

# Analiza sličnosti peptida temeljena na latentnom prostoru autoenkodera

---

**Novosel, Bruno**

**Master's thesis / Diplomski rad**

**2023**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:190:513051>

*Rights / Prava:* [Attribution 4.0 International](#)/[Imenovanje 4.0 međunarodna](#)

*Download date / Datum preuzimanja:* **2024-07-10**



*Repository / Repozitorij:*

[Repository of the University of Rijeka, Faculty of Engineering](#)



SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij računarstva

Diplomski rad

**ANALIZA SLIČNOSTI PEPTIDA TEMELJENA NA LATENTNOM  
PROSTORU AUTOENKODERA**

Rijeka, rujan 2023.

Bruno Novosel

0069078199

SVEUČILIŠTE U RIJECI  
**TEHNIČKI FAKULTET**

Diplomski sveučilišni studij računarstva

Diplomski rad

**ANALIZA SLIČNOSTI PEPTIDA TEMELJENA NA LATENTNOM  
PROSTORU AUTOENKODERA**

Mentor: doc. dr. sc. Goran Mauša

Rijeka, rujan 2023.

Bruno Novosel  
0069078199

## IZJAVA

Izjavljujem da sam ovaj rad izradio samostalno, uz vodstvo i stručnu pomoć mentora doc. dr. sc. Gorana Mauše i asistenta Marka Njirjaka.

---

Bruno Novosel

## ZAHVALA

Zahvaljujem mentoru doc. dr. sc. Goranu Mauši i asistentu Marku Njirjaku na pomoći i stručnom vođenju kojim su me vodili kroz izradu ovog rada i što su mi uvijek bili na raspolaganju.

# Sadržaj

1. UVOD .....	1
2. PEPTIDI .....	2
2.1. Što su peptidi? .....	2
2.2. Antimikrobna svojstva.....	4
2.3. Antiviralna svojstva.....	6
3. AUTOENKODERI .....	7
3.1. Autoenkoderi općenito .....	7
3.2. Varijacijski autoenkoderi.....	9
4. MODEL.....	14
4.1. Alati .....	14
4.1.1. Python.....	14
4.1.2. Jupyter Notebook .....	14
4.1.3. Tensorflow .....	16
4.1.4. Keras.....	16
4.1.5. Ostale Python knjižnice.....	17
4.2. Podaci .....	18
4.3. Metode .....	20
4.3.1. Enkoder .....	21
4.3.2. Dekoder .....	23
4.3.3. Model varijacijskog autoenkodera .....	24
4.3.4. Obrada podataka.....	26
4.3.5. Optimizacija hiperparametara .....	27
4.3.6. Trening .....	29
5. REZULTATI.....	31
5.1. Rezultati treninga.....	31
5.2. Vizualizacije .....	37
5.3. Korelacijska analiza.....	42
6. ZAKLJUČAK .....	48
LITERATURA.....	50
POPIS KRATICA .....	52

Sažetak .....	53
Abstract .....	53
PRILOZI.....	54
Programski kod .....	54

## 1. UVOD

Zahvaljujući napretku medicine i znanosti, otkrivena su i proizvedena mnoga cjepiva i lijekovi koji su značajno doprinijeli suzbijanju raznih bolesti kod ljudi. Unatoč tome, mnoge bolesti i dalje predstavljaju veliku prijetnju ljudskom zdravlju te se neprestano traže nova rješenja za njihovo suzbijanje. U tom kontekstu, peptidi su se istaknuli kao obećavajuće novo rješenje pri liječenju. Iako se još uvijek smatraju pomalo nekonvencionalnim pristupom liječenju raznih bolesti i infekcija i relativno su brojčano ograničeni, peptidi su pokazali obećavajuće rezultate u mnogim istraživanjima [1]. U prirodi postoje peptidi s različitim funkcijama, ali ograničena raznolikost sekvenci unutar ovih molekula ograničava i spektar njihove primjene pa se zbog toga pokušava sintetizirati nove peptide koji će imati željena svojstva. Međutim, predviđanje svojstava novo sintetiziranih peptida i poboljšanje postojećih metoda predviđanja još uvijek ima prostora za napredak.

Temeljem dosadašnjih uspjeha metoda strojnog učenja na ovom području, u ovom radu koristi se varijacijski autoenkoder kako bi veliki broj peptida opisanih njihovim fizikalno-kemijskim značajkama prikazali u latentnom prostoru autoenkodera. S obzirom na to da su se brojne fizikalno-kemijske značajke peptida pokazale kao potencijalni indikatori antimikrobne ili antiviralne učinkovitosti [2], u latentnom prostoru tada možemo promatrati sličnost među pojedinim vrstama i podvrstama peptida sa željenim svojstvima. Želimo ispitati hoće li različite skupine peptida biti vizualno odvojene u latentnom prostoru, odnosno imaju li pojedine skupine ujedno i različite fizikalno-kemijske značajke. U tom slučaju, moguća buduća primjena mogla bi biti da neki novi peptid prikazemo u latentnom prostoru autoenkodera i ovisno o tome u koju skupinu bude smješten procijenimo vrijedi li ga ispitivati laboratorijski. Također, može se postaviti pitanje postoji li korelacija između takve sličnosti, dobivene u latentnom prostoru autoenkodera, i sličnosti izračunate prema sekvencama pojedinih peptida.

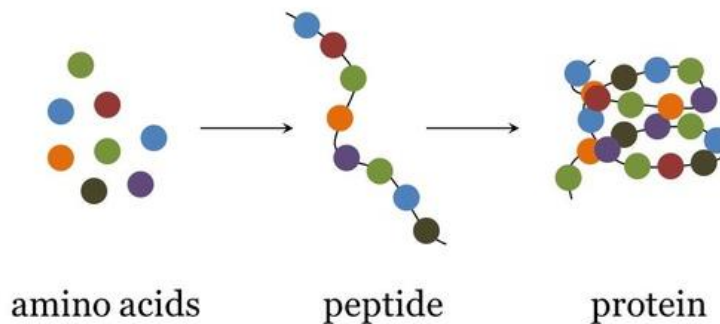
Ovaj rad predstavlja metode i rezultate istraživanja koji prikazuju potencijal korištenja strojnog učenja za istraživanje peptida u svrhu otkrivanja novih lijekova. Korištenjem ovakvih računalnih metoda, cilj je ubrzati otkrivanje i dizajn antimikrobnih i antiviralnih peptida s poboljšanom učinkovitošću.



## 2. PEPTIDI

### 2.1. Što su peptidi?

Peptidi su biomolekule prisutne u svim živim organizmima. Kao i proteini, sastavljeni su od aminokiselina povezanih peptidnim vezama u polipeptidne lance, a razlikuju se po broju aminokiselina od kojih se sastoje. Polipeptidne lance do 50 aminokiselina smatramo peptidima, a s više od 50 smatramo proteinima [3]. Pojednostavljeni prikaz aminokiselina, peptida i proteina nalazi se na slici 2.1.



*Slika 2.1. Pojednostavljeni prikaz aminokiselina, peptida i proteina (preuzeto iz [4])*

U prirodi se pojavljuje dvadeset standardnih aminokiselina. Svaka aminokiselina ima jedinstvenu strukturu i svojstva te one među sobom mogu stvarati peptidne veze i tako se povezivati u lance. Popis svih prirodnih aminokiselina prikazan je slikom 2.2.

Ovisno o broju aminokiselina od kojih se sastoje, peptidi se mogu klasificirati kao dipeptidi, tripeptidi, oligopeptidi ili polipeptidi. Također se mogu podijeliti na linearne i cikličke, ovisno o načinu povezivanja aminokiselina.

Peptidi imaju raznolike uloge u biološkim sustavima, kao što su signalizacija, obrana od patogena, regulacija enzimskih aktivnosti i strukturna podrška. Jedna od posebno bitnih karakteristika za ovaj rad je da peptidi mogu imati antimikrobnu aktivnost, boreći se na taj način protiv bakterija, gljivica, parazita i virusa. Takvi peptidi, poznati kao antimikrobni peptidi, imaju široki spektar djelovanja i važni su u razvoju novih lijekova.

<b>Amino Acid</b>	<b>3-Letters</b>	<b>1-Letter</b>
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamic acid	Glu	E
Glutamine	Gln	Q
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

*Slika 2.2. Popis svih prirodnih aminokiselina i njihove oznake (preuzeto iz [5])*

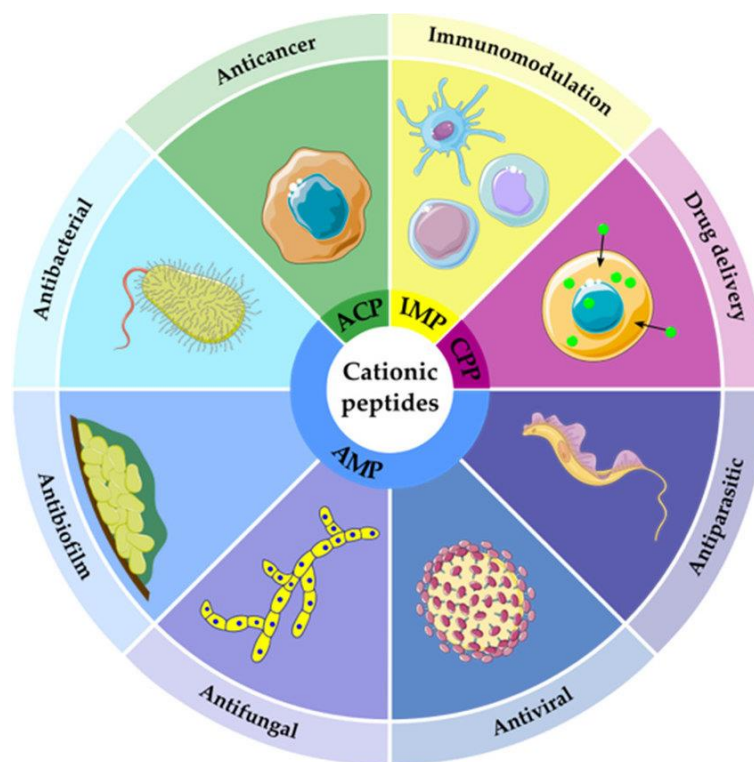
Interes istraživača za razvoj novih peptida i proučavanje struktura i funkcija ciljnih receptora znatno je porastao posljednjih godina. Suvremeno medicinsko i biokemijsko istraživanje ima značajan fokus na peptide zbog njihove selektivnosti, specifičnosti i snažne interakcije s ciljanim proteinima. Svojom veličinom i površinom omogućuju preciznije vezanje na ciljne molekule. Mogu imati vrlo selektivno djelovanje, smanjujući rizik od nuspojava, ali brzo se metaboliziraju te imaju kratko djelovanje u tijelu. Aktivnost peptida može se produljiti uvođenjem raznih modifikacija.

Peptidi također imaju neke prednosti u usporedbi s proteinima kada govorimo o liječenju, iako proteini zauzimaju sve veći udio na farmaceutskom tržištu posljednjih nekoliko godina. Proteini, iako su često vrlo sigurni i učinkoviti, moraju se proizvoditi u bioreaktorima koji koriste cijele stanice, a njihovo pročišćenje i strukturalna analiza često su složeni i skupi. S druge strane, peptidi se

često mogu dobiti kemijski, a njihovo pročišćenje i analiza su mnogo jednostavniji. Također, sve je više primjera oralno djelotvornih peptida, što ih čini poželjnijima jer se lijekovi zasnovani na proteinima gotovo uvijek moraju ubrizgati u tijelo [3].

## 2.2. Antimikrobna svojstva

Antimikrobni peptidi (AMP) su peptidi koji imaju antimikrobna svojstva i time igraju ključnu ulogu u obrani organizama od mikrobnih infekcija. Često se nazivaju i kationskim peptidima jer su kao cjelina pozitivno nabijeni, iako su zapravo samo podvrsta kationskih peptida. Simboličan prikaz podjele kationskih peptida u skupine prema njihovim funkcijama, među kojima se nalaze i AMP prikazan je slikom 2.3. Oni su dio prirodnog imunološkog sustava prisutnog kod gotovo svih živih organizama, uključujući ljude, životinje, biljke i mikroorganizme. AMP se ističu svojom sposobnošću da selektivno ciljaju i ubiju mikrobe, uključujući bakterije, gljivice, parazite i viruse.

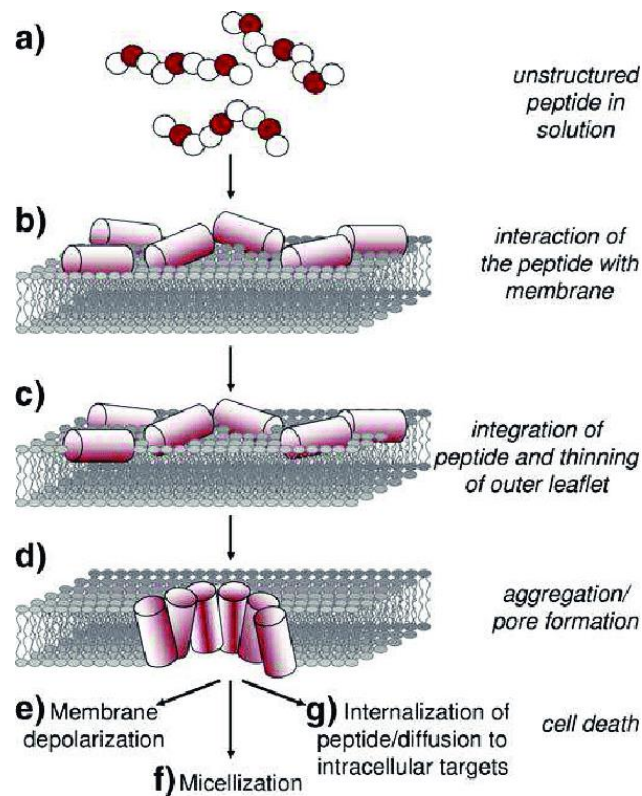


Slika 2.3. Podjela kationskih peptida prema funkcijama (preuzeto iz [6])

AMP imaju raznoliku strukturu koja može uključivati linearno raspoređene aminokiseline, cikličke strukture, kao i kombinaciju različitih strukturnih motiva. Ovi peptidi često sadrže hidrofobne i hidrofilne regije, što im omogućuje interakciju s membranama mikroba. Mnogi AMP imaju pozitivno nabijene aminokiseline, kao što su lizin i arginin, što im pomaže u vezivanju za negativno nabijene komponente mikrobnih membrana [7].

AMP djeluju na mikrobe na različite načine, ciljajući njihove vitalne strukture i funkcije. Neki peptidi djeluju na mikrobne membrane, oštećujući njihovu strukturu i uzrokujući curenje unutarstaničnih tvari. Drugi peptidi prodiru u unutrašnjost mikroba i ciljaju vitalne procese kao što su sinteza proteina i DNA replikacija. Pojednostavljeni primjer djelovanja peptida na ciljnu molekulu prikazan je slikom 2.4. Također, neki peptidi imaju imunomodulatorna svojstva, potičući imunološki odgovor organizma i potencijalno regulirajući upalne procese.

Kada govorimo o razvoju lijekova, jedna od ključnih prednosti AMP je to što su mikroorganizmi manje skloni razvoju otpornosti na njihovo djelovanje za razliku od ostalih lijekova. Otpornost na antimikrobne lijekove postala je globalni problem, ali mnogi AMP djeluju na mikrobe putem mehanizama koji se razlikuju od konvencionalnih antibiotika, što smanjuje vjerojatnost razvoja otpornosti.



Slika 2.4. Primjer djelovanja antimikrobnih peptida (AMP) na membranu ciljne molekule (preuzeto iz [8])

### 2.3. Antiviralna svojstva

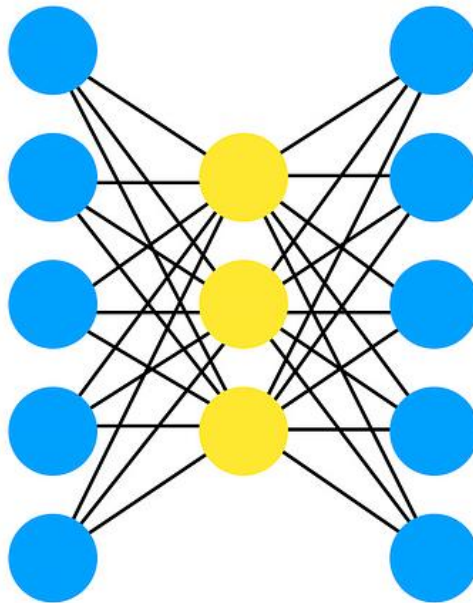
Antiviralni peptidi (AVP) su podvrsta AMP koja ima gotovo sve iste karakteristike, ali su specifični po tome što imaju sposobnost suzbijanja virusnih infekcija. Kao i kod AMP, AVP koriste razne mehanizme kojima suzbijaju infekcije, npr. mogu blokirati vezanje virusa, spriječiti spajanje virusa sa stanicama domaćina, prekinuti proces signaliziranja virusa ili spriječiti razmnožavanje virusa u stanicama domaćina [1]. Osim toga, AVP mogu modulirati imunološki odgovor domaćina i poticati stvaranje protuvirusnih citokina.

Predviđanje antiviralnih svojstava prije sinteze novih peptida izuzetno je važno za razvoj lijekova i prevenciju virusnih infekcija. Mnoge metode strojnog učenja i računalnog modeliranja koriste se za predviđanje antiviralnih svojstava peptida na temelju njihovih fizikalno-kemijskih značajki i strukture. Ova područja istraživanja doprinose razvoju boljih modela za predviđanje antiviralne aktivnosti peptida i optimizaciju njihovih svojstava.

### 3. AUTOENKODERI

#### 3.1. Autoenkoderi općenito

Autoenkoderi su posebna vrsta neuronskih mreža čiji je cilj što vjernije rekonstruirati podatke sa ulaza na izlazu. Sastoje se od tri glavna dijela: enkodera, latentnog prostora i dekodera, vrlo pojednostavljeno prikazano na slici 3.1. Možemo reći da autoenkoder uči kako kroz enkoder sažeti podatke u latentni prostor i prikazati ih pomoću manjeg broja značajki, a pritom minimizirati grešku rekonstrukcije na izlazu dekodera.



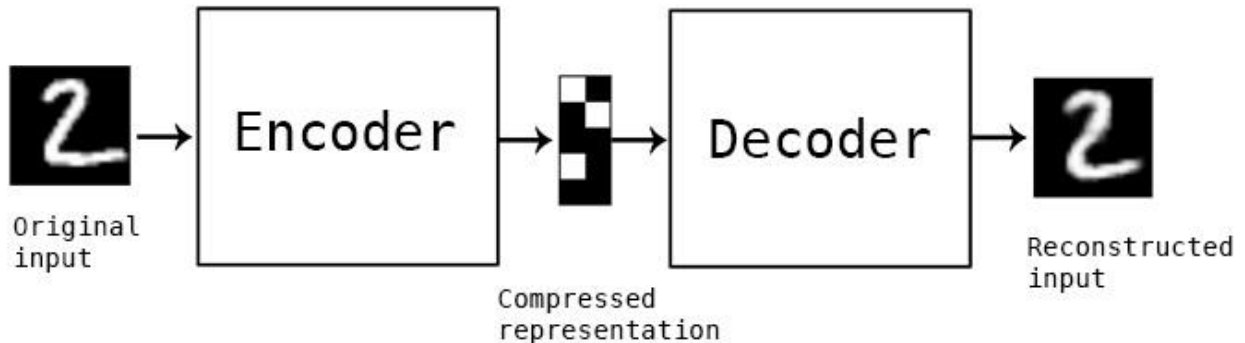
*Slika 3.1. Pojednostavljeni prikaz autoenkodera (enkoder – lijevi sloj, latentni prostor – srednji sloj, dekoder – desni sloj) (preuzeto iz [9])*

Međutim, autoenkoder ne bi imao svrhu kada bi samo preslikavao podatke s ulaza na izlaz. Ono što ga čini korisnim je to da kroz proces rekonstrukcije podataka model uči koje su najvažnije značajke ulaznih podataka te kada je istreniran može se koristiti kako bi na temelju naučenog stvorio nove podatke uz dodatak ili izuzetak određenih značajki [10]. Takav primjer možemo vidjeti i na slici 3.2. gdje autoenkoder uzima sliku znamenke napisane rukom, sažima ju u mali broj dimenzija, uči koje su najvažnije značajke te na kraju pomoću tih značajki rekonstruira sliku koja je vrlo slična originalnoj slici na ulazu.

Svaki dio autoenkodera ima svoju konkretnu ulogu u cijelome procesu. Enkoder je gusto povezana mreža i ulaz u autoenkoder. Zadaća mu je da uzme ulazne podatke te ih sažme i prikaže u latentnom prostoru, čineći tako novi prikaz podataka koji ima smanjenu dimenzionalnost.

Latentni prostor radi sa sažetim prikazom podataka. Konstruiran je tako da odredi najvažnije dijelove promatranih podataka odnosno značajke podataka koje su najvažnije za dobru rekonstrukciju. Cilj mu je odrediti koje značajke podataka trebaju biti očuvane, a koje se mogu ukloniti. Također treba uzeti u obzir i napraviti dobar omjer između veličine latentnog prikaza odnosno koliko će on biti sažet te relevantnosti značajki.

Dekoder je zadužen da uzme sažete podatke iz latentnog prostora koje je kreirao enkoder i pretvori ih ponovno u prikaz sa istim brojem dimenzija kao što su imali i početni ulazni podaci.



*Slika 3.2. Slika prolazi kroz enkoder, postaje sažeti prikaz podataka te na izlazu postaje rekonstruirana ulazna slika (preuzeto iz [11])*

Postoje tri vrlo bitne značajke koje definiraju rad autoenkodera [11]:

- 1) Specifični su za podatke na kojima uče – bit će u mogućnosti raditi samo s podacima koji su vrlo slični onima na kojima je model treniran, npr. autoenkoder koji je treniran na slikama na kojima se nalaze ljudska lica će raditi vrlo loše sa slikama na kojima se nalazi drveće jer su značajke koje je autoenkoder naučio specifične za ljudska lica
- 2) Imaju gubitke – podaci na izlazu koji su prošli kroz dekompresiju će biti lošije kvalitete u usporedbi s ulaznim podacima
- 3) Uče automatski iz primjera podataka – vrlo je lako trenirati algoritam da radi dobro na raznim tipovima ulaznih podataka, odnosno nije potrebno mijenjati algoritam već je dovoljno algoritmu dati prikladne podatke za trening

Autoenkoderi imaju vrlo široki spektar primjene, ali najčešće su sljedeće[10]:

- Smanjenje dimenzionalnosti - prolaskom kroz enkoder podaci dolaze u latentni prostor gdje su prikazani pomoću manjeg broja dimenzija u odnosu na izvorne ulazne podatke te se tako mogu lakše razumjeti ili vizualizirati; također može pomoći mrežama visokog kapaciteta naučiti korisne značajke podataka što upućuje na to da autoenkoderi mogu biti korišteni za poboljšanje treninga drugih tipova neuronskih mreža
- Uklanjanje šuma iz podataka - često se koristi na slikama; osim uklanjanja šuma, autoenkoderi mogu ispraviti i druge tipove oštećenja na slikama kao što su mutne slike ili dopuniti dijelove slike koji nedostaju
- Ekstrakcija značajki - kao i smanjenje dimenzionalnosti može biti korisno za poboljšanje treninga drugih tipova neuronskih mreža jer se mogu koristiti za prepoznavanje značajki skupova podataka koji se koriste za treniranje drugih modela
- Generiranje slika - autoenkoderi mogu generirati slike novih, nepostojećih ljudi ili animiranih likova što pomaže pri konstruiranju sustava za prepoznavanje lica ili automatiziranja određenih aspekata animacije
- Predviđanje sekvence - mogu se koristiti za određivanje promjene strukture podataka kroz vrijeme, što znači da mogu generirati idući podatak u sekvenci i to dovodi do toga da autoenkoderi mogu generirati videozapise
- Stvaranje sustava preporuke - koriste se duboki autoenkoderi, a sustave stvaraju tako što će prepoznati uzorke korištenja i povezati ih s interesima korisnika; u ovoj primjeni enkoder analizira podatke o načinu interakcije, a dekodeer stvara preporuke koje odgovaraju prepoznatim uzorcima

### **3.2. Varijacijski autoenkoderi**

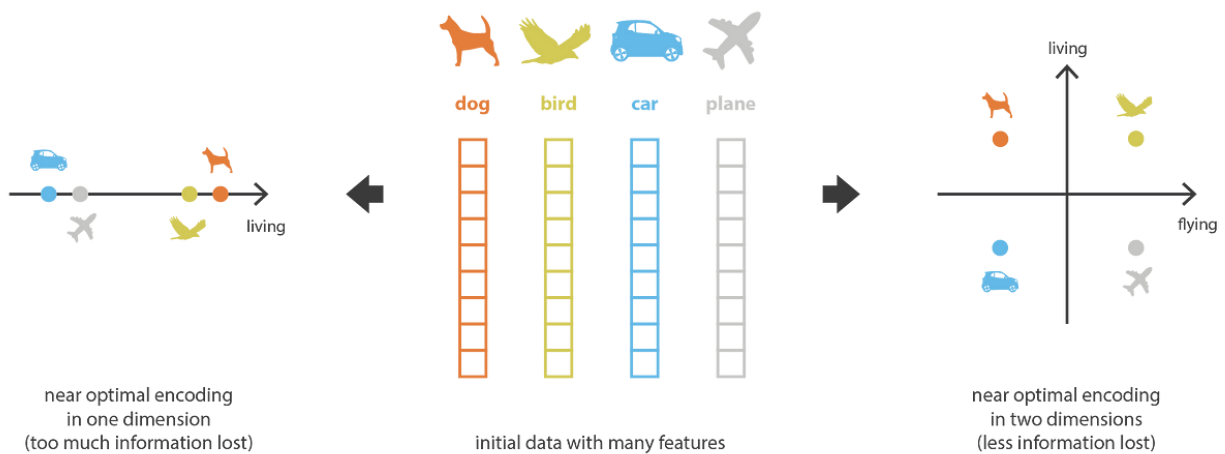
U osnovi, varijacijski autoenkoder (VAE) je autoenkoder čija je distribucija enkodiranih podataka regularizirana tijekom treninga kako bismo osigurali da latentni prostor ima dovoljno dobra svojstva da možemo generirati nove podatke [12].

„Obični“ autoenkoderi često nemaju svojstva koja su potrebna za generiranje novih podataka. Kako bismo to prikazali možemo zamisliti primjer gdje postoje enkoder i dekodeer koji su dovoljno moćni da postavje bilo koji  $N$  početni skup podataka na realnu os, odnosno svaku točku



podataka enkodiraju kao realnu vrijednost u jednoj dimenziji, i dekodiraju ih bez ikakvih gubitaka pri rekonstrukciji. U tom slučaju visoki stupanj slobode koji omogućava autoenkoderu enkodiranje i dekodiranje bez ikakvih gubitaka informacija dovodi do jakog *overfittinga* što znači da će neke točke latentnog prostora nakon dekodiranja vraćati besmislen i nerazumljiv sadržaj. Također treba uzeti u obzir da je cilj smanjenja dimenzionalnosti, uz smanjenje broja dimenzija podataka, i zadržavanje većine informacija u sažetim prikazima, a sažimanjem podataka u samo jednu dimenziju gubimo jako velik dio informacija, dok već sa dvije dimenzije zadržavamo mnogo veći dio informacija, što je vrlo pojednostavljeno prikazano slikom 3.3. Iako je ovaj primjer s jednom dimenzijom u latentnom prostoru poprilično ekstreman, on demonstrira kako autoenkoderi općenito imaju problem s regularnosti latentnog prostora i da na to treba pripaziti.

Ovaj problem nedostatka strukture u enkodiranim podacima u latentnom prostoru je shvatljiv s obzirom na to da zadatak za koji je autoenkoder treniran ne zahtijeva takvu organizaciju. Autoenkoder je treniran isključivo da enkodira i dekodira podatke sa što manje gubitaka, bez obzira na to kako je latentni prostor organiziran. Stoga, ako se arhitekturi autoenkodera ne pridodaje posebna pažnja, normalno je da mreža koristi sve mogućnosti za *overfitting* koje ima kako bi postigla što bolji rezultat.



Slika 3.3. Početni skup podataka (sredina) enkodiran u jednu dimenziju (lijevo) i u dvije dimenzije (desno) (preuzeto iz [12])

Dakle, kako bismo mogli koristiti dekođer autoenkodera u svrhu generiranja novih podataka moramo osigurati regularnost latentnog prostora. Jedno od rješenja za postizanje regularnosti je da tijekom procesa treninga eksplicitno uvedemo regularizaciju. Na taj način dobivamo varijacijski autoenkoder. Njegova arhitektura je ista kao i kod standardnog autoenkodera, ali postoji razlika kod procesa enkodiranja: umjesto enkodiranja ulaznih podataka kao pojedinačne točke, enkodiramo ih kao distribucije u latentnom prostoru. Time dolazimo do sljedećeg načina treniranja modela:

- 1) Ulazni podaci se enkodiraju kao distribucija u latentnom prostoru
- 2) Točka iz latentnog prostora se uzima kao uzorak iz te distribucije
- 3) Uzorkovana točka se dekodira i izračunava se greška rekonstrukcije
- 4) Greška rekonstrukcije se vraća unazad kroz mrežu

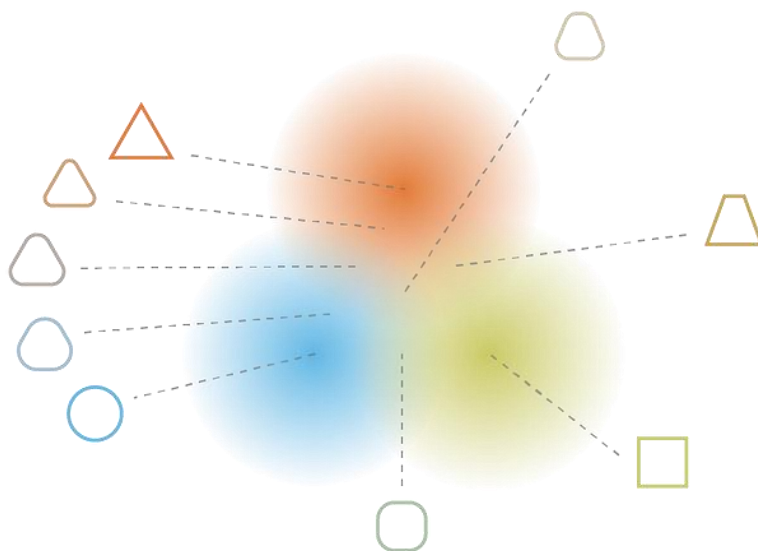
Regularnost koja se očekuje od latentnog prostora kako bi se omogućio proces generiranja novih podataka zahtijeva da latentni prostor bude kontinuiran i potpun. Kontinuiranost osigurava da dvije točke koje su blizu jedna drugoj u latentnom prostoru ne vraćaju potpuno drugačiji sadržaj kada se dekodiraju. Potpunost osigurava da za odabranu distribuciju svaka točka koja se uzme kao uzorak iz latentnog prostora i dekodira vraća smisleni sadržaj. Važnost regularizacije latentnog prostora pojednostavljeno je prikazana slikom 3.4. gdje na lijevom prikazu vidimo latentni prostor bez regularizacije iz kojeg se dekodiraju točke. Točka koja je dekodirana iz sredine latentnog prostora vraća sadržaj koji nije smislen niti nalikuje na bilo koji drugi sadržaj dekodiran iz tog latentnog prostora. Suprotno tome, na desnom prikazu vidimo latentni prostor koji je regulariziran. U ovom slučaju, točka koja je dekodirana iz sredine latentnog prostora vraća smisleni sadržaj, a uz to nalikuje na sve oblike koji su dekodirani iz obližnjeg prostora.



Slika 3.4. Prikaz latentnog prostora i podataka bez regularizacije (lijevo) i s regularizacijom (desno) (preuzeto iz [12])

Sama činjenica da VAE enkodira ulazne podatke kao distribucije umjesto pojedinačnih točaka nije dovoljno da osigura kontinuitet i potpunost. Zbog toga je potrebno regularizirati matricu kovarijance i srednju vrijednost distribucija koje vraća enkoder. U praksi se to postiže osiguravanjem da distribucija bude što sličnija normalnoj distribuciji (centrirana i reducirana). Na taj način se osigurava da matrica kovarijance bude slična jediničnoj matrici, sprječavajući točkaste distribucije, i da srednja vrijednost bude blizu nuli, sprječavajući enkodirane distribucije da budu predaleko jedne od drugih.

Cijeli postupak regularizacije dovodi do toga da sprječavamo model da enkodira podatke daleko jedne od drugih u latentnom prostoru i potiče enkodirane distribucije da se preklapaju koliko god je to moguće, zadovoljavajući na taj način uvjete kontinuiteta i potpunosti i stvarajući gradijent u podacima enkodiranim u latentnom prostoru. Npr. točka u latentnom prostoru koja je na pola puta između srednjih vrijednosti dvije enkodirane distribucije koje dolaze od različitih podataka u treningu trebala bi biti dekodirana u nešto što bi odgovaralo vrijednosti između podatka koji je dao prvu distribuciju i podatka koji je dao drugu distribuciju. Ovo možemo vidjeti i na slici 3.5. gdje npr. dekodiramo točku iz prostora između točaka koje su dekodirane kao geometrijski likovi krug i kvadrat. Sadržaj te dekodirane točke je tada geometrijski lik koji istovremeno nalikuje na krug i na kvadrat.



*Slika 3.5. Latentni prostor i dekodirani podaci uz prisutnu regularizaciju (preuzeto iz [12])*

Dodatno treba napomenuti dvije funkcije koje računaju gubitke kod VAE, a koje osiguravaju regularnost i bit će spomenute u kasnijim poglavljima: gubici rekonstrukcije i Kullback-Leibler divergencija (KL divergencija). Kada se minimiziraju gubici rekonstrukcije, to za mrežu znači da mora učiniti da izlazni podaci izgledaju što više kao ulazni podaci i da pri tome može koristiti sve što ima na raspolaganju. To će osigurati da model grupira uzorke koji su slični jedni drugima, odnosno svaka klasa će biti u jednoj skupini, a slične klase će biti enkodirane blizu jedna drugoj te je tako zadovoljen kontinuitet latentnog prostora.

Međutim, kada uzimamo u obzir samo gubitke rekonstrukcije nemamo pravilo koje osigurava da se grupacije donekle preklapaju i da su na taj način povezane jer u namjeri modela da minimizira gubitke rekonstrukcije, on može u potpunosti razdvojiti grupacije enkodiranih podataka te tada nemamo zadovoljenu potpunost latentnog prostora. U tom slučaju, uz izračun gubitaka rekonstrukcije dodajemo i KL divergenciju. KL divergencija je specifična za VAE i ne postoji kod klasičnih autoenkodera. KL divergencija zapravo izračunava divergenciju između dvije distribucije vjerojatnosti, odnosno njen gubitak se povećava kada se distribucija vjerojatnosti koju je generirao enkoder razlikuje od standardne normalne distribucije. To znači da je model prisiljen učiti tako da enkoder „gura“ distribucije vjerojatnosti odnosno uzorke što je bliže moguće jedne drugima i tako zadovoljava potpunost latentnog prostora [13].

## 4. MODEL

### 4.1. Alati

Alati pomoću kojih je izrađen model, napravljena vizualizacija i analiza rezultata odabrani su tako da nude što veću moć i više mogućnosti, a da budu relativno jednostavni za korištenje i tumačenje. Programski kod napisan je u programskom jeziku Python unutar Jupyter Notebook sučelja pomoću knjižnica Keras i Tensorflow koje su specijalizirane za strojno učenje.

#### 4.1.1. Python

Python je jedan od najpopularnijih programskih jezika u svijetu već dugi niz godina. S obzirom na to da je Python jezik opće namjene, može se koristiti za razne stvari kao što su izgradnja web stranica i softvera, automatizacija procesa ili analiza podataka. Ono što ga također čini toliko popularnim je činjenica da je vrlo intuitivan i lagan za učenje kod početnika. U ovom radu korištena je verzija Pythona 3.4.10.

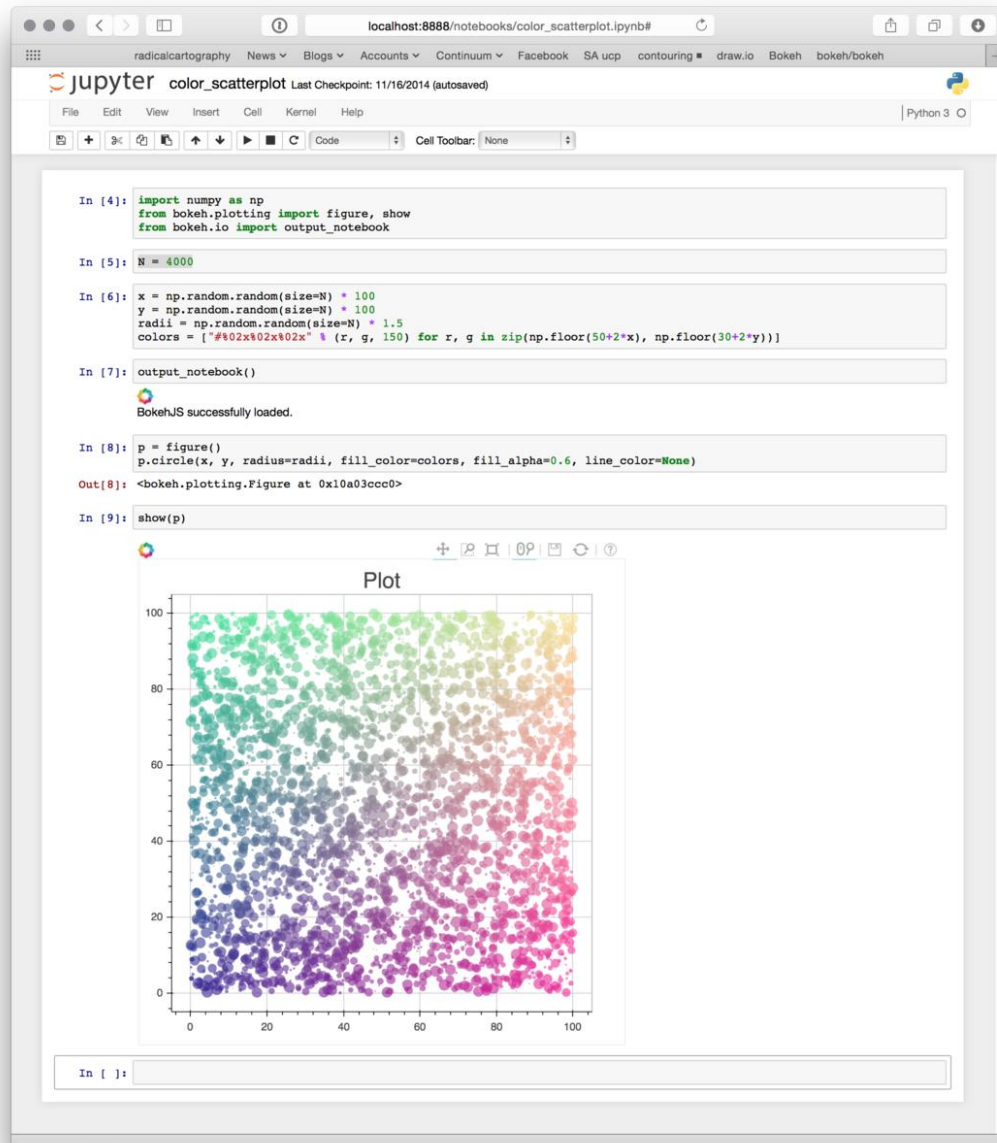
Python je također vrlo moćan jezik kada govorimo o strojnom učenju. Sposoban je provoditi kompleksne statističke izračune, graditi algoritme strojnog učenja, manipulirati podacima i analizirati ih, stvarati razne vizualizacije podataka te izvoditi razne druge zadatke vezane uz rad s podacima. Također, postoje mnoge knjižnice koje dodatno olakšavaju pisanje koda za analizu podataka i strojno učenje te čine kod bržim i efikasnijim, kao što su Tensorflow i Keras [14].

#### 4.1.2. Jupyter Notebook

Jupyter Notebook je aplikacija koja nam pruža okruženje za interaktivno programiranje, analizu podataka i prikaz rezultata. Omogućava korisnicima da kombiniraju kod, tekst, vizualizacije i matematičke formule u jednom dokumentu, stvarajući dinamične i interaktivne „notebookove“ odnosno bilježnice. Alat je u potpunosti otvorenog koda. [15]

Blokovi koda, koji se nazivaju ćelijama, mogu se pokretati zasebno što omogućava izvršavanje željenih dijelova koda u stvarnom vremenu i time interaktivno istraživanje podataka. Ovo je vrlo korisno za eksperimentiranje s različitim pristupima analize podataka ili provjeravanje

ispravnosti koda. U konkretnom slučaju ovog rada, Jupyter Notebook je imao veliku prednost zbog toga što su se analiza podataka, arhitektura modela, izračuni, trening, vizualizacije i interpretacija rezultata, sve moglo prikazati u jednom dokumentu. Slikom 4.1. prikazan je izgled Jupyter Notebook sučelja i kratki kod koji pokazuje mogućnost da se izračuni, obrada podataka i vizualizacija sve obavlja na jednom mjestu.



Slika 4.1. Izgled sučelja i demonstracija mogućnosti Jupyter Notebooka (preuzeto iz [16])

#### 4.1.3. Tensorflow

Tensorflow je besplatna knjižnica otvorenog koda za strojno učenje i umjetnu inteligenciju. Ima vrlo široki spektar primjene, ali posebno se fokusira na rad s dubokim neuronskim mrežama. Prva verzija knjižnica objavljena je 2015. godine, a nova i unaprijeđena verzija pod imenom „Tensorflow 2.0“ objavljena je 2019. godine i ona je korištena u ovom radu. Tensorflow se može koristiti u raznim programskim jezicima, kao što su Python, Javascript, C++ i Java, što mu osigurava veliku fleksibilnost. [17]

Tensorflow ima praktičnu primjenu najčešće za klasifikaciju i detekciju (prepoznavanje objekata na slikama i analiza teksta), generiranje i sintezu (stvaranje umjetnih slika, glazbe ili teksta) te *reinforcement learning* (inteligentni sustavi koji uče putem interakcije s okolinom). Tensorflow uglavnom radi s višedimenzionalnim poljima koja se nazivaju tenzorima i postoje kao zasebni objekti. Neke od prednosti su mu da omogućava rad na više uređaja, uključujući CPU, GPU i TPU (*Tensor Processing Unit*), velika prilagodljivost i kontrola prilikom izgradnje modela te opširna i detaljna dokumentacija.

#### 4.1.4. Keras

Keras je knjižnica otvorenog koda koja pruža Python sučelje za strojno učenje i rad s neuronskim mrežama uz fokus na duboko učenje, odnosno služi kao sučelje za Tensorflow knjižnicu. Keras pokriva sve korake strojnog učenja, od procesiranja podataka preko podešavanja hiperparametara do konačnog modela. Razvijen je s fokusom na omogućavanje brzog eksperimentiranja.

Uz Keras možemo u potpunosti iskoristiti skalabilnost i kompatibilnost s više platformi koju nudi Tensorflow. Keras je dizajniran da smanji kognitivno opterećenje programera tako što će [18]:

- Ponuditi jednostavno, dosljedno sučelje
- Minimizirati broj potrebnih radnji za najčešće slučajeve korištenja
- Pružiti jasne poruke kod grešaka
- Pratiti princip progresivnog otkrivanja: lako je započeti s korištenjem, a zahtjevnije zadatke je moguće riješiti usputnim učenjem dok radimo na modelu
- Pomoći pisati jasan i čitljiv kod

Glavne strukture podataka u Kerasu su slojevi i modeli. Slojevi podrazumijevaju ulazne i izlazne transformacije podataka dok su modeli objekti koji povezuju slojeve i koji se tada mogu trenirati na podacima.

#### 4.1.5. Ostale Python knjižnice

Pandas je moćna i fleksibilna knjižnica otvorenog koda za analizu i manipulaciju podataka. Glavna struktura s kojom radi Pandas je DataFrame, odnosno dvodimenzionalna tablica s označenim redovima i stupcima. U ovom radu je Pandas knjižnica korištena za učitavanje podataka i obradu prije treniranja modela.

Scikit-learn ili skraćeno *sklearn* je knjižnica za strojno učenje i analizu podataka. Sadrži razne algoritme strojnog učenja za klasifikaciju, regresiju i *clustering* te alate za obradu i pripremu podataka. U ovom radu je uz Pandas knjižnicu korištena za obradu podataka prije treniranja modela.

Numpy je knjižnica koja omogućava brzu i učinkovitu manipulaciju numeričkim podacima. Pruža podršku za rad s velikim, višedimenzionalnim poljima i matricama, zajedno s velikim brojem matematičkih funkcija koje olakšavaju rad s tim poljima.

Matplotlib je knjižnica koja se koristi za stvaranje različitih vrsta grafova, grafikona i vizualnih prikaza podataka. Pomoću nje mogu se kreirati statičke, animirane i interaktivne vizualizacije kako bismo lakše predstavili i razumjeli svoje podatke.

Scipy je knjižnica za Python koja proširuje njegove mogućnosti za znanstvenu i tehničku analizu. Sadrži razne module za optimizaciju, linearnu algebru, statistiku, integraciju, interpolaciju, procesiranje signala i slika te druge znanstvene proračune. U ovom radu su iz Scipy knjižnice korišteni moduli za korelacijsku analizu kako bismo obradili krajnje rezultate rada.

Biopython je knjižnica za Python koja nudi niz alata za bioinformatičku analizu i manipulaciju bioloških podataka. Neki od alata omogućavaju rad s biološkim sekvencama, analizu genoma, analizu strukture proteina i slično. Također, omogućava programski pristup bazama podataka koje sadrže biološke informacije. U ovom radu knjižnica Biopython je korištena za izračun sličnosti među peptidima temeljene na njihovim sekvencama.



## 4.2. Podaci

Skup podataka koji je korišten za trening i testiranje modela sastavljen je od dva manja skupa podataka. Prvi sadrži 1050 sekvenci peptida zajedno s oznakom prisutnosti ili odsutnosti antiviralne aktivnosti (603 pozitivnih i 447 negativnih). Njemu je pridodan drugi skup podataka koji sadrži 9409 sekvenci peptida s oznakom prisutnosti ili odsutnosti antimikrobne aktivnosti (3708 pozitivnih i 5701 negativnih). Konačni skup dakle sadrži ukupno 10459 sekvenci peptida, a radi bolje preglednosti informacije o podjeli podataka prikazane su i tablicom 4.1.

Tablica 4.1. Sadržaj skupa podataka korištenog za treniranje i testiranje modela

	Pozitivni	Negativni	Ukupno
Antiviralna svojstva	603	447	1050
Antimikrobna svojstva	3708	5701	9409
Ukupno	4311	6148	10459

Sljedeći korak bio je izračunavanje značajki svih peptida u skupu podataka. Za to je korišten Python paket pod imenom *peptides* te je izračunato 88 značajki za svaki peptid. U nastavku se nalazi sažeti popis svih značajki koje su dostupne za izračun u paketu *peptides* i koje su izračunate za peptide u korištenom skupu podataka [19]:

- Statistike aminokiselina:
  - o Broj pojavljivanja određene aminokiseline u sekvenci peptida
  - o Učestalost pojavljivanja u sekvenci peptida
- QSAR deskriptori:
  - o BLOSUM indeksi
  - o Cruciani svojstva
  - o FASGAI vektori
  - o Kidera faktori
  - o MS-WHIM rezultati
  - o PCP deskriptori
  - o ProtFP deskriptori

- Sneath vektori
- ST-skale
- T-skale
- VHSE-skale
- Z-skale
- Profili sekvenci:
  - Profil hidrofobnosti
  - Profil momenta hidrofobnosti
  - Pozicija membrane
- Fizikalno-kemijska svojstva:
  - Alifatski indeks
  - Indeks nestabilnosti
  - Ukupni teoretski naboj
  - Izoelektrična točka
  - Molekularna masa
- Biološka svojstva
  - Strukturna klasa

Konačni skup podataka sa izračunatim značajkama svih peptida je veličine 10460 redaka te 89 stupaca. Programski ispis podataka iz pripadajuće datoteke prikazan je na slici 4.2., a u kojem je vidljiv izgled konačnog skupa podataka, njegove dimenzije i neke od izračunatih značajki.

	BLOSUM1	BLOSUM2	BLOSUM3	BLOSUM4	BLOSUM5	BLOSUM6	BLOSUM7	BLOSUM8	BLOSUM9	BLOSUM10	...
0	1.070000	-0.093333	0.480000	-0.380000	-0.690000	-0.303333	0.350000	0.293333	0.120000	0.080000	...
1	0.896667	0.390000	-0.430000	-0.013333	-0.130000	-0.963333	0.356667	-0.076667	-0.076667	0.453333	...
2	-0.790000	0.520000	-0.657500	0.147500	0.160000	0.087500	0.247500	0.045000	0.120000	-0.520000	...
3	-0.797500	0.405000	-0.612500	0.025000	0.105000	0.287500	0.170000	0.147500	0.020000	-0.685000	...
4	-0.687500	-0.587500	-0.827500	-0.217500	-0.047500	0.235000	-0.170000	0.202500	-0.137500	0.060000	...
...	...	...	...	...	...	...	...	...	...	...	...
10454	0.138400	-0.480400	0.110400	-0.094400	-0.119200	0.338800	0.312800	0.137200	0.204400	0.052000	...
10455	0.534286	-0.542857	-0.040000	-0.364286	0.488571	0.151429	0.467143	0.027143	0.854286	-0.175714	...
10456	0.008077	-0.551923	-0.281923	-0.137692	0.091923	0.205000	0.106154	-0.323462	0.217692	-0.190000	...
10457	0.395000	-0.175000	0.492500	-0.092500	-0.105000	0.685000	0.772500	0.485000	0.012500	0.142500	...
10458	0.468889	0.175556	0.132222	-0.290000	0.515556	0.192222	-0.045556	-0.107778	-0.087778	0.058889	...

10459 rows × 88 columns

Slika 4.2. Konačni skup podataka s izračunatim značajkama

Treba napomenuti da uz glavni skup podataka postoji i pomoćni skup podataka koji sadrži ranije spomenute oznake aktivnosti svih peptida. Ovaj skup podataka neće biti korišten prilikom treniranja modela jer autoenkoderima nisu potrebni *labeli* za učenje, ali će biti vrlo koristan prilikom vizualizacije kako bismo mogli prikazati peptide odgovarajućom bojom ovisno o vrsti njihove aktivnosti.

### 4.3. Metode

U ovom poglavlju bit će prikazane i objašnjene sve metode kojima se došlo do konačnih rezultata. To uključuje izgradnju modela VAE, enkodera i dekodera, obradu podataka kako bi ih VAE mogao koristiti za trening, optimizaciju parametara modela i sami trening. Većina ovog dijela koda je napisana po uzoru na Kerasov tutorial za VAE [20] i članak o izgradnji VAE sa GitHuba koji je napisao Christian Versloot [21].

Prvi dio koji se mora definirati prije nego što se krene u izgradnju same arhitekture VAE je sloj za uzorkovanje, odnosno *sampling layer*. Kao što mu ime govori, njegov zadatak je da uzima uzorke  $z$  iz latentnog prostora, gdje je  $z$  vektor koji predstavlja enkodirani peptid odnosno njegove značajke. U ovom dijelu definiramo i činjenicu da će broj dimenzija koje će sadržavati latentni

prostor biti 2. Razlog odabira dvodimenzionalnog latentnog prostora je prvenstveno jednostavnija vizualizacija latentnog prostora, iako za tako mali broj dimenzija očekujemo veći gubitak informacija nego što bi to bio slučaj s latentnim prostorom veće dimenzionalnosti.

### 4.3.1. Enkoder

Kreiranje enkodera je proces koji se sastoji od tri koraka: prvo ga moramo definirati, drugi korak je „trik reparametrizacije“ koji nam omogućava da povežemo enkoder s dekoderom i tako definiramo VAE kao cjelinu i na kraju treći korak u kojem instanciramo enkoder. Slikom 4.3. prikazan je kratki pregled enkodera, prikazujući slojeve i oblike njihovih izlaznih podataka.

```

Model: "encoder"
-----
Layer (type)                Output Shape          Param #             Connected to
-----
input_3 (InputLayer)        [(None, 88, 1)]      0                  []
conv1d_5 (Conv1D)           (None, 88, 8)        32                 ['input_3[0][0]']
max_pooling1d_2 (MaxPooling1D) (None, 44, 8)        0                  ['conv1d_5[0][0]']
conv1d_6 (Conv1D)           (None, 44, 16)       400                ['max_pooling1d_2[0][0]']
max_pooling1d_3 (MaxPooling1D) (None, 22, 16)       0                  ['conv1d_6[0][0]']
flatten_1 (Flatten)         (None, 352)          0                  ['max_pooling1d_3[0][0]']
dense_3 (Dense)             (None, 128)          45184              ['flatten_1[0][0]']
z_mean (Dense)              (None, 2)            258                ['dense_3[0][0]']
z_log_var (Dense)           (None, 2)            258                ['dense_3[0][0]']
sampling_1 (Sampling)       (None, 2)            0                  ['z_mean[0][0]',
                        'z_log_var[0][0]']
-----
Total params: 46,132
Trainable params: 46,132
Non-trainable params: 0

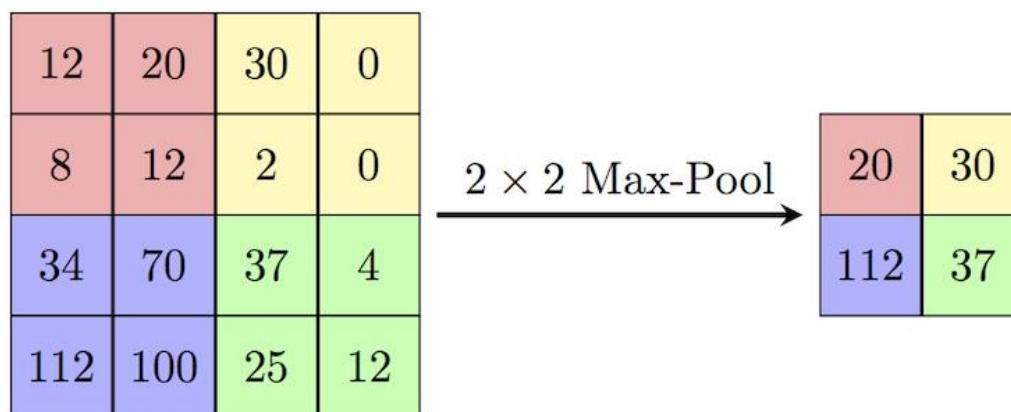
```

*Slika 4.3. Kratki pregled enkodera*

U nastavku su pojašnjeni svi slojevi i instanciranje enkodera:

- **Input\_3** - prvi sloj je ulazni sloj koji prihvaća ulazne podatke u obliku (88, 1), dakle jedan po jedan peptid opisan sa 88 izračunatih značajki
- **Conv1d\_5** - drugi sloj je jednodimenzionalni konvolucijski sloj s 8 filtera; izlazni tenzor sada ima oblik (None, 88, 8)

- **Max\_pooling1d\_2** – ovaj sloj obavlja operaciju *max pooling* na izlaznom tenzoru prethodnog sloja i tako mu smanjuje dimenzije na pola pa je sada izlazni tenzor oblika (None, 44, 8); ova operacija smanjuje dimenzionalnost podataka primjenom max filtera na podregije izvornih podataka; time sprječava *over-fitting* jer pruža apstraktni oblik izvornih podataka i smanjuje računalnu zahtjevnost modela smanjivanjem broja parametara koje treba učiti; slikom 4.4. prikazan je princip *max poolinga*
- **Conv1d\_6** – još jedan jednodimenzionalni konvolucijski sloj, ali ovaj put sa 16 filtera što dovodi do oblika izlaznog tenzora (None, 44, 16)
- **Max\_pooling1d\_3** – još jedan *max pooling* sloj koji dodatno smanjuje dimenzije podataka na oblik (None, 22, 16)
- **Flatten\_1** – ovaj sloj služi da „izravna“ višedimenzionalne tenzore u jednu dimenziju te je sada oblik podataka (None, 352); ovo moramo napraviti zato što idući slojevi zahtijevaju da podaci budu takvog oblika
- **Dense\_3** – *Dense* sloj se klasificira kao potpuno povezani sloj i on mapira tenzor iz prethodnog sloja u prikaz s manje dimenzija u latentnom prostoru te ovdje dolazimo do „najužeg“ dijela odnosno sredine autoenkodera; izlazni podaci su oblika (None, 128)
- **z\_mean** – potpuno povezani sloj koji mapira tenzor iz prethodnog sloja kao srednje vrijednosti u latentnom prostoru; oblik podataka je sada (None, latent\_dim) odnosno (None, 2) jer smo na početku definirali da će broj dimenzija latentnog prostora biti 2
- **z\_log\_var** – slično kao prethodni sloj, još jedan potpuno povezani sloj koji mapira tenzor iz Dense\_3 sloja, ali ovaj put kao logaritam varijance u latentnom prostoru; izlaz je također (None, 2)
- **Sampling\_1** – koristimo prethodno definirani sloj za uzorkovanje koji uzima *z\_mean* i *z\_log\_var* kao ulaz i generira uzorak iz latentnog prostora koristeći „trik reparametrizacije“; trik reparametrizacije služi da izmijenimo parametre uzorka u oblik koji je potreban da bismo mogli koristiti gradijentni spust za preciznu procjenu gradijenata



Slika 4.4. Princip *max pooling* operacije na dvodimenzionalnoj matrici (preuzeto iz [22])

### 4.3.2. Dekoder

Kreiranje dekodera je jednostavniji proces od kreiranja enkodera i sastoji se samo od dva koraka: definicije i instanciranja dekodera. Slikom 4.5. prikazan je kratki pregled dekodera, prikazujući slojeve i oblike njihovih izlaznih podataka.

```
Model: "decoder"
-----
```

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 2)]	0
dense_4 (Dense)	(None, 128)	384
dense_5 (Dense)	(None, 704)	90816
reshape_1 (Reshape)	(None, 88, 8)	0
conv1d_7 (Conv1D)	(None, 88, 16)	400
up_sampling1d_2 (UpSampling1D)	(None, 176, 16)	0
conv1d_8 (Conv1D)	(None, 176, 8)	392
up_sampling1d_3 (UpSampling1D)	(None, 352, 8)	0
conv1d_9 (Conv1D)	(None, 352, 1)	25
cropping1d_1 (Cropping1D)	(None, 88, 1)	0

```
-----
Total params: 92,017
Trainable params: 92,017
Non-trainable params: 0
```

*Slika 4.5. Kratki pregled dekodera*

U nastavku su pojašnjeni svi slojevi dekodera:

- **Input\_4** – prvi sloj je ulazni sloj dekodera koji kao ulaz očekuje točku iz latentnog prostora; podatak koji ovaj sloj dobiva na ulaz je oblika (None, 2) kao i u zadnjem sloju enkodera
- **Dense\_4** – ovaj potpuno povezani sloj mapira točku iz latentnog prostora u prikaz s više dimenzija; izlazni tenzor je oblika (None, 128)
- **Dense\_5** – još jedan potpuno povezani sloj koji mapira tenzor iz prethodnog sloja u prikaz s još više dimenzija što rezultira izlaznim oblikom (None, 704)

- **Reshape\_1** – ovaj sloj preoblikuje prethodni tenzor u tenzor s dimenzijom više i oblikom (None, 88, 8)
- **Conv1d\_7** – jednodimenzionalni konvolucijski sloj sa 16 filtera nakon kojeg izlazni tenzor ima oblik (None, 88, 16)
- **Up\_sampling1d\_2** – ovaj sloj obavlja povećanje uzorkovanja odnosno *upsampling* na prethodnom tenzoru i udvostručuje njegovu duljinu što na kraju daje oblik (None, 176, 16)
- **Conv1d\_8** – još jedan jednodimenzionalni konvolucijski sloj, ovaj put s 8 filtera što dovodi do oblika (None, 176, 8)
- **Up\_sampling1d\_2** – još jedan sloj za povećanje uzorkovanja koji ponovno udvostručava duljinu tenzora i daje mu oblik (None, 352, 8)
- **Conv1d\_9** – konačni jednodimenzionalni konvolucijski sloj s jednim filterom koji izlaznom tenzoru daje oblik (None, 352, 1)
- **Cropping1d\_1** – sloj za izrezivanje koji uklanja potreban broj značajki da bi konačan tenzor imao oblik (None, 88, 1)

#### 4.3.3. Model varijacijskog autoenkodera

U trenutku kada imamo definiran enkoder i dekoder možemo ih povezati i tako dobiti konačnu arhitekturu varijacijskog autoenkodera. Kao što vidimo prema kratkom isječku koda i njegovom ispisu na slici 4.6., ulaz u VAE, odnosno enkoder, je oblika (None, 88, 1). Izlaz iz enkodera, odnosno ulaz u dekoder je oblika (None, 2). Izlaz iz VAE, odnosno dekodera, je također oblika (None, 88, 1). Dakle možemo reći da je izlaz cijelog VAE zapravo izvorni ulaz, enkodiran enkoderom i dekodiran dekoderom.

```
vae = VAE(encoder, decoder)
vae.build(input_shape=(None, 88, 1))
vae.summary()
```

Model: "vae"

Layer (type)	Output Shape	Param #
encoder (Functional)	[(None, 2), (None, 2), (None, 2)]	46132
decoder (Functional)	(None, 88, 1)	92017
=====		
Total params: 138,155		
Trainable params: 138,149		
Non-trainable params: 6		

Slika 4.6. Instanciranje i kratki pregled varijacijskog autoenkodera

Međutim, treba napomenuti kako je za izgradnju modela definirana nova klasa *VAE* koja nasljeđuje klasu *keras.Model* jer su neke metode trebale biti prilagođene za ovaj specifičan slučaj. Unutar klase *VAE* definirana je i konstruktor metoda u kojoj ćemo spojiti enkoder i dekoder u jedan model. Ona prihvaća dva argumenta, naravno enkoder i dekoder, koje smo instancirali ranije. Nakon toga moramo inicijalizirati atribut metrika koje će pratiti gubitke pri treniranju modela. Metrike koje ćemo koristiti su ukupan gubitak, gubitak rekonstrukcije i KL gubitak, odnosno KL divergencija.

Slijedi definiranje *train\_step* i *test\_step* metoda kako bismo prilagodili procese treniranja i testiranja. Prva na redu je *train\_step* metoda. Ona definira kako izgleda jedan korak procesa treniranja modela *VAE*. Unutar metode *train\_step* prosljeđujemo ulazne podatke u enkoder kako bismo dobili latentne varijable *z\_mean*, *z\_log\_var* i *z*, a zatim prosljeđujemo latentnu varijablu *z* u dekoder kako bismo dobili rekonstruirane podatke. Slijede izračuni gubitaka: gubitak rekonstrukcije je srednja kvadrirana greška između ulaznih i rekonstruiranih podataka sumirana preko odgovarajućih osi, gubitak KL divergencije predstavlja razliku između naučene distribucije i standardne Gaussove distribucije, a *total\_loss* odnosno ukupni gubici su suma gubitaka rekonstrukcije i gubitaka KL divergencije. Gubici KL divergencije su prilikom zbrajanja s gubicima rekonstrukcije množeni faktorom  $\beta = 0,27$  kako bi uskladili težine ovih dvaju gubitaka. Razlog i način odabira faktora  $\beta$  bit će predstavljen u poglavlju s rezultatima jer su oni usko povezani. Na



kraju izračunavamo gradijente ukupnih gubitaka s obzirom na težine modela te ih primjenjujemo na iste težine te ažuriramo stanja metrika koje mjere gubitke kako bismo ih mogli pratiti za vrijeme procesa treniranja.

Nakon *train\_step* metode slijedi *test\_step* metoda. Ona definira kako izgleda jedan korak testiranja odnosno validacije modela VAE i vrlo je slična *train\_step* metodi, osim što ne izračunava gradijente i ne ažurira težine s obzirom da se radi o fazi evaluacije. I u ovoj metodi izračunavamo latentne varijable  $z\_mean$ ,  $z\_log\_var$  i  $z$  pomoću enkodera. Zatim rekonstruiramo podatke pomoću dekodera i latentne varijable  $z$ . Slijede izračuni gubitaka na isti način kao i u *train\_step* metodi koje proslijeđujemo definiranim metrikama pomoću kojih će se pratiti proces evaluacije.

#### 4.3.4. Obrada podataka

Nakon definirane arhitekture modela, potrebno je pripremiti ulazne podatke na način da ih model VAE može prihvatiti i efikasno učiti na njima.

Najprije učitavamo podatke iz datoteka koje su opisane u poglavlju 4.2. i koje sadrže sekvence i izračunate značajke peptida te oznake aktivnosti peptida. Slijedi podjela podataka u skupove za trening i testiranje. Podatke dijelimo na način da 20% podataka bude korišteno za testiranje, a ostalih 80% će se koristiti za trening. Zatim odvajamo sekvence peptida u dvije zasebne varijable koje će nam koristiti kasnije pri izračunu sličnosti između peptida temeljene na njihovim sekvencama. Slično tome, odvajamo izračunate značajke peptida u dvije zasebne varijable koje ćemo koristiti prilikom treninga jer želimo da model uči isključivo pomoći izračunatih značajki, bez znanja o sekvencama peptida. Nakon toga normaliziramo podatke koji sadrže izračunate značajke peptida *min-max* postupkom, prema izrazu (4.1.):

$$x_{skalirano} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (4.1.)$$

gdje je:

$x$  izvorna vrijednost koju želimo skalirati,

$x_{min}$  najmanja vrijednost koju podatak može poprimiti,

$x_{max}$  najveća vrijednost koju podatak može poprimiti.

Na ovaj način postavljamo vrijednosti podataka u raspon od 0 do 1 na način da najmanju vrijednost značajke postavljamo kao 0, najveću vrijednost značajke postavljamo kao 1, a ostatak vrijednosti skaliramo u odnosu na minimum i maksimum. To radimo za svaki stupac posebno tako da smo sigurni da će sve značajke biti ispravno normalizirane bez obzira na moguće razlike u njihovim izvornim rasponima vrijednosti. Na kraju svim varijablama koje sadrže naše podatke dodajemo još jednu dimenziju, odnosno novu os kako bi oblik podataka odgovarao onome što model očekuje na ulazu.

#### 4.3.5. Optimizacija hiperparametara

Odabir hiperparametara napravljen je prilagođenom *grid search* metodom koja radi na način da iscrpno isprobava sve kombinacije hiperparametara u treningu, evaluira ih te na kraju vraća najbolju kombinaciju hiperparametara. Isječak koda kojim je odrađena optimizacija hiperparametara prikazan je slikom 4.7.

```

learn_rate = [0.001, 0.005, 0.008, 0.01]
batch_size = [32, 64, 128, 256]

param_grid = dict(learn_rate=learn_rate, batch_size=batch_size)

best_score = None
best_params = {}

for lr in learn_rate:
    for bs in batch_size:
        print("Learning rate: ", lr)
        print("Batch size: ", bs)

        vae = VAE(encoder, decoder)
        vae.compile(optimizer=keras.optimizers.Adam(learning_rate=lr))

        reduce_lr = ReduceLRonPlateau(monitor='loss', factor=0.5, patience=10, verbose=1, min_lr=1e-6)
        early_stop = EarlyStopping(monitor='loss', patience=20, verbose=1, restore_best_weights=True)

        vae.fit(x_train, batch_size=bs, epochs=30, validation_split = 0.2,
                callbacks=[reduce_lr, early_stop])

        score = vae.evaluate(x_train)
        print(score)

        if best_score is None or score < best_score:
            best_score = score
            best_params = {'learn_rate': lr, 'batch_size': bs}

print("Best Hyperparameters:", best_params)
print("Best Score:", best_score)

lr = best_params['learn_rate']
bs = best_params['batch_size']

```

Slika 4.7. Optimizacija hiperparametara

Najprije definiramo prostor pretrage hiperparametara za dvije varijable koje želimo optimizirati, *learning\_rate* i *batch\_size*. Prostor pretrage prikazan je tablicom 4.2. Vrijednosti za prostor pretrage su odabrane na način da smo uzeli neke od najčešćih vrijednosti ovih dvaju hiperparametara te dodali neke koje bi mogle odgovarati za naš model. U samoj metodi stvara se i kompajlira novi model VAE za svaku vrijednost *learning\_rate* iz prostora pretrage, te se model trenira sa svakom vrijednosti *batch\_size* iz prostora pretrage tijekom 30 epoha, validacija se izvršava na 20% podataka određenih za validaciju iz skupa podataka za trening i na kraju se model evaluira na ulaznim podacima kako bismo dobili rezultat uspješnosti modela. Na taj način imamo model VAE za svaku kombinaciju definiranih hiperparametara. Nakon evaluacije, provjeravamo je li trenutni rezultat bolji od prethodnog najboljeg rezultata, odnosno jesu li gubici trenutnog modela manji od gubitaka prethodnih najmanjih gubitaka i u tom slučaju ažuriramo vrijednosti najboljeg

rezultata i najboljih hiperparametara. Nakon što se ispituju sve kombinacije, ispisujemo najbolji rezultat i odgovarajuće najbolje hiperparametre.

*Tablica 4.2. Prostor pretrage hiperparametara*

PARAMETAR	VRIJEDNOSTI			
<i>learn_rate</i>	0,001	0,005	0,008	0,01
<i>batch_size</i>	32	64	128	256

Rezultati opisane metode nalaze se u tablici 4.3. Možemo iščitati da su vrijednosti hiperparametara koje su postigle najbolje rezultate: 0,01 za *learn\_rate* i 64 za *batch\_size*, a vrijednosti gubitaka koje su postignute: 0,689 za ukupne gubitke, 0,484 za gubitke rekonstrukcije i 0,759 za gubitke KL divergencije.

*Tablica 4.3. Rezultati optimizacije hiperparametara*

	Najbolje postignute vrijednosti
<i>Learn_rate</i>	0,01
<i>Batch_size</i>	64
Ukupni gubici	0,689
Gubici rekonstrukcije	0,484
Gubici KL divergencije	0,759

#### 4.3.6. Trening

Nakon optimizacije hiperparametara opisane u prethodnom poglavlju, imamo sve što nam je potrebno kako bismo mogli započeti s kompajliranjem i treniranjem modela VAE.

Prije svega moramo instancirati model VAE tako što ćemo spojiti prethodno definirani enkoder i dekoder. Slijedi kompajliranje modela slično kao i kod optimizacije hiperparametara,

samo što ovaj put kao *learning\_rate* postavljamo vrijednost koju smo dobili kao najbolju prilikom optimizacije hiperparametara. Nakon toga stvaramo instance dvaju povratnih poziva (*callbacks*) koji će dodatno optimizirati treniranje.

Prvi od njih je povratni poziv *ReduceLRonPlateau*. On prati gubitke prilikom treninga i u slučaju da se ukupni gubici prestanu smanjivati tijekom određenog broja epoha, smanjuje *learning\_rate*. Prema definiranim parametrima, ako se tijekom deset epoha treninga ne smanje ukupni gubici, *learning\_rate* se smanjuje na pola, odnosno množi faktorom 0,5. Minimalni dozvoljeni *learning\_rate* je definiran kao  $1e-6$ , što znači da se *learning\_rate* nakon što dostigne tu vrijednost više neće smanjivati.

Drugi povratni poziv jest *EarlyStopping*. On također prati gubitke tijekom treninga, ali u slučaju da se ukupni gubici prestanu smanjivati određen broj epoha u potpunosti zaustavlja trening. Prema definiranim parametrima, ovaj povratni poziv će se aktivirati ako se ukupni gubici ne smanje tijekom 20 epoha treninga. Kako je prethodni povratni poziv postavljen da se aktivira nakon samo 10 epoha, to znači da će se uvijek on aktivirati prije, a *EarlyStopping* će se aktivirati tek u slučaju da nakon smanjenja vrijednosti *learning\_rate* ukupni gubici i dalje ne budu padali. Unutar ovog povratnog poziva definiramo da se nakon njegovog izvršenja vraćaju najbolje težine modela koje su bile postignute tijekom treninga.

U ovom trenutku započinjemo s treniranjem modela. Model treniramo kroz 100 epoha na definiranom skupu podataka za treniranje modela, *batch\_size* postavljamo na vrijednost koju smo dobili kao najbolju prilikom optimizacije hiperparametara te nakon svake epohe provodimo validaciju na predodređenih 20% podataka. Rezultati samog treniranja bit će prikazani u sljedećem poglavlju.

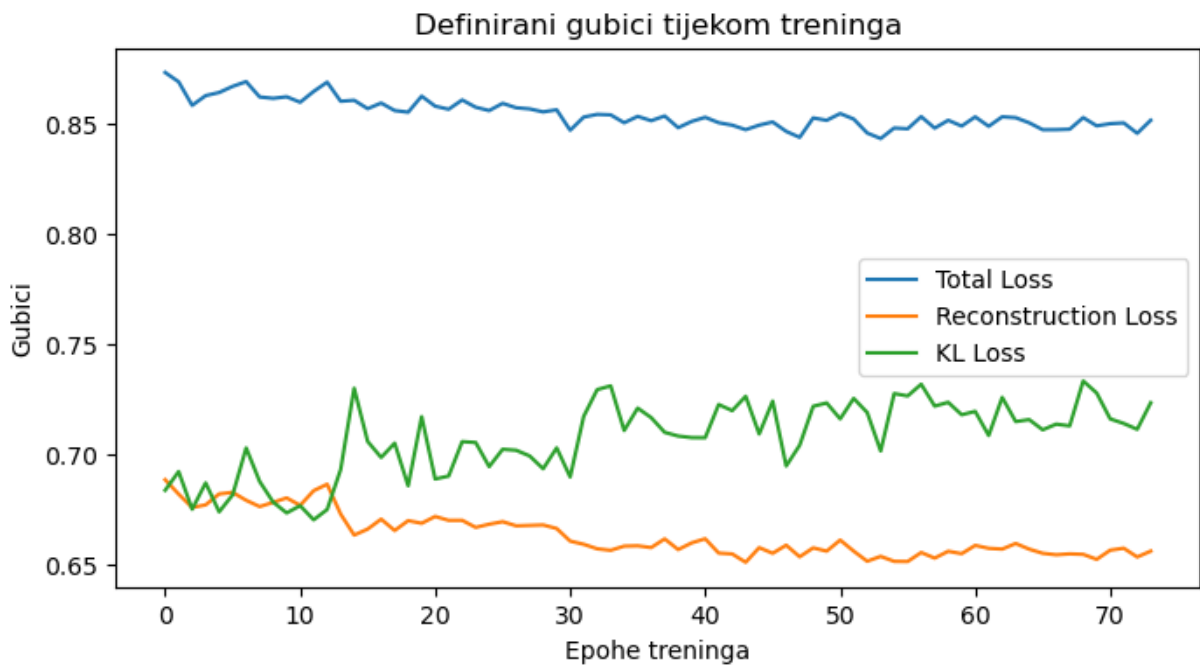
## 5. REZULTATI

### 5.1. Rezultati treninga

Nakon treniranja modela na način objašnjen u prethodnom poglavlju dolazimo do rezultata koje možemo opisati prethodno definiranim metrikama gubitaka. U ovom slučaju putem optimizacije hiperparametara odabrane su vrijednosti 0,01 za *learning\_rate* i 64 za *batch\_size*. U tablici 5.1. nalaze se vrijednosti gubitaka koje su dobivene nakon jednog treninga, a slikom 5.1. prikazano je kako su se kretale vrijednosti gubitaka tijekom treninga.

Tablica 5.1. Vrijednosti gubitaka nakon jednog treninga

	Ukupni gubici	Gubici rekonstrukcije	Gubici KL divergencije
Gubici treninga	0,8413	0,6575	0,7109
Gubici validacije	0,8606	0,6280	0,8614



Slika 5.1. Vizualizacija definiranih gubitaka tijekom treninga

Prema podacima iz tablice 5.1. možemo zaključiti da su gubici rekonstrukcije i gubici KL divergencije dobro izbalansirani jer se njihove vrijednosti nalaze u sličnim rasponima uz prisutnost manjih oscilacija. To nam je bitno jer u isto vrijeme želimo imati i dobru rekonstrukciju podataka i dobra svojstva latentnog prostora. Na slici 5.1. vidimo da ukupni gubici i gubici rekonstrukcije gotovo konstantno padaju tijekom treninga, što smo i htjeli postići, ali na štetu gubitaka KL divergencije. Međutim, to ne predstavlja problem jer uz ovakve gubitke KL divergencije latentni prostor i dalje zadržava dobra svojstva.

Izravno zbrojeni gubici rekonstrukcije i gubici KL divergencije ne daju iznos ukupnih gubitaka zbog njihovih različitih težina, odnosno zbog toga što se prije zbrajanja gubici KL divergencije množe faktorom  $\beta = 0,27$ . Gubicima KL divergencije dodan je taj faktor kako bi smanjili njihov značaj u ukupnim gubicima i tako prisilili model da se više fokusira na smanjivanje gubitaka rekonstrukcije. Bez faktora  $\beta$ , odnosno kada je faktor  $\beta$  jednak 1, naš model krene smanjivati gubitke isključivo smanjivanjem gubitaka KL divergencije. Zbog toga pretpostavljamo da su naši ulazni podaci takvog tipa da je modelu puno lakše smanjiti ukupne gubitke smanjivanjem gubitaka KL divergencije pa to i koristi. U tom slučaju dobivamo izuzetno kvalitetnu regularnost latentnog prostora, ali s druge strane jako lošu rekonstrukciju, što nam naravno nije cilj. Zbog toga je uveden faktor  $\beta$ , a njegovu vrijednost smo dobili tako što smo trenirali četiri različita modela s različitim vrijednostima faktora  $\beta$  koje bi mogle pružiti zadovoljavajuće rezultate i odabrali onu za koju model ima najbolji omjer između kvalitete rekonstrukcije i regularnosti latentnog prostora. U tablici 5.2. nalaze se dobivene vrijednosti gubitaka i metrika koje određuju točnost modela za svaki faktor  $\beta$ . Metrike točnosti bit će predstavljene u nastavku.

*Tablica 5.2. Rezultati modela za pojedine vrijednosti faktora  $\beta$*

$\beta$	Ukupni gubici	Gubici rekonstrukcije	Gubici KL divergencije	NSURLP	PKK	RMSE
0,25	0,8429	0,6356	0,8041	0,091	0,799	0,085
<b>0,27</b>	<b>0,8413</b>	<b>0,6575</b>	<b>0,7109</b>	<b>0,103</b>	<b>0,7917</b>	<b>0,0869</b>
0,3	0,8802	0,7419	0,4529	0,121	0,763	0,092
0,35	0,8946	0,7912	0,3193	0,167	0,739	0,096

Iz tablice 5.2. je vidljivo da je za faktor  $\beta = 0,27$  dobiven najbolji omjer gubitaka rekonstrukcije i gubitaka KL divergencije. Uz ovakav omjer gubitaka, regularnost latentnog prostora je i dalje zadovoljena, što ćemo i vizualno prikazati u sljedećem poglavlju, a vrijednosti metrika kojima mjerimo točnost modela pokazuju da je model uspješan pri rekonstrukciji. Uveli smo tri metrike kojima mjerimo točnost modela: normaliziranu srednju udaljenost rekonstrukcije u latentnom prostoru (NSURLP), Pearsonov koeficijent korelacije (PKK) i kvadratni korijen srednje kvadratne pogreške (RMSE – *Root Mean Square Error*).

NSURLP označava normaliziranu srednju udaljenost rekonstrukcije u latentnom prostoru. Izračunali smo je prema izrazu (5.1.):

$$NSURLP = \frac{\sum_{i=0}^N (p_{ei}[i] - p_{er}[i])}{N * D} \quad (5.1.)$$

gdje je:

$p_{ei}$  točka koja predstavlja enkodirani izvorni peptid,

$p_{er}$  točka koja predstavlja enkodirani rekonstruirani peptid,

$N$  ukupan broj peptida u skupu podataka,

$D$  duljina dijagonale latentnog prostora.

Duljinu  $D$  za izraz (5.1.) odredili smo na način da smo za testni skup podataka enkodiran u latentni prostor odredili koordinate točaka s najmanjim i najvećim iznosima na obje osi i tako odredili granice latentnog prostora unutar kojih se nalaze svi enkodirani podaci. Zatim smo uzeli dijagonalu tog prostora kao najveću moguću udaljenost koja može postojati između izvornog i rekonstruiranog peptida. Naravno, u teoriji ta udaljenost može biti i veća, ali takva rekonstrukcija se niti ne bi mogla smatrati korektnom. Nakon toga izračunavamo udaljenosti između svih peptida enkodiranih u latentni prostor i njihovih pripadajućih rekonstruiranih, odnosno dekodiranih i ponovno enkodiranih, peptida. Te udaljenosti zbrajamo te zatim dijelimo s ukupnim brojem peptida u testnom skupu podataka kako bi dobili srednju udaljenost i na kraju dijelimo s dijagonalom latentnog prostora kako bi tu vrijednost normalizirali. Dakle metriku NSURLP možemo shvatiti i kao postotak greške, s obzirom na to da će se ona u gotovo svakom slučaju nalaziti između 0 i 1, a manja vrijednost označava bolji rezultat, to jest manje odstupanje od izvornih podataka. Za naš konkretni model s



faktorom  $\beta = 0,27$ , iznos metrike jest  $NSURLP = 0,103$ , što bi značilo da model rekonstruira peptide s prosječnom greškom od 10,3%.

PKK označava Pearsonov koeficijent korelacije koji se općenito koristi za mjerenje linearne povezanosti između dvije varijable, recimo X i Y, a računa se prema izrazu (5.2.):

$$PKK_{X,Y} = \frac{cov(X,Y)}{\sigma_X * \sigma_Y} \quad (5.2.)$$

gdje je:

$cov(X,Y)$  kovarijanca između dvije promatrane varijable,

$\sigma_X$  standardna devijacija varijable X,

$\sigma_Y$  standardna devijacija varijable Y.

U ovom slučaju uspoređujemo izvorni skup testnih podataka s dekodiranim, odnosno rekonstruiranim skupom testnih podataka iz latentnog prostora. Prema tome, rekonstruirani skup podataka bi trebao imati slične obrasce i promjene u vrijednostima kao izvorni skup podataka te bi PKK trebao dobro opisati njihovu povezanost, a time i sličnost. Vrijednost PKK će se također uvijek nalaziti između 0 i 1 gdje veća vrijednost označava bolji rezultat, odnosno veću sličnost rekonstruiranih i izvornih podataka. Kao referentnu vrijednost p-vrijednosti uzimamo standardnih 0,05, što će značiti da u slučaju dobivene p-vrijednosti koja je iznosom manja od referentne imamo potvrdu da je iznos PKK statistički značajan i smijemo ga tumačiti kao takvog. U slučaju p-vrijednosti većeg iznosa od referentne, naš PKK neće biti statistički značajan, odnosno ne možemo biti sigurni u njegovu vjerodostojnost. Dobiveni iznos  $PKK = 0,7917$  označava izraženu linearnu vezu, što možemo protumačiti i kao sličnost dekodiranih i izvornih podataka od 79,17%. Iznos pripadne p-vrijednosti jest jednak nuli, što znači da je dobiveni PKK statistički značajan.

RMSE označava kvadratni korijen srednje kvadratne pogreške. On se izračunava na način da kvadriramo razlike između vrijednosti izvornih i rekonstruiranih podataka, sumiramo ih, izračunamo srednju vrijednost tih razlika i na kraju izračunamo korijen srednje vrijednosti, prema izrazu (5.3.):

$$RMSE = \sqrt{\frac{\sum_{i=0}^N (p_{izv}[i] - p_{rek}[i])^2}{N}} \quad (5.3.)$$

gdje je:

$p_{izv}$  točka koja predstavlja izvorni peptid,

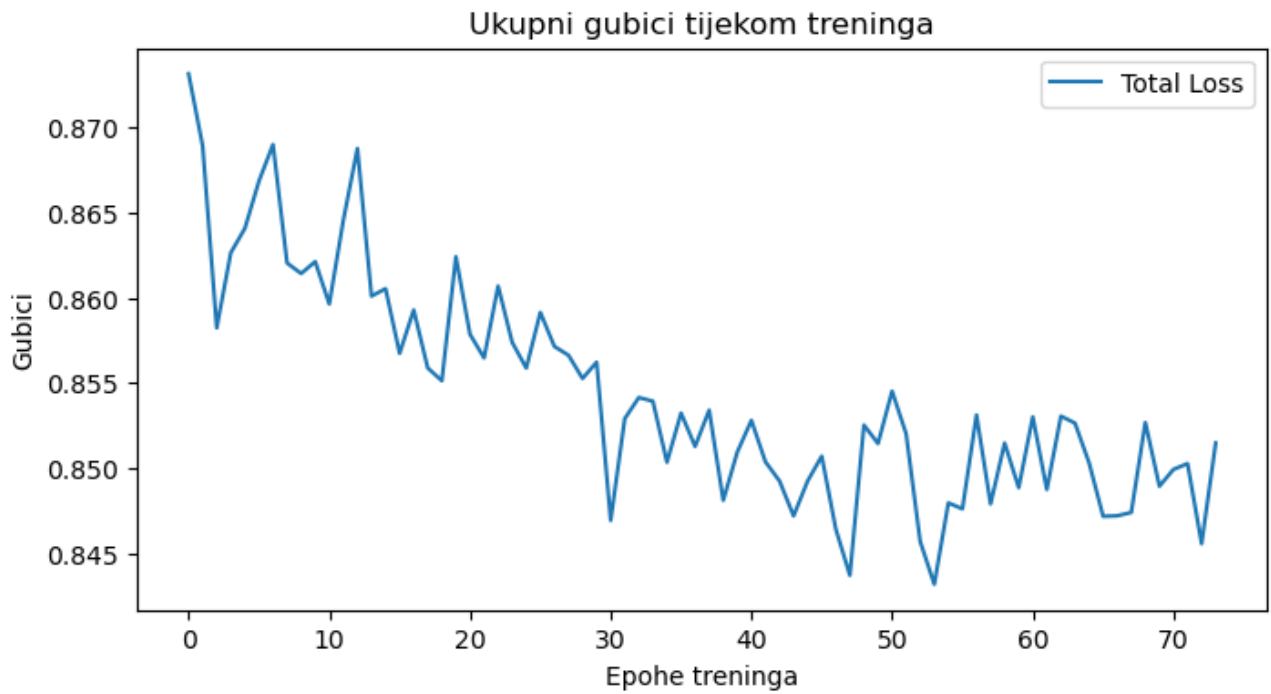
$p_{rek}$  točka koja predstavlja rekonstruirani peptid,

$N$  ukupan broj peptida u skupu podataka.

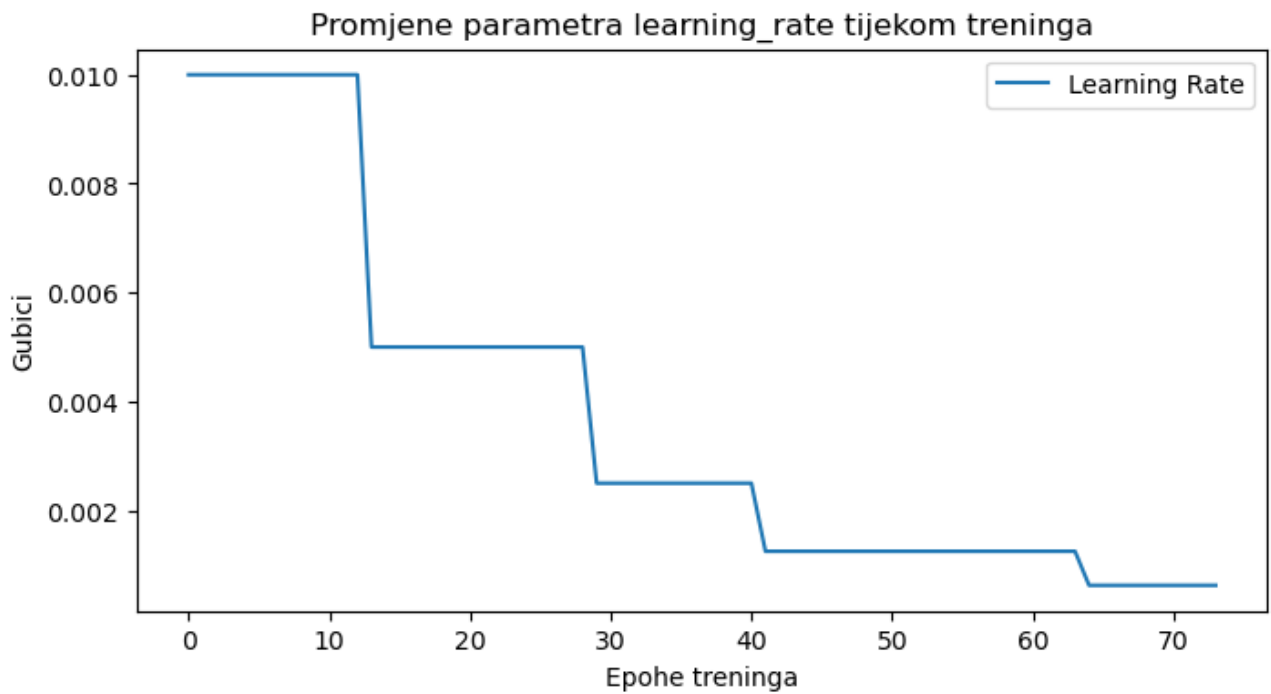
Općenito je metrika RMSE relativna i može poprimiti bilo koju realnu vrijednost, ali s obzirom na to da su naši ulazni podaci normalizirani, samim time će i izlazni podaci biti normalizirani i metrika RMSE će također moći poprimiti samo vrijednosti između 0 i 1 gdje će manja vrijednost označavati manju grešku, odnosno bolji rezultat modela. Za ovaj model je dobiven iznos  $RMSE = 0,0869$ , što možemo protumačiti kao grešku od 8,69%. Ovakav iznos je u skladu s iznosima prethodnih metrika.

Uzimajući u obzir rezultate predstavljenih metrika možemo zaključiti da je model uspješan u rekonstruiranju podataka. Rezultati rekonstrukcije se mogu poboljšati, ali tada bismo izgubili regularnost latentnog prostora koju model VAE zahtijeva. Uz ovakve rezultate, model i dalje zadržava regularnost latentnog prostora, a rekonstrukcija ostaje u prihvatljivim granicama.

Ono što također možemo vidjeti tijekom treninga je kako rade definirani povratni pozivi s obzirom na to kako se kreću ukupni gubici modela. Slikom 5.2. prikazano je kako se kreću ukupni gubici tijekom treninga, a slikom 5.3. prikazano je kako se kreće *learning\_rate* tijekom treninga, što nam ujedno govori i kada se aktiviraju definirani povratni pozivi. Povratni poziv *ReduceLRonPlateau* aktivira se nakon što model prođe 10 epoha treninga u kojima ukupni gubici ne padnu ispod prethodne nabolje postignute vrijednosti, odnosno prethodne najniže vrijednosti gubitaka. Tom aktivacijom, trenutni *learning\_rate* smanjuje se na polovicu dosadašnjeg iznosa, a u ovom slučaju je početna vrijednost iznosila 0,01. Također, nakon epohe 74 aktivira se i povratni poziv *EarlyStopping*, što možemo zaključiti iz toga što je model prestao s treningom prije definiranih 100 epoha koje je trebao proći. Povratni poziv *EarlyStopping* ovdje je aktiviran iz razloga što je namješten da se poziva u slučaju ako ukupni gubici treninga ne padaju tijekom 20 epoha treninga pa možemo pretpostaviti kako gubici nisu padali nakon epohe 54, što je vidljivo i na slici 5.2. Tako dolazimo do 20 epoha tijekom kojih ukupni gubici nisu pali ispod prethodne najbolje vrijednosti i model prestaje s treningom. Na kraju se vraćaju težine modela iz epohe u kojoj su postignuti najmanji gubici, a to je u ovom slučaju epoha 54.



Slika 5.2. Vizualizacija ukupnih gubitaka tijekom treninga

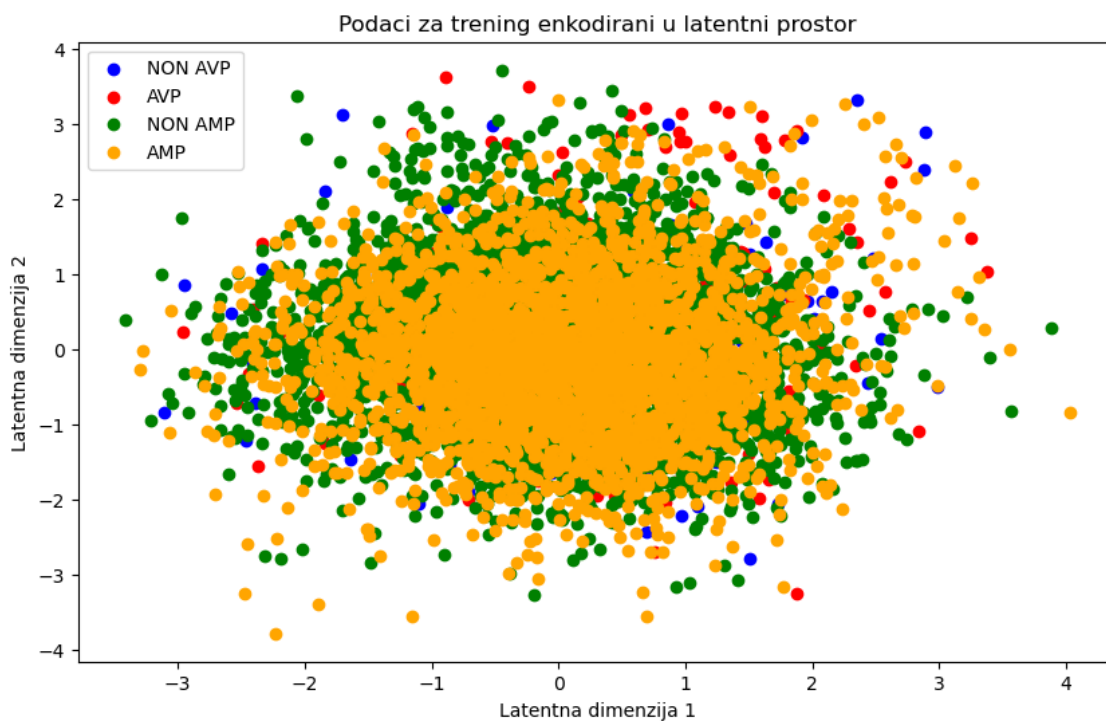


Slika 5.3. Promjene parametra learning\_rate tijekom treninga

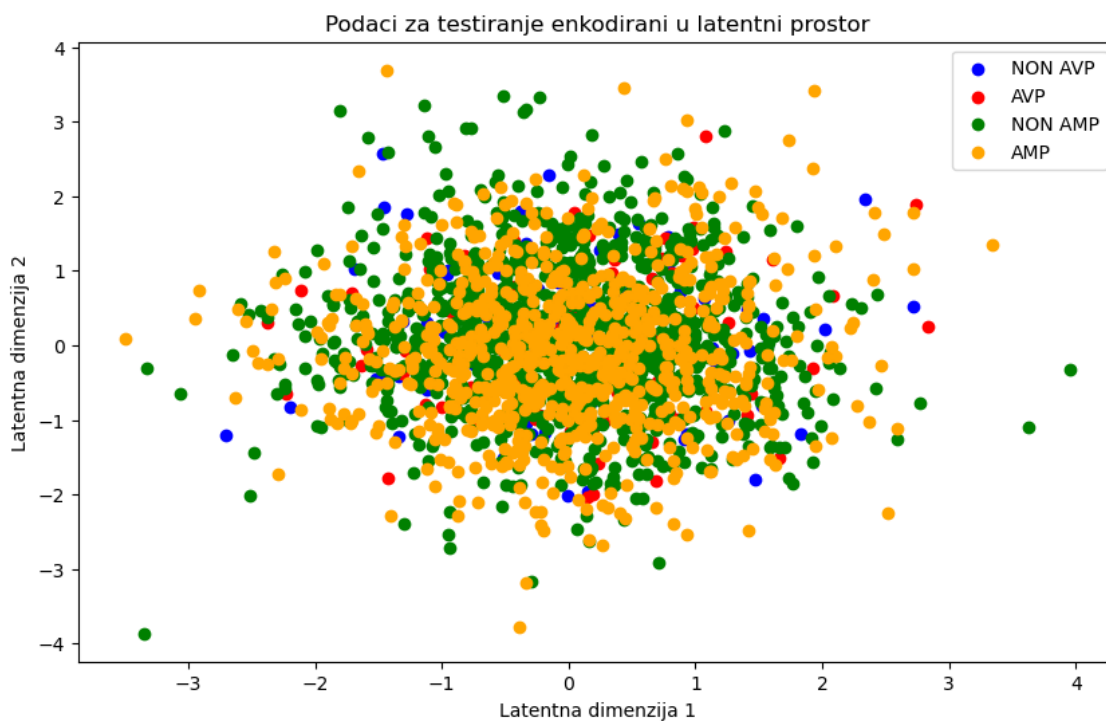
## 5.2. Vizualizacije

U ovom poglavlju bit će prikazane najvažnije vizualizacije podataka kojima će se jednostavno predočiti rezultati rada modela VAE. Razlog zašto je arhitektura modela takva da u latentnom prostoru ima samo dvije dimenzije je upravo ovaj korak vizualizacije jer na taj način možemo vrlo jednostavno prikazati i shvatiti što se događa u latentnom prostoru. Na gotovo svim slikama su peptidi prikazani točkama obojenim prema njihovim oznakama koje odgovaraju vrstama i aktivnostima peptida. U legendama grafova oznake 'AVP' i 'AMP' označavaju peptide s potvrđenim antiviralnim i antimikrobnim aktivnostima, a 'NON AVP' i 'NON AMP' peptide za koje je potvrđeno da nemaju antiviralne i antimikrobne aktivnosti.

Na slikama 5.4. i 5.5. prikazani su ulazni podaci za trening i testiranje enkodirani u latentni prostor. Ovdje je jasno vidljivo da je zadovoljena potpunost koju VAE zahtijeva od latentnog prostora jer se distribucije maksimalno preklapaju i da gotovo i nema „praznog“ prostora u glavnom području latentnog prostora koji bi kad se dekodira mogao davati nejasne rezultate. Ono što bi moglo biti problematično jest kontinuitet latentnog prostora. S obzirom na to da su točke koje predstavljaju peptide obojene različitim bojama prema njihovoj vrsti, vidimo da se sve vrste uglavnom „preklapaju“, odnosno pokrivaju isti dio latentnog prostora što znači da dekodiranjem bilo kojeg dijela latentnog prostora zapravo možemo dobiti bilo koju vrstu peptida. Druga stvar koju možemo pretpostaviti iz toga jest da se značajke peptida prema vrsti aktivnosti možda i ne razlikuju dovoljno da bi ih model VAE mogao razlučiti.

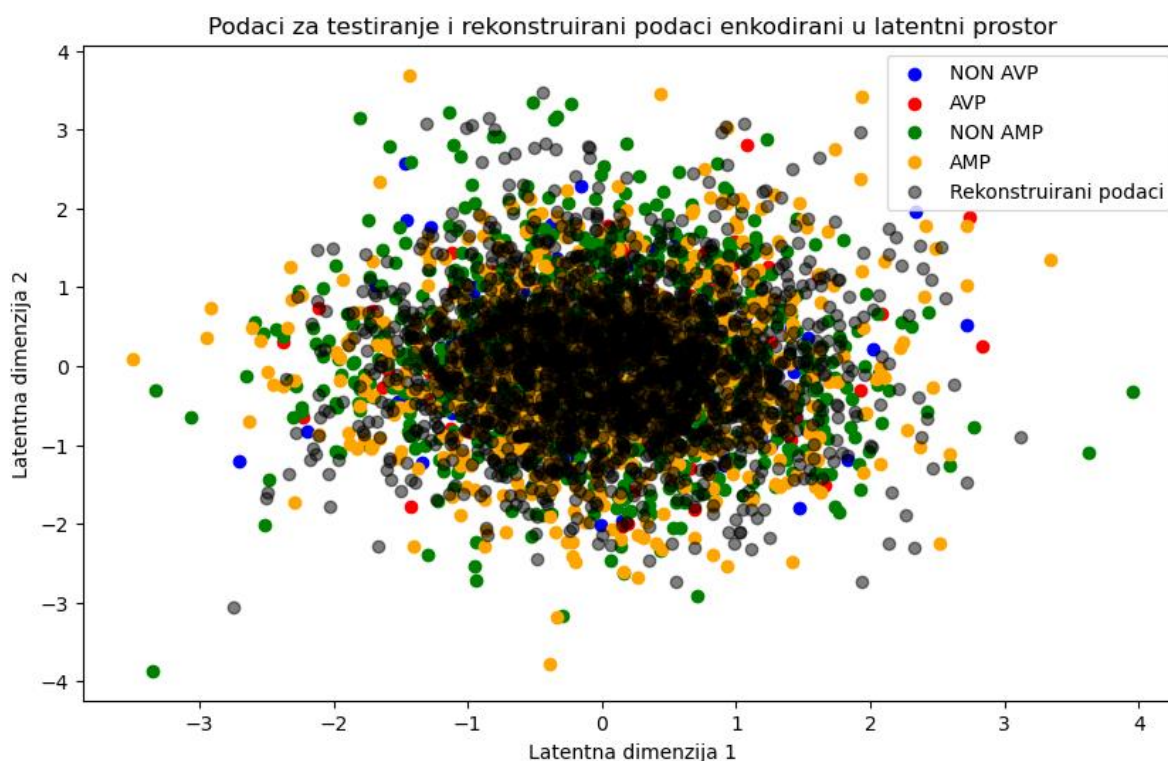


Slika 5.4. Podaci za trening enkodirani u latentni prostor ('NON' u legendi označava nedostatak pripadne aktivnosti peptida)

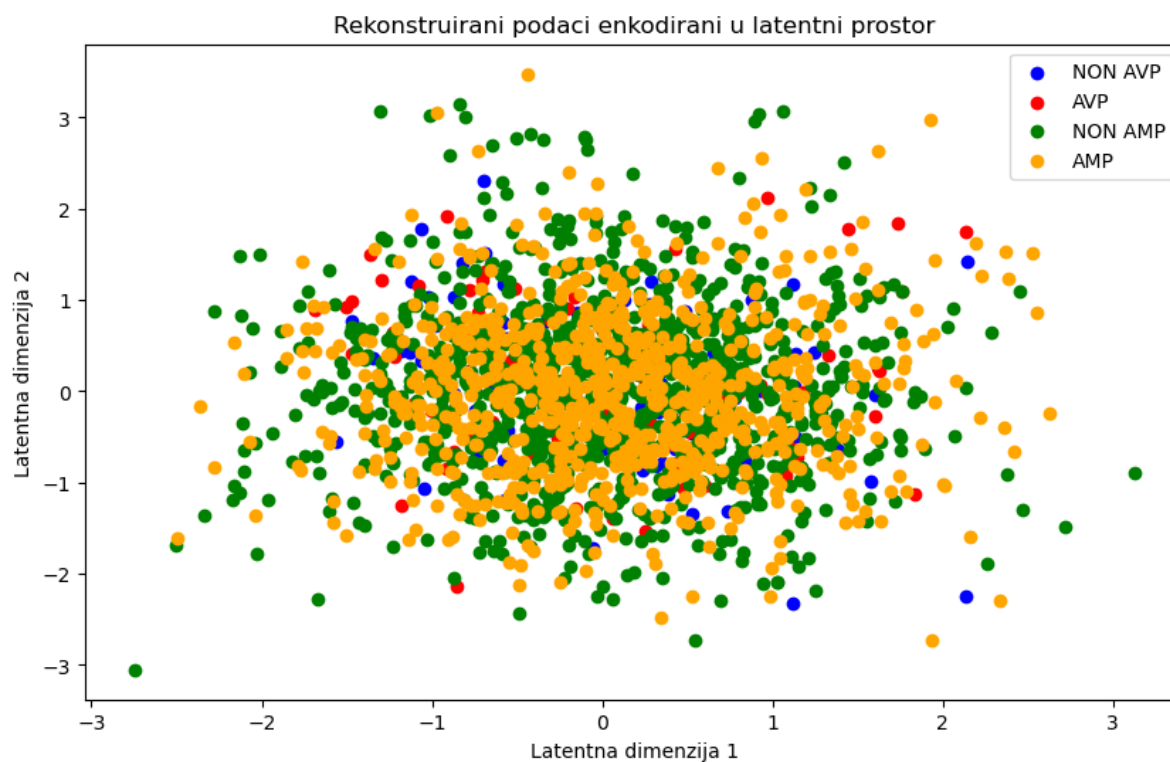


Slika 5.5. Podaci za testiranje enkodirani u latentni prostor ('NON' u legendi označava nedostatak pripadne aktivnosti peptida)

Na slici 5.6. prikazani su podaci za testiranje i rekonstruirani podaci enkodirani u latentni prostor. Ono što vidimo na slici 5.6. jest da kada model dekodira i ponovno enkodira podatke, smješta ih u isti dio latentnog prostora gdje se nalaze enkodirani izvorni podaci. Prema tome, možemo pretpostaviti da je model uspješan u rekonstrukciji. Radi bolje preglednosti, rekonstruirane točke prikazane su različitim bojama prema vrsti peptida na slici 5.7. koju možemo usporediti sa slikom 5.5. da bismo usporedili izvorne i rekonstruirane podatke. Međutim, to nije dovoljno da bismo mogli sa sigurnošću odrediti koliko su dobro rekonstruirane konkretne točke jednostavno zbog prevelikog broja točaka koje se preklapaju i zbog toga što ne možemo razlučiti konkretne točke unutar neke boje odnosno vrste.

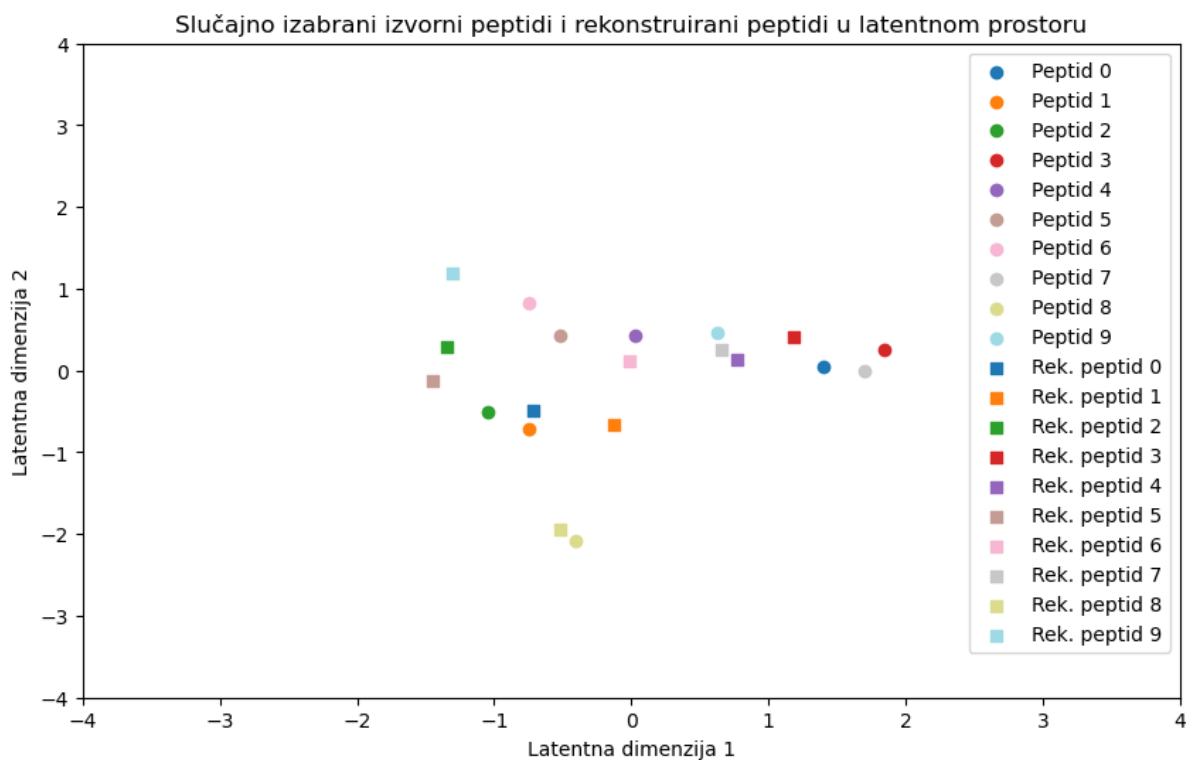


Slika 5.6. Podaci za testiranje i rekonstruirani podaci enkodirani u latentni prostor ('NON' u legendi označava nedostatak pripadne aktivnosti peptida)



*Slika 5.7. Rekonstruirani podaci za testiranje enkodirani u latentni prostor ('NON' u legendi označava nedostatak pripadne aktivnosti peptida)*

Kako bismo lakše vizualno procijenili koliko dobro model rekonstruira izvorne podatke, stvorili smo vizualizaciju prikazanu slikom 5.8. Ovaj put smo nasumično odabrali 10 peptida iz skupa podataka za testiranje koje smo prikazali enkodirane u latentni prostor svakog svojom bojom bez obzira na antimikrobnu i antiviralnu aktivnost. Te iste peptide smo zatim dekodirali i ponovno enkodirali u latentni prostor kako bi ih mogli prikazati. Peptidi prikazani krugom označavaju izvorne peptide, a peptidi prikazani kvadratom označavaju pripadajuće rekonstruirane peptide. Kako se radi o jako malom broju peptida i kako je svaki od njih prikazan svojom bojom, možemo točno vidjeti koliko je svaki rekonstruirani peptid udaljen od pripadajućeg izvornog peptida. Vidimo da model rekonstruira podatke na korektan način, ali uz prisutnu grešku.



Slika 5.8. Slučajno izabrani izvorni peptidi i rekonstruirani peptidi u latentnom prostoru

Kako bismo vizualne rezultate rekonstrukcije sa slike 5.8. i kvantitativno defnirali, udaljenosti između svakog pojedinog izvornog peptida i njemu pripadajućeg rekonstruiranog peptida izračunate su i prikazane tablicom 5.3. U prvom retku tablice nalaze se vrijednosti koje predstavljaju relativnu udaljenost između svake pojedine izvorne točke i njoj pripadajuće rekonstruirane točke, a u drugom retku nalazi se prosječna relativna udaljenost između izvornih i rekonstruiranih točaka. Vidimo da se izmjerene udaljenosti razlikuju, ali uglavnom su niske s obzirom na veličinu latentnog prostora, kao i prosječna udaljenost između prikazanih točaka te i prema ovome parametru možemo zaključiti da je model uspješan u dekodiranju točaka iz latentnog prostora.



Tablica 5.3. Udaljenosti između izvornih i rekonstruiranih točaka

Udaljenost između pojedinih točaka	2,18	0,62	0,85	0,68	0,79	1,08	1,02	1,07	0,17	2,06
Prosječna udaljenost	<b>1,06</b>									

### 5.3. Korelacijska analiza

Jedan od ciljeva ovog rada je provjeriti postoji li korelacija između sličnosti peptida prema njihovim sekvencama i sličnosti peptida prema njihovim fizikalno-kemijskim značajkama. Da bismo to ustanovili moramo provesti korelacijsku analizu te dvije varijable.

Prije svega moramo izračunati sličnost među peptidima na osnovu njihove sekvence, odnosno dobiti kvantitativni podatak njihove sličnosti. To radimo pomoću knjižnice Biopython koja sadrži funkciju za računanje sličnosti peptida i proteina prema njihovim sekvencama tako što na njima provodi *global sequence alignment* odnosno globalno poravnanje sekvenci. Naša metodologija uključuje petlje koje prolaze kroz dijelove skupova podataka koji sadrže sekvence peptida iz skupova podataka za trening i testiranje te u svakom koraku petlje promatramo dvije različite sekvence peptida. Pettle prolaze kroz 50 peptida, od kojih svakog uspoređuju sa svakim od idućih 50 peptida u skupu podataka. Dakle, u svakom koraku uzimamo dvije sekvence peptida, pretvaramo ih u *Seq* objekte s kojima knjižnica Biopython radi, na njima se provodi globalno poravnanje sekvenci, odabire se najbolje poravnanje te vraća njihova sličnost u obliku decimalnog broja u rasponu od 0 do 1 gdje broj bliži nuli označava manju sličnost, a bliži jedinici veću sličnost. Možemo reći da dobivena vrijednost označava postotak sličnosti dvaju peptida. Na taj način dobivamo dvije matrice, jednu s podacima za trening i drugu s podacima za testiranje, koje sadrže izračunate sličnosti među peptidima. Razlog zašto prolazimo samo kroz dijelove skupova podataka je taj što bi istovremeno promatranje cjelovitih skupova podataka rezultiralo jako velikim matricama koje bi sadržale desetke milijuna brojeva i samim time bile računalno previše zahtjevne za izračun, a matrice od 2500 brojeva koje smo mi dobili bit će sasvim dovoljne za korelacijsku analizu.

Sljedeći korak je izračunavanje sličnosti među peptidima na osnovu njihovih fizikalno-kemijskih značajki i za ovo će nam koristiti naš model VAE. Pretpostavljamo da bi peptidi koji

imaju slične značajke trebali biti enkodirani vrlo blizu u latentnom prostoru. Dakle, kao mjeru sličnosti između dva peptida prema njihovim značajkama uzet ćemo njihovu udaljenost u latentnom prostoru. Ovaj dio je vrlo sličan izračunavanju sličnosti prema sekvencama. Pomoću dvije petlje raspona 50 prolazimo kroz enkodirane podatke za trening i testiranje i računamo euklidsku udaljenost  $d$  između svake dvije točke, odnosno peptida, prema izrazu (5.4.):

$$d_{(p,q)} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} \quad (5.4.)$$

gdje je:

$p$  prva promatrana točka,

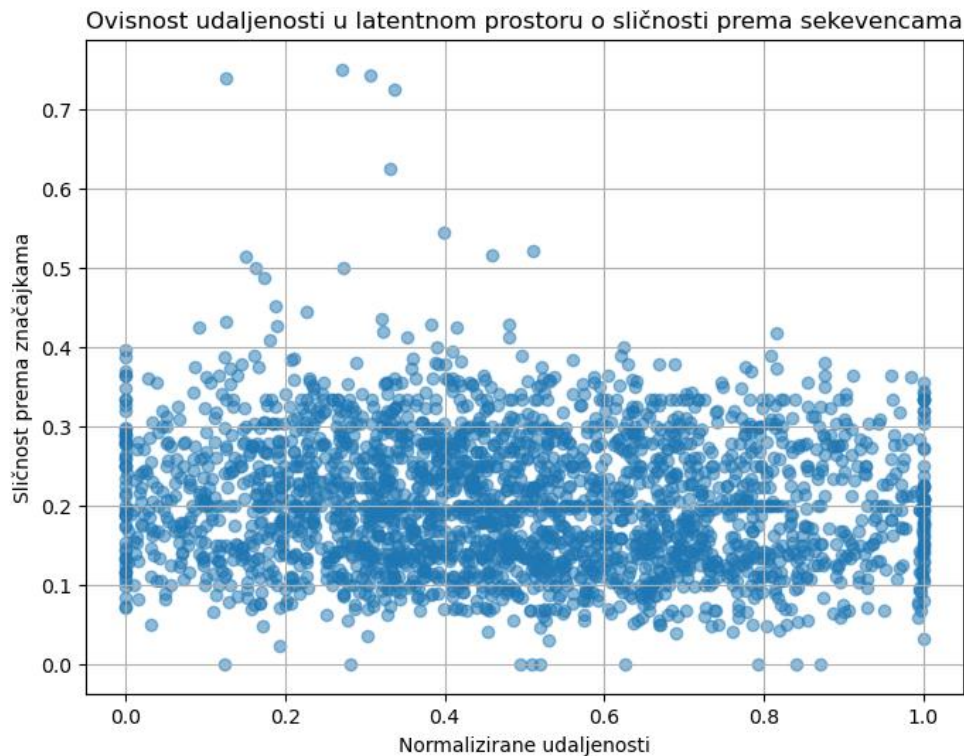
$q$  druga promatrana točka.

Bitno je napomenuti da ovdje uključujemo iste one peptide za koje smo računali sličnost prema sekvencama kako bismo ih mogli pravilno usporediti. Dobivamo dvije liste s udaljenostima, jednu s podacima za trening i drugu s podacima za testiranje, unutar kojih normaliziramo vrijednosti u raspon od 0 do 1 kako bismo ih mogli uspoređivati sa sličnostima prema sekvenci koje su u istom rasponu. Konačni rezultat su ponovno matrice koje sadrže 2500 decimalnih brojeva koji ovaj put predstavljaju sličnost među peptidima na osnovi njihovih značajki odnosno udaljenosti u latentnom prostoru VAE.

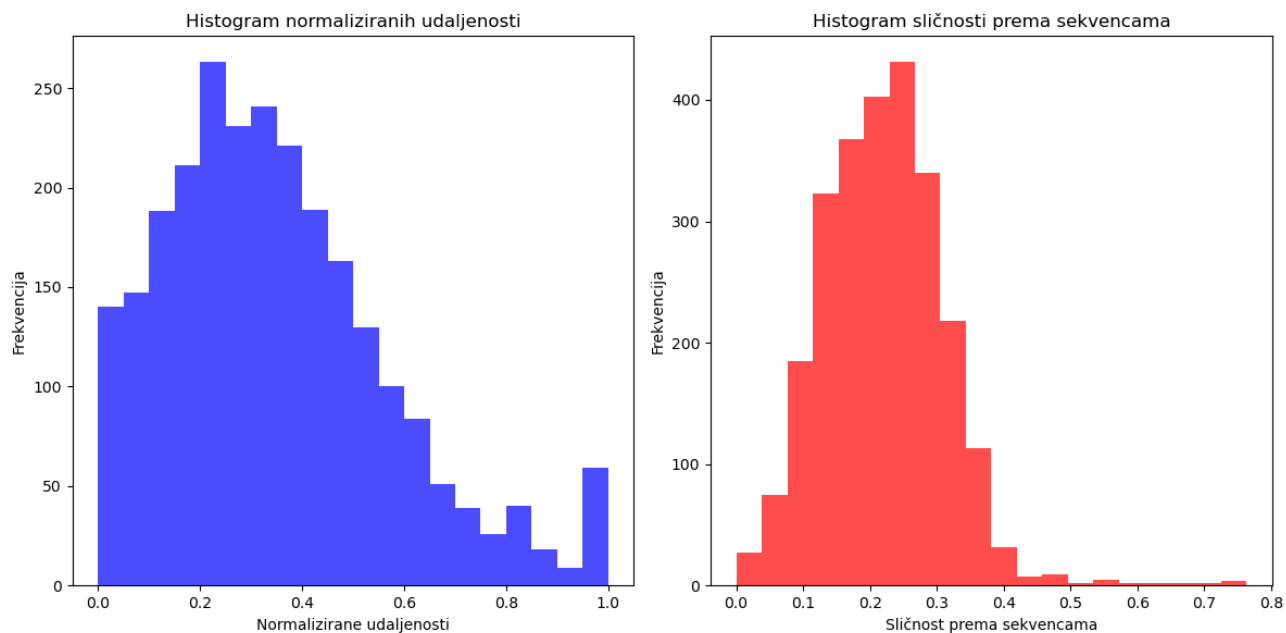
Nakon što smo pripremili matrice koje sadrže sve potrebne vrijednosti možemo provesti korelacijsku analizu. U knjižnici Scipy već postoje gotove funkcije koje računaju koeficijente korelacije i njihove p-vrijednosti. Za korelacijsku analizu naših podataka odabran je Pearsonov test koeficijenta korelacije, koji mjeri linearnu vezu između dviju varijabli, iz razloga što varijable koje uspoređujemo ispunjavaju sve uvjete koji su potrebni da bi se Pearsonov test proveo:

- **kontinuiranost** – obje varijable mogu poprimiti bilo koju realnu vrijednost u rasponu od 0 do 1 te ih smatramo kontinuiranima u istom rasponu
- **linearna veza** – među varijablama mora postojati približna linearna veza, odnosno većina točaka bi se trebala nalaziti oko jednog zamišljenog pravca; to je provjereno jednostavnom vizualizacijom ovisnosti jedne varijable o drugoj, na slici 5.9. koja prikazuje da se uistinu većina točaka nalazi oko jednog pravca, ali se već može naslutiti nedostatak pozitivne ili negativne linearne veze jer je taj zamišljeni pravac gotovo paralelan s osi  $x$

- **normalna distribucija** – podaci moraju biti približno normalno distribuirani; to je potvrđeno vizualizacijom histograma distribucije podataka, prikazano slikom 5.10. gdje vidimo da distribucije oba skupa podataka imaju približni oblik 'zvona' koji upućuje na normalnu distribuciju
- **nedostatak outliersa** – ne smiju postojati točke koje značajno odskakuju od linearne veze; u našem slučaju postoji nekoliko takvih točaka, vidljivo i na slici 5.9., ali s obzirom na to da su one malobrojne u usporedbi s količinom ostatka podataka možemo ih zanemariti



*Slika 5.9. Ovisnost udaljenosti peptida u latentnom prostoru o sličnosti prema značajkama*



*Slika 5.10. Vizualni prikaz distribucija podataka udaljenosti u latentnom prostoru i sličnosti prema sekvencama*

Pearsonov koeficijent korelacije može poprimiti vrijednosti između  $-1$  i  $1$ . Pozitivan koeficijent korelacije blizu broja  $1$  ukazuje na jaku linearnu vezu između dvije varijable gdje bi se povećanjem jedne varijable povećala i druga. Negativan koeficijent korelacije blizu  $-1$  ukazuje na jaku negativnu linearnu vezu između dvije varijable gdje bi se povećanjem jedne varijable druga varijabla smanjila. Koeficijent korelacije koji je blizu  $0$  ukazuje na to da veza između dvije varijable ne postoji. Pripadne  $p$ -vrijednosti koeficijenta korelacije mogu poprimiti vrijednosti između  $0$  i  $1$ , a one predstavljaju vjerojatnost da je promatrani koeficijent korelacije dobiven slučajno, odnosno da ne postoji snažan dokaz da su dvije varijable zaista u korelaciji. Niska  $p$ -vrijednost, obično manja od  $0,05$ , potvrđuje da je dobiveni koeficijent korelacije statistički značajan i da se smije tumačiti kao takav.  $P$ -vrijednost viša od referentne ukazuje na to da je dobivena korelacija možda slučajna i da ne postoji snažan dokaz koji bi potvrdio vezu između dvije varijable.

Da bismo dodatno osigurali vjerodostojnost dobivenog koeficijenta korelacije, opisani postupak računanja sličnosti prema sekvencama i sličnosti prema fizikalno-kemijskim značajkama po dijelovima skupova podataka ponovljen je  $10$  puta, gdje smo u svakom pojedinom postupku promatrali različite peptide, izračunali koeficijent korelacije i pripadnu  $p$ -vrijednost za taj dio podataka i na kraju usrednjili dobivene rezultate. U tablici 5.4. prikazani su usrednjeni koeficijenti

korelacije i p-vrijednosti za naše podatke. U tablici 5.5. prikazana je statistika mjerenja koeficijenata korelacije kroz 10 različitih ispitivanja. P-vrijednost koju uzimamo kao granicu značajnosti iznosi uobičajenih 0,05. S obzirom na to da je dobivena p-vrijednost  $< 0,05$ , te prema tablici 5.4. iznosi  $3,998e-08$  za skup podataka za trening i  $1,089e-11$  za skup podataka za testiranje, smijemo tumačiti dobiveni koeficijent korelacije kao statistički značajan. Dobivene vrijednosti koeficijenta korelacije ukazuju na to da bi mogla postojati i blaga negativna veza između ovih dviju varijabli. Međutim, dobivene vrijednosti koeficijenata korelacije osim što su negativne su relativno blizu nuli, što upućuje na to da stvarna linearna veza između ove dvije varijable ne postoji, odnosno da sličnost peptida prema njihovim sekvencama i sličnost prema njihovim značajkama nisu povezane.

Iz toga slijedi da nam prikazi peptida pomoću sekvenci i pomoću fizikalno-kemijskih značajki nose različite informacije, odnosno da nam analiza svakog od tih prikaza nadopunjuje ukupno znanje o peptidima koje proučavamo. Dakle, pretpostavljamo da dva peptida koji imaju vrlo slične sekvence mogu imati potpuno različite fizikalno-kemijske značajke. Također, s ovim saznanjem možemo otkriti koje molekule su slične u oba prikaza i tako brže doći do otkrića peptida sa željenim svojstvima ili lakše uočiti nepravilnosti gdje recimo malom promjenom u sekvenci u velikoj mjeri mijenjamo fizikalno-kemijske značajke. Pritom treba uzeti u obzir vrlo mali broj dimenzija latentnog prostora zbog kojeg je zasigurno izgubljen dio informacija te bi ponavljanjem pokusa na VAE s većim brojem dimenzija mogli dobiti drugačije rezultate.

*Tablica 5.4. Rezultati korelacijske analize*

	Koeficijent korelacije	p-vrijednost
Skup podataka za trening	-0,17	$3,998e-08$
Skup podataka za testiranje	-0,21	$1,089e-11$

Tablica 5.5. Statistika mjerenja koeficijenta korelacije

	Koeficijent korelacije	p-vrijednost
Skup podataka za trening	-0,26	2,48e-41
	-0,11	1,058e-07
	-0,14	5,755e-13
	-0,24	8,573e-34
	-0,23	1,46e-31
	-0,12	1,046e-09
	-0,10	2,93e-07
	-0,17	3,89e-18
	-0,19	2,198e-21
	-0,20	1,186e-24
Skup podataka za testiranje	-0,28	6,314e-46
	-0,25	2,857e-36
	-0,13	1,037e-10
	-0,25	1,558e-38
	-0,27	3,517e-45
	-0,14	5,225e-12
	-0,19	5,291e-23
	-0,15	5,227e-15
	-0,18	5,219e-19
	-0,23	3,611e-32

## 6. ZAKLJUČAK

S obzirom na obećavajuće rezultate primjene peptida u medicini za liječenje raznih bolesti, postalo je vrlo važno fokusirati se na otkrivanje novih i djelotvornih peptida. Razne metode strojnog učenja pokazale su se vrlo učinkovitim za otkrivanje novih peptida sa željenim svojstvima za razliku od otkrivanja novih peptida u laboratorijima što je vrlo skupo, sporo i neefikasno.

U ovom radu proveli smo istraživanje o primjeni varijacijskih autoenkodera za analizu sličnosti peptida s antimikrobnim i antiviralnim svojstvima kako bismo pokušali bolje shvatiti vezu između njihovih sekvenci i fizikalno-kemijskih značajki. Enkodirali smo peptide opisane njihovim fizikalno-kemijskim značajkama u latentni prostor VAE kako bismo ih mogli lakše proučavati i vizualizirati, uz zadržavanje što više informacija o podacima. To je uključivalo prikupljanje i organizaciju podataka, obradu podataka, kodiranje samog autoenkodera, treniranje modela, optimizaciju hiperparametara, osmišljanje metrika koje će mjeriti točnost modela, vizualizacije latentnog prostora, izračune sličnosti na temelju sekvenci peptida te sličnosti na temelju njihovih fizikalno-kemijskih značajki, odnosno njihove udaljenosti u latentnom prostoru, i na kraju provođenje korelacijske analize tih dviju varijabli.

Glavni cilj rada bila je analiza sličnosti i razlika različitih skupina peptida u latentnom prostoru kako bismo istražili imaju li pojedine skupine ujedno i različite fizikalno-kemijske značajke. Uzimajući u obzir vrijednosti predstavljenih metrika, koje ukazuju na to da naš model dekodira podatke uz grešku od 8,69% i da je sličnost dekodiranih podataka s izvornim podacima 79,17% te da dekodirane podatke ponovno enkodira uz grešku od 10,3%, možemo reći da naš model VAE postiže vrlo dobre rezultate pri enkodiranju peptida u latentni prostor i dekodiranju peptida. Uz to zadržava dobra svojstva latentnog prostora koja omogućavaju kvalitetnu vizualizaciju kakva je i prikazana u rezultatima rada. Međutim, kao krajnji rezultat nije dobiveno željeno vizualno razdvajanje peptida u latentnom prostoru te možemo zaključiti da peptidi s različitim antimikrobnim i antiviralnim svojstvima mogu imati slične fizikalno-kemijske značajke. Nedostatak razdvajanja peptida prema skupinama moguć je i zbog male dimenzionalnosti latentnog prostora od samo dvije dimenzije, gdje je moguć značajan gubitak informacija. To otvara mogućnost da se u budućnosti napravi slično istraživanje s VAE većih dimenzija i ispita hoće li tada peptidi različitih aktivnosti biti odvojeni u latentnom prostoru.

Ono što je ipak ostvareno jest VAE dizajniran da nauči smislene prikaze peptida u dvodimenzionalnom latentnom prostoru, koji je uspješan u rekonstrukciji peptida iz latentnog prostora i koji se pokazao kao moćan pristup istraživanju strukturnih veza među peptidima te može biti korišten za razna istraživanja u bioinformatici, otkrivanju lijekova i molekularnoj biologiji. Također, ono na što su ukazali rezultati korelacijske analize, gdje su dobiveni iznosi Pearsonovog koeficijenta korelacije od  $-0,17$  za skup podataka za trening i  $-0,21$  za skup podataka za testiranje, jest da sličnost peptida prema njihovoj sekvenci i sličnost prema fizikalno-kemijskim značajkama nisu povezane, no to ne možemo tvrditi sa potpunom sigurnošću s obzirom na to da dobiveni iznosi koeficijenta korelacije već uz malu promjenu mogu ukazivati na blagu negativnu linearnu povezanost i trebalo bi provesti dodatna istraživanja kako bi potvrdili te rezultate. Ovim radom i istraživanjem VAE u kontekstu analize peptida pokazujemo veliki potencijal koji imaju VAE u daljnjim pokušajima da unaprijedimo naše razumijevanje ovih kompleksnih molekula.



## LITERATURA

- [1] Kuan Y. Chang, Je-Ruei Yang: Analysis and prediction of highly effective antiviral peptides based on random forests, s Interneta, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3734225/>, 7. kolovoza 2023.
- [2] Vilas Boas LCP, Campos ML, Berlanda RLA, de Carvalho Neves N, Franco OL: Antiviral peptides as promising therapeutic drugs, s Interneta, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7079787/>, 8. kolovoza 2023.
- [3] Peptides Guide, s Interneta, <https://peptidesguide.com>, 18. srpnja 2023.
- [4] Peptide Sciences, s Interneta, <https://www.peptidesciences.com/information/peptides-vs-proteins/>, 18. srpnja 2023.
- [5] Courtney Simons: Structure of amino acids and proteins, s Interneta, <https://cwsimons.com/structure-of-amino-acids-and-proteins/>, 5. srpnja 2020.
- [6] Hitchhiking with Nature: Snake Venom Peptides to Fight Cancer and Superbugs - Scientific Figure on ResearchGate, s Interneta, [https://www.researchgate.net/figure/Principal-functions-of-cationic-peptides-ACP-anticancer-peptides-AMP-antimicrobial\\_fig1\\_340674862](https://www.researchgate.net/figure/Principal-functions-of-cationic-peptides-ACP-anticancer-peptides-AMP-antimicrobial_fig1_340674862), 7. kolovoza 2023.
- [7] Sara Wilcox: Cationic peptides: A new hope, s Interneta, <https://www.scq.ubc.ca/cationic-peptides-a-new-hope/>, 8. kolovoza 2023.
- [8] Protein Esterification: Design, Antibacterial and Safety Assessment - Scientific Figure on ResearchGate, s Interneta, [https://www.researchgate.net/figure/Fig-1-Model-for-the-mechanism-of-action-of-cationic-antimicrobial-peptides\\_fig1\\_282186504](https://www.researchgate.net/figure/Fig-1-Model-for-the-mechanism-of-action-of-cationic-antimicrobial-peptides_fig1_282186504), 7. kolovoza 2023.
- [9] Shreya Chaudhary: A High-level guide to autoencoders, s Interneta, <https://towardsdatascience.com/a-high-level-guide-to-autoencoders-b103ccd45924>, 8. kolovoza 2023.
- [10] Daniel Nelson: What is an autoencoder?, s Interneta, <https://www.unite.ai/what-is-an-autoencoder/>, 9. kolovoza 2023.
- [11] Francois Chollet: Building autoencoders in Keras, s Interneta, <https://blog.keras.io/building-autoencoders-in-keras.html>, 9. kolovoza 2023.
- [12] Joseph Rocca: Understanding variational autoencoders, s Interneta, <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>, 10. kolovoza 2023.
- [13] Christian Vesloot: What is a variational autoencoder (VAE)?, s Interneta, <https://github.com/christianversloot/machine-learning-articles/blob/main/what-is-a-variational-autoencoder-vae.md>, 10. kolovoza 2023.

- [14] Coursera: What is Python used for? A beginner's guide, s Interneta, <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>, 10. kolovoza 2023.
- [15] Jupyter Notebook Documentation, s Interneta, [https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what\\_is\\_jupyter.html](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html), 11. kolovoza 2023.
- [16] Bokeh documentation: First steps 7: Displaying and exporting, s Interneta, [https://docs.bokeh.org/en/latest/docs/first\\_steps/first\\_steps\\_7.html](https://docs.bokeh.org/en/latest/docs/first_steps/first_steps_7.html), 15. kolovoza 2023.
- [17] Wikipedia: Tensorflow, s Interneta, <https://en.wikipedia.org/wiki/TensorFlow>, 11. kolovoza 2023.
- [18] Tensorflow documentation, s Interneta, <https://www.tensorflow.org/guide/keras>, 11. kolovoza 2023.
- [19] Peptides.py documentation, s Interneta, <https://peptides.readthedocs.io/en/stable/>, 12. kolovoza 2023.
- [20] Francois Chollet: Variational autoencoder, s Interneta, <https://keras.io/examples/generative/vae/>, 14. kolovoza 2023.
- [21] Christian Versloot: How to create a variational autoencoder with Keras, s Interneta, <https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-create-a-variational-autoencoder-with-keras.md>, 14. kolovoza 2023.
- [22] Computer Science Wiki: Max-pooling/Pooling, s Interneta, <https://computersciencewiki.org/index.php/Max-pooling / Pooling>, 14. kolovoza 2023.

## POPIS KRATICA

AMP – antimikrobni peptidi

AVP – antiviralni peptidi

VAE – varijacijski autoenkoder

KL divergencija – Kullback-Leibler divergencija

NSURLP – normalizirana srednja udaljenost rekonstrukcije u latentnom prostoru

PKK – Pearsonov koeficijent korelacije

RMSE – *Root Mean Square Error* (kvadratni korijen srednje kvadratne pogreške)

## Sažetak

Unatoč brzom napretku medicine i znanosti, mnoge bolesti i dalje predstavljaju veliku prijetnju ljudskom zdravlju. U tom kontekstu, peptidi su se istaknuli kao obećavajuće novo rješenje pri liječenju. Razne metode strojnog učenja pokazale su se vrlo učinkovitima za otkrivanje novih peptida sa željenim svojstvima. U ovom radu proveli smo istraživanje o primjeni varijacijskih autoenkodera za analizu sličnosti peptida s antimikrobnim i antiviralnim svojstvima kako bismo pokušali bolje shvatiti vezu između njihovih sekvenci i fizikalno-kemijskih značajki. Kao krajnji rezultat dobiven je VAE koji uspješno uči smislene prikaze peptida u dvodimenzionalnom latentnom prostoru, koji je uspješan u rekonstrukciji peptida iz latentnog prostora i koji se pokazao kao moćan pristup istraživanju strukturnih veza među peptidima. Potreban je dodatan trud i ulaganja u slična istraživanja jer imaju velik potencijal za ostvarenje napretka u medicini i poboljšanje kvalitete ljudskog života.

Ključne riječi: antimikrobni peptidi, antiviralni peptidi, autoenkoderi, varijacijski autoenkoderi, latentni prostor, korelacijska analiza

## Abstract

Despite the quick progress in medicine and science, there are still many diseases that pose a threat to human health. In this context, peptides stood out as a new promising treatment solution. Various machine learning methods have proven very effective in discovering new peptides with desired properties. In this paper, we conducted research on the application of variational autoencoders for similarity analysis of peptides with antimicrobial and antiviral properties so we could better understand the connection between their sequences and physicochemical properties. As the end result we got a variational autoencoder that successfully learns meaningful representations of peptides in a two-dimensional latent space, that is successful in reconstructing peptides from latent space and has proven to be a powerful approach to researching structural bonds between peptides. Additional effort and investment in similar research is needed because of its great potential for achieving progress in medicine and improving the quality of human life.

Keywords: antimicrobial peptides, antiviral peptides, autoencoders, variational autoencoders, latent space, correlation analysis

# PRILOZI

## Programski kod

```
# In[1]:

#IMPORT KNJIŽNICA I ARHITEKTURA MODELA

#Import knjižnica
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.callbacks import ReduceLRonPlateau, EarlyStopping, Callback
import matplotlib.pyplot as plt

#Klasa za uzorkovanje iz latentnog prostora
class Sampling(layers.Layer):

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon

latent_dim = 2

#Arhitektura enkodera
encoder_inputs = keras.Input(shape=(88, 1))
x = layers.Conv1D(8, 3, activation='relu', padding='same',
dilation_rate=2)(encoder_inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(16, 3, activation='relu', padding='same', dilation_rate=2)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Flatten()(x)
encoder = layers.Dense(128, activation='relu')(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(encoder)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(encoder)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()

#Arhitektura dekodera
```

```

latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(128, activation='relu')(latent_inputs)
x = layers.Dense(88 * 8)(x)
x = layers.Reshape((88, 8))(x)
x = layers.Conv1D(16, 3, activation='relu', padding='same')(x)
x = layers.UpSampling1D(2)(x)
x = layers.Conv1D(8, 3, activation='relu', padding='same')(x)
x = layers.UpSampling1D(2)(x)
decoded = layers.Conv1D(1, 3, activation='sigmoid', padding='same')(x)
decoded = layers.Cropping1D(cropping=(0, 264))(decoded)
decoder = keras.Model(latent_inputs, decoded, name="decoder")
decoder.summary()

#Klasa VAE
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker =
keras.metrics.Mean(name="reconstruction_loss")
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)

            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum((data - reconstruction) ** 2, axis=(1, 2))
            )

            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) -
tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))

            total_loss = reconstruction_loss + 0.27 * kl_loss

```

```

grads = tape.gradient(total_loss, self.trainable_weights)
self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
self.total_loss_tracker.update_state(total_loss)
self.reconstruction_loss_tracker.update_state(reconstruction_loss)
self.kl_loss_tracker.update_state(kl_loss)
return {
    "loss": self.total_loss_tracker.result(),
    "reconstruction_loss": self.reconstruction_loss_tracker.result(),
    "kl_loss": self.kl_loss_tracker.result(),
}

```

```

def test_step(self, data):
    x = data

    z_mean, z_log_var, z = self.encoder(x, training=False)
    reconstructed_x = self.decoder(z, training=False)

    reconstruction_loss = tf.reduce_mean(
        tf.reduce_sum((x - reconstructed_x) ** 2, axis=(1, 2))
    )

    kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var))
    kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))

    total_loss = reconstruction_loss + 0.27 * kl_loss

    results = {m.name: m.result() for m in self.metrics}
    results["reconstruction_loss"] = reconstruction_loss
    results["kl_loss"] = kl_loss
    results["total_loss"] = total_loss
    return results

```

```
# In[2]:
```

```
#OBRADA PODATAKA
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
#Učitavanje podataka i podjela na skupove za trening i testiranje
```

```
features = pd.read_csv('features.csv')
```

```
labels = pd.read_csv('labels.csv')
```

```
(x_train, x_test, y_train, y_test) = train_test_split(
```

```

features, labels, test_size=0.2,
random_state=42)

#Odvajanje sekvenci peptida od ostatka podataka
x_train_seq = x_train.values[:, 0]
x_test_seq = x_test.values[:, 0]

#Odvajanje izračunatih značajki peptida
x_train_features = x_train.values[:, 1:]
x_test_features = x_test.values[:, 1:]

#Normalizacija podataka
scaler = MinMaxScaler()

x_train = scaler.fit_transform(x_train_features)
x_test = scaler.transform(x_test_features)

#Prilagođavanje dimenzionalnosti podataka
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
y_train = np.expand_dims(y_train, -1)
y_test = np.expand_dims(y_test, -1)

# In[3]:

#OPTIMIZACIJA HIPERPARAMETARA

#Prostor pretrage
learn_rate = [0.001, 0.005, 0.008, 0.01]
batch_size = [32, 64, 128, 256]

param_grid = dict(learn_rate=learn_rate, batch_size=batch_size)

best_score = None
best_params = {}

#Petlje u kojima se obavlja testiranje svih kombinacija hiperparametara iz prostora
pretrage
for lr in learn_rate:
    for bs in batch_size:
        print("Learning rate: ", lr)
        print("Batch size: ", bs)

        vae = VAE(encoder, decoder)
        vae.compile(optimizer=keras.optimizers.Adam(learning_rate=lr))

```



```

    reduce_lr = ReduceLRonPlateau(monitor='loss', factor=0.5, patience=10,
verbose=1, min_lr=1e-6)
    early_stop = EarlyStopping(monitor='loss', patience=20, verbose=1,
restore_best_weights=True)

vae.fit(x_train, batch_size=bs, epochs=30, validation_split = 0.2,
        callbacks=[reduce_lr, early_stop])

score = vae.evaluate(x_train)
print(score)

if best_score is None or score < best_score:
    best_score = score
    best_params = {'learn_rate': lr, 'batch_size': bs}

#Ispis najboljih hiperparametara
print("Best Hyperparameters:", best_params)
print("Best Score:", best_score)

#Spremanje najboljih hiperparametara u varijable za daljnje korištenje
lr = best_params['learn_rate']
bs = best_params['batch_size']

```

```
# In[4]:
```

```
#DEFINIRANJE POVRATNIH POZIVA ZA PRAĆENJE GUBITAKA I LEARNING RATES TIJEKOM
TRENINGA
```

```
learning_rates = []
```

```
class LearningRateScheduler(Callback):
    def on_epoch_begin(self, epoch, logs=None):
        lr = float(tf.keras.backend.get_value(self.model.optimizer.lr))
        learning_rates.append(lr)
```

```
class LossHistoryCallback(Callback):
    def on_train_begin(self, logs=None):
        self.losses = []

    def on_epoch_end(self, epoch, logs=None):
        self.losses.append(logs['loss'])
        self.losses.append(logs['reconstruction_loss'])
        self.losses.append(logs['kl_loss'])
```

```

# In[5]:

#TRENING

#Instanciranje i kompajliranje modela
vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam(learning_rate = lr))

#Instanciranje povratnih poziva
reduce_lr = ReduceLRonPlateau(monitor='loss', factor=0.5, patience=10, verbose=1,
min_lr=1e-6)
early_stop = EarlyStopping(monitor='loss', patience=20, verbose=1,
restore_best_weights=True)
learning_rates = []
lr_scheduler = LearningRateScheduler()
loss_callback = LossHistoryCallback()

#Treniranje modela
history = vae.fit(x_train, batch_size=bs, epochs=100, validation_split = 0.2,
callbacks=[lr_scheduler, loss_callback, reduce_lr, early_stop])

# In[6]:

#VIZUALIZACIJA GUBITAKA I LEARNING RATES TIJEKOM TRENINGA

#Vizualizacija gubitaka
plt.figure(figsize=(8, 4))
plt.plot(history.history['loss'], label='Total Loss')
plt.plot(history.history['reconstruction_loss'], label='Reconstruction Loss')
plt.plot(history.history['kl_loss'], label='KL Loss')
plt.xlabel('Epohe treninga')
plt.ylabel('Gubici')
plt.title('Definirani gubici tijekom treninga')
plt.legend()

plt.figure(figsize=(8, 4))
plt.plot(history.history['loss'], label='Total Loss')
plt.xlabel('Epohe treninga')
plt.ylabel('Gubici')
plt.title('Ukupni gubici tijekom treninga')
plt.legend()

#Vizualizacija learning rates
plt.figure(figsize=(8, 4))

```

```

plt.plot(learning_rates, label='Learning Rate')
plt.xlabel('Epohe treninga')
plt.ylabel('Gubici')
plt.title('Promjene parametra learning_rate tijekom treninga')
plt.legend()

plt.show()

# In[7]:

#Stvaranje enkodiranih i rekonstruiranih podataka za daljnje korištenje
_, _, z_input = encoder.predict(x_train)
reconstructed_z_input = vae.decoder.predict(z_input)
_, _, reconstructed_z_encoded = vae.encoder.predict(reconstructed_z_input)

_, _, z_test_input = encoder.predict(x_test)
reconstructed_z_test_input = vae.decoder.predict(z_test_input)
_, _, reconstructed_z_test_encoded =
vae.encoder.predict(reconstructed_z_test_input)

# In[8]:

#Funkcija za vizualizaciju ulaznih podataka
def plot_input_data(x_data, y_data):
    z_input = x_data
    unique_labels = np.unique(y_data)
    label_colors = ['blue', 'red', 'green', 'orange']
    legend_labels = ['NON AVP', 'AVP', 'NON AMP', 'AMP']

    for label in unique_labels:
        indices = np.where(y_data == label)[0]
        plt.scatter(z_input[indices, 0], z_input[indices, 1],
c=label_colors[label], label=legend_labels[label])

    plt.xlabel("Latentna dimenzija 1")
    plt.ylabel("Latentna dimenzija 2")
    plt.legend()
    plt.show()

#Vizualizacija podataka za trening i testiranje
plt.figure(figsize=(10, 6))
plt.title("Podaci za trening enkodirani u latentni prostor")
plot_input_data(z_input, y_train)

```

```
plt.figure(figsize=(10, 6))
plt.title("Podaci za testiranje enkodirani u latentni prostor")
plot_input_data(z_test_input, y_test)
```

```
# In[9]:
```

```
#Funkcija za vizualizaciju rekonstruiranih podataka preko ulaznih podataka
def plot_input_and_reconstructed_data(x_input_data, x_reconstructed, y_input_data):
    z_input = x_input_data
    recon_data_lsp = x_reconstructed

    unique_labels = np.unique(y_input_data)
    label_colors = ['blue', 'red', 'green', 'orange']
    legend_labels = ['NON AVP', 'AVP', 'NON AMP', 'AMP']

    for label in unique_labels:
        indices = np.where(y_input_data == label)[0]
        plt.scatter(z_input[indices, 0], z_input[indices, 1],
c=label_colors[label], label=legend_labels[label])

    plt.scatter(recon_data_lsp[:, 0], recon_data_lsp[:, 1], c='black', alpha=0.5,
label='Rekonstruirani podaci')

    plt.xlabel("Latentna dimenzija 1")
    plt.ylabel("Latentna dimenzija 2")
    plt.title("Podaci za testiranje i rekonstruirani podaci enkodirani u latentni
prostor")
    plt.legend()
    plt.show()
```

```
plt.figure(figsize=(10, 6))
plot_input_and_reconstructed_data(z_test_input, reconstructed_z_test_encoded,
y_test)
```

```
# In[10]:
```

```
#Funkcija za vizualizaciju rekonstruiranih podataka
def plot_reconstructed_data(x_reconstructed, y_data):
    recon_data_lsp = x_reconstructed

    unique_labels = np.unique(y_data)
    label_colors = ['blue', 'red', 'green', 'orange']
```

```

legend_labels = ['NON AVP', 'AVP', 'NON AMP', 'AMP']

for label in unique_labels:
    indices = np.where(y_data == label)[0]
    plt.scatter(recon_data_lsp[indices, 0], recon_data_lsp[indices, 1],
               c=label_colors[label], label=legend_labels[label])

plt.xlabel("Latentna dimenzija 1")
plt.ylabel("Latentna dimenzija 2")
plt.legend()
plt.show()

#Vizualizacija ulaznih i rekonstruiranih podataka za testiranje
plt.figure(figsize=(10, 6))
plt.title("Podaci za testiranje enkodirani u latentni prostor")
plot_input_data(z_test_input, y_test)

plt.figure(figsize=(10, 6))
plt.title("Rekonstruirani podaci enkodirani u latentni prostor")
plot_reconstructed_data(reconstructed_z_test_encoded, y_test)

# In[11]:

#Vizualizacija 10 nasumično izabranih peptida i njihove rekonstrukcije za usporedbu

#Odabir 10 nasumičnih peptida
selected_indices = np.random.choice(len(x_test), 10, replace=False)

selected_peptides = x_test[selected_indices]

#Enkodiranje i rekonstrukcija
_, _, encoded_selected_peptides = vae.encoder.predict(selected_peptides)
reconstructed_peptides = vae.decoder.predict(encoded_selected_peptides)
_, _, encoded_reconstructed_peptides = vae.encoder.predict(reconstructed_peptides)

#Funkcija za vizualizaciju odabranih podataka
def plot_selected_data(x_data):
    z_input = x_data
    num_peptides = z_input.shape[0]
    unique_colors = plt.cm.tab20(np.linspace(0, 1, num_peptides))

    for i in range(num_peptides):
        plt.scatter(z_input[i, 0], z_input[i, 1], color=unique_colors[i],
label=f'Peptide {i}')

```

```

ax = plt.gca()
ax.set_xlim([-4,4])
ax.set_ylim([-4,4])
plt.xlabel("Latentna dimenzija 1")
plt.ylabel("Latentna dimenzija 2")
plt.legend()
plt.show()

#Funkcija za izračun euklidske udaljenosti
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y)**2))

#Vizualizacija odabranih peptida u izvornom i rekonstruiranom obliku
plt.figure(figsize=(10, 6))
plt.title("Slučajno izabrane izvorne točke u latentnom prostoru")
plot_selected_data(encoded_selected_peptides)

plt.figure(figsize=(10, 6))
plt.title("Rekonstruirane točke u latentnom prostoru")
plot_selected_data(encoded_reconstructed_peptides)

#Izračun udaljenosti između izvornih i rekonstruiranih peptida
distances = []

for i in range(10):
    dist = euclidean_distance(encoded_selected_peptides[i],
    encoded_reconstructed_peptides[i])

    distances.append(dist)

print("Relativna udaljenost između svake pojedine točke i njoj pripadajuće
rekonstruirane točke: \n", distances, "\n")
print("Prosječna relativna udaljenost između izvornih i rekonstruiranih točaka: ",
np.sum(distances)/10)

# In[12]:

#VIZUALIZACIJA 10 NASUMIČNO ODABRANIH PEPTIDA I NJIHOVIH REKONSTRUKCIJA

plt.figure(figsize=(10, 6))
plt.title("Slučajno izabrani izvorni peptidi i rekonstruirani peptidi u latentnom
prostoru")
unique_colors = plt.cm.tab20(np.linspace(0, 1, 10))

for i in range(10):

```

```

plt.scatter(encoded_selected_peptides[i, 0], encoded_selected_peptides[i, 1],
            color=unique_colors[i], label=f'Peptid {i}')

for i in range(10):
    plt.scatter(encoded_reconstructed_peptides[i, 0],
encoded_reconstructed_peptides[i, 1],
                marker='s', color=unique_colors[i], label=f'Rek. peptid {i}')

ax = plt.gca()
ax.set_xlim([-4, 4])
ax.set_ylim([-4, 4])
plt.xlabel("Latentna dimenzija 1")
plt.ylabel("Latentna dimenzija 2")
plt.legend()
plt.show()

# In[13]:

#NORMALIZIRANA UDALJENOST REKONSTRUKCIJE U LATENTNOM PROSTORU

min_values = np.min(z_test_input, axis=0)
max_values = np.max(z_test_input, axis=0)

latent_space_size = np.linalg.norm(max_values - min_values)

print(latent_space_size)

distances_x_test = []

for i in range(len(x_test)):
    dist = euclidean_distance(z_test_input[i], reconstructed_z_test_encoded[i])

    distances_x_test.append(dist)

print("Normalizirana prosječna udaljenost između izvornih i rekonstruiranih točaka:
",
      np.sum(distances_x_test)/latent_space_size/len(x_test))

# In[14]:

#PEARSONOV KOEFICIJENT KORELACIJE

from scipy.stats import spearmanr, kendalltau, pearsonr

```

```
print(pearsonr(x_test.flatten(), reconstructed_z_test_input.flatten()))
```

```
# In[15]:
```

```
#RMSE
```

```
mse = np.mean(np.square(x_test - reconstructed_z_test_input))  
rmse = np.sqrt(mse)
```

```
print(f"RMSE: {rmse}")
```

```
# In[16]:
```

```
#IZRAČUN SLIČNOSTI PEPTIDA PREMA SEKVENCI
```

```
from Bio import pairwise2
```

```
from Bio.Seq import Seq
```

```
def calculate_sequence_similarity(seq1, seq2):
```

```
    seq1 = Seq(seq1)
```

```
    seq2 = Seq(seq2)
```

```
    alignments = pairwise2.align.globalxx(seq1, seq2)
```

```
    best_alignment = alignments[0]
```

```
    seq1_aligned, seq2_aligned, score, begin, end = best_alignment
```

```
    similarity = (score / max(len(seq1), len(seq2)))
```

```
    return similarity
```

```
similarities_train = []
```

```
similarities_test = []
```

```
for i in range(50):
```

```
    row_similarities_train = []
```

```
    row_similarities_test = []
```

```
    for j in range(50,101):
```

```
        sim_train = calculate_sequence_similarity(x_train_seq[i], x_train_seq[j])
```

```
        sim_test = calculate_sequence_similarity(x_test_seq[i], x_test_seq[j])
```

```
        row_similarities_train.append(sim_train)
```

```
        row_similarities_test.append(sim_test)
```



```

        similarities_train.append(row_similarities_train)
        similarities_test.append(row_similarities_test)

similarities_train = np.array(similarities_train)
similarities_test = np.array(similarities_test)

print(similarities_train)
print(similarities_test)

# In[17]:

#IZRAČUN SLIČNOSTI PEPTIDA PREMA UDALJENOSTI U LATENTNOM PROSTORU
distances_train = []
distances_test = []

for i in range(50):
    row_distances_train = []
    row_distances_test = []
    for j in range(50,101):
        dist_train = euclidean_distance(z_input[i], z_input[j])
        dist_test = euclidean_distance(z_test_input[i], z_test_input[j])
        row_distances_train.append(dist_train)
        row_distances_test.append(dist_test)

    distances_train.append(row_distances_train)
    distances_test.append(row_distances_test)

distances_train = np.array(distances_train)
distances_test = np.array(distances_test)

normalized_distances_train = scaler.fit_transform(distances_train)
normalized_distances_test = scaler.fit_transform(distances_test)

print(normalized_distances_train)
print(normalized_distances_test)

```

```
# In[18]:
```

```
#VIZUALIZACIJA OVISNOSTI UDALJENOSTI U LATENTNOM PROSTORU O SLIČNOSTI PREMA
SEKVENCAMA
```

```
flat_normalized_distances_train = normalized_distances_train.flatten()
```

```

flat_similarities_train = similarities_train.flatten()

plt.figure(figsize=(8, 6))
plt.scatter(flat_normalized_distances_train, flat_similarities_train, alpha=0.5)
plt.title("Ovisnost udaljenosti u latentnom prostoru o sličnosti prema
sekevcama")
plt.xlabel("Normalizirane udaljenosti")
plt.ylabel("Sličnost prema značajkama")
plt.grid(True)
plt.show()

```

```
# In[19]:
```

```
#VIZUALIZACIJA DISTRIBUCIJE UDALJENOSTI U LATENTNOM PROSTORU O SLIČNOSTI PREMA
ZNAČAJKAMA
```

```

plt.figure(figsize=(12, 6))

#Histogram udaljenosti
plt.subplot(1, 2, 1)
plt.hist(flat_normalized_distances_train, bins=20, color='blue', alpha=0.7)
plt.title("Histogram normaliziranih udaljenosti")
plt.xlabel("Normalizirane udaljenosti")
plt.ylabel("Frekvencija")

#Histogram sličnosti prema sekvenci
plt.subplot(1, 2, 2)
plt.hist(flat_similarities_train, bins=20, color='red', alpha=0.7)
plt.title("Histogram sličnosti prema sekvencama")
plt.xlabel("Sličnost prema sekvencama")
plt.ylabel("Frekvencija")

plt.tight_layout()
plt.show()

```

```
# In[20]:
```

```
#KORELACIJSKA ANALIZA
```

```

correlation_train, p_value_train = pearsonr(similarities_train.flatten(),
distances_train.flatten())
correlation_test, p_value_test = pearsonr(similarities_test.flatten(),
distances_test.flatten())

```

```

print("Pearsonov koeficijent korelacije za skupove podataka za trening i
testiranje: \n", correlation_train, correlation_test)
print("p-value:", p_value_train, p_value_test, "\n")

# In[21]:

sum_train = 0
sum_test = 0
pval_train = 0
pval_test = 0

N = 50

for x in range(10):

    similarities_train = []
    similarities_test = []

    for i in range(N-50,N):
        row_similarities_train = []
        row_similarities_test = []
        for j in range(N,N+51):
            sim_train = calculate_sequence_similarity(x_train_seq[i],
x_train_seq[j])
            sim_test = calculate_sequence_similarity(x_test_seq[i], x_test_seq[j])
            row_similarities_train.append(sim_train)
            row_similarities_test.append(sim_test)

        similarities_train.append(row_similarities_train)
        similarities_test.append(row_similarities_test)

    similarities_train = np.array(similarities_train)
    similarities_test = np.array(similarities_test)

    distances_train = []
    distances_test = []

    for i in range(N-50,N):
        row_distances_train = []
        row_distances_test = []
        for j in range(N,N+51):
            dist_train = euclidean_distance(z_input[i], z_input[j])
            dist_test = euclidean_distance(z_test_input[i], z_test_input[j])

```

```

        row_distances_train.append(dist_train)
        row_distances_test.append(dist_test)

    distances_train.append(row_distances_train)
    distances_test.append(row_distances_test)

distances_train = np.array(distances_train)
distances_test = np.array(distances_test)

normalized_distances_train = scaler.fit_transform(distances_train)
normalized_distances_test = scaler.fit_transform(distances_test)

    correlation_train, p_value_train = pearsonr(similarities_train.flatten(),
distances_train.flatten())
    correlation_test, p_value_test = pearsonr(similarities_test.flatten(),
distances_test.flatten())

    print("Rezultati ispitivanja pod rednim brojem ", x+1, ":\n")
    print("Pearsonov koeficijent korelacije za skupove podataka za trening i
testiranje: \n", correlation_train, correlation_test)
    print("p-value:", p_value_train, p_value_test, "\n")

    sum_train += correlation_train
    sum_test += correlation_test
    pval_train += p_value_train
    pval_test += p_value_test

N += 50

print("Konačni rezultati ispitivanja:")
print("Usrednjeni koeficijent korelacije za skupove podataka za trening i
testiranje: \n", sum_train/10, sum_test/10)
print("p-value:", pval_train/10, pval_test/10, "\n")

```