)
38
39      if(flip==True):
40          return a_flip[0:num];
41
42      if(flip == False):
43          return a[0:num]
44
45  masses = np.load(mass_dist_path)
46  num = 50000
47  stop = num
48  sort = True
49  cut = 0
50  start = 0
51  rad = 0.8
52  bins = 50
53  indices = []
54  black_box_path = black_box_path
55  black_box_flat = np.load(black_box_path)
56  kwargs = dict(histtype='step', alpha=0.5,  bins=bins, range=[2500,
        4500])
57  black_indices = sorter(black_box_flat, num=num, flip=True)
58
59  #TOP 5% MASS DISTRIBUTION#
60  img = indices_to_mass(black_indices, indices)
61  (n1, bins1, patches1) = plt.hist(img, **kwargs, color='black')
62
63  #TOP 5% SCALING#
64  (n, bins, patches) = plt.hist(masses, **kwargs, color = 'green')
65  n = np.asarray(n)
66  n= n+0.001
67  n3 = np.asarray(n1)
68  bins3 = bins[:len(n1)]
```

```
n2 = n3/n * 10**6/num

#TOP 5% SCALED DISTRIBUTION#
y_max = n2.max()
plt.bar(bins3, n2, width=float(bins[1]-bins[0]), edgecolor='black')
plt.ylim(0, y_max+y_max/10)
ax.set_xlabel("$m$ [GeV]", fontsize=14)
ax.set_title('$N_{bad}\,/ \,N_{exp}$', fontsize=14)
plt.show()
```

Bibliography

[1] LHC Olympics, https://lhco2020.github.io/homepage/, 5.3.2021.

[2] Thomson, M. Modern Particle Physics, 1st ed. Cambride University Press, 2013.

[3] The Top Quark, The Review of Particle Physics, https://pdg.lbl.gov/2017/reviews/rpp2017-rev-top-quark.pdf, 10.4.2021.

[4] Introduction to Renormalization - QCD, University of Southampton, http://www.personal.soton.ac.uk/ab1u06/teaching/qft/qft3/lit/2007-04-20-chen.pdf, 12.4.2021.

[5] The Large Hadron Colider, https://home.cern/science/accelerators/large-hadron-collider, 12.4.2021

[6] An overview of the CMS experiment for CERN guides, https://cds.cern.ch/record/2629323/files/CMSdocumentforGuides.pdf, 13.4.2021.

[7] Pseudorapidity, https://en.wikipedia.org/wiki/Pseudorapidity, 13.4.2021.

[8] Cacciari, M; Gavin, P.; Soyez, G; The anti-k_t jet clustering algorithm. // JHEP 0804:063, 2008

[9] An Introduction to POWHEG. https://indico.cern.ch/event/656211/contributions/2673382/attachments/1498653/2333153/luisoni_powheg.pdf, 14.4.2021.

[10] A Brief Introduction to PYTHIA, http://home.thep.lu.se/~torbjorn/pythia81html/Welcome.html, 14.4.2021.

[11] 11 Main Neural Network Structure Diagrams https://www.programmersought.com/article/69014678496/, 15.4.2021.

[12] Why Sigmoid? https://miro.medium.com/max/2384/1*ACHo09NFhKvYCsOFHxWVbA.png, 15.4.2021.

[13] Gradient Descent, https://en.wikipedia.org/wiki/Gradient_descent, 15.4.2021.

[14] Kingma, P; Adam: A Method for Stochastic Optimization, 3rd International Conference for Learning Representations, San Diego, 2015

[15] Duchi, J.; Hazan, E; Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, Journal of Machine Learning Research 12 (2011) 2121-2159

[16] RMSprop algorithm, https://keras.io/api/optimizers/rmsprop/, 15.4.2021.

[17] Kernels (Image Processing), https://en.wikipedia.org/wiki/Kernel_(image_processing), 16.4.2021.

[18] Convolutional Neural Network — Deep Learning, https://developersbreach.com/convolution-neural-network-deep-learning/

[19] Autoencoder, https://en.wikipedia.org/wiki/Autoencoder, 16.4.2021.

[20] Kullback–Leibler divergence https://en.wikipedia.org/wiki/Kullback-Leibler_divergence, 16.4.2021.

[21] PyJet: The interface between FastJet and NumPy https://github.com/scikit-hep/pyjet

[22] Ronneberger, O.; Fischer, P.; U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015.

[23] NumPy, https://numpy.org/

[24] PyTorch, https://github.com/pytorch/pytorch

[25] Ioffe, S.; Szegedy, I.; Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, arXiv:1502.03167, 2015.

[26] Kalwarczyk, T.; Tabakaba, M.; Holyst, R. Biologistics : diffusion coefficients for complete proteome of Escherichia coli. // Bioinformatics. Vol. 28, 22(2012), str. 2971-2978.

[27] Galli, D. Magnetic fields in star-forming regions : theoretical aspects. // Dense molecular gas around protostars and in galactic nuclei : European workshop

on astronomical molecules 2004 / edited by Baan and Hagiwara. Dordrecht : Springer, 2004. Str. 43-51.

[28] Prezime, I. Naslov doktorskog rada. Doktorski rad. Zagreb : Prirodoslovno-matematički fakultet, 2008.

[29] The Chapman-Kolmogorov equations, (20.12.2008), Mathematics Prelims, http://mathprelims.wordpress.com/2008/12/20/the-chapman-kolmogorov-equations/, 1.6.2013.