SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2816

OTKRIVANJE STRŠEĆIH VRIJEDNOSTI U SLIKOVNIM PODACIMA KORIŠTENJEM METODA STROJNOG I DUBOKOG UČENJA

Martina Sušilović

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2816

OTKRIVANJE STRŠEĆIH VRIJEDNOSTI U SLIKOVNIM PODACIMA KORIŠTENJEM METODA STROJNOG I DUBOKOG UČENJA

Martina Sušilović

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2816

Pristupnica:	**Martina Sušilović (0036507308)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Otkrivanje stršećih vrijednosti u slikovnim podacima korištenjem metoda strojnog i dubokog učenja**

Opis zadatka:

Stršećim vrijednostima u podacima smatraju se primjerci koji po nekim svojstvima odstupaju od većine ostalih u skupu. Prisutnost stršećih vrijednosti može otežati učenje modela, stoga je otkrivanje i uklanjanje anomalija važan korak u obradi podataka. Jedna od popularnih metoda otkrivanja stršećih vrijednosti u slikama su autoenkoderi, neuronske mreže koje najprije kodiraju podatke u reprezentacije manje dimenzionalnosti, a zatim iz takvih reprezentacija rekonstruiraju ulazni podatak. Podaci koje mreža manje uspješno rekonstruira kandidati su za stršeće vrijednosti. Cilj je ovog diplomskog rada ostvariti i vrednovati rezultate autoenkodera te ih usporediti s rezultatima dobivenim nekim uobičajenim algoritmima nenadziranog strojnog učenja za otkrivanje anomalija. Implementacija modela strojnog učenja bit će ostvarena u programskom jeziku Python uz upotrebu prikladnih biblioteka. Uspješnost modela vrednovat će se na javno dostupnim skupovima slikovnih podataka uz sintetičko variranje udjela anomalija u podacima.

Rok za predaju rada: 27. lipnja 2022.

Sadržaj

Uvod

Stršećim vrijednostima nazivaju se primjerci koji se značajno razlikuju od ostalih u skupu podataka. Prisutnost stršećih vrijednosti može otežati učenje modela strojnog i dubokog učenja, stoga je identificiranje i uklanjanje anomalija važan korak u obradi podataka. Otkrivanje stršećih vrijednosti (engl. *outlier detection*, *anomaly detection*) aktivno se istražuje već desetljećima te pronalazi brojne stvarne primjene u područjima internetske sigurnosti, nadzora financija, medicine, računalnog vida, itd. U svrhu otkrivanja stršećih vrijednosti kontinuirano se razvijaju brojni algoritmi strojnog učenja i statističke metode temeljene na odstupanju, udaljenosti, gustoći, rekonstrukciji podataka itd. Posljednjih godina, s razvojem i popularizacijom dubokog učenja, razvija se i tzv. duboko otkrivanje stršećih vrijednosti. Razvijeni duboki modeli u nekim zahtjevnim zadacima nadmašuju ranije korištene metode [1].

Tri su moguća pristupa otkrivanju stršećih vrijednosti: nadzirani, polunadzirani i nenadzirani. Nadzirani pristupi zahtijevaju podatke s oznakama normalnih i stršećih vrijednosti. Polunadzirani pristupi za učenje zahtijevaju isključivo normalne podatke, bez prisutnosti stršećih vrijednosti da bi kasnije među neviđenim podacima mogli prepoznati one koji ne pripadaju normalnom skupu. Nenadzirani pristup ne podrazumijeva nikakve informacije o svojstvima normalnih ili stršećih vrijednosti u skupu, nego se modeli uče na podacima u kojima su već prisutne stršeće vrijednosti, bez oznaka. Nenadzirano otkrivanje stršećih vrijednosti je najučestaliji tip, a zbog manjka informacija i moguće raznolikosti podataka, taj je pristup ujedno i najviše izazovan [14].

Nenadzirano otkrivanje stršećih vrijednosti u slikovnim skupovima podataka važno je u zadacima računalnog vida koji zahtijevaju velike količine podataka. Velik broj slika moguće je, primjerice, preuzeti s internetske tražilice, ali će rezultati pretraživanja sadržavati i dio podataka koji ne odgovaraju zahtjevima te ih je potrebno ukloniti. Postupci nenadziranog strojnog i dubokog učenja mogu u takvim slučajevima olakšati filtriranje skupa podataka i smanjiti troškove ručnog pregledavanja i označavanja slika [2].

U nastavku rada, opisani su neki od najčešće korištenih algoritama nenadziranog strojnog učenja za otkrivanje stršećih vrijednosti. Osim toga, opisana je arhitektura i implementacija dvaju dubokih modela: konvolucijskog i varijacijskog autoenkodera. Uspješnost svakog od

postupaka evaluirat će se na skupu podataka sa slikama betonskih površina s oštećenjima koja predstavljaju stršeće vrijednosti. Otkrivanje i praćenje površinskih pukotina važno je zbog održavanja strukturalne sigurnosti građevine. Ručno pregledavanje površina vremenski je zahtjevno, a u slučajevima nekih građevina i nemoguće, zbog čega se može zamijeniti metodama nenadziranog strojnog učenja.

Uz variranje udjela stršećih vrijednosti i eksperimente s raznim hiperparametima za svaki od modela, cilj je usporediti konvencionalne metode detekcije anomalija s nenadziranim modelima dubokog učenja na slikovnim podacima te evaluirati ponašanje dubokih modela kada za učenje istih nije dostupan označen skup normalnih podataka.

1. Implementirani postupci otkrivanja stršećih vrijednosti

1.1. Postupci nenadziranog strojnog učenja

Strojno učenje koristi se za detekciju stršećih vrijednosti jer su skupovi podataka često preveliki da bi bilo moguće ručno pregledati i identificirati anomalije. Među brojnim metodama nenadziranog strojnog učenja za otkrivanje stršećih vrijednosti odabrano je nekoliko popularnijih čije su implementacije dostupne u programskom jeziku Python u sklopu biblioteke *scikit-learn* [7].

1.1.1. Faktor lokalnih stršećih vrijednosti (LOF)

Faktor lokalnih stršećih vrijednosti (engl. *local outlier factor, LOF*) pripada postupcima otkrivanja stršećih vrijednosti temeljenim na gustoći. Računa odstupanje lokalne gustoće oko primjerka u odnosu na lokalnu gustoću njegovih najbližih susjednih primjeraka u euklidskom prostoru. Stršećim vrijednostima proglašava primjerke oko kojih je gustoća znatno manja u odnosu na susjedne.

U eksperimentima s algoritmom LOF korištena je pretpostavljena metrika za računanje udaljenosti primjeraka, udaljenost Minkowskog, opisana izrazom (1), koja uz parametar $p = 2$ odgovara euklidskoj udaljenosti.

$$d(\boldsymbol{x}, \boldsymbol{y}) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{1}$$

Udaljenost primjerka od njegovog *k*-tog najbližeg susjeda u euklidskom prostoru naziva se *k*-udaljenost (engl. *k-distance*). Udaljenost dohvatljivosti (engl. *reachability distance, rd*) definirana je kao udaljenost primjerka **x** od primjerka **y**, ali ne veća od svojstvene *k*-udaljenosti primjerka **y** što opisuje izraz (2).

$$rd_k(\boldsymbol{x}, \boldsymbol{y}) = max\{k-distance(\boldsymbol{y}), d(\boldsymbol{x}, \boldsymbol{y})\} \tag{2}$$

Taj se rezultat dalje koristi u računanju lokalne udaljenosti dohvatljivosti (engl. *local reachability distance, lrd*) svakog primjerka, prema izrazu (3). Nakon što se izračunaju

udaljenosti dohvatljivosti u odnosu na *k* najbližih susjeda, lokalna udaljenost dohvatljivosti jednaka je inverznoj vrijednosti prosjeka *k* udaljenosti dohvatljivosti.

$$lrd_k(\boldsymbol{x}) = \frac{|N_k(\boldsymbol{x})|}{\sum_{\boldsymbol{y} \in N_k(\boldsymbol{x})} rd(\boldsymbol{x}, \boldsymbol{y})} \tag{3}$$

Konačno, faktor lokalnih stršećih vrijednosti (LOF) u izrazu (4) određuje se kao omjer prosječne vrijednosti *lrd* od *k* susjeda primjerka **x** s njegovom vlastitom vrijednosti *lrd*.

$$LOF_k(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{y} \in N_k(\boldsymbol{x})} lrd_k(\boldsymbol{y})}{|N_k(\boldsymbol{x})|\, lrd_k(\boldsymbol{x})} \tag{4}$$

Vrijednost LOF približno jednaka 1 ili manja upućuje na to da je točka u području slične gustoće kao njezinih *k* najbližih susjeda, što je svojstveno za normalne primjerke. Za stršeće vrijednosti koje su okružene s manje primjeraka LOF tipično iznosi više od 1. Osim stršećih vrijednosti u odnosu na cijelu skup podataka, LOF uz manji *k* pronalazi i lokalne stršeće vrijednosti, tj. primjerke koji odstupaju u odnosu na neku grupu unutar skupa podataka [12]. Nedostatak metode je taj što nije jednostavno odlučiti prag vrijednosti LOF u praksi koji zaista predstavlja stršeću vrijednost, jer je to ovisno o skupu podataka.

Osim LOF-a, razvijeni su brojni slični algoritmi koji na različite načine procjenjuju lokalnu gustoću oko primjerka i primjenjivi su na različite probleme. Primjerice, algoritam faktora stršećih vrijednosti temeljen na povezanosti (engl. *connectivity-based outlier factor*, *COF*) pretpostavlja linearnu distribuciju podataka. Faktor lokalnih stršećih vrijednosti temeljen na grupiranju (engl. *cluster-based local outlier factor*, *CBLOF*) nakon grupiranja podataka procjenjuje gustoću grupa. Algoritam vjerojatnosti lokalnih stršećih vrijednosti (engl. *local outlier probability*, *LoOP*) pretpostavlja da *k*-udaljenosti slijede normalnu razdiobu i kao mjeru stršećih vrijednosti koristi vjerojatnost [20].

1.1.2. Izolacijska šuma

Algoritam izolacijske šume (engl. *isolation forest*, *IF*) razlikuje se od većine ostalih metoda otkrivanja anomalija jer ne modelira normalne podatke, nego nastoji izravno izolirati stršeće vrijednosti iz skupa podataka. Temelji se na ideji da je anomalije jednostavnije izdvojiti prema vrijednosti njihovih značajki nego normalne podatke.

Implementacijski, IF se sastoji od ansambla binarnih stabala odluke. Iz odabranog skupa podataka izdvaja se podskup i dodjeljuje se određenom binarnom stablu. Za grananje stabla

koristi se nasumična vrijednost nasumično odabrane numeričke značajke podataka. Primjerci s vrijednošću značajke manjom od odabrane pripadaju lijevom podstablu, a primjerci s većom vrijednošću značajke desnom podstablu. Postupak se rekurzivno ponavlja sve dok svaki primjerak iz skupa podataka ne bude izoliran u listu stabla. Maksimalna dubina tako izgrađenog stabla ograničena je na $\lceil log_2 n \rceil$, gdje n označava broj primjeraka u skupu podataka [6]. Nasumično particioniranje rezultira znatno kraćim putevima kroz stablo za stršeće vrijednosti nego za normalne podatke. Srednja duljina puta od korijena do čvora koji predstavlja primjerak u svim stablima ansambla koristi se kao mjera za određivanje stršećih vrijednosti [6]. Mjera za određivanje stršećih vrijednosti računa se prema izrazu (5).

$$s(\boldsymbol{x}, n) = 2 - \frac{E(h(\boldsymbol{x}))}{c(n)} \tag{5}$$

Vrijednost h(x) označava duljinu puta kroz stablo, tj. broj bridova kojima je potrebno proći od korijena stabla do lista koji predstavlja primjerak **x**, a E(h(**x**)) prosječnu duljinu puta za **x** u svim stablima u ansamblu. c(n) predstavlja prosječnu duljinu puta u jednom stablu i određuje se prema izrazu (6). Broj primjeraka u skupu označen je s n, a H predstavlja harmonijski broj opisan izrazom (7).

$$c(n) = 2H(n-1) - 2(n-1)/n \tag{6}$$

$$H(n) = \ln(n) + e \tag{7}$$

Primjerci s rezultatom s(**x**, n) približno jednakim 1 su vjerojatno stršeće vrijednosti, dok je za normalne primjerke očekivana vrijednost s(**x**, n) manja od 0.5. Ako skup nema stršećih vrijednosti, rezultati s(**x**, n) bit će približno jednaki 0.5 za sve podatke [6].

Prednosti su ovog postupka jednostavnost i linearna vremenska složenost koje ga čine primjenjivim na podatke s velikim brojem značajki ili velike skupove podataka. Algoritam je brz jer ne podrazumijeva računalno zahtjevne operacije računanja udaljenosti ili gustoće [18].

1.1.3. Stroj potpornih vektora s jednim razredom (OC-SVM)

Otkrivanje stršećih vrijednosti moguće je interpretirati kao klasifikaciju s jednim razredom, pri čemu se anomalije detektiraju u odnosu na neku kategoriju podataka. Stroj potpornih vektora (engl. *support vector machine, SVM*) je učinkovit algoritam strojnog učenja namijenjen klasifikacijskim i regresijskim zadacima. Model je opisan izrazom (8).

$$h(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b} \tag{8}$$

Korisno je svojstvo postupka SVM preslikavanje linearno nerazdvojivih skupova podataka pomoću funkcije Φ u prostor značajki više dimenzije F u kojem problem postaje linearno odvojiv hiperravninom. U ovom slučaju, model je moguće opisati kao u izrazu (9).

$$h(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{w}^T \Phi(\boldsymbol{x}) + \boldsymbol{b} \tag{9}$$

Algoritam SVM klasificira podatke konstruiranjem hiperravnine $h(\mathbf{x}) = 0$ koja razdvaja razrede podataka u višedimenzionalnom prostoru značajki F. Udaljenost hiperravnine do najbližeg primjerka svakog razreda naziva se marginom, a cilj je učenja pronaći hiperravninu maksimalno udaljenu od najbližeg primjerka svake klase, tj. maksimizirati marginu. Potpornim vektorima nazivaju se primjerci najbliži hiperravnini [11]. Primjerci različitih razreda nalazit će se s različitih strana hiperravnine, a funkciju odluke je moguće zapisati izrazom (10).

$$f(\boldsymbol{x}) = sgn(\sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + \ b) \tag{10}$$

U izrazu je:

- n broj primjeraka u skupu podataka;
- α vektor tzv. Lagrangeovih multiplikatora koji svakom primjerku $\boldsymbol{x_i}$ pridružuje težinu između 0 i 1; vrijednost α_i različita je od nule za vektore koji se nalaze na margini, tj. za potporne vektore;
- $K(\boldsymbol{x}, \boldsymbol{x_i})$ jezgrena funkcija koja opisuje sličnost dvaju primjeraka.

Osim najjednostavnije linearne jezgre opisane izrazom (11), uobičajeno je korištenje moćnije Gaussove radijalne bazne funkcije (engl. *radial basis function, rbf*) opisane izrazom (12), čiji je hiperparametar γ.

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}' \tag{11}$$

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2) \tag{12}$$

Stroj potpornih vektora s jednim razredom (engl. *One-Class Support Vector Machine*, *OC-SVM*) je proširenje ideje algoritma SVM originalno opisan u radu [10]. Cilj je ove inačice algoritma pronaći hiperravninu koja razdvaja skup podataka od ishodišta prostora značajki

F. Pri tome se algoritmu može zadati parametar *v* koji određuje maksimalan udio stršećih vrijednosti koji se mogu nalaziti „sa suprotne strane“ hiperravnine u odnosu na većinu podataka u području veće gustoće. Algoritam spada u postupke temeljene na udaljenosti.

1.2. Duboki modeli za nenadzirano učenje

Ranije opisani postupci strojnog učenja nailaze na ograničenja u podacima s velikim brojem značajki, poput slika. Za takve skupove podataka učestalo je korištenje algoritama dubokog učenja. Duboke neuronske mreže su podskup algoritama strojnog učenja implementirane slaganjem slojeva jednostavnih jedinica (tzv. perceptrona) koje primjenjuju zajedničku operaciju nad podacima. Svaki od slojeva prihvaća informacije prethodnog sloja, obavlja određene transformacije i prosljeđuje podatke sljedećem sloju. Razvojem biblioteka za duboko učenje poput *tensorflowa* povećala se dostupnost i jednostavnost implementacije dubokih modela. Razina kompleksnosti neuronskih mreža znatno je veća od ranije opisanih postupaka strojnog učenja, stoga je teško predvidjeti kakve reprezentacije podataka će mreža naučiti i koje će vrijednosti označiti kao stršeće.

Postupak učenja dubokih neuronskih mreža

Za učenje parametara neuronskih mreža koristi se iterativni postupak gradijentnog spusta. Za odabranu funkciju gubitka L koju model nastoji minimizirati, računaju se gradijenti s obzirom na svaki od parametara θ. Zatim se, uzimajući u obzir tzv. stopu učenja α ažuriraju svi parametri prema izrazu (13). θ^t označava vrijednost parametra u iteraciji učenja *t*.

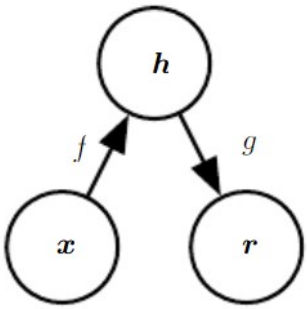
$$\theta^{t+1} = \theta^t - \alpha \frac{\partial \mathrm{L}}{\partial \theta} \tag{13}$$

Težine se ažuriraju od posljednjeg sloja prema prvom, u postupku koji se naziva propagacijom unatrag (engl. *backpropagation*). Važan je odabir stope učenja α. Uz premalu stopu učenja, algoritam će presporo konvergirati k optimalnom rješenju, dok uz preveliku stopu učenja algoritam neće konvergirati.

Autoenkoderi

Autoenkoderi su tip unaprijedne umjetne neuronske mreže čiji je cilj pokušati ulazne podatke preslikati na izlaz. Prvi dio mreže, tzv. koder (engl. *encoder*) podatke koji se dovode na ulaz kodira u bitne latentne značajke tipično manje dimenzionalnosti funkcijom $h = f(x)$,

dok drugi dio mreže, tzv. dekoder (engl. decoder) iz tih značajki nastoji rekonstruirati ulazne podatke funkcijom $r = g(h)$ [9]. Općenita shema autoenkodera prikazana je slikom 1.1.



Sl. 1.1 Općenita struktura autoenkodera; slika je preuzeta iz knjige [9]

Razlika između ulaznih i izlaznih podataka naziva se pogreškom rekonstrukcije, a cilj učenja modela je minimizirati pogrešku, tj. naučiti koder da kodira najvažnije značajke skupa podataka, tako da iz njih dekoder može ponovo rekonstruirati ulazne podatke.

Autoenkoderi spadaju u skupinu algoritama za otkrivanje stršećih vrijednosti temeljenih na rekonstrukciji. Detekcija stršećih vrijednosti uz autoenkodere odvija se u dva koraka. Modeli najprije uče rekonstruirati ulazne slike iz skupa podataka. Ključna ideja je da po završetku učenja mreže, primjerci koje model najlošije rekonstruira, tj. primjerci koji imaju pogrešku višu od nekog proizvoljno zadanog praga izdvajaju se kao stršeće vrijednosti. Duboki autoenkoderi su mreže s više skrivenih slojeva u koderu i dekoderu koji kodiraju i dekodiraju značajke u više koraka.

Mjera uspješnosti rekonstrukcije (SSIM)

Kao mjera za usporedbu ulaznih i izlaznih slikovnih podataka odabrana je mjera indeksa strukturalne sličnosti (engl. *structural similarity index measure*, *SSIM*) opisana u radu [4]. Usporedba dviju slika temelji se na trima značajkama: svjetlini (engl, *luminance*, *l*), kontrastu (engl. *contrast*, *c*) i strukturi (engl. *structure*, *s*), koje opisuju redom izrazi (14), (15) i (16).

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{14}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{15}$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x \sigma_y + c_3} \tag{16}$$

U navedenim izrazima oznake su:

- μ_x prosječna vrijednost piksela slike *x*;
- μ_y prosječna vrijednost piksela slike *y*;
- σ_x^2 varijanca slike *x*;
- σ_y^2 varijanca slike *y*;
- σ_{xy} kovarijanca slika *x* i *y*;
- c_1, c_2 konstante koje se dodaju kako bi se izbjeglo dijeljenje s malim nazivnikom.

Sličnost dviju slika jednakih dimenzija *x* i *y* računa se kao kombinacija triju opisanih značajki prema izrazu (17).

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}. \tag{17}$$

Vrijednosti indeksa strukturalne sličnosti su između -1 i 1. Vrijednost 1 upućuje na veliku sličnost dviju slika, dok vrijednost -1 upućuje na malu sličnost.

Prednosti mjere SSIM u odnosu na uobičajenu funkciju srednjeg kvadratnog odstupanja ulaza i izlaza (engl. *mean squared error, MSE*) su korelacija s ljudskim vizualnim sustavom, jedinstveni maksimum i ograničenost [5]. Kao funkcija gubitka koju duboki modeli tijekom učenja nastoje minimizirati korištena je invertirana srednja vrijednost metrike SSIM dobivene na grupi slika. Vrijednost definirane funkcije gubitka bit će to manja što su originalna i rekonstruirana slika međusobno sličnije. Takva formulacija funkcije gubitka je u skladu s minimizacijskim postupkom gradijentnog spusta korištenog kod učenja neuronskih mreža.

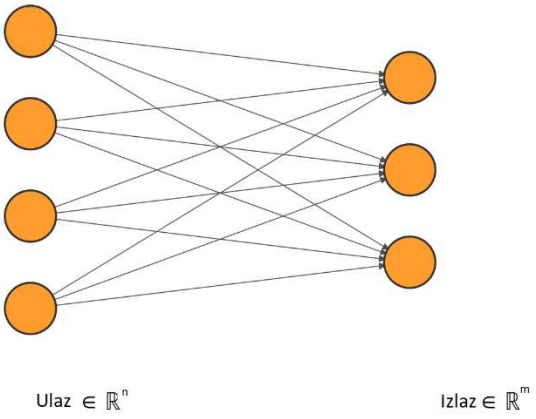
1.2.1. Slojevi u dubokim modelima

Potpuno povezani slojevi

Potpuno povezani slojevi podrazumijevaju da se svaka od *n* vrijednosti ulaza (ili izlaza prethodnog sloja) dovodi do svake od *m* osnovnih jedinica, tj. neurona u sloju koji primjenjuje linearnu transformaciju opisanu izrazom (18).

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{18}$$

Za ulazni vektor $\boldsymbol{x}$ veličine n i željenu dimenzija izlaza m, $\boldsymbol{W}$ predstavlja matricu parametara dimenzija $m \times n$, a $\boldsymbol{b}$ vektor s m parametara. Na početku procesa učenja, parametri se najčešće inicijaliziraju na nasumične vrijednosti. Na slici, linije između ulaznog i izlaznog vektora predstavljaju upravo vrijednosti parametara $w_{i,j}$ za svaku od kombinacija ulaza i izlaza.



Sl. 1.2 Vizualizacija ulaza, izlaza i težina potpuno povezanog sloja

Konvolucijski slojevi

Operacija konvolucije podrazumijeva uzastopnu primjenu filtara (engl. *filter*) na ulazne podatke. Filtar je dvodimenzionalno polje manjih dimenzija od ulaznih slika, a primjenjuje se računanjem skalarnog produkta vrijednosti s dijelom slike veličine filtra. Operacija konvolucije primjenjuje se sistematično, pomicanjem od gornjeg lijevog do donjeg desnog ruba slike za zadani broj piksela. U jednom koraku, filtar se „smjesti“ na regiju ulazne slike te se primjenjuje množenje po elementima, a svi elementi izlazne matrice se sumiraju u jedinstvenu vrijednost. Na slici 1.3 lijevo je vizualizacija 2D konvolucije. U sljedećem koraku, filtar se pomiče i operacija se ponavlja. Vrijednost dvodimenzionalnog izlaza na mjestu (i, j) opisuje izraz (19).

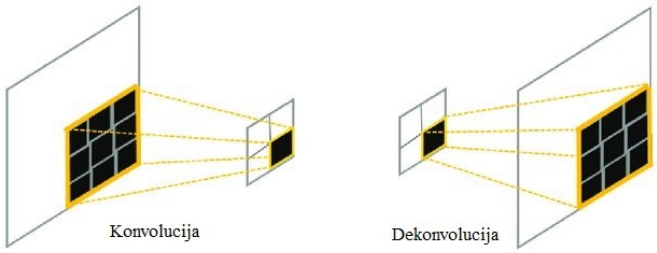
$$S(i,\ j) = (K * I)(i,j) = \sum_{m=m_{min}}^{m_{max}} \sum_{n=n_{min}}^{n_{max}} K(m,n) \cdot I(i+m,\ j+n) \tag{19}$$

U izrazu su:

- K(i, j) vrijednost konvolucijskog filtra na poziciji (i, j)
- I(i, j) piksel ulazne slike na poziciji (i, j)

Konvolucijski slojevi su prikladni za upotrebu s podacima koji imaju topologiju rešetke, kao što su slike [9]. Izlaz iz konvolucijskog sloja naziva se mapom značajki. Za ulaznu sliku dimenzija $n \times n$, filtar dimenzija $f \times f$ i pomak konvolucije jednak 1, dimenzije izlazne slike bit će $(n - f + 1) \times (n - f + 1)$. Dimenzija dubine izlaza odgovara broju filtara u sloju.

Operacija dekonvolucije inverzna je konvoluciji, jer povećava visinu i širinu izlaza u odnosu na ulaz. Operacija ulazne slike ili izlaze prethodnog sloja najprije proširuje nadopunjavanjem nulama, a zatim se primjenjuje ranije opisan postupak konvolucije, kao što je ilustrirano na slici 1.3 desno. Broj redaka i stupaca koji se dodaju nadopunjavanjem nulama ovisi o željenim dimenzijama izlaznih podataka i koraku konvolucije.



Sl. 1.3 Vizualizacija ulaza i izlaza operacija konvolucije i dekonvolucije

U dubokim modelima, podaci koji se dovode na ulaz slojeva su oblika *(veličina mini-grupe × broj kanala × broj redaka × broj stupaca)*, a povratne vrijednosti oblika *(veličina mini-grupe × broj filtera × novi broj redaka × novi broj stupaca)*. Učenje mreže s konvolucijskim slojevima podrazumijeva računanje gradijenata funkcije gubitka s obzirom na vrijednosti filtara i podešavanje vrijednosti filtara.

Aktivacijske funkcije

Uobičajeno je na izlaze svakog od slojeva dodatno primijeniti neku od aktivacijskih funkcija koje unose nelinearnost. Bez aktivacijskih funkcija, ograničena je sposobnost modela za rješavanje nekih kompleksnijih zadataka, jer bi njihovi slojevi predstavljali kompoziciju linearnih transformacija. Jedna od često korištenih aktivacija je funkcija zglobnice (engl. *rectified linear unit, ReLU*). Funkcija izlaze iz sloja koji su veći od 1 propušta, a negativne izlaze zamjenjuje nulom. Opisana je izrazom (20). Druga često korištena aktivacijska funkcija je funkcija sigmoide, koja realne vrijednosti preslikava u vrijednosti između 0 i 1, a opisana je izrazom (21).

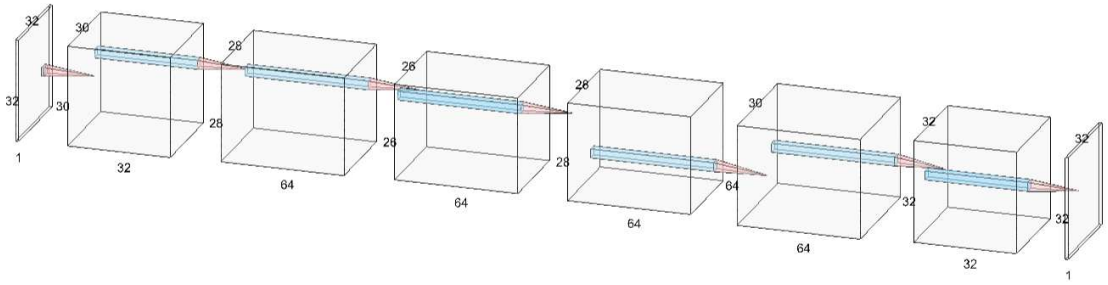
$$ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{20}$$

$$S(x) = \frac{1}{1 + e^{-x}} \tag{21}$$

1.2.2. Konvolucijski autoenkoder

Konvolucijski autoenkoder je jednostavan deterministički model građen od niza konvolucijskih slojeva. U odabranoj arhitekturi mreže, koder je građen od 3 uzastopna konvolucijska sloja s redom 32, 32 i 64 filtara. Veličina pojedinih filtara u svakom od skrivenih slojeva je 3×3 piksela. Dekoder je građen od slojeva u kojima se primjenjuje operacija dekonvolucije u tri sloja s po 64, 32 i 32 filtara. Korak konvolucije je 1. Opisana arhitektura ne smanjuje ukupnu dimenzionalnost latentne reprezentacije na izlazu kodera u odnosu na ulazne podatke, nego ju povećava (engl. *overcomplete autoencoder*). Potencijalni problem s arhitekturama s velikim brojem parametara u skrivenom sloju je mogućnost učenja funkcije identiteta koja bi omogućila mreži da izravno preslika podatke s ulaza na izlaz. Ipak, učenje funkcije identiteta u praksi nije trivijalno, što je opisano u i radu [3]. Za zadatak nenadziranog otkrivanja stršećih vrijednosti na odabranom skupu podataka ovakva arhitektura pokazuje se uspješnijom od arhitektura s tzv. „uskim grlom“ čija je dimenzija latentnih reprezentacija znatno manja od ulaznih podataka (engl. *undercomplete autoencoder*). Mreža s velikim brojem parametara u latentnom sloju sve primjerke rekonstruira znatno bolje, pogreška rekonstrukcije je manja, ali se razlikuje za normalne i stršeće vrijednosti. Kod arhitekture s uskim grlom, model sve primjerke iz skupa za učenje podjednako dobro rekonstruira. Upotreba arhitektura s uskim grlom česta je kod nadziranog i polunadziranog učenja stršećih vrijednosti, pri čemu modeli uče bitne značajke samo normalnih, nekontaminiranih podataka.

Nakon svakog od skrivenih slojeva primijenjena je aktivacijska funkcija zglobnice. Na izlaze posljednjeg sloja kodera primjenjuje se aktivacijska funkcija sigmoide, jer izlazne vrijednosti preslikava u interval [0, 1] kako bi bile normalizirane kao i ulazne slike. Osim toga, na izlaze slojeva primijenjena je i normalizacija mini-grupe (engl. *batch normalization*) koja vrijednosti izlaza sloja za jednu mini-grupu podataka transformira tako da im srednja vrijednost bude 0, a standardna devijacija približno 1. Normalizacija ima učinak stabilizacije procesa učenja i smanjenja broja epoha potrebnih za učenje modela [19].



Sl. 1.4 Ilustracija ulaznih i izlaznih tenzora u konvolucijskom modelu autoenkodera

1.2.3. Varijacijski autoenkoder

Kod standardnih autoenkodera, nije moguće pretpostaviti distribuciju vrijednosti latentnog vektora. Varijacijski autoenkoderi, za razliku od standardnih, ne kodiraju ulazne podatke izravno u vektor latentnih značajki, nego u parametre distribucije iz koje se zatim uzorkuje latentni vektor iz kojeg dekoder može uspješno rekonstruirati ulaze [15]. Rezultat je neprekinut i „gladak" prostor latentnih značajki. Slične vrijednosti latentnog vektora dat će slične rekonstruirane vrijednosti, što ne vrijedi za sve autoenkodere [16]. Primarno su osmišljeni kao generativni modeli (npr. za generiranje slika), ali zbog njihove sposobnosti generiranja kompleksnih distribucija u latentnom prostoru moguće ih je koristiti i za detekciju stršećih vrijednosti. Ideja je primjerke koji su manje vjerojatni u generiranoj distribuciji proglasiti stršećim vrijednostima.

Dekoderski dio modela VAE definiran je funkcijom $p(\mathbf{x}|\mathbf{z})$, koja opisuje distribuciju dekodiranih varijabli uz poznate kodirane varijable. Koderski dio definiran je funkcijom $p(\mathbf{z}|\mathbf{x})$ koja opisuje aposteriornu distribuciju kodiranog latentnog vektora **z** uz poznat vektor **x**. Prema Bayesovom teoremu, vrijedi izraz (22).

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})} \tag{22}$$

Distribucija p(**x**) je često netraktabilna, naročito u visokodimenzionalnom prostoru, pa se p(**z**|**x**) procjenjuje nekom traktabilnom distribucijom q(**z**). Odstupanje ciljne i stvarne distribucije mjeri Kullback-Leiblerova (KL) divergencija opisana izrazom (23). Kako bi se postigla što veća sličnost dviju distribucija, potrebno je minimizirati KL-divergenciju.

$$\min KL(q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{x})) = -\sum q(\boldsymbol{z}) \log \frac{p(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{z})} \tag{23}$$

Uvrštavanjem (22) u izraz za minimizaciju KL divergencije, dolazi se do istovjetnog izraza (24).

$$\begin{aligned} \min KL(q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{x})) &= \min\left[\sum q(\boldsymbol{z}) \log\, p(\boldsymbol{x}|\boldsymbol{z}) - KL(q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))\right] \\ &= \min\left[E_{q(z)} \log p(\boldsymbol{x}|\boldsymbol{z}) - KL(q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))\right] \end{aligned} \tag{24}$$

Prvi dio izraza je očekivanje log-izglednosti $p(\mathbf{x}|\mathbf{z})$ uz $q(\mathbf{z})$. Kako postoji deterministično preslikavanje između vektora $\mathbf{z}$ i izlaza $\hat{\mathbf{x}}$, minimizacija tog očekivanja istovjetna je minimizaciji očekivanja distribucije $p(\mathbf{x}|\hat{\mathbf{x}})$, tj. minimizaciji rekonstrukcijske pogreške između ulaza kodera $\mathbf{x}$ i izlaza dekodera $\hat{\mathbf{x}}$. Drugi dio izraza osigurava da distribucija latentnih vektora $q(\mathbf{z}|\mathbf{x})$ bude što sličnija distribuciji $p(\mathbf{z})$. Uobičajeno je za $p(\mathbf{z})$ pretpostaviti poznatu Gaussovu distribuciju *N(0, 1)* [16].

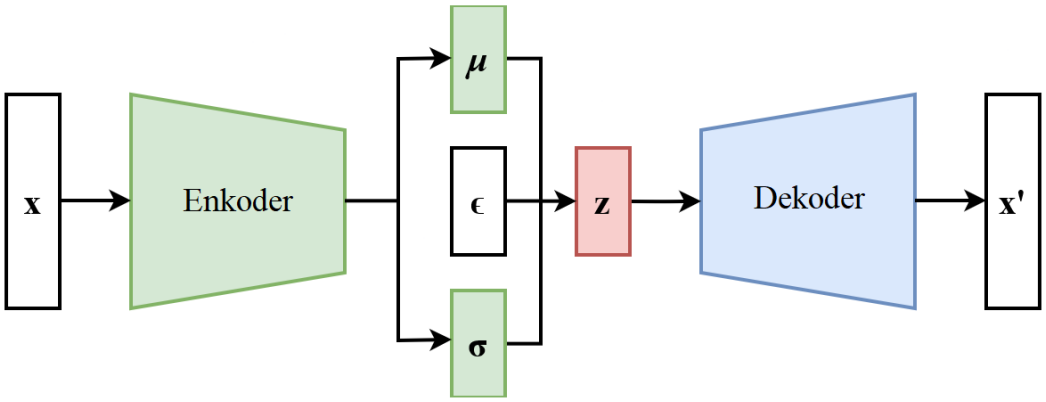
Minimizacija rekonstrukcijske pogreške odgovara ranijoj definiciji autoenkodera. KL-divergencija predstavlja regularizacijski član i usmjerava distribuciju latentnog vektora prema normalnoj. Pojednostavljeni izraz ukupne funkcije gubitka korišten u implementaciji je (25).

$$KL\big(N(\boldsymbol{\mu_x}, \boldsymbol{\sigma_x}), N(0,1)\big) = -\frac{1}{2}\sum_{k=1}^{n}(1 + \log\sigma_k^2 - \mu_k^2 - \sigma_k^2) \tag{25}$$

U izrazu μ_k i σ_k^2 predstavljaju k-tu komponentu vektora srednje vrijednosti, odnosno varijance. Za zadanu veličinu latentnog vektora n, koder će na izlazu dati vektor parametara veličine $2\times n$ koji predstavlja srednje vrijednosti i varijance normalne distribucije za svaku od n značajki. Vrijednosti vektora $\mathbf{z}$ uzorkuju se prema izrazu (26).

$$\boldsymbol{z} = \boldsymbol{\mu}_z + \boldsymbol{\sigma}_z \odot \boldsymbol{\varepsilon} \tag{26}$$

Značajke vektora $\boldsymbol{\varepsilon}$ odabiru se nasumično iz normalne distribucije $N(0, 1)$. Zatim se vrijednosti vektora skaliraju ($\odot$) s varijancom koja je implementirana množenjem vektora po elementima i posmiču za srednje vrijednosti. Taj se postupak naziva reparametrizacijskim trikom te omogućuje da se istovremeno vektor uzorkuje nasumično (nedeterministički) i da ovisi o parametrima distribucije koji se optimiraju postupkom propagiranja unatrag.



Sl. 1.5 Shema modela varijacijskog autoenkodera

Uvođenje KL-divergencije u funkciju gubitka za posljedicu ima veću rekonstrukcijsku pogrešku, stoga je važno odrediti prikladnu vrijednost faktora uz regularizacijski član. Kao mjera uspješnosti rekonstrukcije korištena je ranije opisana mjera SSIM. Ukupni izraz za funkciju gubitka sastavljen od rekonstrukcijskog i regularizacijskog člana opisan je izrazom (27).

$$L = SSIM(\boldsymbol{x}, \boldsymbol{x}') + 0.0001\, KL\big(N(\boldsymbol{\mu_x}, \boldsymbol{\sigma_x}), N(0,1)\big) \tag{27}$$

Koeficijent uz regularizacijski član odabran je u sklopu ovog rada eksperimentalno. Kako je vrijednost rekonstrukcijskog gubitka ograničena na interval [0,1], važno je da vrijednost uz regularizacijski član ne bude prevelika, jer se u tom slučaju previše ograničava sposobnost modela da rekonstruira ulazne podatke te je distribucija previše nalik normalnoj. Koderski dio mreže sastoji se od tri ulančana skrivena sloja. Prva dva sloja su konvolucijska, s po 32 i 64 filtera veličine 3×3 piksela uz korak konvolucije jednak 1. Mape značajki na izlazu drugog konvolucijskog sloja se vektoriziraju i istovremeno dovode na ulaz dva paralelna potpuno povezana sloja. Jedan od slojeva generira srednje vrijednosti distribucije iz koje se uzorkuje latentni vektor. Zbog mogućnosti da mreža generira negativne vrijednosti varijance, drugi potpuno povezani sloj ne daje izravno vrijednosti varijance, nego njezinog logaritma na koji se naknadno primjenjuje eksponencijalna funkcija [16]. Na izlaze potpuno povezanih slojeva ne primjenjuje se nelinearna aktivacijska funkcija, jer često korištene aktivacijske funkcije ograničavaju vrijednosti na određen interval, a parametri na izlazu dekodera općenito mogu biti bilo koji realni brojevi. Na ulaz dekodera dovodi se uzorkovani latentni vektor. Arhitektura dekodera je zrcalna s obzirom na koder. Ulazi se najprije dovode do potpuno povezanog sloja koji transformira reprezentaciju podataka povećavanjem

dimenzionalnosti u oblik iz kojeg je moguće rekonstruirati ulaze, a zatim kroz dva uzastopna dekonvolucijska sloja s po 64 i 32 filtra.

2. Eksperimenti

2.1. Skup podataka

Za zadatak otkrivanja stršećih vrijednosti u slikama s web-stranice [8] odabran je i preuzet skup podataka imena „Detekcija površinskih pukotina“ (engl. *Surface Crack Detection*). Skup sadrži kvadratne trokanalne slike, tj. slike u boji raznih betonskih površina, npr. pločnika, zidova i mostova s raznolikim površinskim strukturama i uvjetima osvjetljenja. Slike su podijeljene u dva razreda s obzirom na to je li na njima vidljivo površinsko oštećenje ili ne. Slike bez oštećenja su označene kao negativne, a slike s oštećenjima kao pozitivne. Dimenzije originalnih slika su 227×227 piksela. U svakom od razreda je po 20.000 primjeraka. Za potrebe evaluacije pristupa za otkrivanje stršećih vrijednosti, iz originalnog skupa podataka izdvojen je podskup slika bez oštećenja koje predstavljaju normalne podatke, dok slike s oštećenjima predstavljaju stršeće vrijednosti. Nekoliko normalnih primjeraka prikazano je na slici 2.1, a na slici 2.2 su dani razni primjeri stršećih primjeraka. Ukupan broj primjeraka u svakom od korištenih podskupa je 2000. Eksperimenti su provedeni na nekoliko verzija skupa podataka koji se međusobno razlikuju prema udjelu stršećih vrijednosti u ukupnom broju primjera. Za učenje i evaluaciju korištene su verzije skupa s 5, 10, 20 i 30% primjeraka slika oštećenja.

Sl. 2.1 Primjerci iz normalnog dijela skupa podataka

Sl. 2.2 Primjerci iz dijela skupa podataka s oštećenjima koji predstavljaju stršeće vrijednosti

Za svaku od varijanti skupa podataka generirane su datoteke *csv* formata s putanjama u lokalnom datotečnom sustavu do primjeraka slika i pripadnim oznakama: 1 za normalne podatke, a -1 za stršeće vrijednosti, kao u primjeru:

```
D:\data\surface_crack_detection\Negative\05912.jpg,1
D:\data\surface_crack_detection\Positive\14698.jpg,-1
```

Kako je riječ o zadatku nenadziranog učenja, algoritmi za učenje zahtijevaju samo slike učitane iz zadanih putanja, bez dodatnih informacija o njihovom sadržaju. Oznake se koriste tek kod evaluacije uspješnosti postupaka, za usporedbu predviđenih i stvarnih stršećih vrijednosti. Putanje i oznake se iz datoteke učitavaju korištenjem metode `read_csv` biblioteke *pandas* u strukturu `dataframe`, prikladnu za daljnju obradu podataka. Slike se iz putanja u strukturi `dataframe` učitavaju korištenjem metoda biblioteke *keras* kako je opisano isječkom koda 2.1. Metodi `load_img` zadaje se putanja do datoteke, željene dimenzije slike, tj. širina, visina i broj kanala, te parametar koji određuje hoće li slika biti učitana u boji ili kao crno-bijela. Metoda `img_to_array` zatim pretvara učitanu sliku u trodimenzionalno *Numpyovo* polje vrijednosti tipa `float32`. Konačno, vrijednosti polja piksela koji predstavljaju sliku su normirane, što ubrzava učenje.

```
from keras.preprocessing.image import load_img, img_to_array
img = img_to_array(load_img(img_path, target_size=(img_w,
img_h, img_ch), color_mode='grayscale')) / 255.
```

Kôd 2.1 Učitavanje u *Numpyovo* polje i normalizacija slike

Učitane slike su crno-bijele, tj. jednokanalne, jer za detekciju oštećenja nije važna boja površine, nego struktura. Visina i širina slike nakon učitavanja je 32 piksela. Slike su znatno smanjene u odnosu na originalne kako bi se ubrzalo učenje algoritama s ograničenim računalnim resursima. Slike ne prikazuju složene strukture i pri smanjivanju ne dolazi do značajnog gubitka informacija. U eksperimentima sa slikama veće visine i širine (64 i 128

piksela), nije uočena bolja točnost, a vrijeme potrebno za učenje je znatno dulje. Slike su spremljene u obliku trodimenzionalnog *Numpyovog* polja dimenzija (32, 32, 1).

Metode strojnog učenja implementirane u sklopu biblioteke *scikit-learn* na ulazu očekuju polje oblika *(broj primjeraka, broj značajki*), te je pojedinačne učitane slike predstavljene trodimenzionalnim poljem potrebno preoblikovati u vektore s 1024 značajke.

```
img = img.reshape(img_w * img_h * img_ch)
```

Metode dubokog učenja implementirane pomoću biblioteke *keras* na ulazu primaju podatke proizvoljnog oblika, stoga učitane slike nije potrebno preoblikovati prije učenja i predikcije. Oblik ulaznih podataka definira se u ulaznom sloju mreže prema obliku ranije učitanih slika.

2.2. Mjere za evaluiranje uspješnosti algoritama

Kod odabira hiperparametara svakog od algoritama prikladnih za skup podataka evaluirane su mjere preciznosti i odziva. Preciznost (engl. *precision*, *P*) se računa prema izrazu (28), a označava udio stvarnih stršećih vrijednosti među primjercima koje model strojnog učenja predviđa kao stršeće. Odziv (engl. *recall*, *R*) označava udio stvarnih stršećih vrijednosti koje model uspješno detektira kao stršeće i opisan je izrazom (29). Preciznost i odziv su fokusirani na stršeće vrijednosti i ne mjere uspješnost prepoznavanja većinskog dijela normalnih podataka.

$$P = \frac{TP}{TP + FP} \tag{28}$$

$$R = \frac{TP}{TP + FN} \tag{29}$$

U izrazima je:

- TP broj točno klasificiranih stršećih vrijednosti (engl. *true positive*),
- FP broj normalnih vrijednosti koje su klasificirane kao stršeće (engl. *false positive*),
- FN broj stršećih vrijednosti koje nisu klasificirane kao stršeće (engl. *false negative*).

Veličina koja kombinira preciznost i odziv i omogućuje izravnu usporedbu uspješnosti modela naziva se F1-mjerom, a računa se kao harmonijska sredina preciznosti i odziva (30). Ako se koristi F1-mjera, pretpostavka je da su preciznost i odziv jednako važni kod evaluacije modela.

$$F1 = \frac{2 \times P \times R}{P + R} \tag{30}$$

Za duboke modele, kod kojih je potrebno odrediti prag koji određuje granicu između normalnih i stršećih vrijednosti, prikazane su PR-krivulje koje ilustriraju kompromis između preciznosti i odziva za različite vrijednosti praga. Vrijednosti pragova postavljene su na svakih 5 percentila vrijednosti funkcije gubitka na odabranom skupu podataka. Modeli s većom površinom ispod PR-krivulje (engl. *area under curve*, *PR-AUC*) su uspješniji u detekciji. Detekcija stršećih vrijednosti može se interpretirati kao klasifikacija s 2 nebalansirana razreda. Idealni bi klasifikator imao preciznost i odziv jednake 1, pa bi krivulja prolazila točkom (1, 1) i bila pravokutna.

2.3. Implementacija i izbor hiperparametara algoritama iz biblioteke *scikit-learn*

2.3.1. Implementacija algoritma izolacijske šume

Implementacija postupka IF dostupna je u sklopu biblioteke *scikit-learn* kao razred `sklearn.ensemble.IsolationForest`. Kod instanciranja primjerka razreda potrebno je zadati broj stabala u ansamblu. Parametar `max_samples` određuje broj primjeraka koji će se koristiti u učenju svakog od stabala. Parametar `max_features` određuje broj značajki prema kojima se može dijeliti skup podataka u stablima. Za oba parametra korištena je pretpostavljena vrijednost `'auto'`, koja podrazumijeva korištenje svih dostupnih primjeraka, odnosno značajki kod izgradnje svakog od stabala odluke. Za parametar `contamination` također je korištena pretpostavljena vrijednost, jer za zadatke nenadziranog učenja nije unaprijed poznat udio stršećih vrijednosti. Pretpostavljena implementacija primjerke za koje je mjera *s(x, n)* veća od 0.5 prepoznaje kao stršeće vrijednosti, kao što je predloženo u originalnom radu [6]. Isječak koda 2.2 opisuje stvaranje i pokretanje modela `IsolationForest`.

```
from sklearn.ensemble import IsolationForest iForest =
IsolationForest(n_estimators=200, max_samples='auto',
contamination='auto')
predictions = iForest.fit_predict(X)
scores = iForest.decision_function(X)
```

Kôd 2.2 Pokretanje detekcije stršećih vrijednosti algoritmom IF iz biblioteke *scikit-learn*

Pozivom metode `fit_predict(X)` gradi se model i vraćaju predikcije za skup `X`. Metoda `decision_function(X)` vraća posmaknutu ranije opisanu mjeru s(**x**, n) za svaki primjerak iz skupa `X`.

Tablica 2.1 Rezultati algoritma IF za različite veličine ansambla na skupu s 5% anomalija

	P	R	F1
50 stabala	0.427	0.838	0.565
100 stabala	0.466	0.875	0.605
200 stabala	0.470	0.885	0.615

Tablica 2.2 Rezultati algoritma IF za različite veličine ansambla na skupu s 10% anomalija

	P	R	F1
50 stabala	0.691	0.765	0.726
100 stabala	0.713	0.783	0.746
200 stabala	0.732	0.782	0.756

Tablica 2.3 Rezultati algoritma IF za različite veličine ansambla na skupu s 20% anomalija

	P	R	F1
50 stabala	0.845	0.539	0.661
100 stabala	0.855	0.548	0.667
200 stabala	0.871	0.550	0.674

Tablica 2.4 Rezultati algoritma IF za različite veličine ansambla na skupu s 30% anomalija

	P	R	F1
50 stabala	0.850	0.319	0.470
100 stabala	0.891	0.313	0.466
200 stabala	0.883	0.366	0.517

Mjere preciznosti, odziva i F1 za svaki od skupova podataka uz različite veličine ansambla prikazane su u tablicama 2.1 – 2.4. Kako je riječ o algoritmu koji koristi puno slučajnosti, rezultati u tablicama dobiveni su usrednjavanjem mjera 5 uzastopnih pokretanja učenja i evaluacije uz isti broj stabala Za veličinu ansambla eksperimentalno je odabrano 200 stabala, jer ta vrijednost hiperparametra daje bolje rezultate nego manje veličine ansambla za sve varijante skupa podataka. Poboljšanja s daljnjim povećanjem broja stabala su neznatna ili ih nema, a vrijeme izvođenja je znatno dulje. Za skupove podataka s manjim udjelom stršećih vrijednosti, modeli imaju nižu preciznost, ali veći odziv. Većina stršećih primjeraka će modelima IF biti otkrivena, ali će među predviđenim stršećim vrijednostima biti velik broj normalnih primjeraka. Za skupove s većim udjelom stršećih vrijednosti, IF može prepoznati anomalije s visokom preciznošću, ali će istovremeno velik dio stršećih vrijednosti ostati neotkriven među normalnim podacima.

Na slici 2.3 izdvojeni su normalni primjerci koje algoritam detektira kao stršeće (FP). Normalne slike detektirane kao stršeće vrijednosti uglavnom su tamniji primjerci. Na slici 2.4 prikazane su stršeće vrijednosti koje algoritam ne detektira (FN) u skupu s 20% stršećih vrijednosti. Slike vizualno ne nalikuju normalnim primjercima i nije jednostavno interpretirati zašto nisu detektirane.

Sl. 2.3 Normalni primjerci koje algoritam IF detektira kao stršeće (FP)

Sl. 2.4 Stršeće vrijednosti koje algoritam IF ne detektira (FN)

2.3.2. Implementacija algoritma LOF

Implementacija algoritma LOF dostupna je kao razred `sklearn.neighbors.LocalOutlierFactor`. Konstruktoru se predaje broj susjednih primjeraka koje postupak uzima u obzir, tj. *k* (parametar `n_neighbors`), kao što je prikazano u kodu 2.3.

```
from sklearn.neighbors import LocalOutlierFactor
lof = LocalOutlierFactor(n_neighbors=200)
predictions = lof.fit_predict(X)
scores = lof.negative_outlier_factor_
```

Kôd 2.3 Pokretanje detekcije stršećih vrijednosti algoritmom LOF iz biblioteke *scikit-learn*

Negativne vrijednosti LOF za primjerke korištene za učenje modela dostupne su u obliku *Numpyovog* polja kroz atribut `negative_outlier_factor_`.

Tablica 2.5 Rezultati za skup podataka s 5% stršećih vrijednosti uz različite vrijednosti parametra *k*

	P	R	F1
k=20	0.221	1.0	0.362
k=50	0.304	1.000	0.466
k=100	0.390	0.960	0.555
k=300	0.543	0.990	0.702
k=500	0.433	1.000	0.604

Tablica 2.6 Rezultati za skup podataka s 10% stršećih vrijednosti uz različite vrijednosti parametra *k*

	P	R	F1
k=20	0.349	0.980	0.515
k=50	0.687	0.690	0.688
k=100	0.744	0.740	0.742
k=300	0.741	0.785	0.762
k=500	0.690	0.995	0.814

Tablica 2.7 Rezultati za skup podataka s 20% stršećih vrijednosti uz različite vrijednosti parametra *k*

	P	R	F1
k=20	0.531	0.855	0.655
k=50	0.676	0.950	0.790
k=100	0.737	0.863	0.795
k=300	0.887	0.765	0.821
k=500	0.919	0.990	0.953

Tablica 2.8 Rezultati za skup podataka s 30% stršećih vrijednosti uz različite vrijednosti parametra *k*

	P	R	F1
k=20	0.661	0.750	0.703
k=50	0.792	0.826	0.809
k=100	0.859	0.785	0.821
k=300	0.961	0.740	0.836
k=500	0.966	0.747	0.842

Tablice 2.5 – 2.8 sadrže rezultate mjere preciznosti i odziva dobivene uz variranje hiperparametra *k* za različite varijante skupa podataka. Skupovi podataka s relativno manjim udjelom stršećih vrijednosti (5% i 10%), najuspješniji su modeli kada u obzir uzimaju 300 susjednih točaka u okruženju svake točke, dok su za skupove podataka s većim udjelom anomalija (20% i 30%) prikladniji modeli s hiperparametrom *k* vrijednosti 500. Obje su vrijednosti parametra *k* velike u odnosu na ukupan broj slika u odabranom skupu podataka, što znači da algoritam nema svojstvo lokalnosti i pronalazi samo globalne stršeće vrijednosti. Ovakvo ponašanje svojstveno je skupovima podataka s puno šuma u normalnim podacima, kod kojih su vrijednosti normalnih značajki raspršene i imaju veća odstupanja. Manje

vrijednosti hiperparametra *k* rezultiraju time da model više normalnih primjeraka klasificira kao stršeće vrijednosti.

Za usporedbu uspješnosti algoritma u odnosu na ostale postupke, za svaku verziju skupa podataka odabran je broj susjednih primjeraka koji daje najbolje rezultate s obzirom na preciznost i odziv.

Slika 2.5 prikazuje nasumično odabrane normalne primjerke iz skupa s 20% anomalija koje algoritam prepoznaje kao stršeće. Slike su uglavnom tamnije i prikazuju površine s više teksture u odnosu na ostatak normalnog dijela skupa. Očekivano je da algoritam više griješi na takvim podacima. Na slici 2.6 su stršeće vrijednosti koje algoritam ne detektira. Oštećenja koja prikazuju su relativno manja i takvi primjerci su po sadržaju uistinu bliži normalnima.

Sl. 2.5 Normalni primjerci koje algoritam LOF detektira kao stršeće (FP)

Sl. 2.6 Stršeće vrijednosti koje algoritam LOF ne detektira (FN)

2.3.3. Implementacija algoritma OC-SVM

OC-SVM implementiran je kao razred `sklearn.svm.OneClassSVM`, prikazano u kodu 2.4.

```
from sklearn.svm import OneClassSVM
svm = OneClassSVM(kernel='rbf')
predictions = svm.fit_predict(X)
scores = svm.decision_function(X)
```

Kôd 2.4 Pokretanje detekcije stršećih vrijednosti algoritmom OC-SVM iz biblioteke *scikit-learn*.

Pri inicijalizaciji objekta konstruktoru je osim tipa jezgrene funkcije (`kernel`) moguće zadati i opcionalan parametar ν (`nu`) koji određuje maksimalan mogući udio stršećih vrijednosti odvojenih hiperravninom te maksimalan udio primjeraka koji predstavljaju

potporne vektore. Kako zadatak nenadziranog otkrivanja stršećih vrijednosti ne pretpostavlja znanje o stvarnom broju stršećih vrijednosti, pri učenju modela OC-SVM korištena je pretpostavljena vrijednost parametra ν=0.5 za sve skupove podataka. Mjera za određivanje i usporedbu stršećih vrijednosti je udaljenost od hiperravnine s predznakom i dostupna je u obliku metode `decision_function(X)`. Udaljenosti su pozitivne za normalne podatke i negativne za stršeće vrijednosti.

Rezultati dobiveni učenjem algoritma s opisanim hiperparametrima prikazani su tablicama 2.9 i 2.10. U eksperimentima je kao jezgrena funkcija odabrana Gaussova radijalna bazna funkcija. Učenje modela s jezgrom *rbf* vremenski je zahtjevnije nego s jednostavnom linearnom baznom funkcijom, ali rezultira boljom preciznošću i odzivom. Za sve inačice skupa podataka vrijednost odziva je 1, što znači da algoritam uspješno prepozna sve stvarne stršeće vrijednosti. Nedostatak postupka je niska preciznost, jer model značajan dio normalnih primjeraka svrstava u stršeće vrijednosti, kako bi ukupan broj detekcija iznosio ν**n*. Preciznost je zato očekivano manja za skupove podataka s manjim stvarnim udjelom anomalija. Iako bi se ovim pristupom izdvojile sve stršeće vrijednosti iz skupa, izgubio bi se i značajan dio korisnih podataka.

Tablica 2.9 Vrijednosti P, R i F1 za algoritam OC-SVM s linearnom jezgrenom funkcijom

	P	R	F1
5% anomalija	0.099	0.990	0.180
10% anomalija	0.199	0.995	0.332
20% anomalija	0.391	0.978	0.332
30% anomalija	0.574	0.957	0.718

Tablica 2.10 Vrijednosti P, R i F1 za algoritam OC-SVM s jezgrenom funkcijom *rbf*

	P	R	F1
5% anomalija	0.099	1.000	0.182
10% anomalija	0.200	1.000	0.334
20% anomalija	0.399	1.000	0.571
40% anomalija	0.601	1.000	0.751

Za skup s 20% stršećih vrijednosti, na slici 2.7 je izdvojeno nekoliko normalnih primjeraka koje OC-SVM prepoznaje kao stršeće (FP). Model s niskom preciznošću otkriva stršeće vrijednosti te velik udio normalnih slika klasificira kao anomalije, što se očituje i na nekim izdvojenim slikama koje ne sadrže mnogo šuma, a svejedno su detektirane kao anomalije.

Sl. 2.7 Normalni primjerci koje algoritam OC-SVM detektira kao stršeće (FP)

Ne postoje stršeće vrijednosti koje implementacija OC-SVM s jezgrom *rbf* ne otkriva, tj, odziv je jednak 1.

2.4. Implementacija dubokih modela

Duboki modeli za detekciju stršećih vrijednosti implementirani su pomoću programskog sučelja Pythonovih biblioteka *tensorflow* i *keras* za duboko učenje. Biblioteka *keras* sadrži gotove implementacije slojeva i modela sa sučeljima koja omogućuju jednostavno definiranje proizvoljnog modela i procesa učenja.

2.4.1. Funkcija gubitka i optimizacijski postupak

Implementacija mjere SSIM dostupna je u sklopu biblioteke *tensorflow*. Funkciji se osim grupa ulaznih i izlaznih slika (parametri `y_true`, `y_pred`) zadaje i treći parametar koji označava dinamički raspon slike, tj. maksimalan dozvoljen raspon vrijednosti piksela.

Implementacija funkcije gubitka temeljene na mjeri SSIM opisana je sljedećim isječkom koda.

```
def SSIMLoss(y_true, y_pred):
    return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred,
    1.0))
```

Kao optimizacijska tehnika učenja autoenkodera korišten je učinkovit iterativni postupak adaptivnih momenata (engl. *Adaptive Moment Estimation,* Adam) koji predstavlja nadogradnju optimizacije gradijentnim spustom. Adam prati prosječno kretanje gradijenata i kvadrata gradijenata funkcije gubitka uz eksponencijalno zaboravljanje. Razlika u korekciji težina u odnosu na uobičajeni postupak gradijentnog spusta je korištenje uprosječenog gradijenta umjesto izravno izračunatog gradijenta za korak *t* [17]. Uprosječeni gradijent (m) i kvadrat gradijenta (v) u koraku *t* računaju se prema izrazima (31) i (32).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{31}$$

$$v_t = \beta_2 v_t - 1 + (1 - \beta_2) g_t^2 \tag{32}$$

β_1 i β_2 su konstante, a vrijednosti vektora $\boldsymbol{m}_t$ i $\boldsymbol{v}_t$ inicijalno iznose 0. Korištenje prosječnog gradijenta može ubrzati postupak učenja [17].

Vrijednosti težina mreže $\boldsymbol{\theta}_{t+1}$ za korak učenja *t+1* korigiraju se prema izrazu (33).

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\frac{v_t}{1-\beta_2^t} + \epsilon}} \frac{m_t}{1-\beta_2^t} \tag{33}$$

α je oznaka stope učenja, a ϵ konstanta za postizanje numeričke stabilnosti. Za parametre β_1, β_2 i ϵ odabrane su vrijednosti predložene u izvornom radu.

Optimizacijski postupak Adam je implementiran u biblioteci *keras* razredom `keras.optimizers.Adam`.

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4,
beta_1=0.9, beta_2=0.999, epsilon=1e-07)
```

2.4.2. Učitavanje podataka za učenje dubokih modela

Podaci na ulazu dubokih mreža su u obliku mini-grupa veličine 8. Kako je cilj autoenkodera rekonstruirati ulazni podatak, za učenje mu se zadaju jednake vrijednosti za ulazne (**X**) i ciljne (**y**) podatke. Na kraju svake epohe, koja označava jedno prikazivanje svih primjeraka iz skupa za učenje mreži, redoslijed učitavanja podataka se mijenja nasumično, da bi se smanjila mogućnost da se model prenauči na ulazne podatke u određenom redoslijedu. Učitavanje podataka implementirano je nasljeđivanjem razreda `keras.utils.Sequence` i nadjačavanjem postojeće metode `__get_data` tako da iz strukture `dataframe` učitava mini-grupe (engl. *batch*) ulaznih i ciljnih podataka, tj. slika.

2.4.3. Implementacija konvolucijskog modela

Konvolucijski model implementiran je koristeći funkcijsko sučelje biblioteke *keras* (engl. *functional API*) kao primjerak razreda `keras.Model`, prema kodu 2.5.

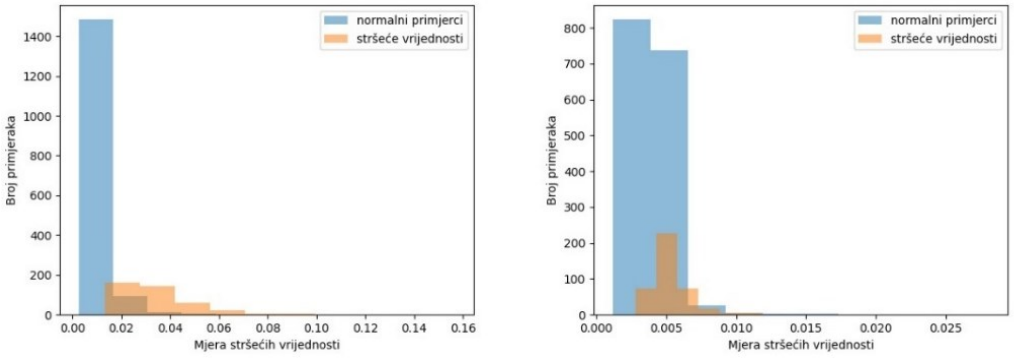
```
def build_CAE(input_shape=(32, 32, 1)):
    # Encoder
    inputs = Input(shape=input_shape)
    code = Conv2D(filters=32, kernel_size=3,
activation='relu')(inputs)
```

```
    code = BatchNormalization()(code)
    code = Conv2D(filters=64, kernel_size=3,
activation='relu')(code)
    code = BatchNormalization()(code)
    code = Conv2D(filters=64, kernel_size=3,
activation='relu')(code)
    code = BatchNormalization()(code)
    # Decoder
    decoded = Conv2DTranspose(64, kernel_size=3,
activation='relu')(code)
    decoded = BatchNormalization()(decoded)
    decoded = Conv2DTranspose(64, kernel_size=3,
activation='relu')(decoded)
    decoded = BatchNormalization()(decoded)
    decoded = Conv2DTranspose(32, kernel_size=3,
activation='relu')(decoded)
    decoded = BatchNormalization()(decoded)
    outputs = Conv2DTranspose(1, kernel_size=3,
padding='same',
activation='sigmoid')(decoded)

    return Model(inputs, outputs)
```

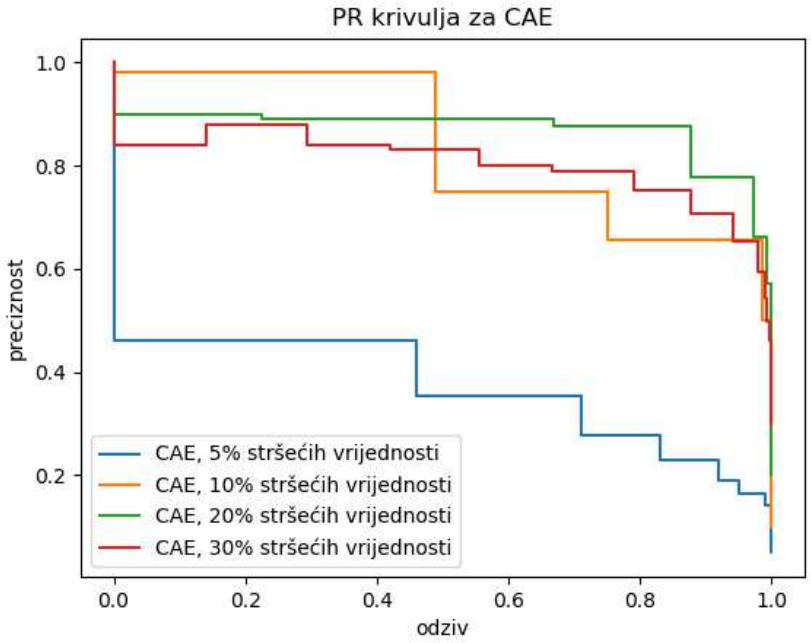
Kôd 2.5 Funkcija za stvaranje novog primjerka modela CAE

Hiperparametri su odabrani pretragom po rešetci (engl. *grid search*): modeli su učeni kroz 10 epoha uz stopu učenja 10^{-4} i veličinu mini-grupe 8. Za znatno veći broj epoha, model se prenauči na relativno malom skupu podataka za učenje, uspješnije rekonstruira i anomalije te djelomično gubi sposobnost razlikovanja normalnih podataka od stršećih.



Sl. 2.8 Histogram mjere stršećih vrijednosti za model treniran na 10 epoha (lijevo) i 50 epoha (desno)

Na slici 2.8 su prikazani rezultati modela učenih na podacima kontaminiranim s 10% stršećih vrijednosti. Lijevi histogram odgovara predikcijama modela učenog na 10 epoha, a desni na 50 epoha. Plavom bojom označena je raspodjela rekonstrukcijskog gubitka za normalne podatke, a narančastom za stršeće vrijednosti.



Sl. 2.9 PR-krivulja za 4 inačice modela CAE

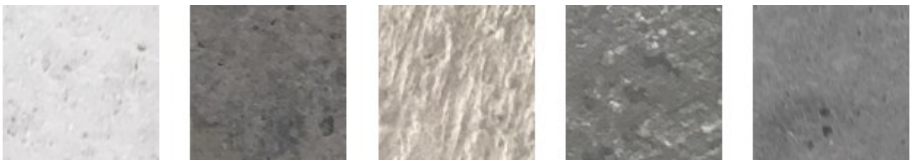
Iz PR-krivulje prikazane slikom 2.9 moguće je zaključiti kako je najuspješniji model CAE učen i vrednovan na podacima s 10% pozitivnih primjeraka. Za tu verziju skupa podataka, najveća je površina ispod PR-krivulje. Model je nešto manje uspješan kod detekcije s 20% ili 30% stršećih vrijednosti. S najmanjim udjelom stršećih vrijednosti (5%) funkcionira približno kao nasumični detektor i nije koristan za detekciju. Za usporedbu s ostalim algoritmima, odabrane su vrijednosti pragova koje maksimiziraju F1mjeru, tj. rezultiraju najboljim omjerom odziva i preciznosti.

Tablica 2.11 sadrži najbolje rezultate modela CAE za svaku od inačica skupa podataka. Model ima malu preciznost za skup podataka s 5% stršećih vrijednosti i nije koristan kao detektor. Na skupovima s većim udjelom anomalija, modeli su znatno uspješniji.

Tablica 2.11 Rezultati modela CAE za skupove podataka s različitim udjelom stršečih vrijednosti

	P	R	F1
5% anomalija	0.355	0.710	0.473
10% anomalija	0.657	0.985	0.788
20% anomalija	0.878	0.878	0.878
30% anomalija	0.774	0.903	0.833

Većina izdvojenih primjeraka FP na slici 2.10 su tamnije boje ili imaju više detalja na površini. Izdvojeni primjerci FN na slici 2.11 uglavnom sadrže manja oštećenja.



Sl. 2.10 Normalni primjerci koje algoritam CAE detektira kao stršeće (FP)

Sl. 2.11 Stršeće vrijednosti koje algoritam CAE ne detektira (FN)

2.4.4. Implementacija varijacijskog autoenkodera

Model VAE, osim postojećih slojeva biblioteke *keras*, sadrži i prilagođeni sloj izveden iz razreda `Layer` za uzorkovanje latentnog vektora iz parametara distribucije. Implementacija je dana u isječku koda 2.6.

```
class Sampling(Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon =
            tf.keras.backend.random_normal(shape=(batch,
            dim))
```

```
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

Kôd 2.6 Implementacija sloja za uzorkovanje iz distribucije koju generira koder

Zbog nesekvencijalne arhitekture i složenije funkcije gubitka, koder i dekoder su implementirani kao dva zasebna modela, prema kodu 2.7.

```
def build_encoder(latent_dim, input_shape=(32, 32, 1)):
    inputs = Input(shape=input_shape)
    x = Conv2D(32, 3, activation="relu", strides=1,
padding="same")(inputs)
    x = Conv2D(64, 3, activation="relu", strides=1,
padding="same")(x)
    x = Flatten()(x)
    z_mean = Dense(latent_dim, name='z_mean')(x)
    z_log_var = Dense(latent_dim, name='z_log_var')(x)
    z = Sampling()([z_mean, z_log_var])
    encoder = Model(inputs, [z_mean, z_log_var, z])
    return encoder

def build_decoder(latent_dim):
    latent_inputs = Input(shape=(latent_dim,))
    x = Dense(32 * 32 * 32, activation="relu")(latent_inputs)
    x = Reshape((32, 32, 32))(x)
    x = Conv2DTranspose(64, 3, activation="relu", strides=1,
padding="same")(x)
    x = Conv2DTranspose(32, 3, activation="relu", strides=1,
padding="same")(x)
    decoder_outputs = Conv2DTranspose(1, 3,
padding="same")(x)
    decoder = Model(latent_inputs, decoder_outputs,
name="decoder")
    return decoder
```

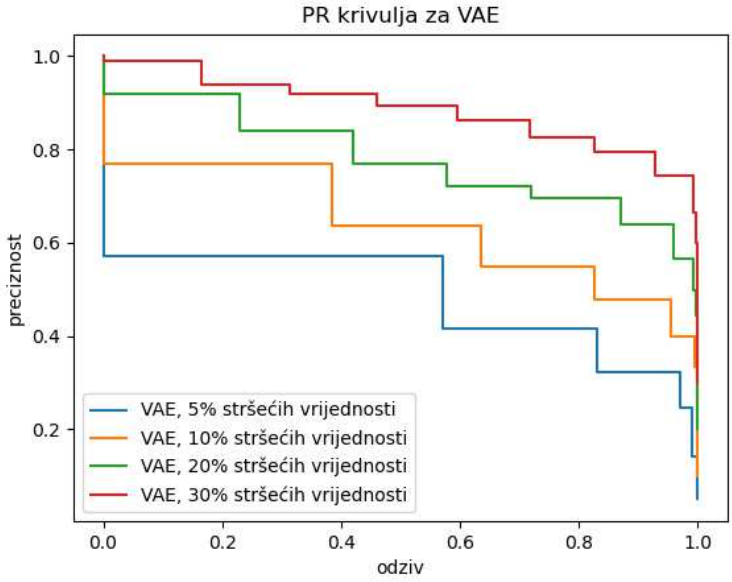
Kôd 2.7 Funkcije za kreiranje koderskog i dekoderskog dijela modela VAE

Tablica 2.12 Utjecaj dimenzije latentnog vektora modela VAE na mjere P, R i F1

	P	R	F1
100	0.068	0.085	0.076
200	0.640	0.960	0.768
300	0.696	0.870	0.773
500	0.696	0.870	0.773

Tablica 2.12 sadrži rezultate modela učenih i evaluiranih na skupu podataka s 20% stršećih vrijednosti s različitim dimenzijama latentnog vektora. Modeli su učeni uz stopu učenja 10^{-3} kroz 10 epoha. Najuspješniji je model s 300 značajki u latentnom vektoru. Uz manje značajki, model gotovo uopće ne razlikuje stršeće od normalnih vrijednosti. Povećanje broja značajki u vektoru iznad 300 ne povećava sposobnost modela da detektira stršeće vrijednosti, a vrijeme učenja se znatno povećava zbog povećanja broja parametara potpuno povezanih slojeva.

Na slici 2.12 je prikaz PR krivulje za četiri modela VAE učenih na različitim inačicama skupa podataka. Model VAE uspješniji je u detekciji stršećih vrijednosti za skupove s većim udjelom anomalija, što se na grafu očituje većom površinom ispod krivulje. S povećanjem udjela stršećih vrijednosti, krivulja se približava krivulji idealnog klasifikatora. Kao i kod modela CAE, za usporedbu s ostalim algoritmima, odabrane su vrijednosti pragova koje maksimiziraju F1-mjeru.



Sl. 2.12 PR krivulja za 4 inačice modela VAE

Tablica 2.13 Rezultati modela VAE za skupove podataka s različitim udjelima stršećih vrijednosti

	P	R	F1
5% anomalija	0.570	0.570	0.570
10% anomalija	0.635	0.635	0.635
20% anomalija	0.696	0.870	0.773
30% anomalija	0.827	0.827	0.827

U tablici 2.12 izdvojeni su najbolji rezultati modela VAE za svaku od inačica skupa podataka. Preciznost i odziv povećavaju se s povećanjem udjela stršećih vrijednosti u podacima.

Izdvojeni normalni podaci prikazani na slici 2.13 iz skupa podataka s 20% anomalija koje model detektira kao stršeće pokazuju neravnije površine s više detalja. Stršeće vrijednosti koje model ne detektira na slici 2.14 su, slično kao za druge postupke, primjerci s manjim oštećenjima.

Sl. 2.13 Normalni primjerci koje algoritam VAE detektira kao stršeće (FP)

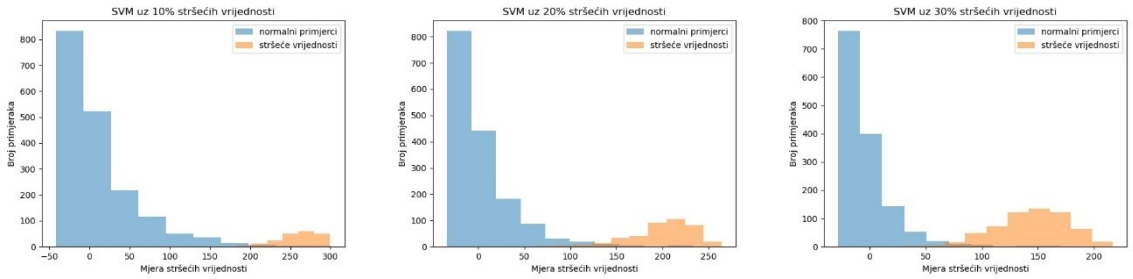
Sl. 2.14 Stršeće vrijednosti koje algoritam VAE ne detektira (FN)

3. Usporedba rezultata

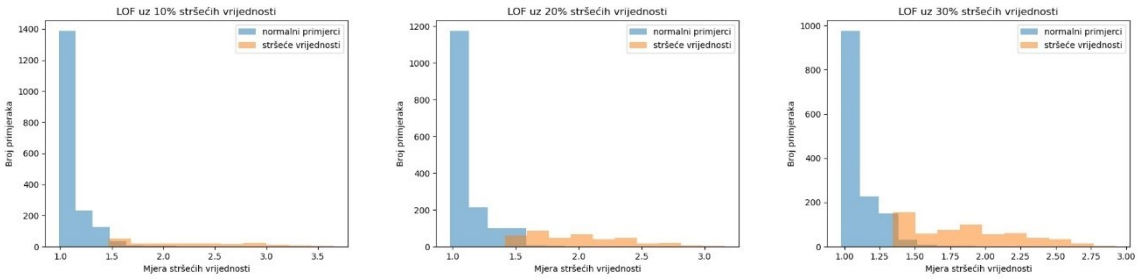
Histogramima na slikama 3.1 – 3.5 prikazana je raspodjela pripadnih mjera za normalne i stršeće vrijednosti sa skupovima podataka s 10%, 20% i 30% anomalija za svaki od implementiranih postupaka. Za algoritme iz biblioteke *scikit-learn*, prikazane su vrijednosti ugrađene mjere stršećih vrijednost svojstvene pojedinom algoritmu. Za duboke modele CAE i VAE, histogrami prikazuju vrijednosti funkcije gubitka rekonstrukcije temeljene na mjeri SSIM. Za vizualizaciju rezultata svakog od postupaka odabran je model s ranije opisanim hiperparametrima koji maksimiziraju uspješnost detekcije. Rezultati za OC-SVM su obrnuti od stvarnih izlaza koje daje implementacija algoritama zbog jednostavnije usporedbe s rezultatima ostalih algoritama.

Stupci histograma stršećih vrijednosti su za svaki od algoritama i skupova podataka pomaknuti udesno u odnosu na stupce normalnih primjerka, što znači da svaki od postupaka u nekoj mjeri uspješno razlikuje normalne podatke od anomalija. Prema histogramima, algoritam OC-SVM najuspješnije razdvaja normalne i stršeće podatke, no u primjeni ima manju preciznost jer je prag postavljen prenisko i velik udio normalnih podataka detektira kao stršeće.

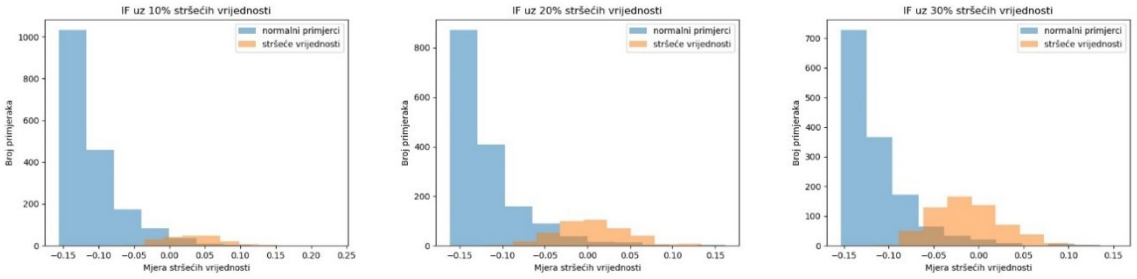
Na slikama 3.4 i 3.5 vidljivo je kako duboki modeli uspješnije prepoznaju stršeće vrijednosti kada je njihov udio u skupu podataka veći.



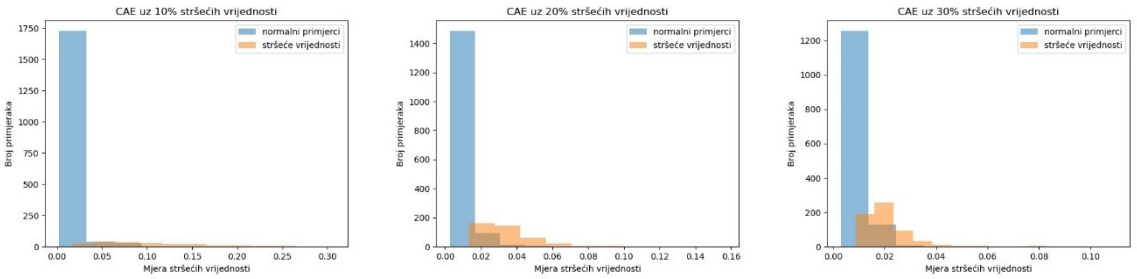
Sl. 3.1 Histogrami mjere anomalija za algoritam OC-SVM za skupove s 10%, 20% i 30% stršećih vrijednosti



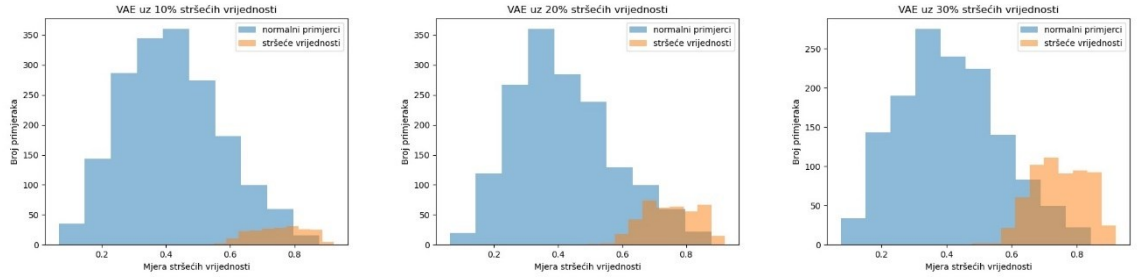
Sl. 3.2 Histogrami mjere anomalija za algoritam LOF za skupove s 10%, 20% i 30% stršećih vrijednosti



Sl. 3.3 Histogrami mjere anomalija za algoritam IF za skupove s 10%, 20% i 30% stršećih vrijednosti



Sl. 3.4 Histogrami mjere anomalija za algoritam CAE za skupove s 10%, 20% i 30% stršećih vrijednosti



Sl. 3.5 Histogrami mjere anomalija za algoritam VAE za skupove s 10%, 20% i 30% stršećih vrijednosti

Tablica 3.1 Rezultati na skupu podataka s 5% stršećih vrijednosti

	P	R	F1
OC-SVM	0.099	1.0	0.182
LOF (k=300)	0.543	0.990	0.702
IF	0.470	0.885	0.615
CAE	0.355	0.710	0.473
VAE	0.570	0.570	0.570

Tablica 3.2 Rezultati na skupu podataka s 10% stršećih vrijednosti

	P	R	F1
OC-SVM	0.200	1.000	0.334
LOF (k=300)	0.805	0.970	0.880
IF	0.730	0.782	0.756
CAE	0.657	0.985	0.788
VAE	0.635	0.635	0.635

Tablica 3.3 Rezultati na skupu podataka s 20% stršećih vrijednosti

	P	R	F1
OC-SVM	0.399	1.000	0.571
LOF (k=500)	0.919	0.990	0.953
IF	0.871	0.550	0.674
CAE	0.878	0.878	0.878
VAE	0.696	0.870	0.773

Tablica 3.4 Rezultati na skupu podataka s 30% stršećih vrijednosti

	P	R	F1
OC-SVM	0.601	1.000	0.751
LOF (k=500)	0.966	0.747	0.842
IF	0.883	0.366	0.517
CAE	0.774	0.903	0.833
VAE	0.827	0.827	0.827

U tablicama 3.1 – 3.4 su najbolji rezultati dobiveni s 5 implementiranih algoritama otkrivanja stršećih vrijednosti za 4 inačice skupa podataka: s 5%, 10%, 20% i 30% stršećih vrijednosti. Za prve tri inačice skupa podataka, očekivano najlošije rezultate daje algoritam OC-SVM koji uvijek 50% cijelog skupa podataka proglašava anomalijama. Preciznost se povećava s povećanjem stvarnog udjela anomalija. Za skup s 30% stršećih vrijednosti, najlošiji rezultat postiže algoritam IF za koji odziv drastično opada s povećanjem kontaminacije skupa podataka. Algoritam LOF s nelokalnim pristupom, tj. dovoljno velikim brojem susjeda k daje najbolji F1 rezultat za sve četiri inačice skupa podataka. Za skupove s 20% i 30%

stršećih vrijednosti, rezultati dobiveni dubokim modelima CAE i VAE su približno jednako dobri kao rezultati algoritma LOF. Dakle, ispitani duboki nenadzirani modeli nisu primjenjivi na skupove s malim udjelima stršećih vrijednosti.

Zaključak

Postoje brojni i raznoliki postupci strojnog učenja za otkrivanje stršećih vrijednosti. Kako se definicija stršeće vrijednosti znatno razlikuje za različite skupove podataka, svaki od algoritama pronalazi primjenu u nekom području te nije moguće proglasiti neki postupak superiornim za svaki zadatak. Nenadzirano otkrivanje stršećih vrijednosti, bez izdvojenog podskupa normalnih podataka za učenje je osobito zahtjevno. Opisani su hiperparametri i rezultati 5 modela strojnog i dubokog učenja: SVM, IF, LOF, dubokog i varijacijskog autoenkodera. Za odabrani skup malih slika namijenjenih detekciji površinskih oštećenja, koji sadrži dosta šuma i u primjercima bez anomalija, gotovo je nemoguće savršeno odvojiti normalne od stršećih vrijednosti nenadziranim algoritmima. Najuspješnijim u detekciji uz različite udjele stršećih vrijednosti pokazao se algoritam LOF s velikim parametrom k koji ne pronalazi lokalne stršeće vrijednosti. Sljedeći korak u rješavanju opisanog problema bio bi istražiti pobliže slične pristupe koji na različite načine određuju faktor lokalnih stršećih vrijednosti. Postupak SVM ne ostvaraje dobre rezultate za zadatak nenadziranog učenja, jer prevelik udio normalnih podataka detektira kao stršeće. Kompleksni algoritmi dubokog učenja, autoenkoderi, primjenjivi su tek za detekciju kada je udio stršećih vrijednosti znatan, tj. veći od 20%. Iako su konvolucijske mreže često korištene u zadacima sa slikovnim podacima, pronalaženje ispravnih hiperparametara je iznimno zahtjevno. Za zadatak nenadziranog učenja, uspješniji se pokazuju modeli CAE i VAE koji ne komprimiraju ulazne podatke u latentne značajke malih dimenzija prije rekonstrukcije. Konačno, rezultate metoda dubokog učenja je zbog kompleksnosti arhitekture teško interpretirati te ih treba koristiti za uistinu izazovne zadatke s velikim slikama, uz puno značajki i primjeraka, kada jednostavniji postupci nisu upotrebljivi.

Literatura

[1] Pang, G., Shen, C., Cao, L., Hengel, A. V. D. *Deep learning for anomaly detection: A review*, ACM Computing Surveys (CSUR) 54,2 (2021), str. 1-38.

[2] Xia, Y., Cao, X., Wen, F., Hua, G., Sun, J. *Learning discriminative reconstructions for unsupervised outlier removal,* Proceedings of the IEEE International Conference on Computer Vision, str. 1511-1519.

[3] Yong, B. X., Brintrup, A. *Do autoencoders need a bottleneck for anomaly detection?*, arXiv, (2022, veljača). Poveznica: https://www.arxiv.org/abs/2202.12637; pristupljeno 3. lipnja 2022.

[4] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. *Image quality assessment: from error visibility to structural similarity*, IEEE transactions on image processing, 13,4 (2004), str. 600-612.

[5] Swati, A. G., *MSE Vs SSIM*, International Journal of Scientific & Engineering Research 4,7 (2013), str. 930-934.

[6] Liu, F. T., Ting, K. M., Zhou, Z. H. *Isolation forest*, Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (2008), str. 413-422.

[7] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Duchesnay, E. *Scikit-learn: Machine learning in Python. the Journal of machine learning research*, 12 (2011), str. 2825-2830.

[8] Pandian, A. *Surface Crack Detection*, Kaggle, (2019). Poveznica: https:// www.kaggle.com/datasets/arunrk7/surface-crack-detection; pristupljeno 4. lipnja 2022.

[9] Goodfellow, I., Bengio, Y., Courville, A. *Deep learning*. 1. izdanje. MIT press, 2016.

[10] Schölkopf, B., Williamson, R. C., Smola, A., Shawe-Taylor, J., Platt, J. *Support vector method for novelty detection*, Advances in neural information processing systems, 12 (1999)

[11] Boser, B. E., Guyon, I. M., Vapnik, V. N. *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on Computational learning theory (1992), str. 144-152.

[12] Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. *LOF: identifying density-based local outliers*, Proceedings of the 2000 ACM SIGMOD international conference on Management of data (2000), str. 93-104.

[13] Tiao, L. *A Tutorial on Variational Autoencoders with a Concise Keras Implementation*, (2021, rujan). Poveznica: https:// tiao.io/post/tutorial-on-variational-autoencoders-with-a-concise-keras-implementation; pristupljeno 5. lipnja 2022.

[14] Yao, R., Liu, C., Zhang, L., Peng, P. *Unsupervised anomaly detection using variational auto-encoder based feature extraction*, 2019 IEEE International Conference on Prognostics and Health Management, (2019), str. 1-7.

[15] Agmon, A. *Hands-on Anomaly Detection with Variational Autoencoders*, Towards Data Science, (2021, srpanj). Poveznica: https://towardsdatascience.com/hands-on-

anomaly-detection-with-variational-autoencoders-d4044672acd5; pristupljeno 6. lipnja 2022.

[16] Jordan, J. *Variational autoencoders*, (2018, ožujak). Poveznica: https://www.jeremyjordan.me/variational-autoencoders/#:~:text=A%20variational%20autoencoder%20(VAE)%20provides,distribution%20for%20each%20latent%20attribute.; pristupljeno 6. lipnja 2022.

[17] *Intuition of Adam Optimizer*, GeeksforGeeks, (2020, listopad). Poveznica: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/; pristupljeno 7.lipnja 2022.

[18] Rajaram, A. *Isolation Forest Outlier Detection Simplified*, Medium, (2022, siječanj). Poveznica: https://medium.com/codex/isolation-forest-outlier-detection-simplified-5d938548bb5c; pristupljeno 10. lipnja 2022.

[19] Brownlee, J. *A Gentle Introduction to Batch Normalization for Deep Neural Networks*, Machine Learning Mastery, (2019, siječanj). Poveznica: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/; pristupljeno 10. lipnja 2022.

[20] Alghushairy, O., Alsini, R., Soule, T., Ma, X. *A review of local outlier factor algorithms for outlier detection in big data streams*, Big Data and Cognitive Computing, 5,1 (2020).

Sažetak

Stršeće vrijednosti su primjerci u skupu podataka koji po vrijednosti nekih značajki odstupaju od ostatka skupa. Otkrivanje i uklanjanje stršećih vrijednosti je važan korak u predobradi podataka, te su u tu svrhu razvijene brojne metode strojnog i dubokog učenja. Osobito je izazovan zadatak nenadzirano otkrivanje stršećih vrijednosti. U ovom su radu, na primjeru skupa podataka betonskih površina s oštećenjima, evaluirane neke od najčešće korištenih metoda otkrivanja anomalija u podacima: lokalni faktor stršećih vrijednosti (LOF), stroj s potpornim vektorima (SVM) te izolacijska šuma (IF). Osim toga, opisani su i najbolji rezultati dobiveni s dva duboka modela za otkivanje stršećih vrijednosti na temelju uspješnosti rekonstrukcije: konvolucijskim autoenkoderom i varijacijskim autoenkoderom. Najuspješnijim algoritmom na ovom problemu pokazao se LOF.

Summary

Outliers are data points whose features deviate significantly from the majority in a dataset. Outlier detection and removal is an important step in data preprocessing and many machine learning and deep learning approaches have been developed to solve this problem. Unsupervised anomaly detection is especially challenging. In this paper, a dataset containing images of concrete surfaces with cracks is used to evaluate some of the most popular machine learning algorithms for anomaly detection: local outlier factor (LOF), support vector machine (SVM) and isolation forest (IF). Additionaly, the best results obtaned from two deep learning reconstruction-based approaches (convolutional autoencoder and variational autoencoder) are described and compared to the results of traditional approaches. LOF was shown to be the most successful algorithm on this problem.

Skraćenice

SVM	*Support Vector Machine*	stroj s potpornim vektorima
VAE	*Variational Autoencoder*	varijacijski autoenkoder
CAE	*Convolutional Autoencoder*	konvolucijski autoenkoder
LOF	*Local Outlier Factor*	faktor lokalnih stršećih vrijednosti
IF	*Isolation Forest*	izolacijska šuma
SSIM	*Structural Similarity Index Measure*	mjera indeksa strukturalne sličnosti
MSE	*Mean Squared Error*	srednje kvadratno odstupanje
Adam	*Adaptive Moments*	postupak promjenjivih momenata
rbf	*radial basis function*	Gaussova radijalna bazna funkcija
FP	*false positives*	detektirani normalni primjerci
FN	*false negatives*	nedetektirane stršeće vrijednosti
TP	*true positives*	detektirane stršeće vrijednosti