Razvoj lijekova s pomoću računalnih simulacija i neuronskih mreža

Ćosić, Stjepan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Stjepan Ćosić

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:
izv. prof. dr. sc. Tomislav Stipančić

Student:
Stjepan Ćosić

Zagreb, 2024.

Istinski znak inteligencije nije znanje, nego mašta.

Albert Einstein

Na prvom mjestu htio bih zahvaliti svojim roditeljima Ivi i Anđelki na bezuvjetnoj ljubavi, podršci i razumijevanju tijekom cijelog mog obrazovanja.

Posebno zahvaljujem svome bratu Slavku i sestrama Luciji, Antoniji i Mariji, koji su mi, osim što su bili neizmjerna podrška često bili i zdrava konkurencija. Njihovi uspjesi i zalaganje motivirali su me da uvijek dajem najbolje od sebe i da stalno radim na vlastitom napretku.

Veliku zahvalnost dugujem i svojim prijateljima Noi Marasoviću, Vilimu Đuri, Marku Belaviću, Dominiku Bastaliću, Antoniu Treppi i Luki Valjku, na neizmjernom ohrabrivanju, strpljenju i pomoći kroz cijeli proces. Vaše prijateljstvo je bilo neprocjenjivo u trenucima kada mi je trebalo malo odmora i zabave.

Na kraju, želim izraziti posebnu zahvalnost svom mentoru, izv. prof. dr. sc. Tomislavu Stipančiću, na nesebičnom dijeljenju svog znanja, strpljenju i podršci. Njegovi savjeti i smjernice bili su od iznimne važnosti za uspješnu izradu ovog rada.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Stjepan Ćosić** JMBAG: **0035239533**

Naslov rada na hrvatskom jeziku: **Razvoj lijekova s pomoću računalnih simulacija i neuronskih mreža**

Naslov rada na engleskom jeziku: **Drugs development using computer simulations and neural networks**

Opis zadatka:

Tijekom ranih faza otkrivanja lijekova, virtualno istraživanje temeljeno na umjetnoj inteligenciji / strojnom učenju / dubokom učenju postalo je neophodan alat. Kod takvih alata koristi se trenirani model za izradu projekcija prilikom stvaranja kataloga malih molekula za određivanje potencijalnih kandidata u ovisnosti o ciljanom proteinu kod stvaranja novih lijekova.

Cilj završnog rada je implementirati i validirati DEEPScreen klasifikator za predviđanje interakcija između lijeka i proteina u ranoj fazi otkrivanja lijekova što je ključno za razvoj u područjima bioinformatike i farmaceutskih znanosti. Navedeni sustav koristi duboku neuronsku mrežu za analizu i klasifikaciju potencijalnih kandidata za lijekove kroz virtualno istraživanje.

U radu je potrebno:

- odabrati te pripremiti prikladan skup podataka za trening modela konvolucijske neuronske mreže za predviđanja ciljanog proteina
- trenirati klasifikator za određivanje prisutnosti ciljanog proteina
- napraviti predviđanja o prisutnosti ciljanog proteina na testnim podacima te na taj način napraviti validaciju korištenog modela.

U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan: 24. 4. 2024.

Zadatak zadao: izv. prof. dr. sc. Tomislav Stipančić

Datum predaje rada:
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva: izv. prof. dr. sc. Petar Ćurković

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Opis oznake
H	skup hipoteza
h	pojedinačna hipoteza
h*	optimalna hipoteza
θ	parametar za definiciju hipoteza
E	empirijska pogreška
D	ulazni skup podataka
TP	ispravno predviđeni pozitivni primjeri
TN	ispravno predviđeni negativni primjeri
FP	negativni primjeri koji su pogrešno klasificirani kao pozitivni
FN	pozitivni primjeri koji su pogrešno klasificirani kao negativni

SAŽETAK

Razvoj lijekova s pomoću računalnih simulacija i neuronskih mreža

Razvoj lijekova je izuzetno kompleksan i dugotrajan postupak zbog izrazito strogih zakona, detaljnog ispitivanja i potrebe za velikim ulaganjima. Financijska sredstva i etička ispravnost upravo su glavni razlozi zašto industrija trenutno traži neke od alternativnih načina ubrzanja najduljeg početnog dijela razvoja. Uporaba umjetne inteligencije i strojnog učenja, koji su izuzetno razvijeni u posljednjem desetljeću, čini se kao pravi izbor za to. Određeni modeli omogućuju predikciju ispitivanja ponašanja kemijskih spojeva iz lijekova s ciljanim proteinima u čovjeku s pomoću 2D zapisanih podataka. Upravo jedan od tih modela je DEEPScreen klasifikacija koja je svojim dosadašnjim radom pokazala izrazito visoku točnost. U ovom radu u potpunosti je obrađena DEEPScreen klasifikacija. Od teoretskog objašnjenja njegovog rada, pa sve do testiranja na dostupnim podacima iz ChEMBL v23 baze podataka. Provedena su testiranja iz kojih su kao vrijednosti vrednovanja dobivene preciznost, odziv, točnost, F1-score i MCC koeficijent, te gubitci tijekom treninga. Obavljeni su treninzi s različitim skupovima podataka i treninzi s istim skupom podataka, ali različitim brojem epoha treninga. Svi rezultati su u konačnici uspoređeni, te su doneseni zaključci o radu napravljenog modela. Primjena DEEPScreen klasifikatora u razvoju lijekova pokazala se kao vrlo uspješna stvar. Po rezultatima obavljenih treninga koji upućuju na dobru predikciju, uporabom DEEPScreen klasifikatora vrlo se lako može ubrzati proces ranog razvoja lijeka od kojeg će cijela farmaceutska industrija i njeni korisnici imati benefita.

KLJUČNE RIJEČI: razvoj lijekova, umjetna inteligencija, strojno učenje, 2D zapisani podaci, DEEPScreen klasifikacija

SUMMARY

Drugs development using comuputer simulations and neural networks

The development of drugs is an extremely complex and lengthy process due to stringent regulations, thorough testing, and the need for substantial investments. Financial resources and ethical considerations are the main reasons why the industry is currently seeking alternative ways to accelerate the longest initial phase of development. The use of artificial intelligence and machine learning, which have significantly advanced in the last decade, seems to be the right choice for this. Certain models allow for the prediction of the behavior of chemical compounds from drugs with targeted proteins in humans using 2D recorded data. One such model is the DEEPScreen classifier, which has demonstrated exceptionally high accuracy in its work so far. This paper fully explores the DEEPScreen classification. It covers everything from the theoretical explanation of its operation to testing on available data from the ChEMBL v23 database. The tests conducted yielded precision, recall, accuracy, F1-score, and MCC coefficient as evaluation metrics, as well as losses during training. Training was conducted with different datasets and with the same dataset but varying the number of training epochs. All results were ultimately compared, and conclusions were drawn about the performance of the developed model. The application of the DEEPScreen classifier in drug development has proven to be very successful. Based on the training results, which indicate good prediction, the use of the DEEPScreen classifier can easily accelerate the early drug development process, benefiting the entire pharmaceutical industry and its users.

KEYWORDS: drug development, artificial intelligence, machine learning, 2D recorded data, DEEPScreen classification

1. UVOD

Čovjek od svojih ranih godina postojanja pokušava sačuvati i poboljšati svoj život, a jedan od načina je stvaranje supstance koja može riješiti individualne zdravstvene probleme poznatijom kao lijek. Razvojem čovječanstva, razvijale su se i ljudske navike u prehrani, osobnoj higijeni i uvjetima stanovanja što je dovelo do dužih života. Zbog svih ovih promjena ljudi su izloženi većem broju mogućih zdravstvenih problema, čak i onim bolestima koje nisu bile toliko česte upravo zbog kratkog života. Daljnjim razvitkom čovječanstva, razvijale su se razna znanstvena područja od kojih velik značaj ima medicina i farmaceutska industrija. Naime, upravo su one razlog značajnog produljenja ljudskih života. Nažalost, osim razvoja samih lijekova i medicine, razvijaju se i nove bolesti. Jedna od značajnih je bila nedavna virusna zaraza COVID-19, koja je 2020. godine uzrokovala globalnu pandemiju, te tako ugrozila industriju, ekonomiju, društvo i ono najgore, ljudske živote. Upravo zbog ovakvih brzorastućih zaraza i bolesti koje mogu dovesti u pitanje zdravlje naših najmilijih, ekonomiju i industriju koja čini osnovnu bazu društva, to jest cjelokupni svijet koji znamo, potrebno je razviti brz način razvijanja lijekova. Moderniziranje ranog razvoja lijekova putem strojnog učenja u farmaceutskoj industriji jedno je od realnih, sve češće implementiranih rješenja. DEEPScreen klasifikacija je metoda, jedna od mnogih, koja nam pruža brz i efikasan način pronalaska komplementarnih spojeva danih proteina iz ljudskog tijela i kemijskih spojeva, možda, budućeg lijeka.

2. RAZVOJ LIJEKOVA

2.1. Tradicionalan razvoj lijekova

Razvoj lijekova već se godinama temelji na detaljnom proučavanju određene bolesti kod oboljelih kako bi se došlo do zaključka o uzročniku, odnosno detaljnih karakteristika poput kemijskog sastava, uvjeta u kojima se uzročnik ponaša aktivnije, to jest pasivnije, te njegovog djelovanja tijekom vremena. Daljnji razvoj se temelji na laboratorijskim ispitivanjima koja mogu biti u kontroliranim uvjetima izvan živog organizma (in vitro) ili ispitivanjima koja uključuju živi organizam (in vivo) kako bi se dobile preliminarne informacije o djelotvornosti, toksičnosti, distribuciji i eliminaciji lijeka.[1] Sveukupni proces razvoja lijeka izrazito je dug, detaljan i jasno definiran od strane zato zaduženih organizacija. Proces razvoja lijekova podijeljen je u 4 faze, kao što je prikazano na slici 1.: rana faza, klinička ispitivanja, odobrenje i postmarketinški nadzor. Cijeli postupak traje oko 10 godina.

Faza	Ispitivana skupina	Svrha	Trajanje
Rani razvoj			
	Laboratorijski uvjeti (kao što su kulture stanica i životinje)	Određivanje kemijskih i fizikalnih karakteristika lijeka te procjena sigurnosti i učinaka lijeka u živim organizmima.	2–6,5 godina
Klinička ispitivanja			
Faza 1	20–80 zdravih dobrovoljaca	Utvrditi osnovnu sigurnost i razinu lijeka u krvi koja se postiže s različitim dozama lijeka	1,5 godina
Faza 2	Do 100 ljudi koji imaju ili bi mogli razviti poremećaj koji se proučava	Utvrditi učinkovitost i raspon doziranja lijeka te utvrditi nuspojave	2 godine
Faza 3	300-30 000 ljudi koji imaju poremećaj koji se proučava	Potvrditi najučinkovitiji režim doziranja te dobiti više informacija o učinkovitosti lijeka i nuspojavama koje nisu viđene tijekom faza 1 i 2. Usporedba lijeka s postojećim lijekovima, s placebom ili oboje.	3,5 godina
Ocjena FDA			
	Vladin pregled svih informacija iz ranog razvoja i kliničkih ispitivanja	Utvrditi je li dokazana učinkovitost i sigurnost lijeka	0,5–1 godina
Faza 4 (postmarketinški nadzor)			
	Svi ljudi koji uzimaju lijek, osobito podskupine kao što su trudnice, djeca i starije osobe	Utvrditi bilo koje probleme koji se nisu pojavili u fazama 1, 2 ili 3, kao naprimjer oni koji se pojavljuju nakon dugo vremena te oni koji se rijetko događaju	Kontinuirano

Slika 1. Podjela razvoja lijekova na faze [2]

2.1.1. Rani razvoj lijekova

Nakon definiranja ideje za lijek koji može biti koristan u rješavanju određenog zdravstvenog poremećaja i definiranja njegovog sastava, kreću testiranja. Uobičajeno se testiranja provode na laboratorijskim životinjama. Iz tih testiranja se prikupljaju podaci o djelovanju lijeka, učinkovitosti, toksičnosti, uzrokovanim posljedicama na reproduktivne sposobnosti i zdravlje potomaka. Kod velike količine lijekova u ovoj fazi dolazi do prekida razvoja zbog prikupljenih informacija o velikoj toksičnosti ili neučinkovitosti. Ostali lijekovi, čija ispitivanja daju obećavajuće podatke o učinkovitosti idu u daljnji razvoj, to jest drugu fazu.[2]

2.1.2. Klinička ispitivanja

Kako bi klinička ispitivanja bila moguća potrebno je za lijek u razvoju prikupiti određene dozvole od nadležnih institucija, pa tako program koji opisuje kliničko ispitivanje mora biti odobren od strane odgovarajućeg etičkog povjerenstva (IRB), a prijava za novi ispitivani lijek (IND) se mora podnijeti Agenciji za hranu i lijekove (FDA). Ako FDA odobri zahtjev, lijek se smije ispitivati kod ljudi. U Republici Hrvatskoj odobrenje za provođenje kliničkih ispitivanja izdaje Ministarstvo zdravstva na temelju pozitivnog mišljenja Središnjeg etičkog povjerenstva.[2]

Kliničko ispitivanje se odvija u tri podfaze: Faza 1, Faza 2 i Faza 3. Faza 1 se sastoji od malog broja najčešće muških ispitanika kojima se daje lijek. Ova faza služi za procjenjivanje sigurnosti i toksičnosti lijekova kod ljudi.

U Fazi 2 se povećava broj ispitanika koji imaju ciljani poremećaj i služi za provjeravanje učinkovitosti lijeka i definiranje potrebne doze. Naime, da lijek uopće dospije na kliničko ispitivanje mora, kao što je već rečeno, zadovoljiti na testiranju kod životinja, no to ne znači da će lijek imati jednako učinkovito djelovanje na čovjeka.

Posljednja podfaza kliničkog ispitivanja, Faza 3, se odvija na još većem broju ispitanika koji su jako slični realnim korisnicima na tržištu koji bi koristili ovaj tip lijeka, te i dalje služi za proučavanje učinkovitosti. Također, u ovoj podfazi se stavlja naglasak na nove nuspojave i usporedbu sa sličnim postojećim lijekovima na tržištu.

Kliničko ispitivanje, osim određivanja učinkovitosti lijeka, usredotočuje se i na vrstu i učestalost nuspojava te na čimbenike koji ljude čine podložnima tim učincima (kao što su dob, spol, drugi poremećaji i upotreba drugih lijekova).[2]

2.1.3. Odobrenje

Nakon prethodnih faza, ispitivanja na životinjama u ranoj fazi i ljudima u kliničkim ispitivanjima, koja ukazuju na sigurnost i učinkovitost lijeka, podnosi se zahtjev za odobrenje lijeka od strane nadležnih institucija (FDA ili EMA) na temelju dobivenih podataka. Oni nadalje pregledavaju sve dobivene podatke i odlučuju je li lijek dovoljno učinkovit i siguran za stavljanje na tržište. Prosječno, samo pet od četiri tisuće lijekova koji su prošli ranu fazu razvoja se ispituju na ljudima, a od tih pet samo jedan bude stavljen na tržište što ukazuje na rigoroznost i detaljnost ispitivanja dobivenih podataka.

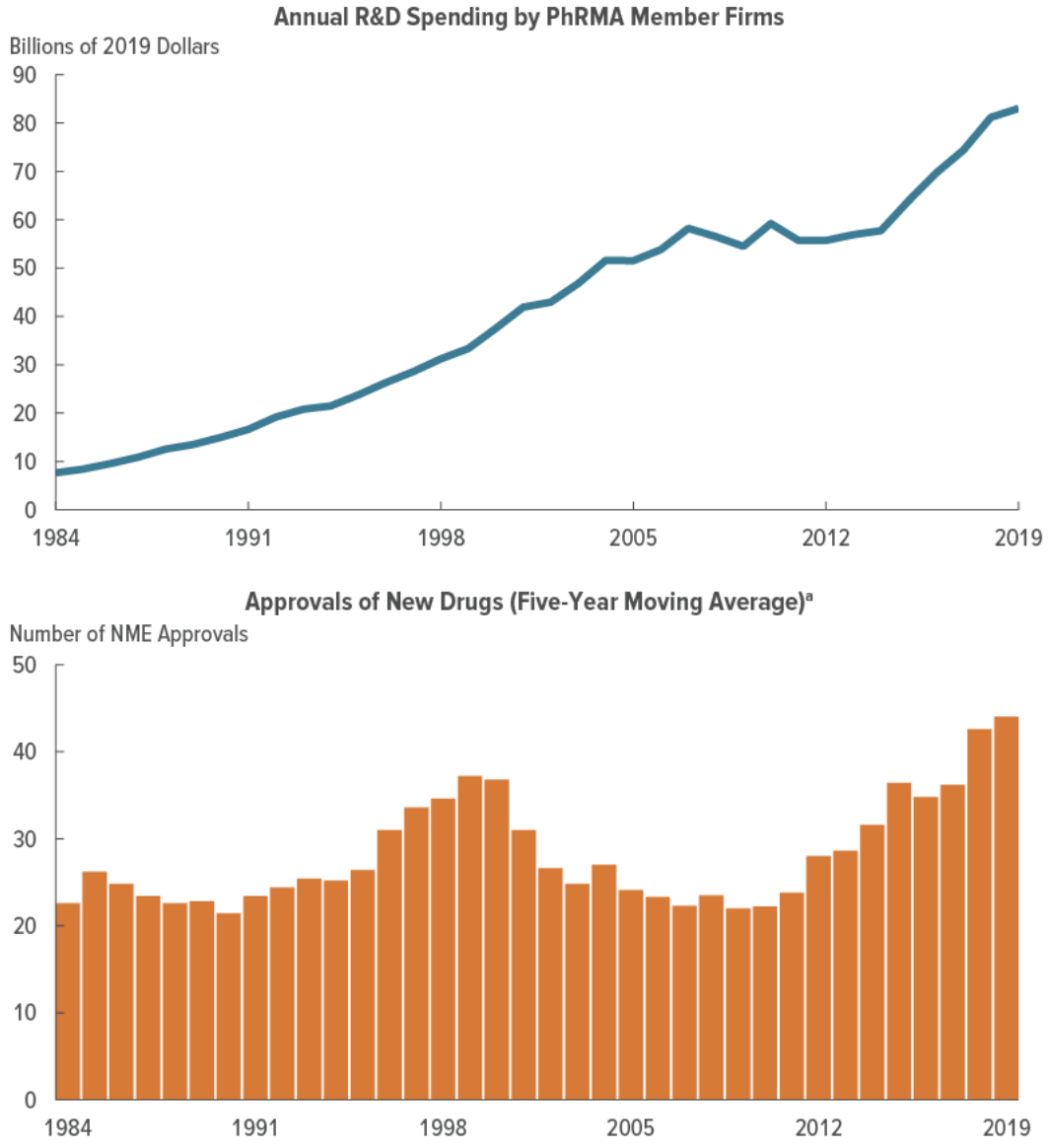
2.1.4. Postmarketinško razdoblje

Nakon stavljanja lijeka na tržište, njegova upotreba se i dalje prati kako bi došlo do zapažanja pojave ozbiljnijih nuspojava. Naime, obavljena dosadašnja ispitivanja mogu ukazati samo na česte nuspojave koje se pojavljuju otprilike jednom na svakih tisuću doza, dok pojava onih važnih koje se javljaju na svakih deset tisuća doza se može otkriti samo dok velika količina ljudi koristi taj lijek, odnosno dok je na tržištu. Ako se pojavi velika količina važnih nuspojava, lijek se može povući s tržišta.

2.2. Izazovi u razvoju lijekova

Razvoj lijekova je dugotrajan i skup proces koji iziskuje visok stupanj odgovornosti i stroga pravila. Istraživanja pokazuju da unatoč tome što se društvo, tehnologija i globalizacija razvijaju izuzetno velikom brzinom, te uzevši u obzir da su ulaganja u razvoj lijekova sve veća, odobravanje novih lijekova na tržištu stagnira. Prema izvještaju Kongresnog ureda za proračun

SAD-a (Congressional Budget Office) [4], povećana ulaganja ne rezultiraju povećanim brojem novih lijekova, pa tako početkom trećeg tisućljeća prosječan godišnji iznos uloženog novca u razvoj lijekova u SAD-u kretao se između trideset i četrdeset milijardi američkih dolara pri čemu se odobrilo četrdesetak lijekova na godišnjoj bazi. U trenutku provedenih zapisa 2019. godine, uložena cifra iznosila je gotovo devedeset milijardi američkih dolara, a dobiveni broj novih lijekova je bio neznatno veći, to jest svega četrdeset i pet lijekova. Slika 2. zorno prikazuje odnose financijskih ulaganja i broja odobrenih lijekova tokom godina za SAD-e.

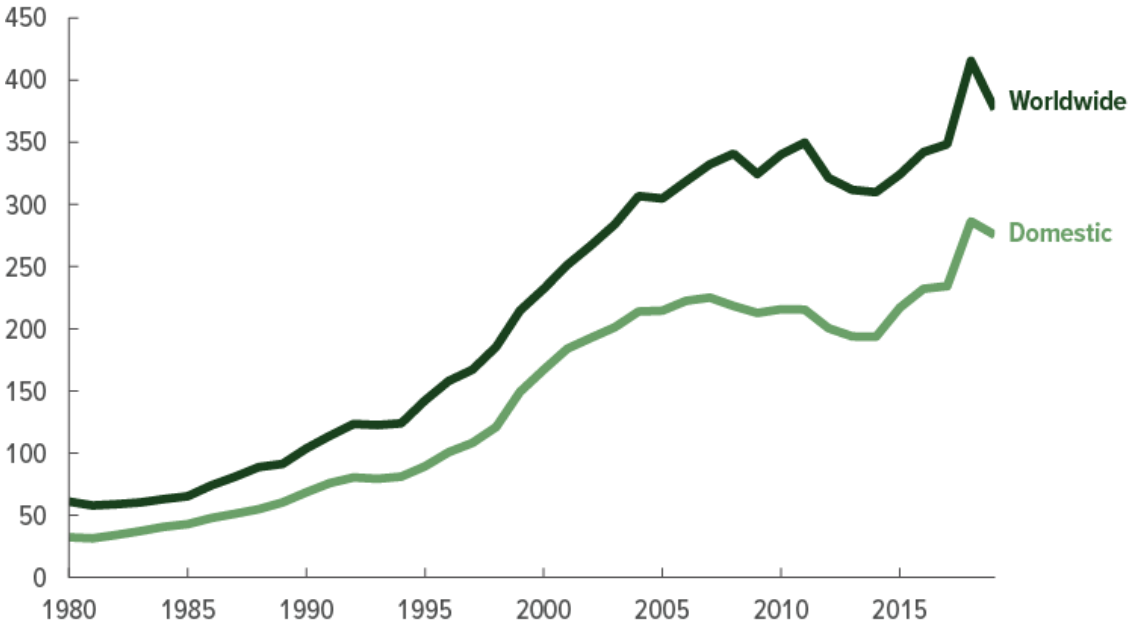


Slika 2. Godišnja ulaganja u razvoj lijekova i količina odobrenih lijekova od 1984. do 2019. godine [4]

Također, jedna od velikih prepreka u razvoju lijekova su etička pitanja prilikom testiranja. Prema članku Europskog parlamenta „Dobrobit i zaštita životinja: Zakonodavstvo EU-a“ potiče se smanjenje korištenja životinja prilikom ispitivanja, zamjenom ispitivanja s nekim alternativnim načinom i poboljšanjem uvjeta ispitivanja (svođenje boli i patnje na najmanje moguće mjere). Sva ta ograničenja ne samo da povećavaju pritisak na ionako već preopterećeni budžet, nego dodatno kompliciraju već zakomplicirane i dugotrajne pravne postupke puštanja lijeka na tržište.[5]

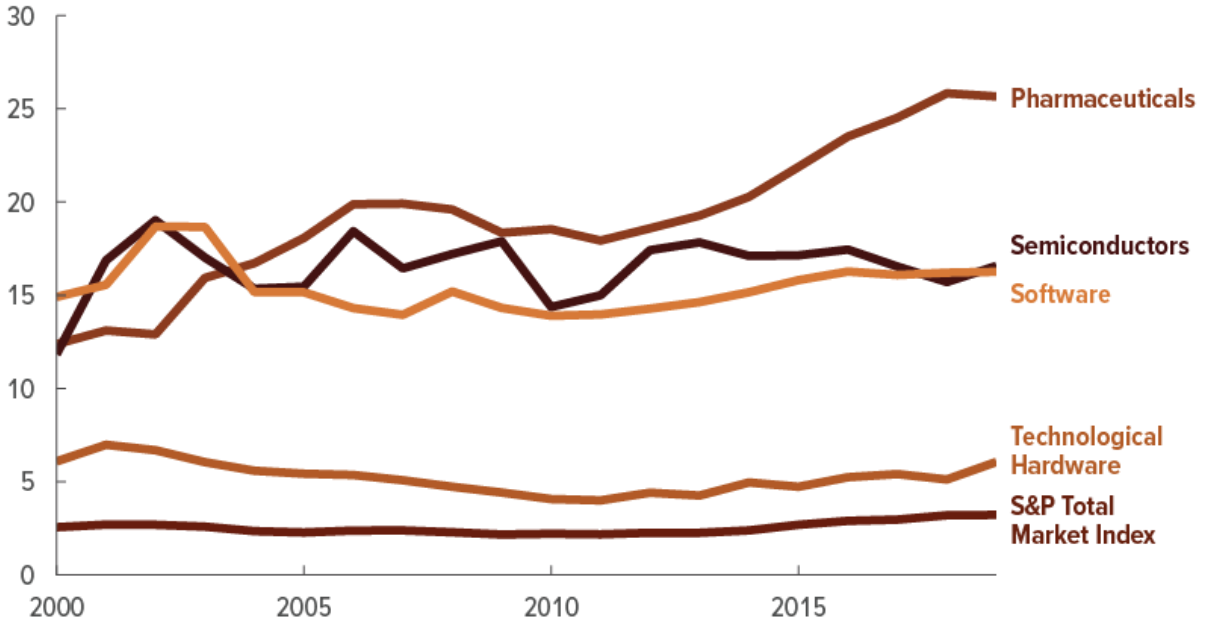
2.3. Primjena modernih tehnologija (strojnog učenja) u razvoju lijekova

Farmaceutska industrija nije samo bitan dio društva nego i bitan dio svjetske ekonomije. Kao što smo mogli i pročitati u prethodnom poglavlju 2.2., ulaganja su izuzetno velika i to s opravdanim razlogom. Prodaja lijekova značajno raste, uz blage anomalije poput svjetske ekonomske krize 2008. godine, čemu svjedoči rast prihoda ispitanih farmaceutskih kompanija, kako američkih, kako svjetskih, što i svjedoči slika 3..



Slika 3. Rast prodaje lijekova u svijetu i SAD-u [4]

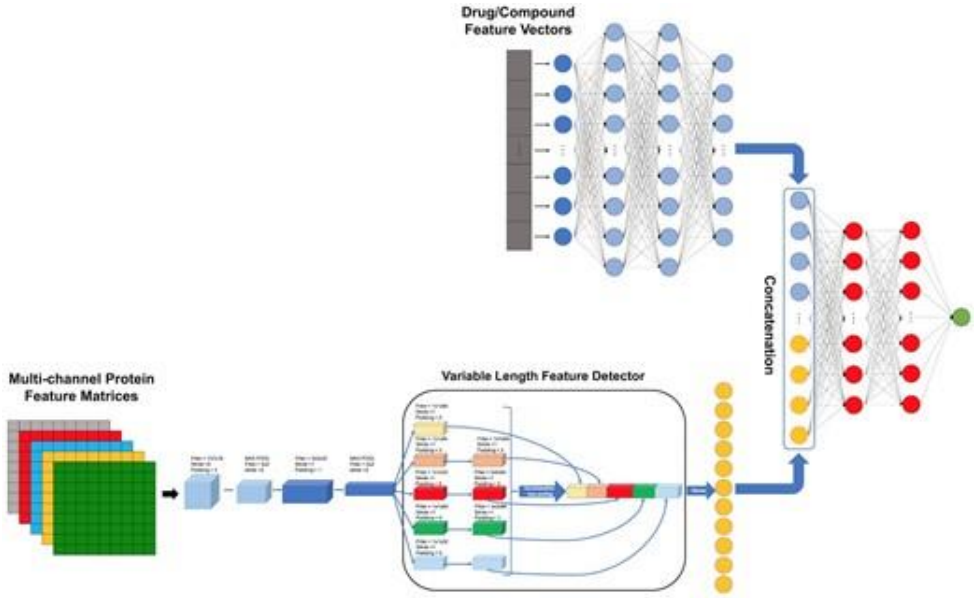
Cvjetanje farmaceutske industrije dodatno pokazuju i povećanje ulaganja u posljednja dva desetljeća kako je i prikazano na slici 4.. Farmaceutske kompanije ulažu otprilike devetnaest posto svojih neto prihoda u daljnji razvoj tehnologije za razvoj lijekova, što je više od poluvodičkih i softverskih kompanija koje ulažu oko petnaest posto. [4]



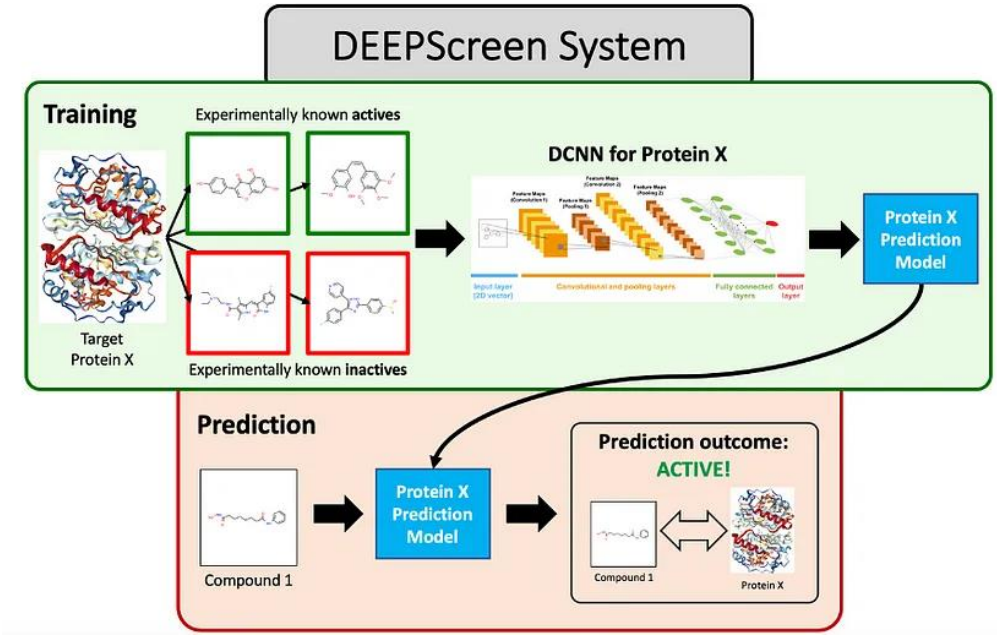
Slika 4. Ulaganja u svjetske industrije [4]

Velika uloga farmaceutske industrije u svjetskoj ekonomiji iziskuje nove oblike razvoja lijekova kako bi se skratilo vrijeme razvoja. Jedan od načina ubrzavanja procesa razvoja je uključivanje visoke tehnologije poput umjetne inteligencije, neuronskih mreža, virtualnih simulacija i njima sličnih. Softverske tehnologije osmišljene za proces razvoja lijekova uglavnom su namijenjene za zamjenu početne rane faze razvoja u kojima se provodi testiranje na životinjama. Softveri se baziraju na simuliranju spajanja određenih proteina i kemijskih spojeva. Nakon virtualnih simulacija i ispitivanja stvara se usporedba s onim realnim rezultatima koja služi za poboljšavanje, to jest učenje samog softvera. Ubacivanjem ovakvog oblika tehnologije u ovaj proces omogućuje se i prenamjena postojećih lijekova, što dodatno ubrzava proces razvoja novih lijekova.

Metode strojnog učenja koji se koriste u razvoju lijekova mogu se podijeliti u dvije grupe: na one s jednim ulazom, to jest samo kemijskim spojem, i one s parnim ulazom, protein i kemijski spoj. Dvije metode koje se ističu su DEEPScreen kao binarni klasifikator prikazan na slici 6. i MDEEPred kao regresor s parnim ulazom prikazan na slici 5.. [6] [7]



Slika 5. MDEEPred metoda [7]

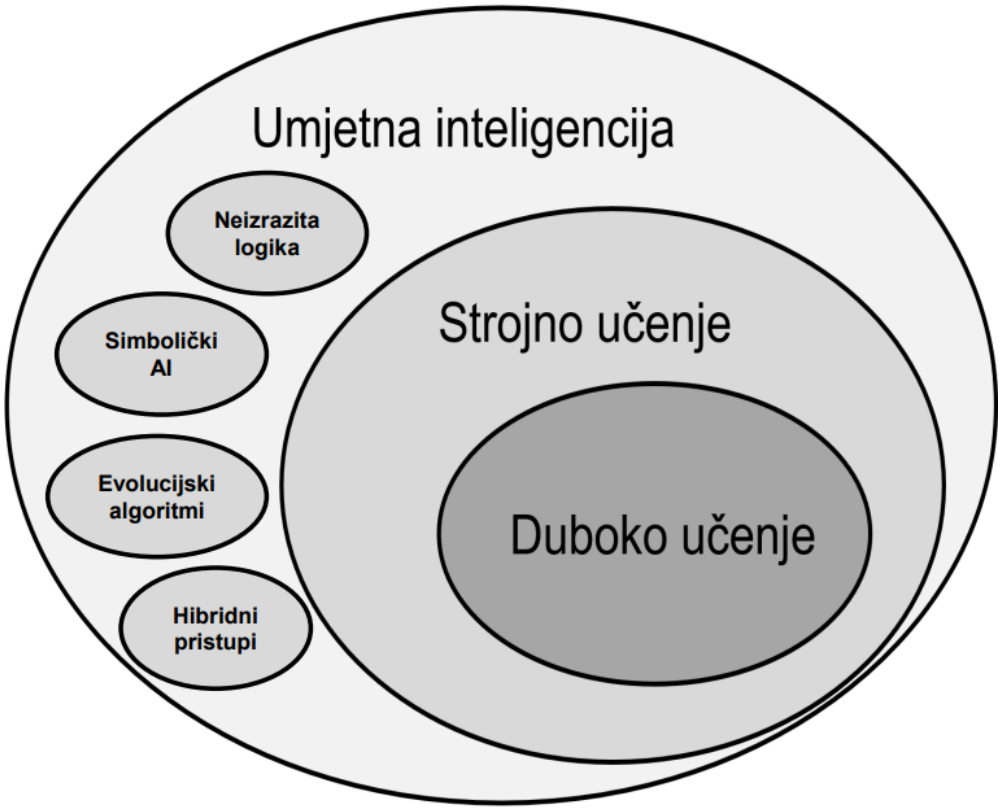


Slika 6. DEEPScreen metoda [6]

3. DEEPSCREEN KLASIFIKATOR

3.1. Umjetna inteligencija i neuronske mreže

Umjetna inteligencija je izrazito važna tehnologija koju, iako nismo toga svjesni, susrećemo svaki dan samim našim djelovanjem na internetu. To je tehnologija koja prima veliku količinu podataka od samih ljudi i tehničkih i prirodnih sustava preko senzora ili nekih drugih uređaja za dobivanje informacija. Razvojem društva i globalizacije gomilaju se velike količine podataka (Big Dana, BD) iz kojih je moguće povući veliku količinu znanja za daljnji razvoj, te upravo zbog takvih karakteristika ljudske stvarnosti umjetna inteligencija već sada ima važnost u čovječanstvu koja samo može rasti. Umjetnu inteligenciju mnogi smatraju vrhunskim alatom kojim bi se rasteretilo čovjeka prilikom obavljanja monotonog dugotrajnog posla za koji je svejedno potreban nekakav oblik ljudske inteligencije. Postoje različite podskupine umjetne inteligencije prikazane na slici 7. od kojih je daleko najzastupljenije strojno učenje (duboko učenje).[8]



Slika 7 Podskupovi umjetne inteligencije [8]

Strojno učenje je grana umjetne inteligencije koja se bavi razvojem i proučavanjem algoritama koji uče bez eksplicitnog programiranja. [8] Zadatak algoritama strojnog učenja bazira se na pronalaženju uzoraka i poveznica na temelju kojih se steći uvid, te odlučivati i predviđati, to jest stvaranju novog znanja iz BD. Velikim porastom količine podataka u svijetu kojem živimo, ponajviše uzrokovano globalizacijom, povećava se i učestalost korištenja strojnog učenja. Ono nam omogućuje da lakše primijetimo anomalije u svijetu oko nas, te da ga lakše shvatimo i to sve iz BD. Primjena strojnog učenja je široka i postaje vrlo važna u mnogim područjima poput financija, računalne vizije, biologije, proizvodnje energije i industrijske proizvodnje.[9]

Svaki model strojnog učenja mora sadržavati komponente poput modela, funkcije pogreške i optimizacijskog postupka. [3]

Model je skup hipoteza kojima je definiran način predviđanja izlaza na temelju ulaza. On se može temeljiti na neuronskim mrežama, stablima odluke ili linearnoj regresiji. Također, može biti parametarski i neparametarski.

$$H = \{h(x;\theta)\}_\theta$$

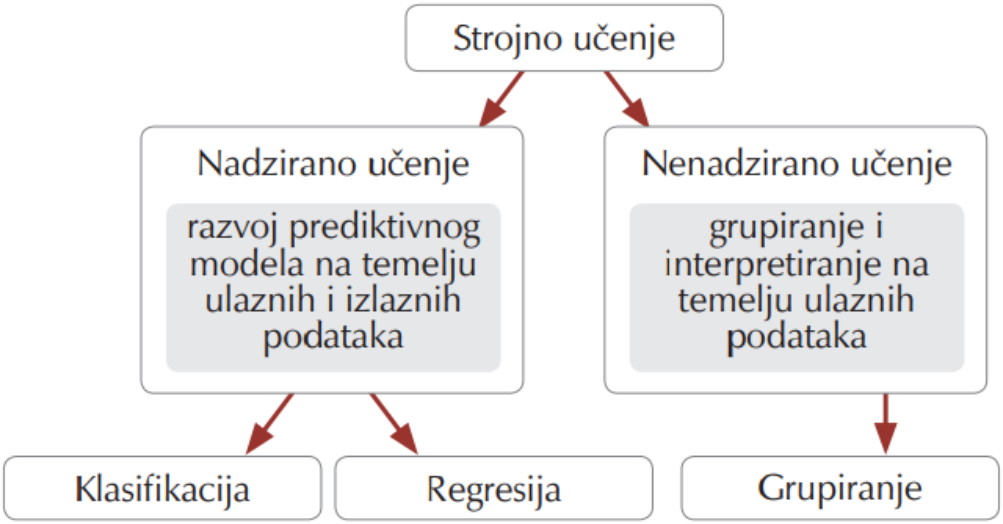
Funkcijska pogreška, kao i što sam naziv kaže, je funkcija koja ukazuje na to kolika se greška javlja između predviđenih podataka i onih stvarnih, te se tim principom mjeri je li model dobar ili loš.

$$E(h|D) \text{ odnosno } \mathrm{E}(\theta|D)$$

Optimizacijski postupak služi za smanjivanje funkcijske pogreške promjenom parametara modela. Ovaj postupak se najčešće bazira na nekom optimizacijskom algoritmu kojim se pokušavaju modelirati parametri tako da izlazi budu što točniji u odnosu na stvarne.

$$h^* = \underset{\mathrm{h}\in\mathrm{H}}{argmin}\, E(h|D)$$

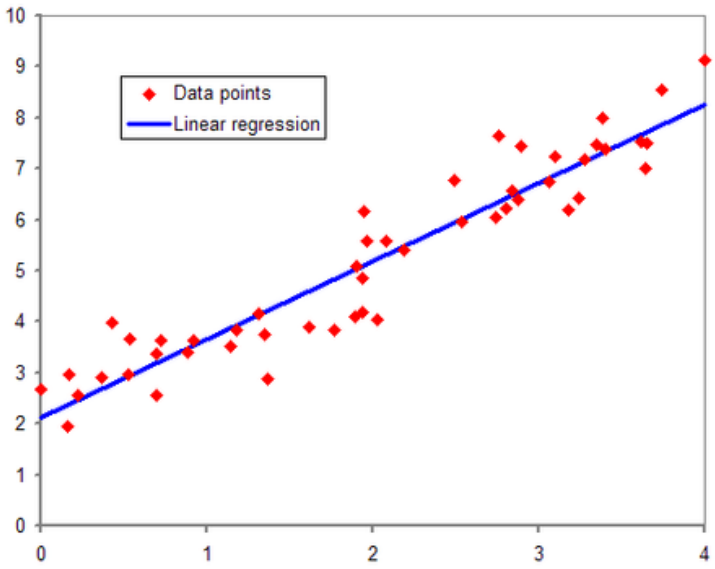
Cilj strojnog učenja je omogućiti optimalne hipoteze h* koje najbolje generalizira skup podataka, na temelju precizno definiranog modela, funkcijske pogreške i optimizacijskog postupka. Osnovna podjela strojnog učenja je na nadzirano i nenadzirano strojno učenje, što zorno prikazuje slika 8.. Nadzirano strojno učenje se zasniva na učitavanju realnih podataka dobivenih u stvarnim eksperimentima koji služe kao ulazi za uvježbavanje, kako bi u konačnici algoritam bio sposoban predviđati, dok nenadzirano učenje traži skrivene uzorke i inherentne strukture u ulaznim podacima bez poznavanja izlaza.[9]



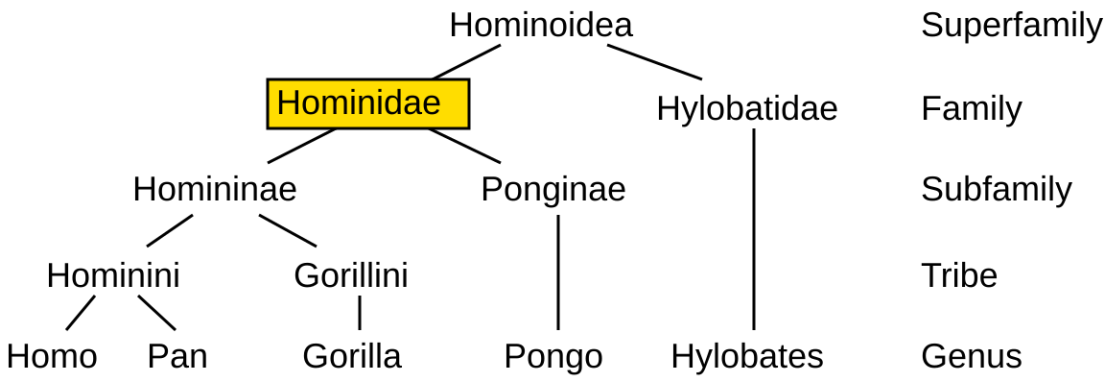
Slika 8. Podjela strojnog učenja [9]

3.1.1. Nadzirano strojno učenje

Za nadzirano strojno učenje potrebne su poznate ulazne i izlazne skupine podataka za određeni proces na koji želimo prilagoditi algoritam, te ga uvježbavamo za predikciju. Za ovaj oblik učenja koriste se postupci regresije i klasifikacije prikazani na slikama 9. i 10.. Regresija je postupak kojim se predviđaju kontinuirane varijable, dok klasifikaciju koristimo za predviđanje diskretnih odziva razvrstavanjem ulaza u kategorije.[9]



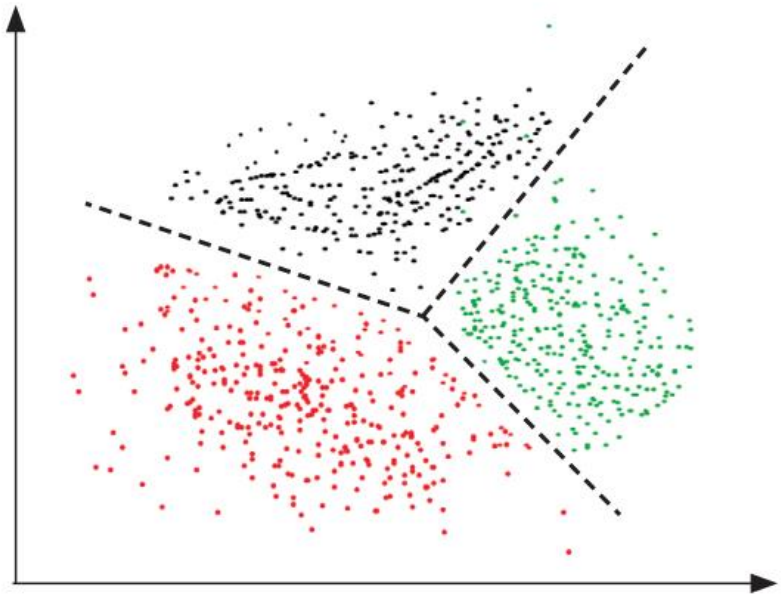
Slika 9. Linearna regresija



Slika 10. Klasifikacija

3.1.2. Nenadzirano strojno učenje

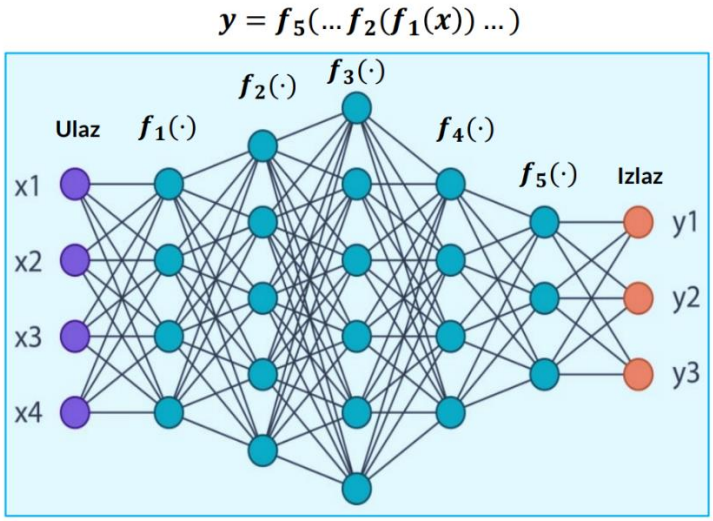
Nenadzirano strojno učenje pronalazi skrivene uzorke i inherentne strukture u ulaznim podacima, a na temelju njih prepoznaje odnosno grupira povezane podatke. Najčešći postupak korišten pri ovom obliku učenja je grupiranje. Grupiranje je jako sličan postupak klasifikaciji, međutim postoje male razlike. Kod klasifikacije postoje unaprijed definirane klase/skupine u kojima će završiti pojedini podaci, dok grupiranje svodi na dijeljenje podataka u grupe na temelju njihove sličnosti kao što prikazuje slika 11.. Kod grupa nastalih grupiranjem granice ne moraju biti fiksne, to jest moguće ih je prilagoditi.[9]



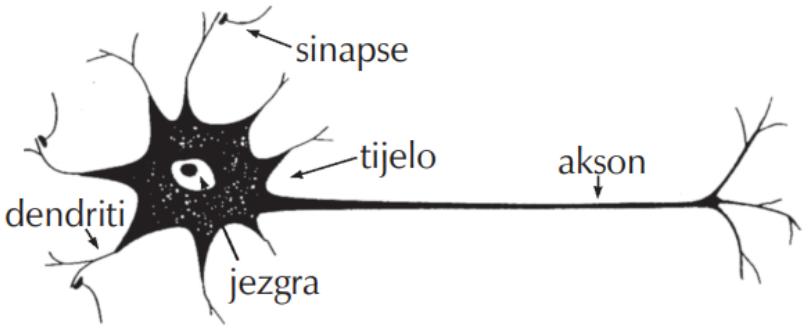
Slika 11. Grupiranje

3.2. Umjetne Neuronske mreže

Umjetne neuronske mreže su skup matematičkih funkcija koje su poredane u slojeve i međusobno povezane ulazima odnosno izlazima. Kao što se može zaključiti po samom imenu, neuronske mreže su dobile ime po ljudskim živčanim stanicama koje imaju svoje dijelove poput dendrita, tijela stanice i aksona. Unatoč tome što to nisu iste stvari, imaju jako slične zadaće. Dok kod čovjeka neuron služi za prijenos određenog podražaja zbog kojeg se generira određena reakcija u čovjekovom mozgu, umjetni neuron će primiti određeni ulazni podatak i na temelju matematičkih funkcija generirati određeni izlaz. (Slika 12. i 13.)

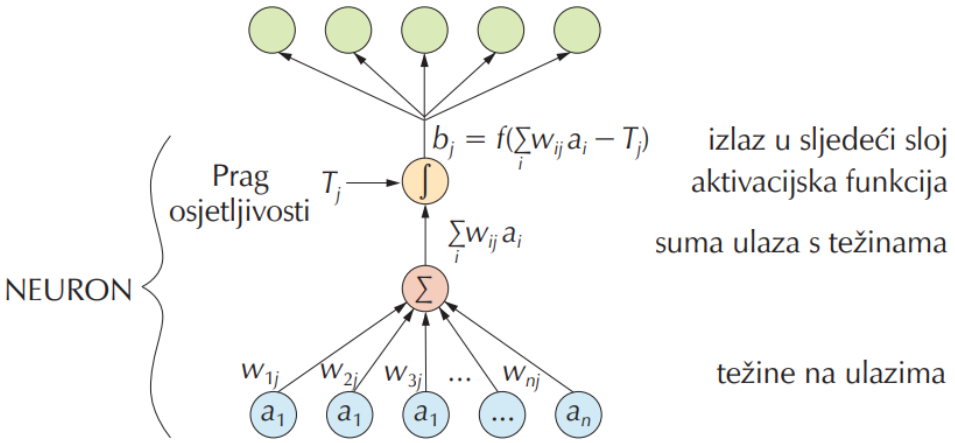


Slika 12. Umjetna neuronska mreža [8]



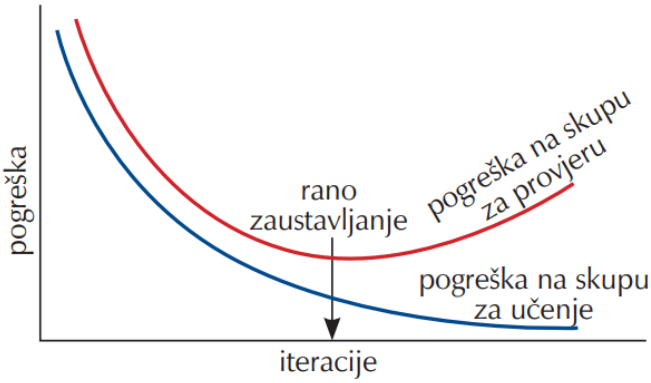
Slika 13. Ljudski neuron [10]

Prvi rad o umjetnim neuronskim mrežama objavili su McCulloch i Pitts, u kojem su opisali rad jednostavnog oblika neuronske mreže imenom Perceptron, čiji dizajn se nalazi na slici 14.. Ona i danas služi kao osnovni primjerni model kako umjetni neuroni rade. Svaka umjetna neuronska mreža sastoji se od ulaznog, izlaznog i skrivenog sloja. Broj skrivenih slojeva može biti različit, a njihovim brojem definira se je li neuronska mreža duboka ili plitka. Slojevi su međusobno spojeni, tako da se signali unutar tih veza množe s težinskim faktorom w_i koji su analogni ljudskim dendritima. Otežani ulazni signali zbrajaju se i uspoređuju s pragom osjetljivosti neurona, T_i. Ako se dogodi da je zbroj otežanih signala veći od praga osjetljivosti nelinearna aktivacijska funkcija f generira izlazni signal neurona iznosa b_i.[10]



Slika 14. Način rada Perceptrona [10]

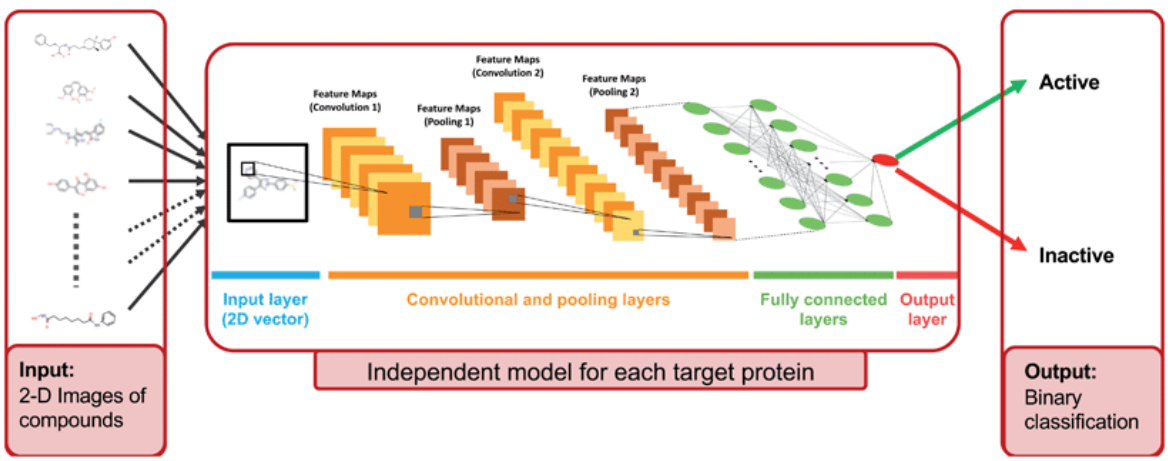
Proces učenja umjetnih neuronskih mreža odvija se na način iterativnog podešavanja vrijednosti težinskih faktora na temelju funkcijske pogreške, to jest razlike dobivenih izlaza u odnosu na stvarne. Za svaki prolaz podataka kroz mrežu generiraju se funkcijske greške, a s time se korigiraju težinski faktori, stoga je poželjno imati više grupa podataka npr. dvije, Jedna skupina podataka će služiti za trening, a druga za provjeru. Prilikom učenja potrebno je primijetiti i jasno definirati idealni broj prolaza podataka kroz mrežu kako bi se spriječilo pretreniranje. Pretreniranje je pojava kada nam s povećanjem broja prolaza raste i funkcijska pogreška, zbog čega je nužno rano zaustavljanje kako bismo dobili što točniji model.[10] (Slika 15.)



Slika 15. Pretreniranje [10]

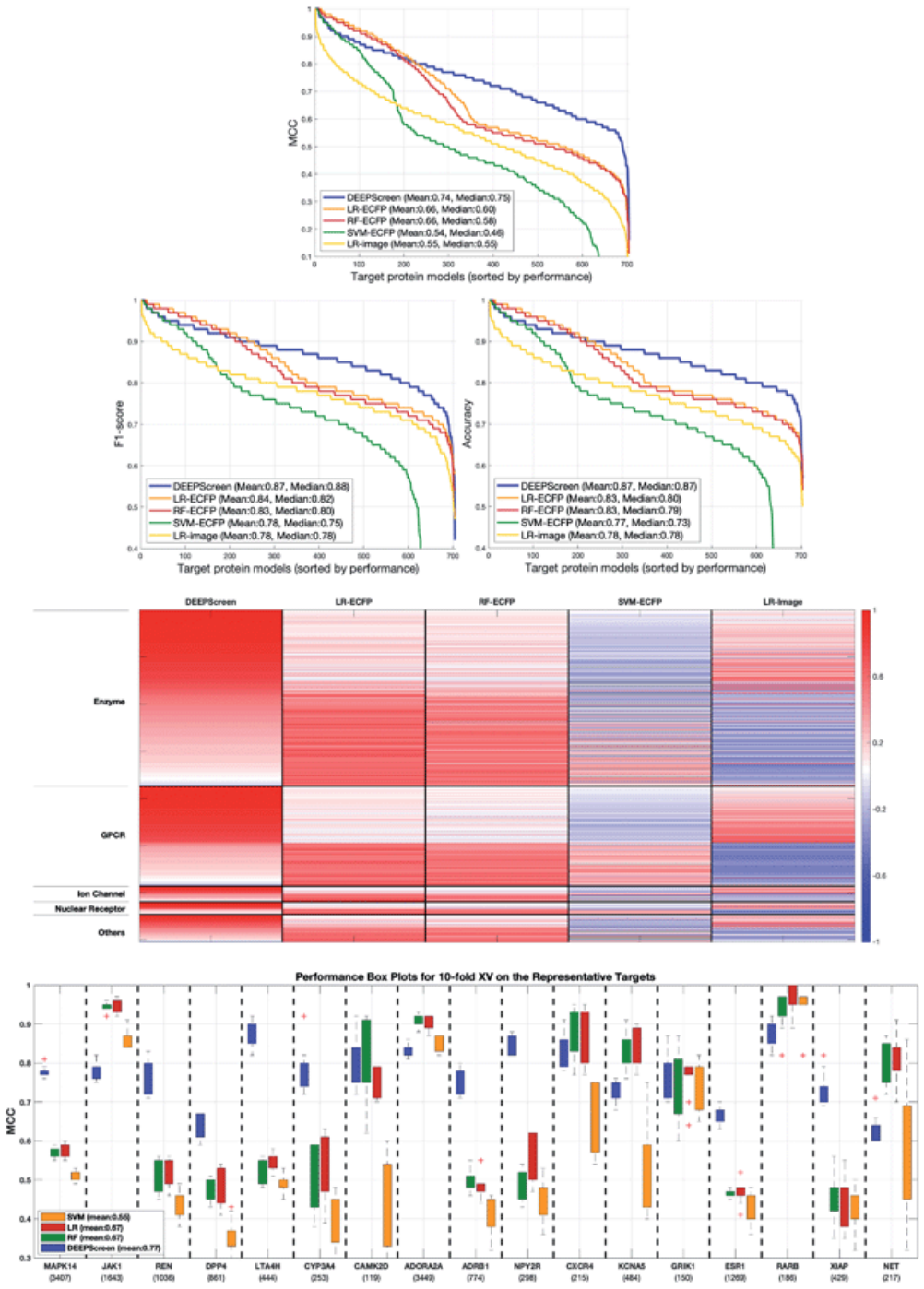
3.3. Opis rada DEEPScreen klasifikatora

Rana faza lijekova, ispitivanje na životinjama, jedna je od najvažnijih faza jer se u njoj identificiraju fizičke interakcije između kandidata za lijekove i ciljanih biomolekula. Ispitivanje na životinjama, kao i cijeli proces razvoja lijeka, dugotrajan je i skup, pa se u posljednje vrijeme sve više koriste sustavi za predikciju koji automatski predviđaju nove interakcije lijek-cilj (DTI). DEEPScreen je jedan od nepoznatijih sustava za predikciju DTI velikih razmjera za ranu fazu otkrivanja lijekova. Rad DEEPScreen klasifikatora, prikazan na slici 16., se bazira na dubokim konvolucijskim neuronskim mrežama i koristi lako dostupne 2D strukturne prikaze spojeva kao ulazne podatke. Upravo, taj oblik 2D podataka velika je prednost DEEPScreen-a u odnosu na ostale sustave za rani razvoj lijekova, jer DEEPScreen inherentno uči složene karakteristike iz 2D prikaza, pri čemu postiže vrlo točne predikcije za razliku od konvencionalnih deskriptora koji pokazuju ograničene performanse. [6]

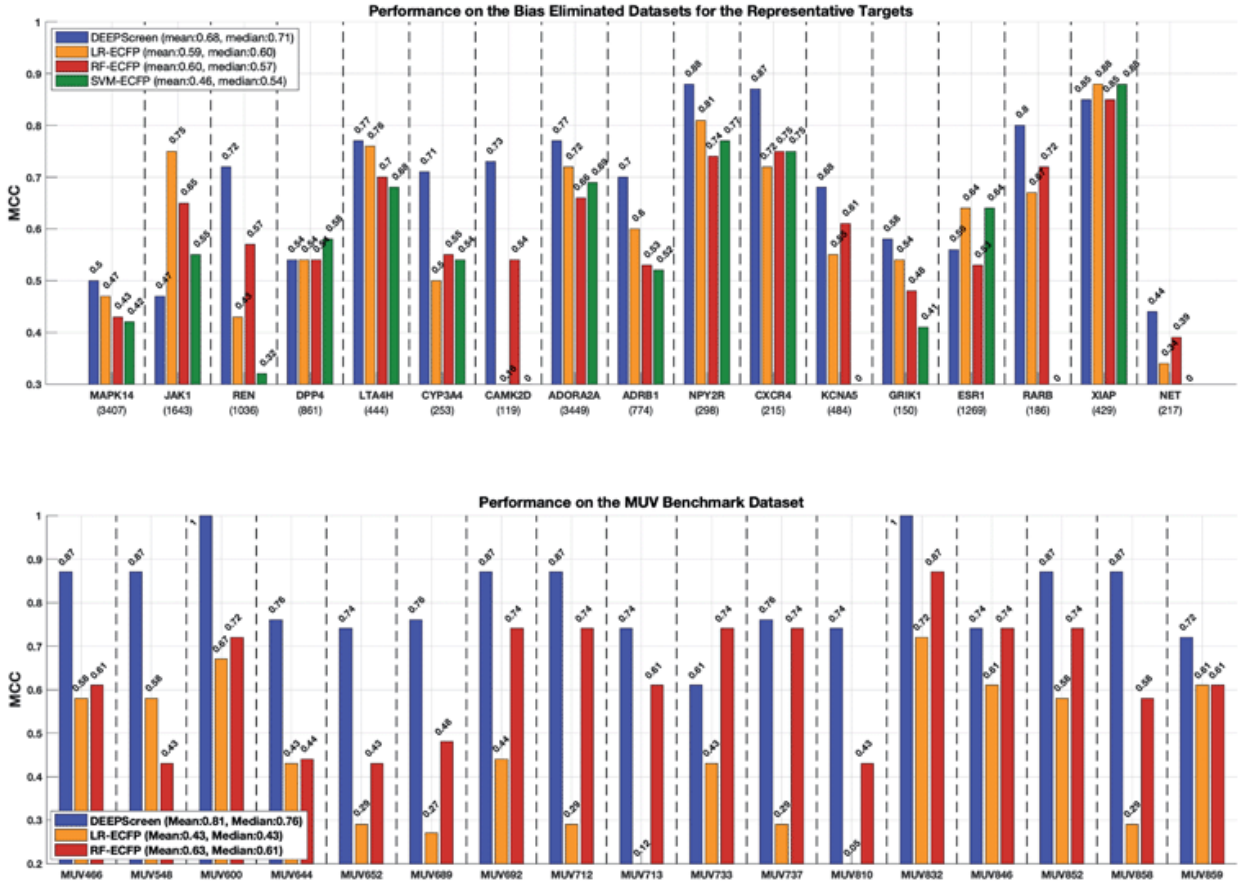


Slika 16. Proces DEEPScreen klasifikacije [6]

DEEPScreen sustavi su testirani na više od 700 ciljanih proteina, podvrgnuti rigoroznim testovima optimizacije hiperparametara i uspoređivani s najmodernijim metodama na više referentnih skupova podataka kao što je prikazano na slikama 17. i 18.. Učinkovitost ovog digitalnog alata može se istaknuti u tome da se nekoliko lijekova za rak, koji su dobiveni DEEPScreen metodom, trenutno testiraju in vitro na stanicama raka i pokazuju veliku mogućnost za uspjeh u otkrivanju novog lijeka.[6]



Slika 17. Usporedba DEEPScreen klasifikatora s drugim suvremenim metodama u primjeni ranog razvoja lijekova [6]

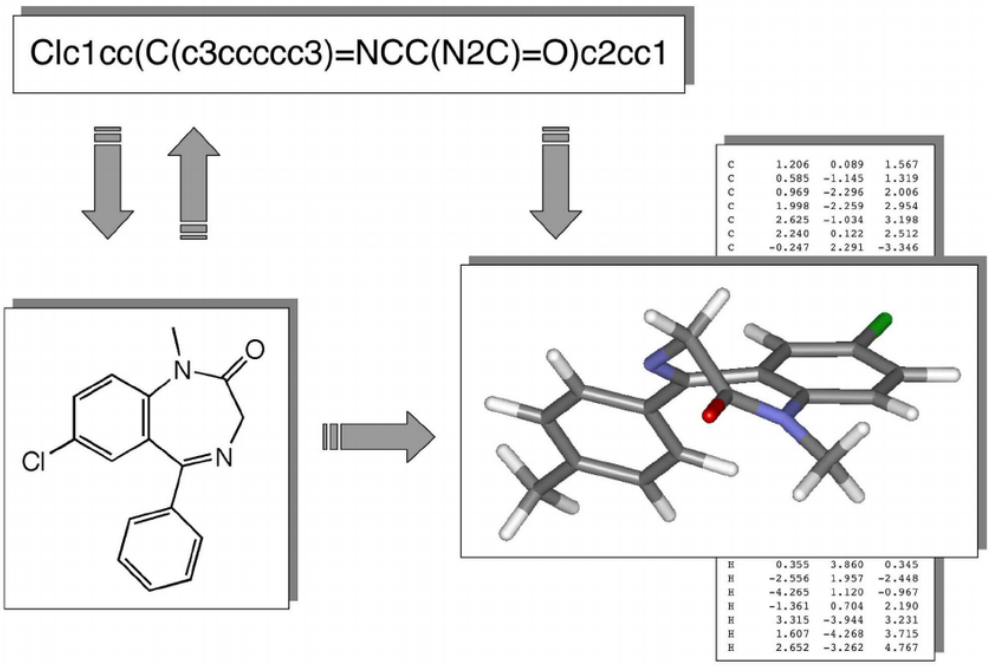


Slika 18. Usporedba rezultata efikasnosti benchmark testova DEEPScreen klasifikatora u odnosu na druge suvremene metode s primjenom u ranom razvoju lijekova [6]

DEEPScreen se također može koristiti i u područjima prenamjene lijekova za in silico screening kemogenomskog prostora. Ovim načinom uporabe DEEPScreen-a osiguravaju se nove količine DTI koje se mogu eksperimentalno ispitati.[6]

3.4. Tehničke specifikacije potrebne za rad (podaci)

U DEEPScreen sustavu podaci, to jest kemijski spojevi, su prikazani s pomoću 2D slika molekularne strukture određenog broja piksela. 2D prikazi molekula lako su dostupni u različitim kemijskim i bioaktivnim bazama podataka, no također ih je moguće generirati samostalno. Jedan od načina generiranja 2D slika molekularnih struktura je generiranje s pomoću pojednostavljenog sustava za unos molekularnih struktura linijskim nizom (SMILES) prikazan na slici 19..[6]



Slika 19. Pretvorba podataka potrebnih za DEEPScreen iz SMILES oblika u 2D slike [6]

SMILES je postupak prikazivanja kemijskih struktura na sažet i lako čitljiv način. Za pisanje SMILES-a koristi se jednostavan vokabular koji se sastoji od simbola atoma, veza i nekoliko gramatičkih pravila poput: velikim slovima su označeni simboli elemenata, malim aromatski atomi, brojevi za redoslijed atoma i simboli za oblik veze. Najveća prednost SMILES-a je da jednostavno predstavlja veze, može služiti za generiranje 2D struktura (npr. DEEPScreen) i zauzima manje memorijskog prostora. SMILES najčešće služi za pohranjivanje podataka koji će služiti za generiranje 2D strukture jer zauzima 50-70 % manje prostora od ekvivalentnih tablica povezanosti, pa čak i binarnih tablica povezanosti.[11]

Veličina generiranih 2D struktura je proizvoljna. Tijekom testiranja pokazano je da generiranje slika dimenzija 100 x 100 nije adekvatno za crtanje molekula, a slike dimenzija 400 x 400 su bile prevelike za treniranje zbog povećane složenosti. Također, prilikom istog testiranja, pri kojem je korištena jednaka količina CPU snage, povećanjem dimenzije sa 100 x 100 na 200 x 200 značajno se povećala točnost predikcije, za čak oko 17 %. Daljnjim povećanjem dimenzija generiranih slika, s 200 x 200 na 400 x 400, nije utvrđeno povećanje točnosti predikcije.

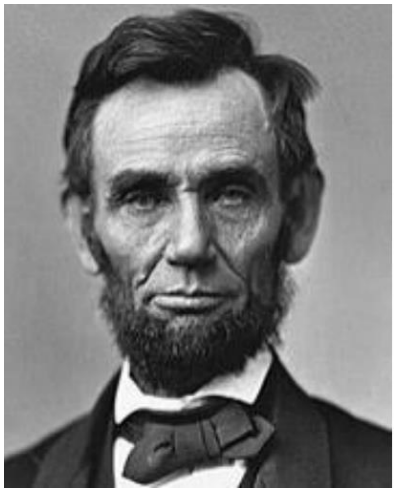
3.5. Praktična primjena DEEPScreen klasifikatora

Primjena DEEPScreen klasifikatora je razvitkom umjetne inteligencije, a s time i umjetnih neuronskih mreža, sve se više pojavljuje u sveopćoj industriji. Naime, svugdje gdje je moguće izvući značajke iz velikih količina podataka generiranih putem slika, ima prostora za primjenu ovakvih sustava. U današnjoj industriji ponajviše se uz razvoj lijekova koristi u auto-industriji za prepoznavanja objekata na cesti kod novijih modela automobila, koji teže autonomnom upravljanju, detektiranje grešaka na proizvodima u proizvodnim pogonima, detekciju lica razvijenih sigurnosnih sustava kod mobilnih i ostalih osobnih uređaja, sigurnosnih kamera i praćenja sumnjivih aktivnosti, u poljoprivredi za praćenje zdravlja usijava s pomoću slika dronova, te razvijanju medicinskih uređaja za pregledavanje i analizu medicinskih slika, snimaka i skenova.

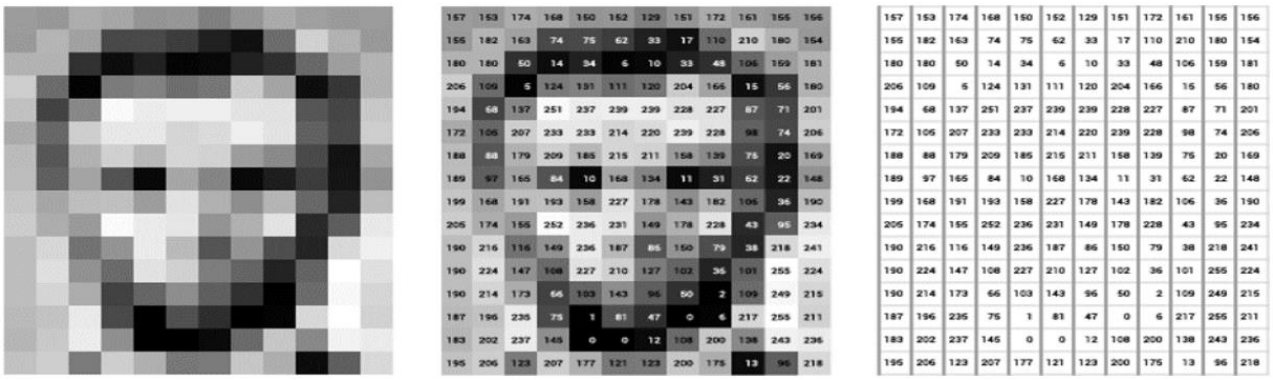
4. STROJNO UČENJE (TRENING)

4.1. Arhitektura dubokih konvolucijskih neuronskih mreža

Duboke konvolucijske neuronske mreže(DCNN) su skupina umjetnih neuronskih mreža koje se sastoje od izmjeničnih, konvolucijskih i pooling slojeva, te su specijalizirane za automatsko izdvajanje karakteristika slikovnih podataka kako je i prikazano na slikama 20. i 21.. DCNN su najnaprednija metoda u području obrade slike, a rade na principu korištenja malih prozora na ulazu vektora karakteristika, za fazu treniranja i testiranja, kao detektor karakteristika. Ove neuronske mreže, također, uče različite karakteristike iz ulaza zanemarujući njihov apsolutni položaj unutar ulaznog vektora.[6]

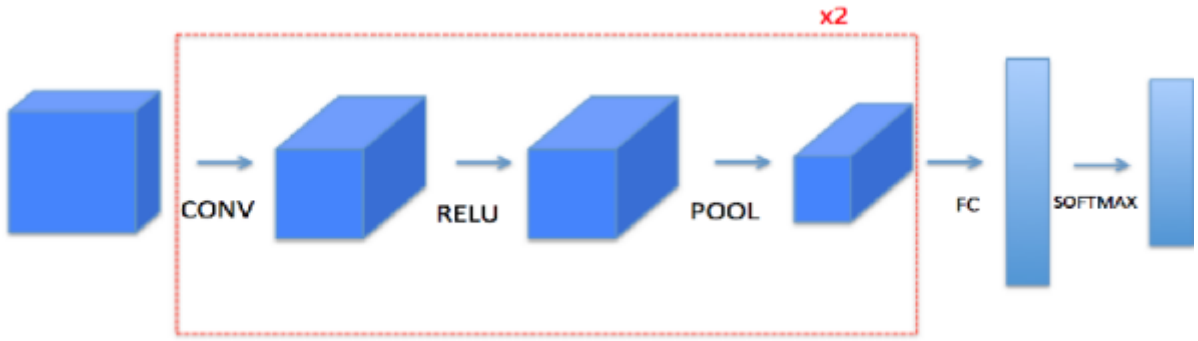


Slika 20. Slika Abrahama Lincolna [8]



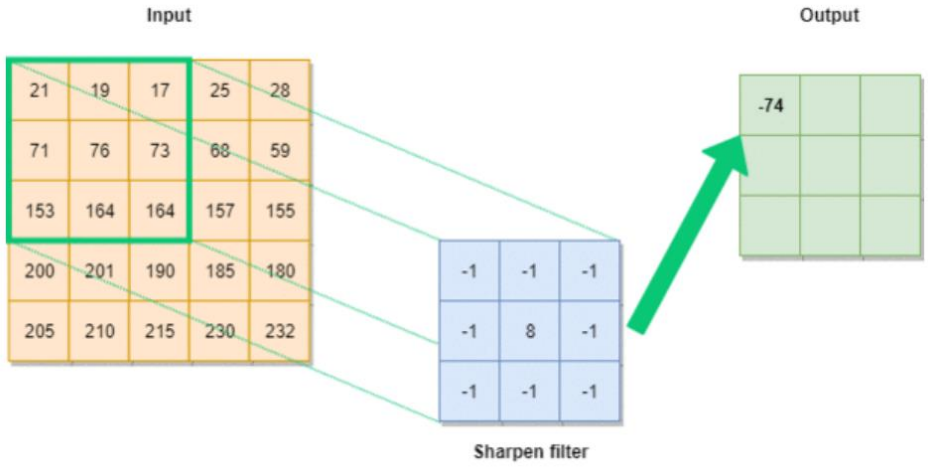
Slika 21. Digitalna analiza slike Abrahama Lincolna [8]

Glavne komponente koje svaka DCNN mora imati su ulazni i konvolucijski sloj, aktivacijska funkcija, sloj sažimanja i potpuno povezani sloj koji su prikazani na slici 22..[13]



Slika 22. Komponente umjetnih kovolucijskih neuronskih mreža [11]

Početkom učenja DCNN-a potrebno je unijeti ulazne podatke, to jest sliku koja se u ulaznom sloju preformulira tako da se zadrže vrijednosti piksela u obliku matrice vrijednosti. U konvolucijskom sloju s pomoću filtera značajki detektiraju se značajke iz izvorne slike. Filteri značajki su ranije definirane matrice koje su opisane težinom koja je definirana njihovom dimenzijom. Filteri klize preko matrice, izdvajaju značajke i na temelju tih rezultata stvaraju kartu značajki. Pri cijelom tom postupku koji se nalazi na slici 23., filter čuva prostorne odnose između piksela.[8]

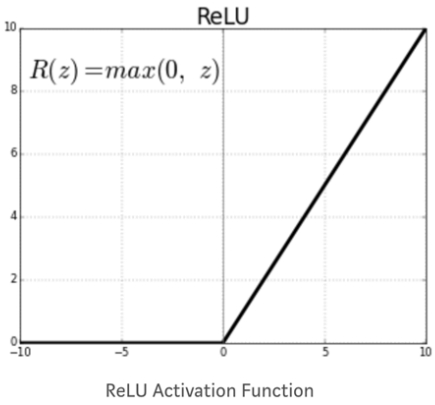


Slika 23. Upotreba filtera značajki i stvaranje karte značajki [9]

Aktivacijska funkcija ima ulogu dodavanja nelinearnosti u skrivene slojeve mreže. Najčešće se koristi ReLU funkcija koja djeluje pozitivno na uklanjanje zasićenosti gradijenta za pozitivne ulazne vrijednosti.[8]

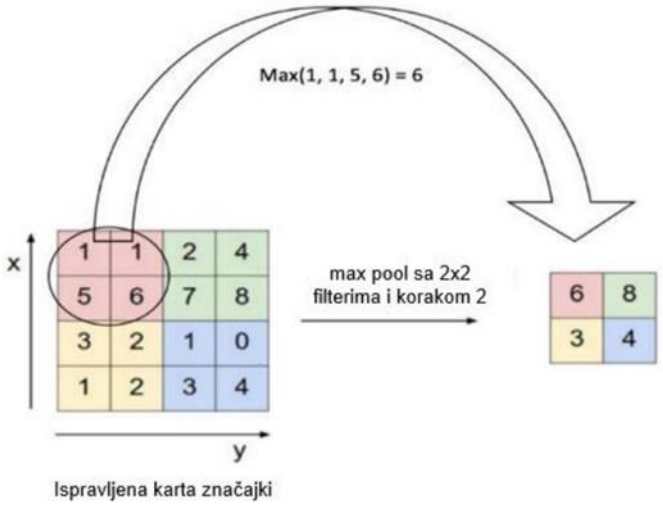
$$F(x) = \max(0, x)$$

Također, ReLU funkcija ne mijenja dimenzije ulaznog volumena. (Slika 24.)[9]



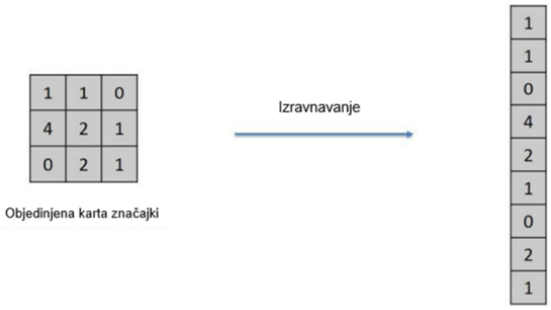
Slika 24. Primjer ReLU funkcije

Nadalje, u sloju sažimanja smanjuju se dimenzije visine i širine vrijednosti karte značajki nakon ispravljanja ReLU funkcijom, pri čemu dubina ostaje sačuvana.(Slika 25.)[8]



Slika 25. Rad sloja sažimanja [8]

Metodom izravnavanja sve vrijednosti sačuvane u obliku matrice prebacuju se u vektorski oblik kako bi se pojednostavio daljnji proces učenja. U konačnici potpuno povezani sloj, čiji su neuroni u potpunosti povezani s neuronima prethodnog sloja, izbacuje rezultat u obliku vektora s konačnim vrijednostima. (Slika 26.)[8]



Slika 26. Proces izravnavanja [8]

U glavnom dijelu ovog završnog rada, strojno učenje u svrhu razvoja lijekova bit će obavljeno s pomoću umjetne konvolucijske neuronske mreže arhitekture kao što je detaljno prikazano u ovom poglavlju. Neuronska mreža korištena za ovaj rad razvijena je s pomoću PyTorch biblioteke, models. Osim ubacivanja biblioteke, kod još sadrži definiranje s pomoću nn.modela koji je osnova za sve modele u PyTorchu, konvolacijskih mreža i Batch normalizacijskih slojeva, pooling i potpuno povezanih slojeva, te forward metode kojom je definiran protok podataka kroz mrežu. Također, posljednji dio koda, forward metoda, sadrži Dropout regularizaciju koja smanjuje prenaučenost, čineći model robusnijim. Cijeli kod se nalazi u Prilozima.

4.2. Prikupljanje i primjena podataka

Podatci su sirov, neobrađena činjenica za razumijevanje, analizu i donošenje odluke. Nemaju vrijednost, sve do trenutka donošenja odluke, kada postaju osnovni gradivni element koji se prikuplja, pohranjuje, obrađuje i analizira radi otkrivanja određenog uzorka. Na temelju pohranjenih podataka može se modelirati fenomen, donositi predikcije i izvlačiti korisni uvidi.[9]

Podaci su jedan od najvažnijih dijelova strojnog učenja, jer upravo su oni izvor svog znanja na temelju kojeg će se optimizirati model, stoga je potrebno detaljno pripremiti podatke, filtrirati da ostanu samo oni najpotrebniji i najkorisniji, prikupiti određenu količinu koja će zadovoljiti potrebe učenja i u konačnici imati pravilan oblik podataka. DEEPScreen klasifikator, kao što je već rečeno u poglavlju 3.4., za svoje učenje koristi podatke u 2D slikovnom obliku, te će svi naši podaci korišteni za treniranje modela za razvoj lijekova u ovom radu biti tog oblika.

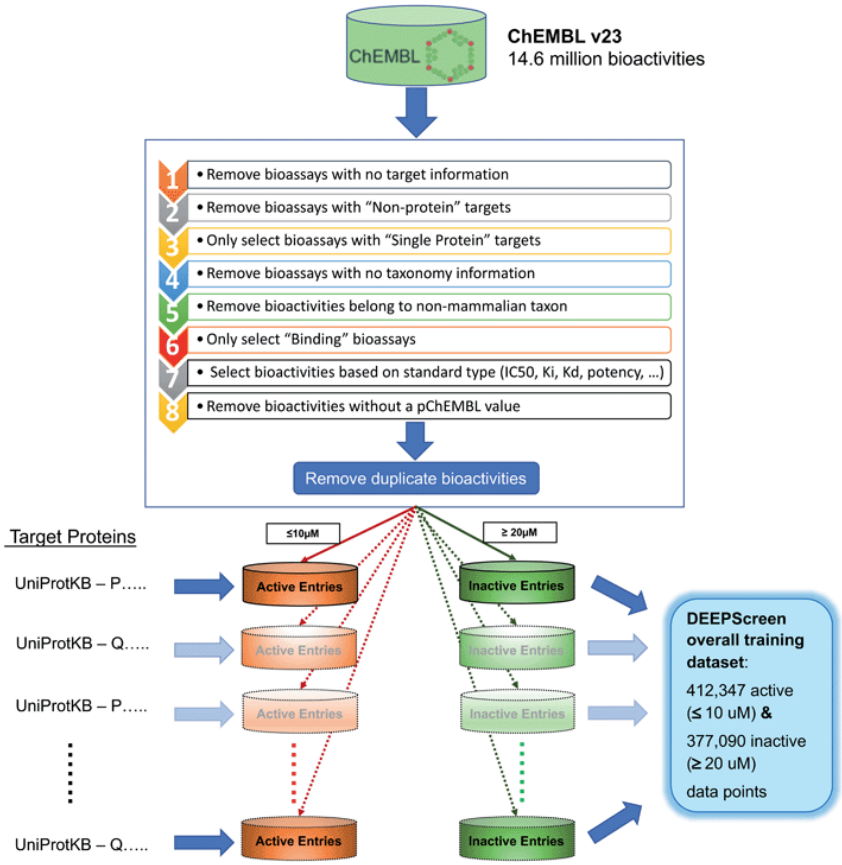
Podaci korišteni u ovom završnom radu preuzeti su iz baze podataka ChEMBL v23 gdje se već nalaze svi potrebni podaci za treniranje DEEPScreen-a. Na ovaj skup podataka primjenjuju se

nekoliko filtera i dodatnih obrada koji su sveli ukupan broj podataka, 14 675 320, na broj od 412 347 aktivnih podataka i 377 090 neaktivnih podataka. Obradom svih ovih podataka i detaljnim filtriranjem, dobiven je pouzdan skup podataka, koji predstavlja standard za testiranje u budućim DTI istraživanjima za 704 ciljana proteina.

Svi podatci su podijeljeni na trening, validacijske i testne dijelove kako bi se trenirali, optimizirali i evaluirali prediktivni model. Što se tiče same podjele podataka, one mogu biti na dva načina. Prvi način je nasumična podjela podataka gdje se podaci razdvajaju nasumično i bez posebnog razmatranja. Ovakav način rada je dobar pokazatelj performansi modela u predviđanju novih poveznica sličnih struktura. Drugi, koji ima širu upotrebu, je dijeljenje podataka u skupine na temelju sličnosti, pri čemu se razdvajaju slični podaci kako ne bi došlo do pojave sličnih podataka u skupu za trening i treniranje. [8] Kako bi se to spriječilo upravo se provode filtracije i obrada podataka koje smo ranije naveli. Podaci koji će biti korišteni u ovom radu filtrirani su prema vrsti cilja, taksonomiji, vrsti testa i standardnoj vrsti.

Upravo sličnost podataka je jedan od dva najveća problema prilikom razvoja ovakvog virtualnog screening sustava, što može stvoriti relativno trivijalno treniranje i testiranje modela, pa se preporučuje detaljna obrada podataka. Drugi problem, koji ugrožava naše učenje, je pojava negativne selekcijske pristranosti. Ovaj problem stvara međusobna sličnost neaktivnih podataka koja je potpuno nepovezana s njihovim svojstvima povezivanja. Ovaj problem je riješen tako da su se podaci podijelili na temelju bio-aktivnosti. Nakon obavljenih treninga svi oni s bioaktivnosti ispod i uključujući 10μM su proglašeni aktivni, a oni s iznad i uključujući 20μM neaktivni. Nakon podjele podataka namješten je sličan broj aktivnih i neaktivnih podataka kako ne bi došlo do pretreniranja sustava.

Sva ova obrada podataka, poput filtriranja, stvaranje skupova za treniranje, testiranje i validaciju, obrada bio-aktivnosti i pretvaranja iz SMILES-a u 2D su obavljeni s pomoću data_processing koda koji se može pronaći u poglavlju Prilozi, a način filtriranja je prikazan na slici 27..



Slika 27. Priprema podataka za kvalitetnu DEEPScreen klasifikaciju [6]

4.3. Implementacija DEEPScreen klasifikatora

Ovaj DEEPScreen klasifikator modeliran je u Phytonu s pomoću dostupnih biblioteka, naredbi na temelju već ranije objašnjene logike. Model se sastoji od pet kodova od kojih su dva, data_processing i models, već spomenuta tijekom ovog rada u kontekstu arhitekture umjetne konvolucijske neuronske mreže, to jest obrade podataka. U ovom poglavlju biti će ukratko obrađena i objašnjena građa i svrha svih korištenih programskih kodova koji se mogu pronaći na kraju završnog rada u poglavlju Prilozi.

4.3.1. data_processing

data_processing je kod, kao što je već napomenuto u poglavlju 4.2., koji služi za detaljnu pripremu podataka. Za ovaj proces koristi se mnoštvo biblioteka od kojih su važnije os, sys za rad sa sustavnim operacijama i sustavom datoteka, OpenCV za obradu slike, torch za rad s dubokim učenjem, numpy za numeričke podatke, pandas za rad podacima u tablicama i manje bitne poput json, random, warnings i subprocess. Sve ove biblioteke čine jedan sveobuhvatan sustav za obradu podataka. Ovaj kod s pomoću RDKit-a generira podatke iz SMILES oblika u 2D slike koje sprema u PNG oblik, definira optimalnu veličinu slike, obrađuje podatke o bioaktivnosti i sprema ih u TSV oblik, dijeli aktivne i neaktivne spojeve, obogaćuje podatke neaktivnih spojeva za bolje rezultate, dijeli podatke za trening, testiranje i vrednovanje, te učitava listu ciljanih proteina.

4.3.2. evaluation_metrics

Metrika performansi modela klasifikacije izvršava se s pomoću evaluation_metrics koda kojim zapravo evaluiramo rezultate samog treninga. Ovim kodom dobivamo rezultate poput preciznosti, odziva, F1-scorea, točnosti i Matthewsovog korelacijskog koeficijenta, koje izračunava s pomoću scikit-learn funkcije. Također, kod u svom radu osim izračunavanja metrike obuhvaća konverziju rezultata ako je potrebno, provjeravanje duljine i ispisivanje vrijednosti rezultata s pomoću get_list_of_scores funkcije. Biblioteke koje koristi su sklearn, biblioteka koja pruža jednostavan alat za regresiju, analizu podataka i modeliranje sustava, te numpy.

4.3.3. main_training

main_training je kod s pomoću kojeg se definiraju osnovni potrebni podaci za izvođenje treninga poput toga koja će ZIP datoteka služiti kao izvor podataka, koliko je neurona u određenom sloju, koji je naziv ispitivanja, broj epoha, stopa učenja, stopa ispadanja i naziv modela. S pomoću ovog koda može se mijenjati način učenja i bilježiti rezultati za usporedbu s drukčijim načinom

učenja, s drugim brojem epoha, stopom učenja ili stopom ispadanja. Također, ovaj kod služi za pokretanje samog učenja, to jest pokretanjem ove skripte započet će i učenje. U svom radu koristi biblioteku argparse koja pokreće funkciju za obuku modela strojnog učenja.

4.3.4. models

models je kod kojim je opisana arhitektura umjetne neuronske konvolucijske mreže. Bazira se na učenju s pomoću značajki koje skuplja sa slike po objašnjenjima iz poglavlja 4.1. Također, kod je već spomenut i način njegova rada objašnjen u poglavlju 4.1.

4.3.5. train_deepscreen

train_deepscreen je kod koji je osnova funkcioniranja ovog modela. Ovaj kod omogućava treniranje, testiranje i validaciju podataka tako što ispituje dostupnost GPU-a ili CPU-a, te dostupan uređaj iskorištava kao stroj za strojno učenje. Kod sadrži veliku količinu biblioteka poput numpy, torch, pandas cv2, i drugih s pomoću kojih procjenjuje performanse modela, obrađuje podatke, sprema najbolje rezultate i predikcije, kreira direktorije i sprema stanje modela i predikcije. Funkcija train_validation_test_training je glavni dio koda za obavljanje cijelog postupka treninga i evaluaciju modela. Za svaku epohu trenira, testira i vrednuje rad modela pri čemu sprema najbolje rezultate, to jest performanse.

4.4. Evaluacija modela

Kako bismo u potpunosti bili upoznati s performansama našeg DEEPScreen sustava potrebno je priložiti rezultate u nekom standardnom obliku kako bi se rezultati mogli usporediti s drugim razvijenim modelima. Kao što smo već spomenuli u poglavlju 4.3.2. kod evaluation_metrics će biti zadužen za bilježenje i računanje rezultata. Metrike u kojima će biti prikazani rezultati su preciznost, odziv, F1-score, točnost i Matthewsov korelacijski koeficijent, odnosno najčešći oblik koji se pojavljuje u strojnom učenju.

Preciznost je metrika koja nam govori o odnosu ispravno razvrstanih pozitivnih vrijednosti podijeljenih sa zbrojem ispravno razvrstanih pozitivnih i netočno razvrstanih pozitivnih slučajeva. Pokazuje koliko je slučajeva model ispravno odredio kao točne. Vrijednosti se nalaze u rasponu [0,1], a cilj je biti što bliže broju 1, što je vrijednost savršeno napravljenog modela.[6]

$$Preciznost = \frac{TP}{TP + FP}$$

Odziv nam govori o omjeru ispravno razvrstanih pozitivnih vrijednosti podijeljenih sa zbrojem ispravno razvrstanih pozitivnih i pogrešno razvrstanih negativnih vrijednosti. Ova metrika ukazuje na to koliko je model točno odredio pozitivne vrijednosti u odnosu na ukupan broj točnih vrijednosti. Također, raspon vrijednosti je [0,1] i idealna vrijednost je 1.[6]

$$Odziv = \frac{TP}{TP + FN}$$

Harmonijska sredina preciznosti i odziva je izražena s pomoću F1-score metrike koja je izražena u postotcima. Ovom metrikom ukazuje se izravna kvaliteta klasifikatora, odnosno što je postotak veći to je klasifikator bolje modeliran.[8]

$$F1 - score = \frac{2 \times Preciznost \times Odziv}{Preciznost + Odziv}$$

Točnost je metrika kojom se iskazuje omjer ispravno razvrstanih primjera, pri čemu se misli i na pozitivne i negativne, te ukupnog broja slučajeva. Vrijednosti su također u rasponu [0,1], i teži se broju 1.[6]

$$Točnost = \frac{TP + TN}{TP + TN + FP + FN}$$

Matthewsov korelacijski koeficijent ili MCC je jedina metrika čije se vrijednosti nalaze u rasponu [-1,1]. Njome se izražava savršenost odnosno nesavršenost klasifikacije modela. 1 označava savršenu, 0 slučajnu, a -1 nesavršenu klasifikaciju.[8]

$$MCC = \frac{TP \times TN + FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

Svaka od ovih metrika je korisna, i osnovni cilj naših modela je da dođu što bliže idealnoj vrijednosti, no ne i na nju. Naime, ostvarivanje savršenih rezultata u evaluaciji nekog od modela, najčešće znači da je model dosta plitak i ne razvijen, da skupina podataka nije dobro pripremljena, velika ili raznolika, to jest da postoji dosta prostora za nadogradnju odnosno poboljšanje modela. Također, ni niski odnosno ne savršeni rezultati nisu poželjni jer oni pokazuju da model nije dovoljno dobar za rad i potrebna je dorada. Kao metrike evaluacije rezultata strojnog učenja mogu se još koristiti osjetljivost i specifičnost, no oni su izostavljeni iz ovog rada.

5. REZULTATI

Tijekom ovog završnog rada provedeno je učenje na nekoliko različitih skupina podataka kako bi se ocijenio rada modela prema ranije navedenim metrikama. Prilikom učenja, također su se, osim podatka mijenjali brojevi epoha kako bi došli do zaključka koji broj epoha bi bio optimalan za provođenje ovaj oblik strojnog učenja s ovim razvijenim modelom.

5.1. Analiza rezultata

5.1.1. Rezultati prilikom promjene skupine podataka

Prilikom svakog strojnog učenja, osim pripreme i podjele podataka na one za trening, testiranje i vrednovanje, potrebno je mijenjati skupove podataka. Prema preporukama stručnjaka koji su provodili testove strojnog učenja savjet je da se koristi velika količina podataka kako bi se mogli koristiti potpuno novi podaci za svaki novi pokrenuti ciklus, ili ako ih nema dovoljno da se kombiniraju skupovi podataka međusobno kako bi imali nove nekorištene skupove.

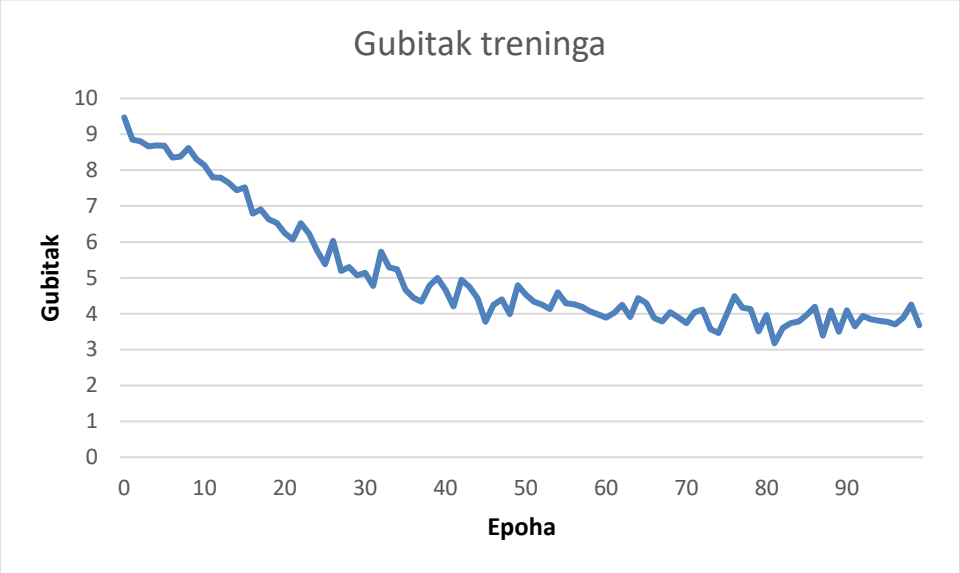
U ovom završnom radu korišteni su već ranije pripremljeni skupovi podataka iz ranije navedenih izvora kojih je bilo dosta, te nije bilo potrebe za kombiniranjem istih. Strojno učenje je provedeno i vrednovano na 4 različite skupine podataka kroz 100 epoha. Cilj ovakvog treninga bio je uvid u način ponašanja modela prilikom promjene skupine podataka uz identične parametare modela.

Prva grupa podataka na kojim je prvi put ispitan napravljeni model je grupa pod nazivom ChEMBL 202. Ona se sastoji od jednostavnijih oblika kemijskih spojeva prikazanih s pomoću 2D slika, što ju čini najboljim izborom za prvi test.

```
Test Precision: 0.6568627450980392
Test Recall:    0.8589743589743589
Test F1-Score:  0.7444444444444445
Test Accuracy:  0.6461538461538462
Test MCC:       0.22153547011515104
Test TP:        67
Test FP:        35
Test TN:        17
Test FN:        11
```

Slika 28. Rezultati 1. testa

Rezultati strojnog učenja na slici 28. prikazuju relativno visoke vrijednosti metrika vrednovanja, tako da preciznost iznosi oko 66 %, odziv visokih 85 %, F1-Score 75 %, točnost 65 % i MCC 22 %.



Slika 29. Gubitci tokom 1. treninga

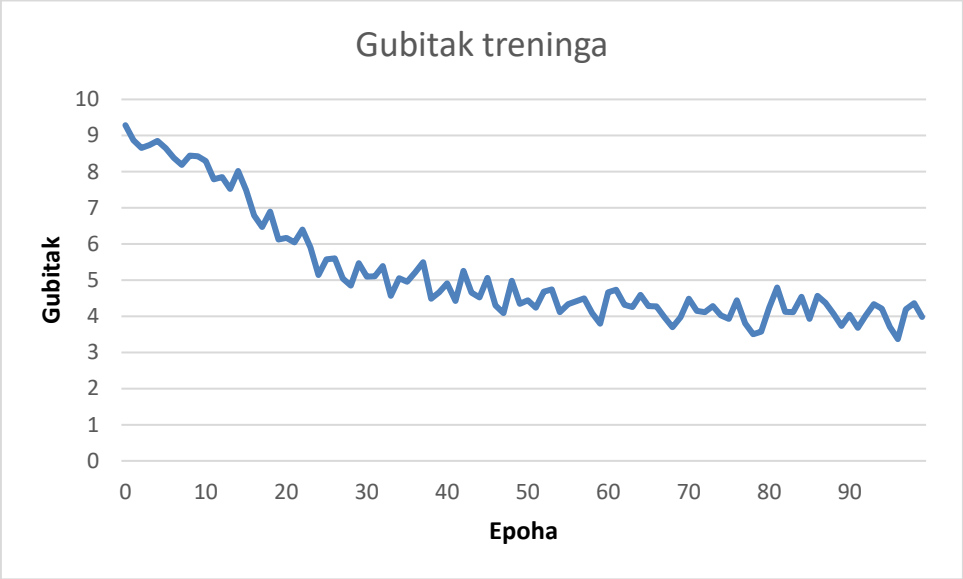
Grafom na slici 29. prikazano je kako su se postupno smanjivali gubitci predviđanja strojnog učenja. U početku se to učenje drastično smanjivalo, no približavanjem kraju učenja ustalilo se oko vrijednosti 3,65 do 4,2.

Druga skupina na kojima je provedeno testiranje je grupa podataka pod nazivom ChEMBL 203. Ona se sastoji od većeg broja jednako složenih kemijskih spojeva na kojoj će se provesti strojno učenje, također kroz 100 epoha.

```
Test Precision: 0.9039301310043668
Test Recall:    0.9282511210762332
Test F1-Score:  0.915929203539823
Test Accuracy:  0.8978494623655914
Test MCC:       0.7863002548034017
Test TP:        207
Test FP:        22
Test TN:        127
Test FN:        16
```

Slika 30. Rezultati 2. testa

Rezultati na slici 30., nakon promjene grupe podataka ukazuju na poboljšan rad modela svojim visokim vrijednostima. Preciznost, odziv i F1-Score nalaze se blago iznad 90 %, dok je točnost blago ispod 90 %. MCC iznosi skoro 79 % što je znatno više nego u prethodnom slučaju.



Slika 31. Gubitci tokom 2. treninga

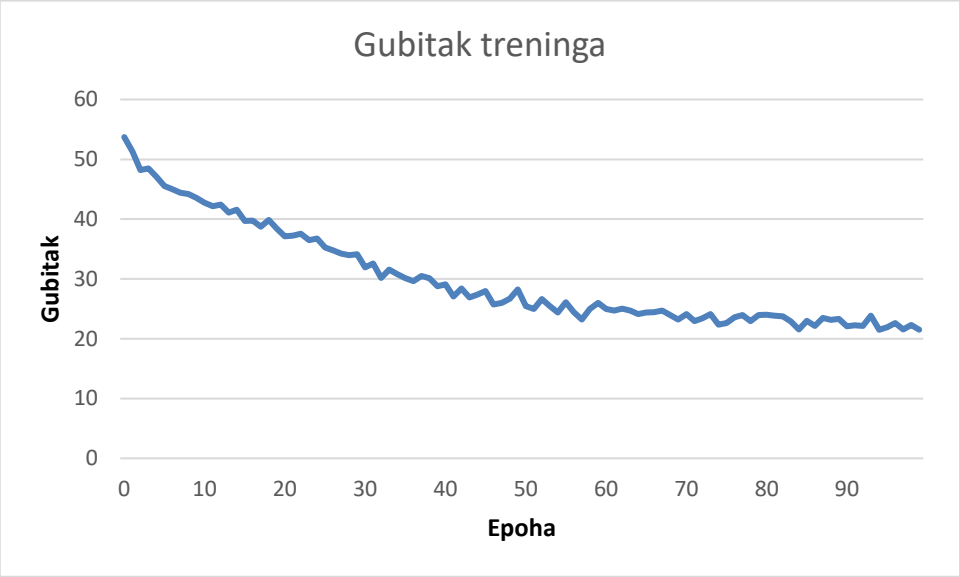
Graf gubitaka na slici 31. ukazuje da je model odmah na prvoj epohi radio manje greške prilikom predviđanja. Također u konačnici se ustalilo na vrijednostima nižim od onih u prethodnom slučaju, to jest 3,5 do 4.

Kako bismo dodatno provjerili rad svog modela, sljedeća skupina podataka koja je korištena prilikom treninga, ChEMBL 204, posjeduje veću količinu podataka koji su pritom složeniji kemijski spojevi za koje je potrebno više vremena za obradu.

```
Test Precision: 0.7564575645756457
Test Recall:    0.8779443254817987
Test F1-Score:  0.8126858275520317
Test Accuracy:  0.7570694087403599
Test MCC:       0.48324460060836744
Test TP:        410
Test FP:        132
Test TN:        179
Test FN:        57
```

Slika 32. Rezultati 3. testa

Dobiveni rezultati prikazani na slici 32. ukazuju da je kvaliteta rada modela blago opala u odnosu na prethodne treninge, no to je i bilo očekivano zbog složenijih kemijskih spojeva. Preciznost je oko 75 %, odziv oko 88 %, F1-Score 81 %, točnost 76 % i MCC oko 48 %. Također na smanjenje nekih od metrika utjecala je i znatno veća količina podataka.



Slika 33. Gubitci nakon 3. treninga

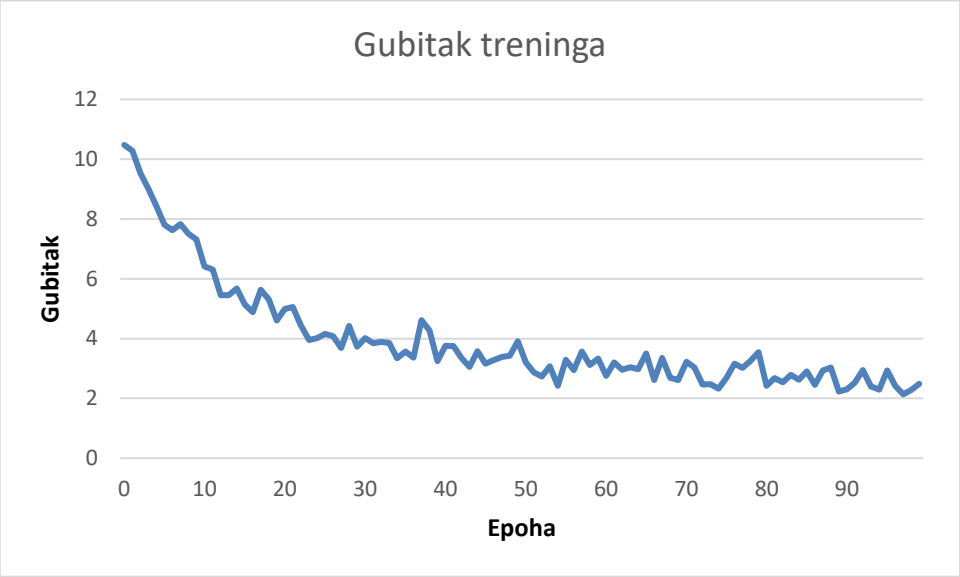
U priloženom grafu na slici 33. može se primijetiti da su gubici bili izrazito visoki prilikom ovog treninga, toliko da su u samom početku prelazili i vrijednost od 50. Daljnjim treningom gubici su se smanjivali i u konačnici ustalili na vrijednost od oko 22.

Kako bismo uvidjeli hoće li naš model napredovati i poboljšavati svoje performanse ako mu nastavimo davati kompleksnije spojeve obavili smo još jedan trening. Ovaj put je u pitanju skupina podataka ChEMBL 205 koja također sadrži kompliciranije kemijske spojeve, no u manjoj količini.

```
Test Precision: 0.8631578947368421
Test Recall:    0.9111111111111111
Test F1-Score:  0.8864864864864865
Test Accuracy:  0.8590604026845637
Test MCC:       0.7027864570656595
Test TP:        82
Test FP:        13
Test TN:        46
Test FN:        8
```

Slika 34. Rezultati 4. testa

Smanjenjem količine kompliciranijih kemijskih spojeva ukazuje se na nešto poboljšano učenje. Prema slici 34., preciznost je porasla za gotovo 10 %, odziv i F1-Score za oko 5-7 %, točnost za oko 10 % i MCC za oko 30 %.



Slika 35. Gubitci tokom 4. treninga

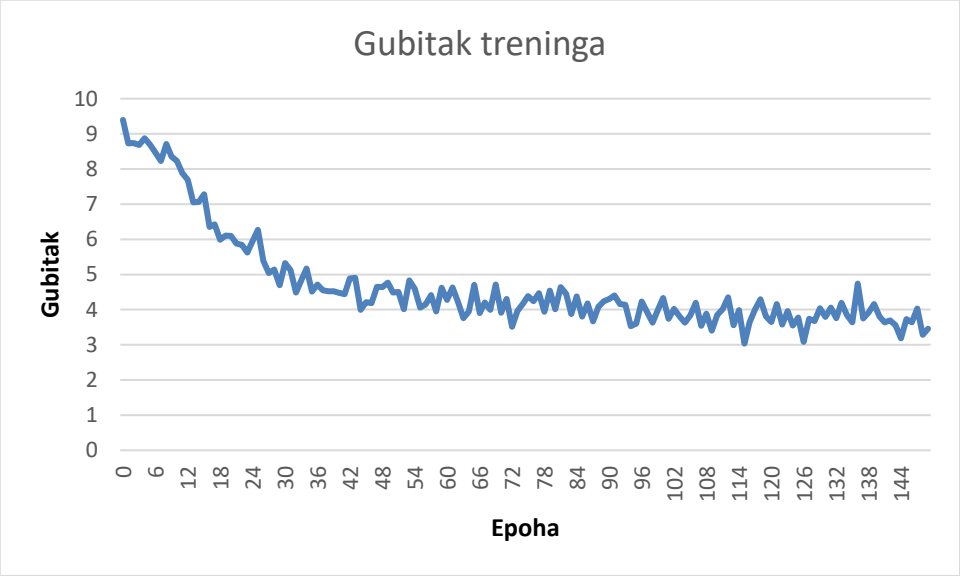
U gornjem grafu na slici 35. možemo vidjeti da su gubitci prilikom treninga znatno smanjeni, čak i oni početni. Značajno smanjenje količine podataka utjecalo je na izrazito veliko smanjenje gubitaka prilikom treninga, posebno konačno ustaljenih vrijednosti koje se vrte oko 2,5.

5.1.2. Rezultati prilikom promjene broja epoha

Osim promjene količine i vrste samih podataka, za bolji uvid u rad našeg modela, može se i pokušati promijeniti broj epoha. Promjenom broja epoha možemo uvidjeti hoće li se gubici prilikom treninga nastaviti smanjivati i kakve su promjene kod metrika vrednovanja modela. Mijenjanje broja epoha i promatranje u svrhu zapažanja promjena bit će obavljeno na istoj skupini podataka radi precizne usporedbe. Skupina podataka koja će biti korištena tijekom ovog vrednovanja je skupina ChEMBL 202 koja je korištena na početku i čije rezultate treninga sa 100 epoha već su prikazani u prethodnom poglavlju.

```
Test Precision: 0.8225806451612904
Test Recall:    0.6538461538461539
Test F1-Score:  0.7285714285714285
Test Accuracy:  0.7076923076923077
Test MCC:       0.43383357978902654
Test TP:        51
Test FP:        11
Test TN:        41
Test FN:        27
```

Slika 36. Rezultati ponovljenog 1. testa s 150 epoha

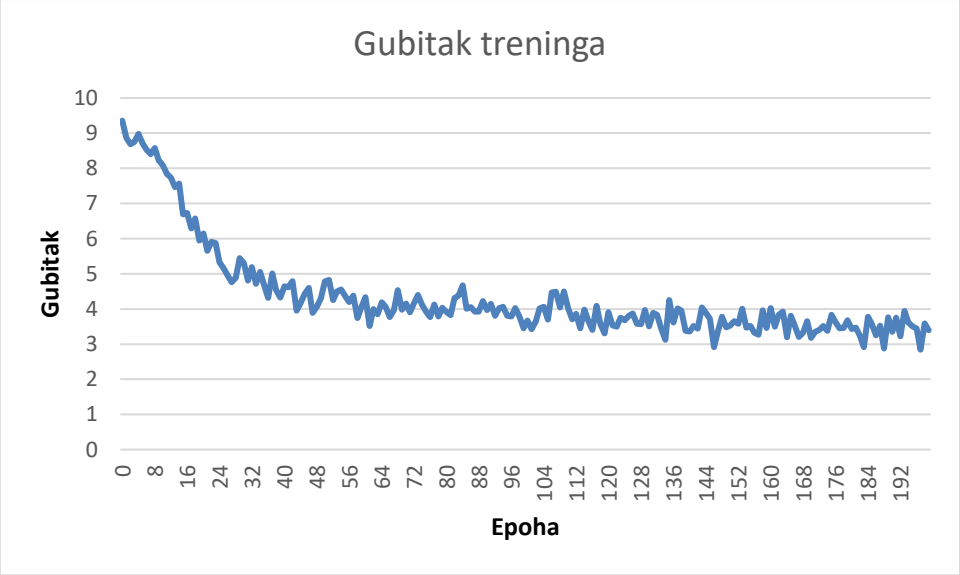


Slika 37. Gubitci tokom ponovljenog 1. testa s 150 epoha

Nakon promjene broja epoha 1. testa sa 100 na 150, prema rezultatima sa slika 36. i 37., primjećuje se da se gubici tijekom treninga ne smanjuju nego se vrte oko sličnih vrijednosti kao i nakon provedenih 100 epoha. Također vrijednosti preciznosti, odziva, F1-score i MCC su se snizile u odnosu na one koje smo dobili nakon treninga sa 100 epoha, što upućuje na to da daljnje promjene vrlo vjerojatno neće značajno utjecati na smanjenje gubitaka, a povećavat će uspješnost rada modela.

```
Test Precision: 0.7
Test Recall:    0.8076923076923077
Test F1-Score:  0.75
Test Accuracy:  0.676923076923077
Test MCC:       0.30618621784789724
Test TP:        63
Test FP:        27
Test TN:        25
Test FN:        15
```

Slika 38. Rezultati ponovljenog 1. testa s 200 epoha



Slika 39. Gubitci tokom ponovljenog 1. testa s 200 epoha

Radi dodatne provjere zaključka kako povećanje broja epoha neće dovesti do značajnih promjena i da će se pojaviti pretreniranost modela, odrađen je još jedan trening. Taj dodatni trening imao je sveukupno 200 epoha, a njegovi rezultati, prikazani na slikama 38. i 39., bili su kao što se i pretpostavljalo. Naime, broj gubitaka se nije smanjivao nego je i dalje varirao oko vrijednosti dobivene u treningu sa 100 epoha, dok se uspješnost modela nastavila smanjivati.

5.2. Diskusija

Rezultati provedenih DEEPScreen klasifikacija s pomoću napravljenog modela možda nisu odmah na prvim testovima pokazali izrazito visoke vrijednosti vrednovanja efikasnosti modela, no pokazano je postupno povećanje istih tijekom provođenja dodatnih testova. Naime, model je pokazao da obavlja zadatka i da optimizira svoje težinske funkcije kako bi dao što bolje rezultate. Također, promjenom kompleksnosti podataka pokazano je da će model reagirati na njih nešto lošije, ali provođenjem dodatne količine treninga te će se vrijednosti efikasnosti postepeno poboljšavati. Promjenom broja epoha primijećeno da se nakon 100 epoha količina gubitaka ne mijenja značajno, dok se validacijske vrijednosti smanjuju što ukazuje na pretreniranost. Tim podatkom došlo se do zaključka da je potrebno dobro ispitati rad modela kako bi se našao optimalni iznos broja epoha. Također ovaj problem se može riješiti i nadogradnjom algoritma s early stopping funkcijom kojom model sam shvaća kada dođe do pretreniranosti nakon čega zaustavlja zadatak.

6. ZAKLJUČAK

Razvoj lijekova je vrlo profitabilno, ali i izuzetno osjetljivo područje. Kao industrija čije proizvode konzumiraju ljudi i koja ima izrazito visoku koncentraciju kemijski jakih spojeva nosi veliku odgovornost. Kao što je već tijekom ovog rada rečeno, razvijanjem ljudskih života potrebno je relativno brzo i sigurno razvijati lijekove koji će pridonijeti poboljšanju ljudskih života. Korištenje strojnog učenja kao zamjena za testiranje lijekova na životinjama pri ranom razvoju lijekova, osim što je etički bolje rješenje, značajno ubrzava razvoj lijekova preskakanjem onih toksičnih. Kroz ovaj završni rad pokazano je upravo jedna od tih metoda strojnog učenja, DEEPScreen. U ovom radu ukazano je na sve razloge potrebe njegova korištenja, njegove prednosti i mane, usporedbu s drugim metodama, način rada i područja primjene. Razvijenim modelom u Pythonu, na kojem je obavljeno nekoliko učenja, pri čemu je pokazana praktična primjena, pri kojoj su dobiveni zadovoljavajući rezultati validacijskih vrijednosti. Rezultati i ponašanje samog modela bili su uspješni, odnosno pokazali su ispravnost rada modela. Završni rad je zbog svoje kompleksnosti sveden na osnove, te ima jako puno prostora za unapređenja koja su u konačnici spomenuta, poput uvođenja kompleksnijih spojeva i early stoppinga koji bi mogli poboljšati rad modela. Umjetna inteligencija, odnosno strojno učenje je područje koje potrebno u suvremenom svijetu. Industrija postaje sve podložnija njemu, jer umjetna inteligencija ima izrazito široku primjenu što omogućuje strahoviti razvoj svih industrija. Područje umjetne inteligencije tek je otvoreno, a već je duboko zagaženo u njega što nam omogućava pregršt kvalitetnih izvora i dobru bazu za daljnji napredak koji će se jednostavno morati dogoditi.

LITERATURA

[1] dr. sc. Romančík Martin, dr. med., magistar javnog zdravstva, „Klinička ispitivanja“ https://www.janssenwithme.hr/hr-hr/bolesti/rak-prostate/podrska/klinicka-ispitivanja , 2.8.2024.

[2] Shalini S. Lynch, PharmD „Razvoj lijekova“ https://www.hemed.hr/Default.aspx?sid=12072 , 2.8.2024

[3] Domagoj Lasić „Primjena dubokih neuronskih mreža za prenamjenu lijekova“ https://repozitorij.mef.unizg.hr/islandora/object/mef:3071/datastream/PDF/download , 2.8.2024

[4] Izvještaj Kongresnog ureda za proračun SAD-a https://www.cbo.gov/publication/57126 , 3.8.2024

[5] Članak 20200624STO81911 Europskog parlamenta „Dobrobit i zaštita životinja: Zakonodavstvo EU-a“ https://www.europarl.europa.eu/pdfs/news/expert/2020/8/story/20200624STO81911/20200624STO81911_hr.pdf , 3.8.2024.

[6] AS Rifaioglu, E Nalbat, V Atalay, MJ Martin, R Cetin Atalay, T Doğan, “DEEPScreen: high performance drug–target interaction prediction with convolutional neural networks using 2-D structural compound representations”, Chemical Science, 11:2531–2557, 2020 https://doi.org/10.1039/C9SC03414E , 7.8.2024.

[7] AS Rifaioglu, R Cetin Atalay, D Cansen Kahraman, T Doğan, MJ Martin, V Atalay, “MDeePred: novel multi-channel protein featurization for deep learning-based binding affinity prediction in drug discovery”, Bioinformatics, 37: 693–704, https://doi.org/10.1093/bioinformatics/btaa858 , 7.8.2024.

[8] Prezentacija s predavanja Prof.dr.sc. Dragutin Lisjak i Doc.dr.sc. Davor Kolar, kolegiji „Uvod u znanosti o podacima“ , 7.8.2024.

[9] Nenad Bolf „Osvježimo znanje“ Kem. Ind. 70 (9-10) (2021) 591–593, „Strojno učenje“ https://hrcak.srce.hr/file/382926 , 14.8.2024.

[10] Nenad Bolf, Ž. Ujević Andrijić „Osvježimo znanje“, Kem. Ind. 68 (5-6) (2019) 219–220, „Umjetne neuronske mreže“ https://hrcak.srce.hr/file/322233 , 14.8.2024.

[11] Santu Chall „SMILES - Simplified Molecular Input Line Entry System“, https://www.linkedin.com/pulse/smiles-simplified-molecular-input-line-entry-system-santu-chall-hwjjc , 16.8.2024.

[12] Jonsdottir SO, Jorgensen FS, Brunak. S.. Prediction methods and databases within chemoinfromatics: emphasis on drugs and drug candidates. Bioinformatics 21: 2145-2160 https://www.researchgate.net/publication/8018882_Jonsdottir_SO_Jorgensen_FS_Brunak_S_Prediction_methods_and_databases_within_chemoinfromatics_emphasis_on_drugs_and_drug_candidates_Bioinformatics_21_2145-2160 16.8.2024.

[13] K. Džomba „Konvolucijske neuronske mreže“, https://repozitorij.pmf.unizg.hr/islandora/object/pmf%3A5124/datastream/PDF/view 16.8.2024.

PRILOZI

data_processing

```
import os
import sys
import cv2
import json
import torch
import random
import warnings
import subprocess
import numpy as np
import pandas as pd
from operator import itemgetter
from torch.utils.data import Dataset
from torch.utils.data.sampler import SubsetRandomSampler, BatchSampler,
SequentialSampler
"""
from rdkit import Chem
from rdkit.Chem import Draw
from rdkit.Chem.Draw import DrawingOptions
import cairosvg
"""


warnings.filterwarnings(action='ignore')

project_file_path = "{}DEEPScreen".format(os.getcwd().split("DEEPScreen")[0])
training_files_path = "{}/training_files".format(project_file_path)
result_files_path = "{}/result_files".format(project_file_path)
trained_models_path = "{}/trained_models".format(project_file_path)

IMG_SIZE = 200


def get_chemblid_smiles_inchi_dict(smiles_inchi_fl):
    chemblid_smiles_inchi_dict = pd.read_csv("{}/{}".format(training_files_path,
smiles_inchi_fl), sep="\t",

 index_col=False).set_index('chembl_id').T.to_dict('list')

    """
    for key in chemblid_smiles_inchi_dict.keys():
        print(key, chemblid_smiles_inchi_dict[key])
        break
    """
    return chemblid_smiles_inchi_dict


def save_comp_imgs_from_smiles(tar_id, comp_id, smiles):
    mol = Chem.MolFromSmiles(smiles)
    DrawingOptions.atomLabelFontSize = 55
    DrawingOptions.dotsPerAngstrom = 100
    DrawingOptions.bondLineWidth = 1.5
```

```
    Draw.MolToFile(mol, os.path.join(training_files_path, "target_training_datasets",
tar_id, "imgs", "{}.svg".format(comp_id)), size= ( IMG_SIZE , IMG_SIZE ))
    cairosvg.svg2png(url=os.path.join(training_files_path,
"target_training_datasets", tar_id, "imgs", "{}.svg".format(comp_id)),
write_to=os.path.join(training_files_path, "target_training_datasets", tar_id,
"imgs", "{}.png".format(comp_id)))
    subprocess.call(["rm", os.path.join(training_files_path,
"target_training_datasets", tar_id, "imgs", "{}.svg".format(comp_id))])


def create_preprocessed_bioact_file(chembl_filtered_bioact_fl, chembl_version):
    raw_dataset_df = pd.read_csv("{}/{}".format(training_files_path,
chembl_filtered_bioact_fl), sep="\t", index_col=False)
    # keys are compound protein pairs values are list of bioactivities
    annot_dict = dict()
    for ind, row in raw_dataset_df.iterrows():
        chembl_tid = row["Target_CHEMBL_ID"]
        chembl_cid = row["Compound_CHEMBL_ID"]
        standard_units = row["standard_units"]
        row["year"] = row["year"]

        if standard_units in ["uM", "nM", "M"]:
            if standard_units == "nM":
                row["standard_value"] = round(row["standard_value"] / pow(10,3), 3)
            elif standard_units == "M":

                row["standard_value"] = round(row["standard_value"] / pow(10,6), 3)
            else:
                row["standard_value"] = round(row["standard_value"], 3)
            # standard_units = "uM"
            row["standard_units"] = "uM"
            try:
                annot_dict["{},{}".format(chembl_tid, chembl_cid)].append(list(row))
            except:
                annot_dict["{},{}".format(chembl_tid, chembl_cid)] = [list(row)]

    out_fl =
open("{}/{}_preprocessed_filtered_bioactivity_dataset.tsv".format(training_files_path
, chembl_version), "w")
    out_fl.write("\t".join(list(raw_dataset_df.columns)))
    for key in annot_dict.keys():
        if len(annot_dict[key])>1:
            median_std_val = 0.0

            annot_dict[key]  = sorted(annot_dict[key], key=itemgetter(6))
            # print(annot_dict[key])

            if len(annot_dict[key])%2==1:
                median = int(len(annot_dict[key])/2)
                median_bioactivity = annot_dict[key][median]
                out_fl.write("\n" + "\t".join([str(col) for col in
median_bioactivity]))

            else:
                median = int(len(annot_dict[key])/2)

                median_std_val = (annot_dict[key][median][6]+annot_dict[key][median-
1][6])/2
                annot_dict[key][median][6] = median_std_val
                median_bioactivity = annot_dict[key][median]
```

```
                out_fl.write("\n" + "\t".join([str(col) for col in
median_bioactivity]))
        else:

            out_fl.write("\n"+"\t".join([str(col) for col in annot_dict[key][0]]))

    out_fl.close()


def get_uniprot_chembl_sp_id_mapping(chembl_uni_prot_mapping_fl):
    id_mapping_fl = open("{}/{}".format(training_files_path,
chembl_uni_prot_mapping_fl))
    lst_id_mapping_fl = id_mapping_fl.read().split("\n")
    id_mapping_fl.close()
    uniprot_to_chembl_dict = dict()
    for line in lst_id_mapping_fl[1:-1]:
        # 'B2GV46', 'CHEMBL3886125', 'Free fatty acid receptor 3', 'SINGLE PROTEIN'
        uniprot_id, chembl_id, prot_name, target_type = line.split("\t")
        if target_type=="SINGLE PROTEIN":
            if uniprot_id in uniprot_to_chembl_dict:
                uniprot_to_chembl_dict[uniprot_id].append(chembl_id)
            else:
                uniprot_to_chembl_dict[uniprot_id] = [chembl_id]
    count = 0
    for u_id in uniprot_to_chembl_dict:
        if len(uniprot_to_chembl_dict[u_id])>1:
            count += 1

    return uniprot_to_chembl_dict


def get_chembl_uniprot_sp_id_mapping(chembl_mapping_fl):
    id_mapping_fl = open("{}/{}".format(training_files_path, chembl_mapping_fl))
    lst_id_mapping_fl = id_mapping_fl.read().split("\n")
    id_mapping_fl.close()
    chembl_to_uniprot_dict = dict()
    for line in lst_id_mapping_fl[1:-1]:
        # 'B2GV46', 'CHEMBL3886125', 'Free fatty acid receptor 3', 'SINGLE PROTEIN'
        uniprot_id, chembl_id, prot_name, target_type = line.split("\t")
        if target_type=="SINGLE PROTEIN":
            if chembl_id in chembl_to_uniprot_dict:
                chembl_to_uniprot_dict[chembl_id].append(uniprot_id)
            else:
                chembl_to_uniprot_dict[chembl_id] = [uniprot_id]
    count = 0
    for chembl_id in chembl_to_uniprot_dict:
        if len(chembl_to_uniprot_dict[chembl_id])>1:
            count += 1


    print(count)

    return chembl_to_uniprot_dict


def get_act_inact_list_for_a_target(target, fl):
    act_list = []

    inact_list = []

    with open("{}/{}".format(training_files_path, fl))  as f:
```

```
        for line in f:
            if line != "":
                line=line.split("\n")[0]
                chembl_part, comps = line.split("\t")
                chembl_target_id, act_inact = chembl_part.split("_")
                if chembl_target_id == target:
                    if act_inact == "act":
                        act_list = comps.split(",")
                    else:
                        inact_list = comps.split(",")
                        break

    return act_list, inact_list


def get_act_inact_list_for_all_targets(fl):
    act_inact_dict = dict()
    with open("{}/{}".format(training_files_path, fl))  as f:
        for line in f:
            if line != "":
                line=line.split("\n")[0]
                chembl_part, comps = line.split("\t")
                chembl_target_id, act_inact = chembl_part.split("_")
                if act_inact == "act":
                    act_list = comps.split(",")
                    act_inact_dict[chembl_target_id] = [act_list, []]
                else:
                    inact_list = comps.split(",")
                    act_inact_dict[chembl_target_id][1] = inact_list
    return act_inact_dict


def create_act_inact_files_for_all_targets(fl, chembl_version, act_threshold,
inact_threshold):
    pre_filt_chembl_df = pd.read_csv("{}/{}".format(training_files_path, fl),
sep="\t" ,index_col=False)

    act_rows_df = pre_filt_chembl_df[pre_filt_chembl_df["standard_value"]<=10.0]
    inact_rows_df = pre_filt_chembl_df[pre_filt_chembl_df["standard_value"] >= 20.0]
    # 'Target_CHEMBL_ID', 'Compound_CHEMBL_ID', 'pchembl_value',
    target_act_inact_comp_dict = dict()

    for ind, row in act_rows_df.iterrows():
        chembl_tid = row['Target_CHEMBL_ID']
        chembl_cid = row['Compound_CHEMBL_ID']

        if chembl_tid in target_act_inact_comp_dict:
            target_act_inact_comp_dict[chembl_tid][0].add(chembl_cid)
        else:
            target_act_inact_comp_dict[chembl_tid] = [set(), set()]
            target_act_inact_comp_dict[chembl_tid][0].add(chembl_cid)

    for ind, row in inact_rows_df.iterrows():
        chembl_tid = row['Target_CHEMBL_ID']
        chembl_cid = row['Compound_CHEMBL_ID']
        if chembl_tid in target_act_inact_comp_dict:
            target_act_inact_comp_dict[chembl_tid][1].add(chembl_cid)
        else:
            target_act_inact_comp_dict[chembl_tid] = [set(), set()]
            target_act_inact_comp_dict[chembl_tid][1].add(chembl_cid)
```

```
    act_inact_comp_fl =
open("{}/{}_preprocessed_filtered_act_inact_comps_{}_{}.tsv".format(training_files_pa
th, chembl_version, act_threshold, inact_threshold), "w")
    act_inact_count_fl =
open("{}/{}_preprocessed_filtered_act_inact_count_{}_{}.tsv".format(training_files_pa
th, chembl_version, act_threshold, inact_threshold), "w")

    for targ in target_act_inact_comp_dict:

        str_act = "{}_act\t".format(targ) +
",".join(target_act_inact_comp_dict[targ][0])
        act_inact_comp_fl.write("{}\n".format(str_act))

        str_inact = "{}_inact\t".format(targ) +
",".join(target_act_inact_comp_dict[targ][1])
        act_inact_comp_fl.write("{}\n".format(str_inact))

        # write act inact count
        str_act_inact_count = "{}\t{}\t{}\n".format(targ,
len(target_act_inact_comp_dict[targ][0]), len(target_act_inact_comp_dict[targ][1]))
        act_inact_count_fl.write(str_act_inact_count)

    act_inact_count_fl.close()
    act_inact_comp_fl.close()


def create_act_inact_files_similarity_based_neg_enrichment_threshold(act_inact_fl,
blast_sim_fl, sim_threshold):
    data_point_threshold = 100

    uniprot_chemblid_dict =
get_uniprot_chembl_sp_id_mapping("chembl27_uniprot_mapping.txt")
    chemblid_uniprot_dict =
get_chembl_uniprot_sp_id_mapping("chembl27_uniprot_mapping.txt")
    all_act_inact_dict = get_act_inact_list_for_all_targets(act_inact_fl)
    new_all_act_inact_dict = dict()
    count = 0
    for targ in all_act_inact_dict.keys():
        act_list, inact_list = all_act_inact_dict[targ]
        if len(act_list)>=data_point_threshold and
len(inact_list)>=data_point_threshold:
            count += 1
    print(count)

    seq_to_other_seqs_score_dict = dict()
    with open("{}/{}".format(training_files_path, blast_sim_fl)) as f:
        for line in f:
            parts = line.split("\t")
            # print(parts)
            u_id1, u_id2, score = parts[0].split("|")[1], parts[1].split("|")[1],
float(parts[2])
            if u_id1!=u_id2:
                if u_id1 in seq_to_other_seqs_score_dict:
                    seq_to_other_seqs_score_dict[u_id1][u_id2] = score
                else:
                    seq_to_other_seqs_score_dict[u_id1] = dict()
                    seq_to_other_seqs_score_dict[u_id1][u_id2] = score
                if u_id2 in seq_to_other_seqs_score_dict:
                    seq_to_other_seqs_score_dict[u_id2][u_id1] = score
                else:
                    seq_to_other_seqs_score_dict[u_id2] = dict()
```

```
                    seq_to_other_seqs_score_dict[u_id2][u_id1] = score

    for u_id in seq_to_other_seqs_score_dict:
        seq_to_other_seqs_score_dict[u_id] = {k: v for k, v in
sorted(seq_to_other_seqs_score_dict[u_id].items(), key=lambda item: item[1],
reverse=True)}


    count = 0
    for chembl_target_id in all_act_inact_dict.keys():

        count += 1
        # print(count, len(all_act_inact_dict.keys()), chembl_target_id)
        target_act_list, target_inact_list = all_act_inact_dict[chembl_target_id]
        target_act_list, target_inact_list = target_act_list[:], target_inact_list[:]
        uniprot_target_id = chemblid_uniprot_dict[chembl_target_id][0]
        if uniprot_target_id in seq_to_other_seqs_score_dict:
            for uniprot_other_target in
seq_to_other_seqs_score_dict[uniprot_target_id]:
                if
seq_to_other_seqs_score_dict[uniprot_target_id][uniprot_other_target]>=sim_threshold:

                    try:
                        other_target_chembl_id =
uniprot_chemblid_dict[uniprot_other_target][0]
                        other_act_lst, other_inact_lst =
all_act_inact_dict[other_target_chembl_id]
                        set_non_act_inact = set(other_inact_lst) -
set(target_act_list)
                        set_new_inacts = set_non_act_inact - (set(target_inact_list)
& set_non_act_inact)
                        target_inact_list.extend(list(set_new_inacts))
                    except:
                        pass

        new_all_act_inact_dict[chembl_target_id] = [target_act_list,
target_inact_list]
    count = 0
    """
    for targ in new_all_act_inact_dict.keys():
        act_list, inact_list = new_all_act_inact_dict[targ]
        if len(act_list)>=100 and len(inact_list)>=100:
            print(targ, len(act_list), len(inact_list))
        if len(inact_list) >= len(act_list):
            inact_list = inact_list[:len(act_list)]
        else:
            act_list = act_list[:int(len(inact_list)*1.5)]
            new_all_act_inact_dict[targ] = [act_list, inact_list]
        if len(act_list)>=100 and len(inact_list)>=100:
            count += 1
            print(targ, len(act_list), len(inact_list))
    print(count)
    """


    act_inact_comp_fl = open(
        "{}/{}_blast_comp_{}.txt".format(training_files_path,
act_inact_fl.split(".tsv")[0], sim_threshold), "w")
    act_inact_count_fl = open(
        "{}/{}_blast_count_{}.txt".format(training_files_path,
act_inact_fl.split(".tsv")[0], sim_threshold), "w")
```

```
    for targ in new_all_act_inact_dict.keys():
        if len(new_all_act_inact_dict[targ][0])>=data_point_threshold and
len(new_all_act_inact_dict[targ][1])>=data_point_threshold:

            while "" in new_all_act_inact_dict[targ][0]:
                new_all_act_inact_dict[targ][0].remove("")

            while "" in new_all_act_inact_dict[targ][1]:
                new_all_act_inact_dict[targ][1].remove("")


            str_act = "{}_act\t".format(targ) +
",".join(new_all_act_inact_dict[targ][0])
            act_inact_comp_fl.write("{}\n".format(str_act))

            str_inact = "{}_inact\t".format(targ) +
",".join(new_all_act_inact_dict[targ][1])
            act_inact_comp_fl.write("{}\n".format(str_inact))

            # write act inact count
            str_act_inact_count = "{}\t{}\t{}\n".format(targ,
len(new_all_act_inact_dict[targ][0]), len(new_all_act_inact_dict[targ][1]))
            act_inact_count_fl.write(str_act_inact_count)

    act_inact_count_fl.close()
    act_inact_comp_fl.close()


def create_final_randomized_training_val_test_sets(neg_act_inact_fl,
smiles_inchi_fl):
    chemblid_smiles_dict = get_chemblid_smiles_inchi_dict(smiles_inchi_fl)
    act_inact_dict = get_act_inact_list_for_all_targets(neg_act_inact_fl)
    for tar in act_inact_dict:
        os.makedirs(os.path.join(training_files_path, "target_training_datasets",
tar, "imgs"))
        act_list, inact_list = act_inact_dict[tar]
        if len(inact_list) >= len(act_list):
            inact_list = inact_list[:len(act_list)]
        else:
            act_list = act_list[:int(len(inact_list) * 1.5)]

        random.shuffle(act_list)
        random.shuffle(inact_list)

        act_training_validation_size = int(0.8 * len(act_list))
        act_training_size = int(0.8 * act_training_validation_size)
        act_val_size = act_training_validation_size - act_training_size
        training_act_comp_id_list = act_list[:act_training_size]
        val_act_comp_id_list =
act_list[act_training_size:act_training_size+act_val_size]
        test_act_comp_id_list = act_list[act_training_size+act_val_size:]

        inact_training_validation_size = int(0.8 * len(inact_list))
        inact_training_size = int(0.8 * inact_training_validation_size)
        inact_val_size = inact_training_validation_size - inact_training_size
        training_inact_comp_id_list = inact_list[:inact_training_size]
        val_inact_comp_id_list =
inact_list[inact_training_size:inact_training_size+inact_val_size]
        test_inact_comp_id_list = inact_list[inact_training_size+inact_val_size:]
```

```
        print(tar, "all training act", len(act_list),len(training_act_comp_id_list),
len(val_act_comp_id_list), len(test_act_comp_id_list) )
        print(tar, "all training inact", len(inact_list),
len(training_inact_comp_id_list), len(val_inact_comp_id_list),
              len(test_inact_comp_id_list))
        tar_train_val_test_dict = dict()
        tar_train_val_test_dict["training"] = []
        tar_train_val_test_dict["validation"] = []
        tar_train_val_test_dict["test"] = []
        for comp_id in training_act_comp_id_list:
            try:
                save_comp_imgs_from_smiles(tar, comp_id,
chemblid_smiles_dict[comp_id][0])
                tar_train_val_test_dict["training"].append([comp_id, 1])
            except:
                pass
        for comp_id in val_act_comp_id_list:
            try:
                save_comp_imgs_from_smiles(tar, comp_id,
chemblid_smiles_dict[comp_id][0])
                tar_train_val_test_dict["validation"].append([comp_id, 1])
            except:
                pass

        for comp_id in test_act_comp_id_list:
            try:
                save_comp_imgs_from_smiles(tar, comp_id,
chemblid_smiles_dict[comp_id][0])
                tar_train_val_test_dict["test"].append([comp_id, 1])
            except:
                pass

        for comp_id in training_inact_comp_id_list:
            try:
                save_comp_imgs_from_smiles(tar, comp_id,
chemblid_smiles_dict[comp_id][0])
                tar_train_val_test_dict["training"].append([comp_id, 0])
            except:
                pass
        for comp_id in val_inact_comp_id_list:
            try:
                save_comp_imgs_from_smiles(tar, comp_id,
chemblid_smiles_dict[comp_id][0])
                tar_train_val_test_dict["validation"].append([comp_id, 0])
            except:
                pass
        for comp_id in test_inact_comp_id_list:
            try:
                save_comp_imgs_from_smiles(tar, comp_id,
chemblid_smiles_dict[comp_id][0])
                tar_train_val_test_dict["test"].append([comp_id, 0])
            except:

                pass
        random.shuffle(tar_train_val_test_dict["training"])
        random.shuffle(tar_train_val_test_dict["validation"])
        random.shuffle(tar_train_val_test_dict["test"])

        with open(os.path.join(training_files_path, "target_training_datasets", tar,
'train_val_test_dict.json') , 'w') as fp:
            json.dump(tar_train_val_test_dict, fp)
```

```
class DEEPScreenDataset(Dataset):
    def __init__(self, target_id, train_val_test):
        self.target_id = target_id
        self.train_val_test = train_val_test
        self.training_dataset_path =
"{}/target_training_datasets/{}".format(training_files_path, target_id)
        self.train_val_test_folds =
json.load(open(os.path.join(self.training_dataset_path, "train_val_test_dict.json")))
        self.compid_list = [compid_label[0] for compid_label in
self.train_val_test_folds[train_val_test]]
        self.label_list = [compid_label[1] for compid_label in
self.train_val_test_folds[train_val_test]]
        # print(self.label_list)
    def __len__(self):
        return len(self.compid_list)

    def __getitem__(self, index):

        comp_id = self.compid_list[index]
        img_path = os.path.join(self.training_dataset_path, "imgs",
"{}.png".format(comp_id))
        img_arr = cv2.imread(img_path)
        if random.random()>=0.50:
            angle = random.randint(0,359)
            rows, cols, channel = img_arr.shape
            rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
            img_arr = cv2.warpAffine(img_arr, rotation_matrix, (cols, rows),
cv2.INTER_LINEAR,
                                                    borderValue=(255, 255, 255))  #
cv2.BORDER_CONSTANT, 255)

        img_arr = np.array(img_arr) / 255.0
        img_arr = img_arr.transpose((2, 0, 1))
        label = self.label_list[index]

        return img_arr, label, comp_id


def get_train_test_val_data_loaders(target_id, batch_size=32):
    training_dataset = DEEPScreenDataset(target_id, "training")
    validation_dataset = DEEPScreenDataset(target_id, "validation")
    test_dataset = DEEPScreenDataset(target_id, "test")

    train_sampler = SubsetRandomSampler(range(len(training_dataset)))
    train_loader = torch.utils.data.DataLoader(training_dataset,
batch_size=batch_size,
                                               sampler=train_sampler)

    validation_sampler = SubsetRandomSampler(range(len(validation_dataset)))
    validation_loader = torch.utils.data.DataLoader(validation_dataset,
batch_size=batch_size,
                                                sampler=validation_sampler)

    test_sampler = SubsetRandomSampler(range(len(test_dataset)))
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
                                               sampler=test_sampler)

    return train_loader, validation_loader, test_loader
```

```
def get_training_target_list(chembl_version):
    target_df = pd.read_csv(os.path.join(training_files_path,
"{}_training_target_list.txt".format(chembl_version)), index_col=False, header=None)
    # print(target_df)
    # print(list(target_df[0]), len(list(target_df[0])))
    return list(target_df[0])
```

evaluation _metrics

```
import numpy as np
import copy
from math import sqrt
from scipy import stats
from sklearn import preprocessing, metrics
from sklearn.metrics import confusion_matrix

def prec_rec_f1_acc_mcc(y_true, y_pred):
    performance_threshold_dict  = dict()

    ### ADDED on 28 July 2020 by YIP YEW MUN ###
    y_true_tmp = []
    for each_y_true in y_true:
        y_true_tmp.append(each_y_true.item())
    y_true = y_true_tmp

    y_pred_tmp = []
    for each_y_pred in y_pred:
        y_pred_tmp.append(each_y_pred.item())
    y_pred = y_pred_tmp
    ### ADDED on 28 July 2020 by YIP YEW MUN ###

    precision = metrics.precision_score(y_true, y_pred)
    recall = metrics.recall_score(y_true, y_pred)
    f1_score = metrics.f1_score(y_true, y_pred)
    accuracy = metrics.accuracy_score(y_true, y_pred)
    mcc = metrics.matthews_corrcoef(y_true, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    performance_threshold_dict["Precision"] = precision
    performance_threshold_dict["Recall"] = recall
    performance_threshold_dict["F1-Score"] = f1_score
    performance_threshold_dict["Accuracy"] = accuracy
    performance_threshold_dict["MCC"] = mcc
    performance_threshold_dict["TP"] = tp
    performance_threshold_dict["FP"] = fp
    performance_threshold_dict["TN"] = tn
    performance_threshold_dict["FN"] = fn


    return performance_threshold_dict

def get_list_of_scores():
    return ["Precision", "Recall", "F1-Score", "Accuracy", "MCC", "TP", "FP", "TN",
"FN"]
```

main_training

```
import argparse
from train_deepscreen import train_validation_test_training


parser = argparse.ArgumentParser(description='DEEPScreen arguments')
parser.add_argument(
    '--targetid',
    type=str,
    default="CHEMBL286",
    metavar='TID',
    help='Target ChEMBL ID')
parser.add_argument(
    '--model',
    type=str,
    default="CNNModel1",
    metavar='MN',
    help='model name (default: CNNModel1)')
parser.add_argument(
    '--fc1',
    type=int,
    default=512,
    metavar='FC1',
    help='number of neurons in the first fully-connected layer (default:512)')
parser.add_argument(
    '--fc2',
    type=int,
    default=256,
    metavar='FC2',
    help='number of neurons in the second fully-connected layer (default:256)')
parser.add_argument(
    '--lr',
    type=float,
    default=0.001,
    metavar='LR',
    help='learning rate (default: 0.001)')
parser.add_argument(
    '--bs',
    type=int,
    default=32,
    metavar='BS',
    help='batch size (default: 32)')
parser.add_argument(
    '--dropout',
    type=float,
    default=0.25,
    metavar='DO',
    help='dropout rate (default: 0.25)')
parser.add_argument(
    '--epoch',
    type=int,
    default=100,
    metavar='EPC',
    help='Number of epochs (default: 100)')
parser.add_argument(
    '--en',
    type=str,
    default="my_experiment",
```

```
    metavar='EN',
    help='the name of the experiment (default: my_experiment)')



if __name__ == "__main__":
    args = parser.parse_args()
    print(args)
    train_validation_test_training(args.targetid, args.model, args.fc1, args.fc2,
args.lr, args.bs,
                                   args.dropout, args.epoch, args.en)
```

models

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from operator import itemgetter

class CNNModel1(nn.Module):
    def __init__(self, fully_layer_1, fully_layer_2, drop_rate):
        super(CNNModel1, self).__init__()

        self.conv1 = nn.Conv2d(3, 32, 2)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, 2)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, 2)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 64, 2)
        self.bn4 = nn.BatchNorm2d(64)
        self.conv5 = nn.Conv2d(64, 32, 2)
        self.bn5 = nn.BatchNorm2d(32)

        self.pool = nn.MaxPool2d(2, 2)
        self.drop_rate = drop_rate
        self.fc1 = nn.Linear(32*5*5, fully_layer_1)
        self.fc2 = nn.Linear(fully_layer_1, fully_layer_2)
        self.fc3 = nn.Linear(fully_layer_2, 2)

    def forward(self, x):
        # print(x.shape)
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        # print(x.shape)
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        # print(x.shape)
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        # print(x.shape)
        x = self.pool(F.relu(self.bn4(self.conv4(x))))
        # print(x.shape)
        x = self.pool(F.relu(self.bn5(self.conv5(x))))
        # print(x.shape)

        x = x.view(-1, 32*5*5)
        x = F.dropout(F.relu(self.fc1(x)), self.drop_rate)
        x = F.dropout(F.relu(self.fc2(x)), self.drop_rate)
        x = self.fc3(x)

        return x
```

```
# TODO: Create other models
```

train_deepscreen

```
import os
import sys
import cv2
import json
import torch
import random
import warnings
import subprocess
import numpy as np
import pandas as pd
import torch.nn as nn
from models import CNNModel1
from torch.autograd import Variable
from data_processing import get_train_test_val_data_loaders
from evaluation_metrics import prec_rec_f1_acc_mcc, get_list_of_scores

warnings.filterwarnings(action='ignore')
torch.manual_seed(123)
np.random.seed(123)
use_gpu = torch.cuda.is_available()



project_file_path = "{}DEEPScreen".format(os.getcwd().split("DEEPScreen")[0])
training_files_path = "{}/training_files".format(project_file_path)
result_files_path = "{}/result_files".format(project_file_path)
trained_models_path = "{}/trained_models".format(project_file_path)

def save_best_model_predictions(experiment_name, epoch, validation_scores_dict,
test_scores_dict, model, project_file_path, target_id, str_arguments,

 all_test_comp_ids, test_labels, test_predictions):

    if not os.path.exists(os.path.join(trained_models_path, experiment_name)):
        os.makedirs(os.path.join(trained_models_path, experiment_name))

    torch.save(model.state_dict(),
               "{}/{}/{}_best_val-{}-state_dict.pth".format(trained_models_path,
experiment_name,

 target_id, str_arguments))
    # print(all_test_comp_ids)
    str_test_predictions = "CompoundID\tLabel\tPred\n"
    for ind in range(len(all_test_comp_ids)):
        str_test_predictions += "{}\t{}\t{}\n".format(all_test_comp_ids[ind],
                                                      test_labels[ind],
                                                      test_predictions[ind])
    best_test_performance_dict = test_scores_dict
    best_test_predictions = str_test_predictions
    return validation_scores_dict, best_test_performance_dict, best_test_predictions,
str_test_predictions
```

```
def get_device():
    device = "cpu"
    if use_gpu:
        print("GPU is available on this device!")
        device = "cuda"
    else:
        print("CPU is available on this device!")
    return device


def calculate_val_test_loss(model, criterion, data_loader, device):
    total_count = 0
    total_loss = 0.0
    all_comp_ids = []
    all_labels = []
    all_predictions = []
    for i, data in enumerate(data_loader):
        img_arrs, labels, comp_ids = data
        img_arrs, labels = torch.tensor(img_arrs).type(torch.FloatTensor).to(device),
torch.tensor(labels).to(device)
        total_count += len(comp_ids)
        y_pred = model(img_arrs).to(device)
        loss = criterion(y_pred.squeeze(), labels)
        total_loss += float(loss.item())
        all_comp_ids.extend(list(comp_ids))
        _, preds = torch.max(y_pred, 1)
        all_labels.extend(list(labels))
        all_predictions.extend(list(preds))


    return total_loss, total_count, all_comp_ids, all_labels, all_predictions


def train_validation_test_training(target_id, model_name, fully_layer_1,
fully_layer_2, learning_rate, batch_size, drop_rate, n_epoch, experiment_name):
    arguments = [str(argm) for argm in
                 [target_id, model_name, fully_layer_1, fully_layer_2, learning_rate,
batch_size, drop_rate, n_epoch, experiment_name]]

    str_arguments = "-".join(arguments)
    print("Arguments:", str_arguments)

    device = get_device()
    exp_path = os.path.join(result_files_path, "experiments", experiment_name)

    if not os.path.exists(exp_path):
        os.makedirs(exp_path)


    best_val_test_result_fl = open(
        "{}/best_val_test_performance_results-{}.txt".format(exp_path,str_arguments),
"w")
    best_val_test_prediction_fl = open(
        "{}/best_val_test_predictions-{}.txt".format(exp_path,str_arguments), "w")

    train_loader, valid_loader, test_loader =
get_train_test_val_data_loaders(target_id, batch_size)
    model = None
    if model_name == "CNNModel1":
        model = CNNModel1(fully_layer_1, fully_layer_2, drop_rate).to(device)
```

```
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    criterion = nn.CrossEntropyLoss()
    optimizer.zero_grad()

    best_val_mcc_score, best_test_mcc_score = 0.0, 0.0
    best_val_test_performance_dict = dict()
    best_val_test_performance_dict["MCC"] = 0.0

    for epoch in range(n_epoch):
        total_training_count = 0
        total_training_loss = 0.0
        print("Epoch :{}".format(epoch))
        model.train()
        batch_number = 0
        all_training_labels = []
        all_training_preds = []
        print("Training mode:", model.training)
        for i, data in enumerate(train_loader):
            batch_number += 1
            # print(batch_number)
            # clear gradient DO NOT forget you fool!
            optimizer.zero_grad()
            img_arrs, labels, comp_ids = data
            img_arrs, labels =
torch.tensor(img_arrs).type(torch.FloatTensor).to(device),
torch.tensor(labels).to(device)

            total_training_count += len(comp_ids)
            y_pred = model(img_arrs).to(device)
            _, preds = torch.max(y_pred, 1)
            all_training_labels.extend(list(labels))
            all_training_preds.extend(list(preds))

            loss = criterion(y_pred.squeeze(), labels)
            total_training_loss += float(loss.item())
            loss.backward()
            optimizer.step()
        print("Epoch {} training loss:".format(epoch), total_training_loss)
        training_perf_dict = dict()
        try:
            training_perf_dict = prec_rec_f1_acc_mcc(all_training_labels,
all_training_preds)
        except:
            print("There was a problem during training performance calculation!")
        # print(training_perf_dict)
        model.eval()
        with torch.no_grad():  # torch.set_grad_enabled(False):
            print("Validation mode:", not model.training)

            total_val_loss, total_val_count, all_val_comp_ids, all_val_labels,
val_predictions = calculate_val_test_loss(model, criterion, valid_loader, device)

            val_perf_dict = dict()
            val_perf_dict["MCC"] = 0.0
            try:
                val_perf_dict = prec_rec_f1_acc_mcc(all_val_labels, val_predictions)
            except:
                print("There was a problem during validation performance
calculation!")
```

```
            total_test_loss, total_test_count, all_test_comp_ids, all_test_labels,
test_predictions = calculate_val_test_loss(
                model, criterion, test_loader, device)

            test_perf_dict = dict()
            test_perf_dict["MCC"] = 0.0
            try:
                test_perf_dict = prec_rec_f1_acc_mcc(all_test_labels,
test_predictions)
            except:
                print("There was a problem during test performance calculation!")

            if val_perf_dict["MCC"] > best_val_mcc_score:
                best_val_mcc_score = val_perf_dict["MCC"]
                best_test_mcc_score = test_perf_dict["MCC"]

                validation_scores_dict, best_test_performance_dict,
best_test_predictions, str_test_predictions = save_best_model_predictions(
                    experiment_name, epoch, val_perf_dict, test_perf_dict,
                    model,project_file_path, target_id, str_arguments,
                    all_test_comp_ids, all_test_labels, test_predictions)

        if epoch == n_epoch - 1:
            score_list = get_list_of_scores()
            for scr in score_list:
                best_val_test_result_fl.write("Test {}:\t{}\n".format(scr,
best_test_performance_dict[scr]))
            best_val_test_prediction_fl.write(best_test_predictions)

            best_val_test_result_fl.close()
            best_val_test_prediction_fl.close()
```