SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2068

PREPOZNAVANJE PLOČICA DRUŠTVENE IGRE CARCASSONNE

Lovro Nidogon

Zagreb, lipanj, 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 2068

Pristupnik:	**Lovro Nidogon (0036551310)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Siniša Šegvić

Zadatak: **Raspoznavanje pločica društvene igre Carcassonne**

Opis zadatka:

Društvena igra Carcassonne sastoji se od pločica koje se mogu razvrstati u konačni broj razreda. Pločice koje pripadaju istom razredu mogu sadržavati različite dekoracije pa njihovo strojno raspoznavanje predstavlja zanimljiv izazov. U okviru rada, potrebno je odabrati okvir za unatražno automatsko diferenciranje te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati metode za ravninska projekcijska preslikavanja te raspoznavanje slika. Sakupiti i označiti skupove slika za učenje, validaciju i ispitivanje. Oblikovati postupke za pronalaženje i raspoznavanje pločica u slikama igre. Komentirati i vrednovati generalizacijsku uspješnost postupaka. Predložiti smjerove budućeg rada. Radu priložiti izvorni kod razvijenih postupaka. Omogućiti reproduciranje eksperimenata potrebnim objašnjenjima i dokumentacijom. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 23. lipnja 2025.

Zahvaljujem prof. dr. sc. Siniši Šegviću i asistentu univ. mag. ing. Ivanu Martinoviću na pomoći tijekom izrade završnog rada. Zahvaljujem roditeljima i prijateljima na bezuvjetnoj potpori tijekom sve tri godine studiranja.

Sadržaj

1. Uvod

U digitalno doba, razni oblici pametnih uređaja postaju dio svakodnevice velikog dijela ljudske populacije. Od mobitela do autonomnih vozila, svi uređaji moraju imati neki način kako razumjeti podatke iz stvarnoga svijeta. Ovo predstavlja izazov koji je jedna od ključnih tema moderne računarske znanosti.

Računalni vid (engl. *computer vision*) područje je računarskih znanosti usko vezano uz umjetnu inteligenciju koja se bavi stjecanjem, obradom, analizom i razumijevanjem slike i općenito višedimenzionalnih podataka iz realnog svijeta. Svojstva objekata poput oblika, veličine, boja i međusobnih položaja koriste se kako bi se naučili modeli za prepoznavanje uzoraka.

Jedan od najpoznatijih problema kojima se računalni vid bavi je klasifikacija slika (engl. *image classification*). Klasifikacija slika zadatak je u kojem se slike na temelju svojih karakteristika nastoje klasificirati u jednu od konačnog broja klasa. Klasifikacija može biti binarna (dvije klase) ili višeklasna (više od dvije klase). Primjer binarnog klasifikacijskog problema je prepoznavanje sadrži li slika psa u sebi (klase su "sadrži psa" i "ne sadrži psa").

U ovom radu bavili smo se klasifikacijom slika društvene igre Carcassonne, čiji raznolik dizajn pločica predstavlja zanimljiv izazov klasifikacije. U samoj igri, posložene pločice tvore igraču ploču što čini zadatak izdvajanja pločica na temelju njihovih svojstava još jednim izazovom koji se može riješiti primjenom tehnika računalnog vida.

Kako bismo riješili problem klasifikacije pločica, trenirali smo različite modele. Njihove uspješnosti usporedili smo na vlastitom skupu podataka, a kombinacijom izdvajanja pločica iz igraće ploče i njihovom klasifikacijom, moguće je i usporediti koliko dobro modeli rekonstruiraju ploču.

Rezultat rada je skripta koja segmentira ploču na pločice i koristeći eksperimentom dokazano najbolji model, klasificira svaku pločicu čime se dobiva rekonstrukcija s kojom se kasnije može dodatno raditi.

Nakon provedenih eksperimenata, analizirali smo rezultate i komentirali što smo mogli bolje napraviti te smjerove u kojima bi ovaj projekt mogao rasti u budućnosti.

2. Društvena igra Carcassonne

Društvena igra Carcassonne temelji se na postavljanju igračih pločica kvadratnoga oblika kako bi se stvorio tlocrt po uzoru na srednjovjekovni grad. Glavne komponente koje sačinjavaju pločice su gradovi, polja, ceste i samostani.

(a) Primjer pločice koja sadrži grad i cestu **(b)** Primjer pločice koja sadrži samostan i cestu

Slika 2.1. Primjeri različitih pločica društvene igre

Polje je definirano kao područje omeđeno cestama i gradovima. Pločice se prilikom igranja postavljaju jedna po jedna te moraju biti postavljene da se smisleno nastavljaju na jednu ili više prethodno postavljenih pločica.

Kako se ne bi ulazilo preduboko u pravila igre, bitno je znati da je za bodovanje na kraju igre potrebno prepoznati obrube polja i gradova na završnoj ploči. Problem je moguće proširiti u zadatak automatskog prebrojavanja bodova što uključuje dodavanje i naknadno prepoznavanje igračih figurica koje predstavljaju artefakte prilikom klasifikacije i segmentacije kao ploča na slici 2.2.a

Problem se može rastaviti na dva dijela: segmentacija završne ploče u pojedine pločice te klasifikacija pojedinih pločica u konačan broj klasa.

Ovaj rad se bavio samo rekonstrukcijom ploče s namjerom prepoznavanja obruba polja, iz ovog razloga neke pločice koje imaju različite uloge u igri smatraju se istom klasom ako ne čine razliku u samom obliku polja koje tvore.

(a) Primjer ploče s postavljenim igračim figuricama

(b) Primjer ploče bez igraćih figurica

Slika 2.2. Primjeri igračih ploča

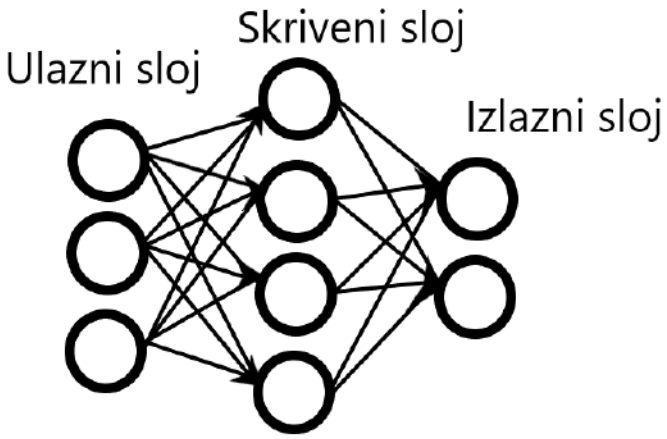
3. Strojno učenje

Strojno učenje predstavlja proces programiranja računala na način da se optimizira određeni kriterij uspješnosti koristeći podatkovne primjere ili prethodno iskustvo. Imamo model koji je definiran određenim parametrima, a učenje se sastoji od izvođenja algoritma koji optimizira te parametre modela na temelju dostupnih podataka ili prethodnog iskustva. Modeli strojnog učenja pokušavaju što bolje predvidjeti svojstva neviđenih podataka. Strojno učenje se dijeli na nadzirano, nenadzirano i podržano učenje, ovisno o vrsti podataka. Kada koristimo modele u računalnom vidu, potrebno je iz podataka stvarnoga svijeta izvlačiti značajke - vektore koji predstavljaju ključne informacije o podacima, na primjer konture, oblike, boje ili osvjetljenje. One se dovode na ulaz modela strojnoga učenja i služe kako bi model bolje prepoznavao sličnosti i razlike među podacima.[1]

U ovom smo radu koristili nadzirano učenje za treniranje i fino ugađanje konvolucijskih modela te nenadzirano učenje za treniranje algoritma k-sredina.

3.1. Umjetne neuronske mreže

Umjetne neuronske mreže su strukture podataka koje se sastoje od međusobno povezanih slojeva umjetnih neurona. Modelirane su po uzoru na ljudski mozak i imaju sposobnost učenja na skupovima podataka. Slojevi se dijele na ulazni, skriveni i izlazni sloj. Ulazni sloj prihvaća podatak koji se obrađuje, skriveni slojevi obrađuju podatke prethodnih slojeva, a izlazni sloj kao rezultat vraća rezultat obrade [2].



Slika 3.1. Arhitektura neuronske mreže, na slici je prikazana struktura koja ima samo jedan skriveni sloj. U praksi, najčešće koristimo arhitekture u kojima postoji više međusobno povezanih skrivenih slojeva.

3.1.1. Umjetni neuroni

Umjetni neuroni su međusobno povezani težinama. Ulazni signali označavaju se sa $x_1, x_2...x_n$, a njihove težine sa $w_1, w_2...w_n$. Težinska suma označava se kao *net* i iznosi 3.1

$$net = \sum_{i=1}^{n} w_i x_i - \theta \tag{3.1}$$

gdje je θ vrijednost praga. Vrijednost praga se često označava kao $\theta = -w_0$ što pretvara formulu za *net* u

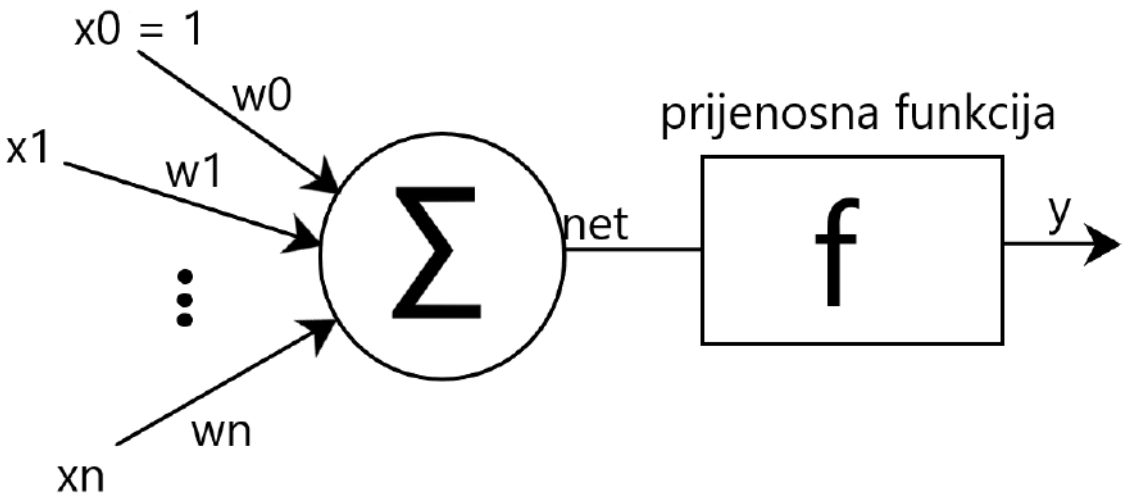
$$net = \sum_{i=0}^{n} w_i x_i \tag{3.2}$$

pod uvjetom da dodamo ulazni signal konstantnog iznosa $x_0 = 1$.

Izlaz neurona y ima vrijednost

$$y = f(net) = f(\sum_{i=0}^{n} w_i x_i) \tag{3.3}$$

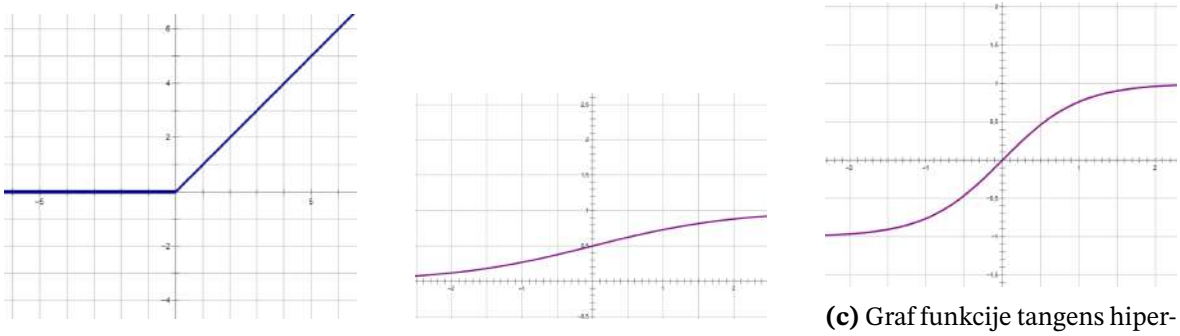
gdje je f aktivacijska ili prijenosna funkcija.



Slika 3.2. Arhitektura umjetnog neurona, na slici je prikazano kako se težinska suma ulaza u tijelu neurona sumira te nakon prolaska kroz prijenosnu funkciju postavlja na izlaz neurona.

3.1.2. Aktivacijske funkcije

Aktivacijske funkcije prihvaćaju linearnu kombinaciju ulaznih signala *net* te je pretvaraju u izlaz neurona. Poželjno svojstvo aktivacijske funkcije je da nije linearna, jer kada bi svaki neuron koristio linearnu aktivacijsku funkciju, izlaz cijele neuronske mreže bio bi linearna kombinacija ulaznih neurona što je ekvivalentno kao da imamo samo jedan neuron. Neke od često korištenih aktivacijskih funkcija su $ReLu(x) = max(0, x)$, $sigmoid(x) = \frac{1}{1+e^{-x}}$, $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



(a) Graf funkcije ReLU

(b) Graf funkcije sigmoide

(c) Graf funkcije tangens hiperbolni

Slika 3.3. Grafovi čestih aktivacijskih funkcija

3.1.3. Funkcija pogreške

Funkcija gubitka L je funkcija koja mapira izlaz neuronske mreže u realan broj. Optimizacijski problemi često kao svoj cilj imaju minimizaciju vrijednosti funkcije pogreške

E koja je definirana kao očekivana vrijednost funkcije gubitka. Jedna od funkcija koju često koristimo kao funkciju gubitka je funkcija polovične sume srednjih kvadrata odstupanja:

$$E = \frac{1}{N}\sum_{s=1}^{N} L(s) = \frac{1}{N}\sum_{s=1}^{N}\frac{1}{2}\sum_{i=1}^{N_o}(t_{s,i} - o_{s,i})^2 \tag{3.4}$$

Gdje $o_{s,i}$ predstavlja i-ti element vektora na izlazu neurona s, a $t_{s,i}$ predstavlja očekivanu vrijednost koju bi i-ti element vektora na izlazu neurona s trebao postići.

3.1.4. Učenje umjetnih neuronskih mreža

Učenje umjetne neuronske mreže postižemo ugađanjem vrijednosti težina signala između neurona. Cilj nam je učenja smanjiti iznos funkcije pogreške, a često to postižemo primjenom algoritma propagacije pogreške unazad. Učenje se provodi u epohama na skupu podataka za treniranje. Na kraju svake epohe provodimo evaluaciju na skupu za validaciju koji služi kako bismo imali uvid u ponašanje modela prilikom treniranja. Nakon treniranja, model evaluiramo na skupu za testiranje pomoću kojeg pokušavamo ustanoviti generalizacijske sposobnosti modela.

Jako je bitno da su skupovi podataka za validaciju, testiranje i treniranje međusobno nezavisni kako bismo spriječili prenaučavanje modela. Prenaučavanje modela jedan je od najvećih problema kojim se područje računalnog vida bavi, a opisano je malim iznosom funkcije pogreške na skupu za treniranje i velikom greškom na skupu za testiranje i validaciju. Pojava prenaučavanja nam govori da model nije dobro razvio generalizacijske sposobnosti.

Čest način učenja neuronskih mreža je gradijentni spust. Gradijentni spust kao osnovnu ideju ima kretanje u suprotnom smjeru od smjera vektora gradijenta kako bismo pronašli globalni minimum funkcije. Ovaj algoritam se temelji na matematičkoj činjenici da vektor gradijenta uvijek ima smjer najvećeg rasta funkcije. U kontekstu treniranja neuronskih mreža, vrijednosti težina neuronske mreže iterativno se mijenjaju na način da se nakon svake iteracije učenja njihovoj vrijednosti oduzme gradijent funkcije pogreške s obzirom na tu težinu pomnožen hiperparametrom η koju nazivamo stopa učenja

(engl. *learning rate*).

$$w_{i,j}^{k} -= \eta * \frac{\partial E}{\partial w_{i,j}^{k}} \tag{3.5}$$

Odabir hiperparametra η od velike nam je važnosti - premale vrijednosti dovode do spore konvergencije i zapinjanja u lokalnim minimumima, a prevelike vrijednosti dovode do nedovoljne preciznosti i divergencije.

Kako bismo uspjeli pronaći gradijent funkcije pogreške s obzirom na svaku težinu u neuronskoj mreži, koristimo algoritam propagacije pogreške unazad. Algoritam propagacije pogreške unazad pokušava pronaći parcijalnu derivaciju funkcije pogreške s obzirom na težine neuronske mreže:

$$\frac{\partial E}{\partial w_{i,j}^{k}} = \frac{\partial E}{\partial o_{j}^{k}} \frac{\partial o_{j}^{k}}{\partial net_{j}^{k}} \frac{\partial net_{j}^{k}}{\partial w_{i,j}^{k}} \tag{3.6}$$

gdje je $net = \sum_{i=1}^{n} w_{i,j}^{k} o_{i}^{k-1}$.

Vrijednost derivacije iznosa sume *net* s obzirom na težinu $w_{i,j}$ iznosi:

$$\frac{\partial}{\partial w_{i,j}^{k}} (\sum_{l=1}^{n} w_{l,j}^{k} o_{l}^{k-1}) = o_{i}^{k-1} \tag{3.7}$$

gdje je o_{i}^{k-1} iznos i tog ulaznog signala, a $w_{l,j}^{k}$ težina od neurona l nižeg sloja $k-1$ do neurona j višeg sloja k.

Vrijednost derivacije izlaza neurona o_{j}^{k} s obzirom na sumu net_{j}^{k} istoga neurona možemo dobiti derivacijom aktivacijske funkcije u točki net_{j}^{k}, što za funkciju sigmoide iznosi:

$$\frac{\partial o_{j}^{k}}{\partial net_{j}^{k}} = \frac{\partial}{\partial net_{j}^{k}} \sigma(net_{j}^{k}) = \sigma(net_{j}^{k})(1 - \sigma(net_{j}^{k})) = o_{j}^{k}(1 - o_{j}^{k}) \tag{3.8}$$

Vrijednost $\frac{\partial E}{\partial o_{j}^{k}}$ dobiva se rekurzivno preko vrijednosti $\frac{\partial E}{\partial net_{i}^{k+1}}$ sljedećeg sloja. $\frac{\partial E}{\partial net_{j}^{k}}$ se također naziva i pogreška neurona.

$$\frac{\partial E}{\partial o_j^k} = \sum_{i=1}^{N} \frac{\partial E}{\partial net_i^{k+1}} \frac{\partial net_i^{k+1}}{\partial o_j^{k+1}} = \sum_{i=1}^{N} \frac{\partial E}{\partial net_i^{k+1}} w_{i,j}^{k+1} \tag{3.9}$$

Ovdje primjećujemo rekurzivnu ovisnost o iznosima $\frac{\partial E}{\partial net_j^{k+1}}$ višeg sloja $k+1$. Izlazni neuroni nemaju sloj viši od sebe, a iznos te iste parcijalne derivacije za posljednji sloj lagano se izračunava derivacijom funkcije pogreške po iznosima. Zbog ovoga algoritam propagacije pogreške zahtijeva da funkcija pogreške mora biti diferencijabilna.

Na početku algoritma, sve težine u neuronskoj mreži postavljamo na slučajne vrijednosti. Nakon toga, predodređen broj epoha ili sve dok se ne ispuni neki drugi uvjet za zaustavljanje, radimo sljedeće:

Za svaki podatak iz skupa za treniranje ponavljamo:

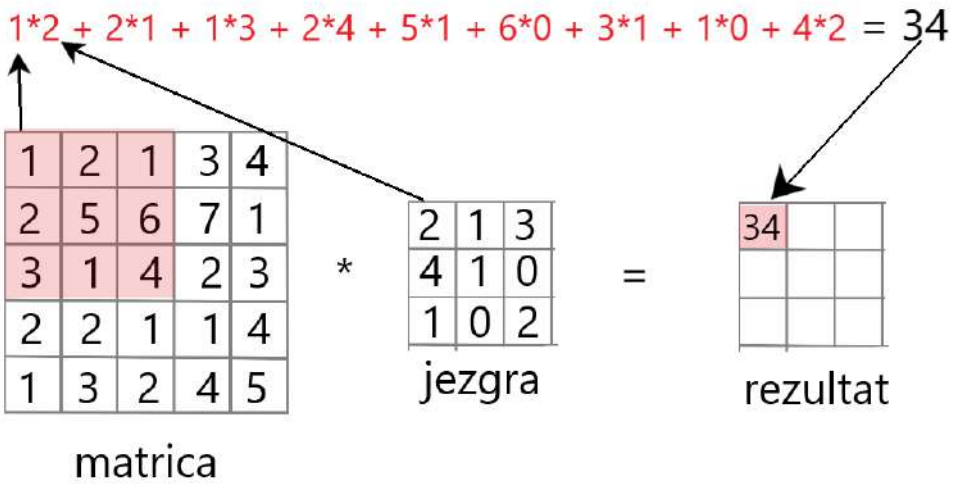
- Postavi podatak x na ulaz mreže.
- Izračunaj izlaz svih neurona gdje je o^K izlaz posljednjeg sloja. Ovo se naziva i unaprijedni prolazak (engl. *forward pass*).
- Odredi iznos funkcije pogreške $\delta_i^K = \sum_{i=1}^{N} E(o_i^K)$ za izlazne neurone.
- Odredi pogrešku neurona $\frac{\partial E}{\partial net_i^K}$ za posljednji sloj.
- Rekurzivno prolaskom slojevima od viših prema nižim, izračunaj pogreške neurona $\delta_i^k = \frac{\partial E}{\partial net_i^k}$
- Za svaku težinu $w_{i,j}$, dodaj parcijalnu derivaciju pomnoženo faktorom učenja η. Ovo je primjena algoritma gradijentnog spusta. $w_{i,j}^k - = \eta \frac{\partial E}{\partial w_{i,j}^k} = \eta \delta_i^k o_i^{k-1}$

3.1.5. Konvolucijske neuronske mreže

Konvolucija je binarna matematička operacija nad dvjema funkcijama takva da kao rezultat daje treću funkciju koja se dobije kao integral umnoška prve funkcije i posmaknute druge funkcije zrcaljene s obzirom na y-os.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \tag{3.10}$$

Diskretna 2D konvolucija se interpretira na način da matrica koja se naziva jezgra (engl. *kernel*) prolazi po matrici nad kojom se vrši konvolucija i u rezultantnu matricu upisuje sumu umnožaka pojedinačnih elemenata.



Slika 3.4. Primjer diskretne konvolucije matrice i jezgre dimenzije 3×3. Ilustrirani postupak se ponavlja na način da centar matrice jezgre iterira po retcima i stupcima ulazne matrice.

Konvolucijske neuronske mreže su arhitektura neuronskih mreža koje u konvolucijskim slojevima množe dobivenu matricu na ulazu s konvolucijskom jezgrom. Ovime se postiže apstrahiranje u matricu značajki. Ovakve mreže također sadrže potpuno povezane slojeve neurona te slojeve sažimanja koji smanjuju dimenzije ulaza. Sažimanje može biti lokalno ili globalno, a najčešće korištena sažimanja su sažimanje maksimumom i sažimanje srednjom vrijednošću.

3.2. *Transformeri*

Transformeri su arhitektura dubokog učenja koju su razvili istraživači iz Googla, a prvi put predstavili u radu "Attention is All You Need" 2017. godine [3]. Ova arhitektura koristi mehanizam pažnje kako bi odredila važnost pojedinih dijelova ulaznih podataka (npr. riječi u rečenici ili dijelovi slike). Danas su transformeri temelj mnogih naprednih modela poput: BERT [4], GPT-2 [5], GPT-3 [6], itd. Prilikom obrade teksta, transformeri tekst dekomponiraju u tokene koje zatim pretvaraju u vektore značajki. U računalnom vidu koristimo arhitekturu vizualnih transformera (ViT [7]). Vizualni transformeri rade slično kao i tekstualni s glavnom razlikom da sliku dekomponiraju u isječke (engl. *patches*) koji igraju ulogu prije spomenutih tokena. [7]

3.2.1. DINOv2

DINOv2 [8] skupina je modela otvorenog izvora koje je razvila Meta AI i bazirani su na vizualnim transformatorima. Služe za samonadzirano učenje reprezentacija iz neoznačenih podataka za razne modele računalnog vida. Samonadzirano učenje detaljnije je objašnjeno u sljedećem poglavlju. DINOv2 modeli mogu učiti iz bilo kakvog skupa slika i nije ih potrebno fino ugađati [1], što ih čini korisnim kao osnovicom (engl. *backbone*) za druge modele. DINOv2 modeli su predtrenirani na velikom skupu podataka koji se sastoji od raznih drugih skupova podataka poput Cityscapes [9] i ImageNet-1k [10]

3.3. Vrste učenja

Po načinu učenja, strojno učenje dijelimo na tri glavne skupine: nadzirano učenje (engl. *supervised learning*), nenadzirano učenje (engl. *unsupervised learning*) i podržano učenje (engl. *reinforcement learning*). U ovom radu koristimo nadzirano i nenadzirano učenje.

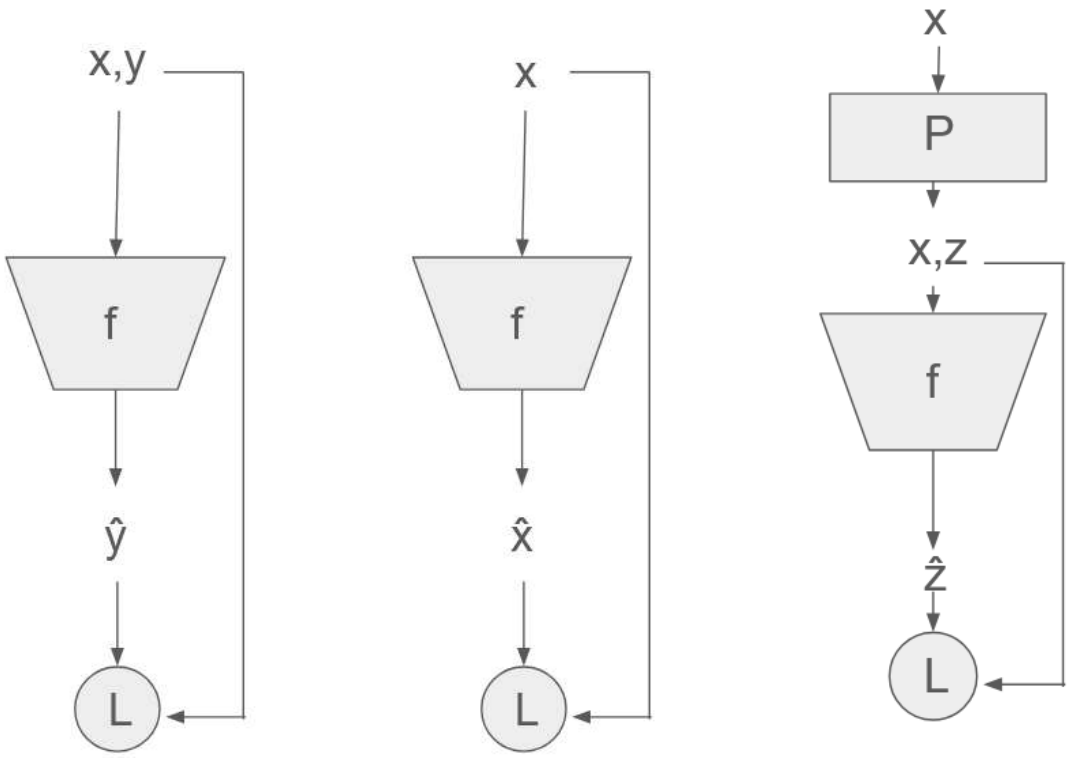
Nadzirano učenje grana je strojnog učenja u kojoj podaci imaju oznaku kojoj klasi pripadaju te se na temelju toga vrši proces učenja. Algoritmi nadziranog strojnog učenja uče kako predvidjeti kojoj klasi će neki podatak pripasti na temelju prijašnjih podataka. Neke vrste nadziranog učenja su: regresije, neuronske mreže, naivni Bayesov klasifikator te stabla odluke.

Nenadzirano učenje grana je strojnog učenja gdje podaci za treniranje nemaju oznaku odredišne klase. Ovakvi modeli uče kako prepoznati uzorke u podacima, a neke vrste su algoritam k-sredina, analiza svojstvenih komponenti te autoenkoderi.

Kao posebna vrsta nenadziranog učenja, razvilo se samonadzirano učenje. Samonadzirano učenje (engl. *self-supervised learning*) kombinira elemente nadziranog i nenadziranog učenja, a proces učenja se sastoji od dva koraka. U prvome koraku model samonadziranog učenja generira smislene pseudo-oznake i potencijalno modificira podatke iz neoznačenog skupa podataka, kreirajući novi skup podataka. U drugom koraku model se trenira na novom skupu podataka s pseudo-oznakama. Prednost ovakvih modela je što ne ovise o označenim podacima koji su skupi za prikupiti, a istovremeno korištenjem

[1] Fino ugađanje je proces u kojem se predtrenirani model dodatno uči kako bi se poboljšao za neki specifični zadatak

pseudo-oznaka dobivamo mogućnost učenja općenitih reprezentacija koje se lako mogu koristiti za različite zadatke [11].



(a) Tijek nadziranog učenja **(b)** Tijek nenadziranog učenja **(c)** Tijek samonadziranog učenja

Slika 3.5. Na slici su uspoređeni tokovi učenja za različite vrste učenja. x označava podatke, a y oznake ciljne klase. f je funkcija koja klasificira podatak x. P predstavlja prvi korak samonadziranog učenja gdje se generiraju pseudo-oznake z. L predstavlja funkciju gubitka.

3.4. Metričko učenje

Metričko učenje pristup je strojnom učenju u kojem nastojimo izmjeriti udaljenost podataka učenjem funkcije ugrađivanja koja mapira podatke u prostor ugrađivanja. Zadatak svakog modela metričkog učenja je da značajke podatka i grupira bliže značajkama pripadnika iste klase j nego značajkama pripadnika drugih klasa k koristeći neku metriku D. [12]

$$D(f_i, f_j) < D(f_i, f_k); \forall i, j \in y; \forall k \notin y \tag{3.11}$$

gdje je f_i vektor značajki podataka i.

3.4.1. Kontrastni gubitak

Kako bismo uspjeli trenirati model koji značajke iste klase ugrađuje bliže nego značajke različitih klasa, moramo definirati prikladnu funkciju gubitka. U metričkom ugrađivanju želimo definirati funkciju gubitka koja velike udaljenosti značajki podataka iste klase kažnjava kao i male udaljenosti značajki podataka različitih klasa. Primjer funkcije gubitka koja ovo zadovoljava je funkcija kontrastnog gubitka L_{con} koja na svoj ulaz prima dva vektora značajki f_i i f_j te ovisno o tome pripadaju li istoj klasi, ima različiti iznos.

$$L_{con} = \begin{cases} ||f_i - f_j||^2 & \text{ako pripadaju istoj klasi} \\ \max(\alpha - ||f_i - f_j||^2, 0) & \text{inače} \end{cases} \tag{3.12}$$

gdje se α naziva marginom i može se smatrati minimalnom udaljenošću koju trebaju imati značajke različitih klasa.

3.4.2. Trojni gubitak

Kontrastni gubitak prilikom svakog koraka treniranja uzima dva podatka iz skupa za treniranje, korištenjem alternativne funkcije gubitka koja prilikom svakog koraka uzima tri podatka možemo ubrzati proces učenja modela. Trojni gubitak nadogradnja je na kontrastni gubitak i uzima u obzir trojku značajki f_a, f_p i f_n gdje redom značajke pripadaju podacima koji se nazivaju sidro (engl. *anchor*), pozitiv i negativ. Pozitiv je nasumično uzorkovan pripadnik iste klase kao sidro, dok je negativ nasumično odabran pripadnik neke druge klase. Funkciju trojnog gubitka koristili smo u radu za svako metričko učenje i fino ugađanje.

$$L_{trip} = \sum_{a,p,n \subset N} max(||f_a - f_p||^2 - ||f_a - f_n||^2 + \alpha, 0) \tag{3.13}$$

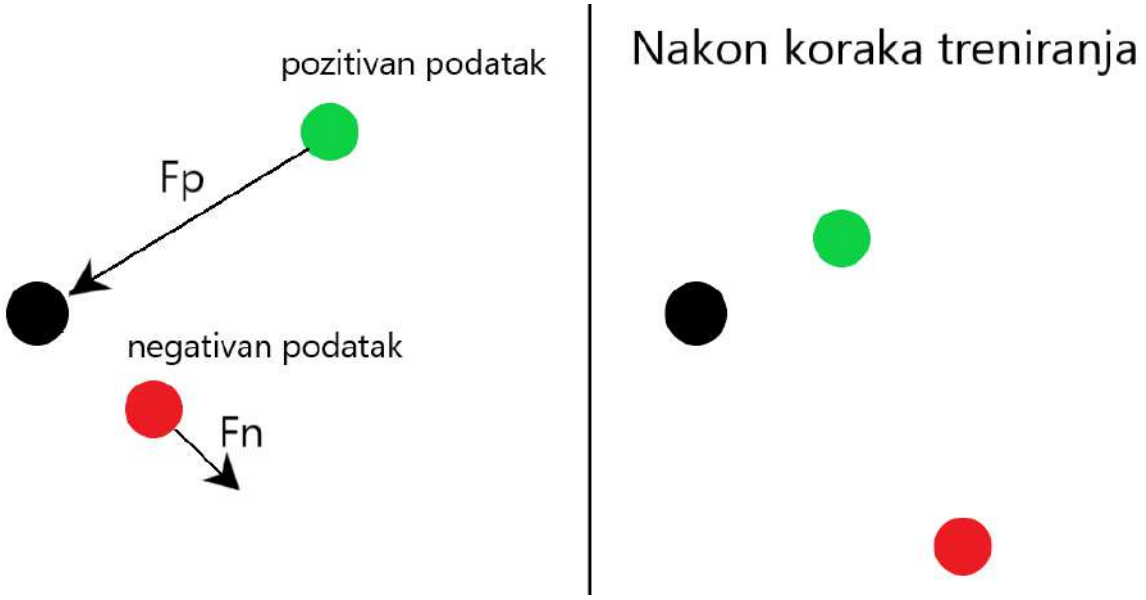
Zanimljivo je interpretirati parcijalne derivacije funkcije gubitka s obzirom na vektore značajki koje se koriste kako bi se ažuriralo ugrađivanje značajki:

$$\frac{\partial L_{trip}}{\partial f_a} = 2(f_n - f_p) \tag{3.14}$$

$$\frac{\partial L_{trip}}{\partial f_p} = 2(f_p - f_a) \tag{3.15}$$

$$\frac{\partial L_{trip}}{\partial f_n} = 2(f_a - f_n) \tag{3.16}$$

Ovo možemo interpretirati kao da na negativni podatak djeluje sila u smjeru $f_n - f_a$ koja djeluje radijalno od sidra. Analogno, na pozitivan primjerak djeluje sila u smjeru $f_a - f_p$ koja djeluje radijalno prema sidru. Kako je "sila" privlačenja proporcionalna samoj udaljenosti, možemo primijetiti da ovakvo učenje ima efekt opruge.



Slika 3.6. Interpretacija koraka učenja kao sile na pozitivni i negativni uzorak. Nakon učenja primjećujemo da je pozitivan podatak bliže sidru, a negativan podatak udaljen. Udaljenosti za koliko se podaci približe ili udalje su proporcionalni udaljenosti prije približavanja, kao i u slučaju opruge.

3.5. Algoritam k sredina

Algoritam k-sredina popularan je oblik nenadziranog učenja pomoću kojeg se skup od n obzervacija pokušava grupirati u k grupa na način da je ukupna suma kvadratnih Euklidskih udaljenosti minimalna. Zadatak je NP-težak što znači da nije rješiv u polinomijalnom vremenu, ali zato postoje heuristički algoritmi koji konvergiraju prema lokalnom minimumu.

Najčešće korištena implementacija algoritma k-sredina je takozvani naivni algoritam k-sredina ili Lloydov algoritam. Ovaj se algoritam sastoji od koraka [13] dodjele i ažuri-

ranja. U koraku dodjele svakoj se obzervaciji dodjeljuje sredina koja je najbliža po Euklidskoj udaljenosti. U koraku ažuriranja, svaka sredina poprima vrijednost aritmetičke sredine svih obzervacija kojima je ona dodijeljena. Ovaj algoritam se koristi u radu prilikom pozivanja metode `KMeans.fit_predict` biblioteke scikit-learn [14].

3.6. Bipartitna sparivanja

Bipartitni graf je graf kojeg možemo podijeliti u dva skupa vrhova gdje ne postoji brid između dva vrha pripadnika istoga skupa. Bipartitno sparivanje se u teoriji grafova odnosi na problem pronalaska skupa bridova, takvih da niti jedan par bridova ne dijeli zajednički vrh. Potpuno sparivanje je podvrsta sparivanja u kojoj svaki čvor grafa pripada nekom od uparenih bridova. U ovom radu pokušavamo pronaći potpuno bipartitno sparivanje između grupa dobivenih algoritmom k-sredina i klasa pločica.

Problem podjele (engl. *Task assignment problem*) u teoriji grafova uključuje pronalaženje sparivanja na težinskom bipartitnom grafu, takvog da je ukupna suma težina bridova minimalna. Težine bridova bipartitnog grafa zapisuju se u matricu cijene M gdje $m_{i,j}$ predstavlja cijenu između i-tog vrha u prvom skupu i j-tog čvora u drugom skupu.

3.6.1. Mađarski algoritam

Mađarski algoritam (engl. *Hungarian algorithm*) nam nudi rješenje na problem podjele u polinomijalnom vremenu od $O(n^4)$ [15]. Koraci algoritma su sljedeći:

1. Za svaki red matrice cijene, minimalna vrijednost retka se oduzima svakom elementu tog istog retka.

2. Prethodni korak također se ponavlja za svaki stupac matrice cijene.

3. Nasumično se u svakom retku pokušava odabrati element s vrijednošću 0 takav da niti jedan drugi element matrice cijene s vrijednošću 0 u istom stupcu ili retku nije odabran.

4. Svaki stupac koji sadrži jedno od odabranih polja se prekriva.

5. Sve dok svi elementi s vrijednošću nula nisu prekriveni ponavljaj:

(a) Pronađi element vrijednosti 0 koji nije prekriven i označi ga

(b) Ako pronađeni element u svojem retku ima neki od odabranih elementa, prekrij cijeli redak i otrkij stupac u kojem se nalazi taj odabrani element.

(c) U suprotnom pronalazi se alternirajući put koji kreće iz najnovije označene nule, a izmjenjuju se koraci traženja odabrane nule u istom stupcu i traženja označene nule u istom retku. Kada se pronađe takav put, sve označene nule postaju odabrane, a odabrane nule postaju označene. Kako ovakav put nužno završava označenom nulom, broj odabranih nula povećava se za 1.

6. Ako je broj odabranih elemenata s vrijednosti 0 jednak $min(n, m)$ gdje su n i m dimenzije matrice cijene, algoritam se zaustavlja. Inače, pronalazi se nepokriveni element s najmanjom vrijednošću te se oduzima od svakog neprekrivenog elementa i dodaje svakom dvostruko pokrivenom elementu.

7. Ako je element $m_{i,j}$ odabran, u optimalnom rješenju problema sparivanja odabire se brid između vrhova i i j.

3.7. Houghova transformacija

Za klasificiranje pločica posebno je bitno iz same slike uspjeti kvalitetno izdvojiti pločice. Mrežu pločica određujemo na temelju konture cijele ploče, a za pronalazak iste konture od ključne nam je važnosti Houghova transformacija. Houghova linijska transformacija je transformacija pomoću koje možemo detektirati ravne linije u slici. Potrebno je predprocesirati sliku kako bi detekcija rubova bila moguća.

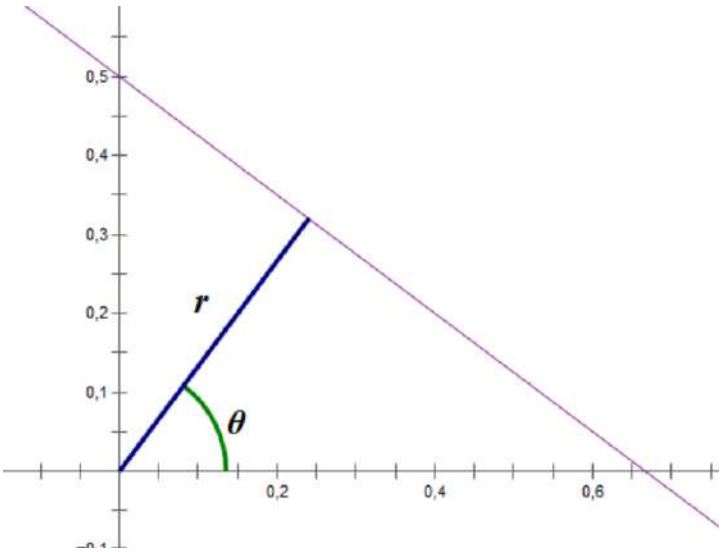
3.7.1. Koordinatni sustavi

Kako bismo razumijeli Houghovu linijsku transformaciju, bitno je detaljno znati odnose između dva najčešće korištena koordinatna sustava:

Kartezijev koordinatni sustav, gdje su sve točke u prostoru zapisane kao par (x, y) gdje x i y predstavljaju udaljenost od apscise i ordinate. U ovakvom koordinatnom sustavu pravac ima oblik $y = mx + b$ gdje je m nagib pravca, a b sjecište pravca i ordinate.

Polarni koordinatni sustav je sustav gdje je svaka točka zapisana u obliku (r, ϕ) gdje je r udaljenost točke od središta, a ϕ kut koji pravac od središta do točke zatvara s apscisom. U polarnom sustavu, jednadžba pravca glasi:

$$y = (-\frac{cos\theta}{sin\theta})x + (\frac{r}{sin\theta}) \tag{3.17}$$

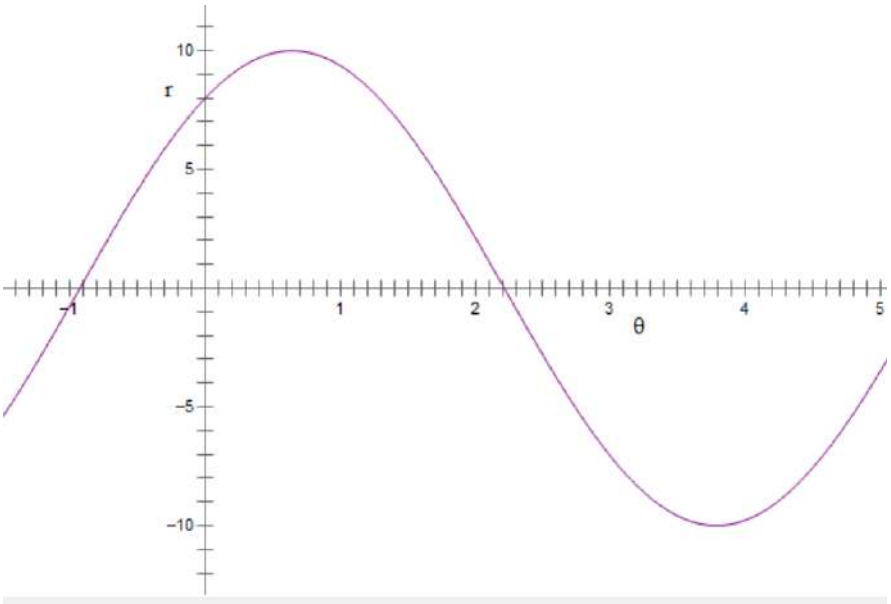


Slika 3.7. Objašnjenje parametara pravca $y = -\frac{2}{3}x + 0.5$u polarnom koordinatnom sustavu. Parametar r je udaljenost pravca od ishodišta, a θ se dobiva kao kut koji visina iz ishodišta na pravac zatvara s osi apscise.

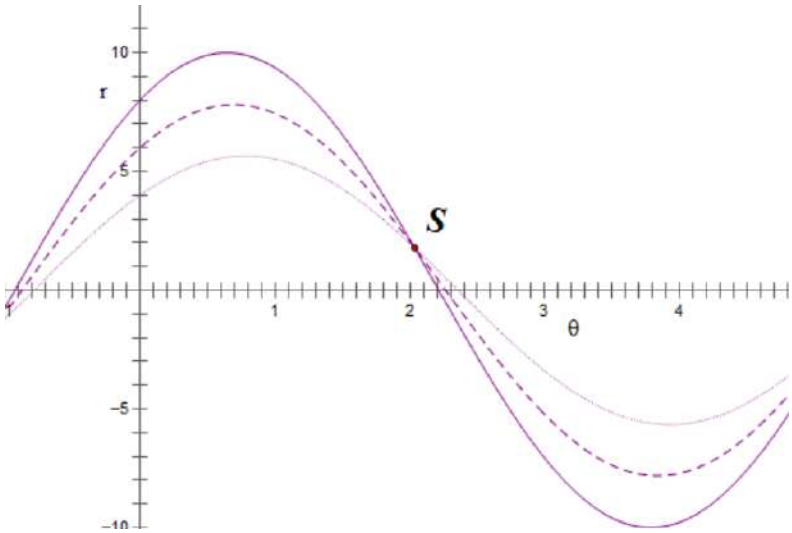
3.7.2. Familije krivulja

Familija krivulja skup ne krivulja koje dijele neko zajedničko svojstvo. Familija krivulja koja sadrži sve pravce koji zadovoljavaju svojstvo da prolaze kroz neku točku T, može se prikazati sinusoidom u Kartezijevom koordinatnom sustavu gdje apscica predstavlja parametar θ, a ordinata parametar r iz jednadžbe 3.17 Točnije, familija krivulja za neku točku (x_0, y_0) ima jednadžbu $r(\theta) = x\cos(\theta) + y\sin(\theta)$.

Ako postoji sjecište dviju familiju krivulja, ono se interpretira kao krivulja koja zadovoljava svojstva obije familije. U ovom slučaju sjecište dvije sinusoide u koordinatnom sustavu (r, θ) predstavlja pravac koji prolazi kroz dvije točke kojima pripadaju familije koje se sijeku.



Slika 3.8. Familija krivulja za točku ($x0 = 8, yo = 6$) u Kartezijevom koordinatnom sustavu



Slika 3.9. Primjer sjecišta familija krivulja točaka $(8, 6)$, $(6, 5)$ i $(4, 4)$ u točki $S = (2.03, 1.79)$. Iz ovog zaključujemo da pravac s parametrima $r = 2.03, \theta = 1.79$ prolazi kroz sve tri točke, prebacivanjem u Kartezijev koordinatni sustav dobivamo pravac $y = 0.5x + 2$

3.7.3. Detekcija linija u slici

Ako je slika obrađena na način da su obrubi jasno definirani (npr. crno-bijela slika), moguće je za svaku točku koja pripada rubu konstruirati familiju krivulja. Zbog svojstva da sjecište više familija krivulja određuje krivulju koja zadovoljava sva svojstva, ako se više familija krivulja koje imaju svojstvo da prolaze kroz neku točku sijeku na istom mjestu, to znači da postoji linija koja prolazi kroz sve te točke, kao što je prikazano na slici 3.9. Prije izvođenja moramo odrediti granicu koliko se točaka mora nalaziti na istom

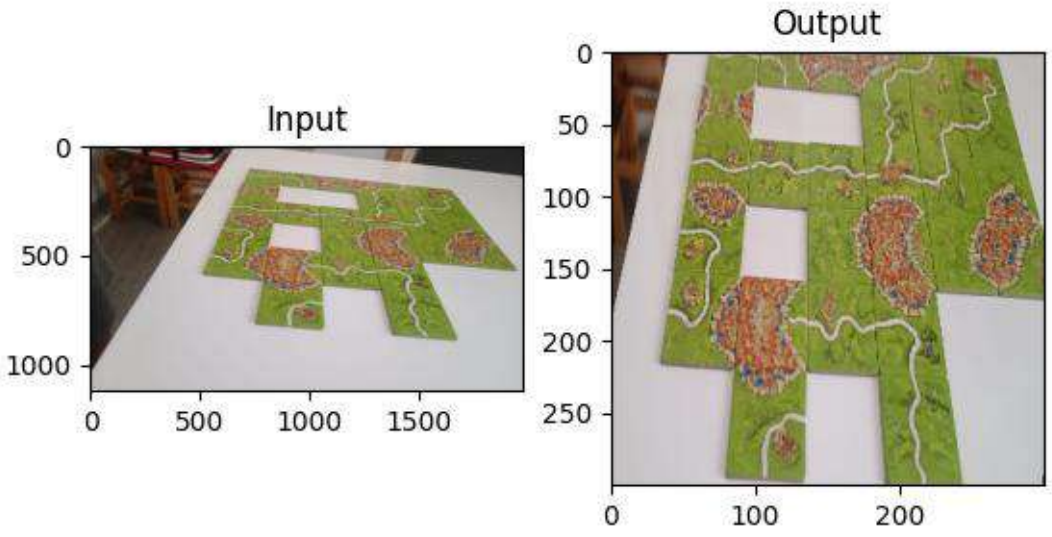
pravcu kako bi se smatralo ravnom linijom.

3.8. Projekcijska transformacija

Projekcijska transformacija postupak je koji smo koristili se kako bi linearnom transformacijom preslikali skup točaka slike na način da dobijemo dojam kao da je slika slikana iz različitog kuta. Rezultat transformacije je matrica M koja preslikava točke na način:

$$\begin{bmatrix} t_i x_i' \\ t_i y_i' \\ t_i \end{bmatrix} = M \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{3.18}$$

vrijedi da su $dst(i) = (x_i', y_i')$, $src(i) = (x_i, y_i)$, gdje su $src(i)$ koordinate kuta četverokuta koji želimo transformirati u odgovarajuće koordinate $dst(i)$ za $i \in \{0, 1, 2, 3\}$



Slika 3.10. Izravnavanje slike projekcijskom transformacijom

4. Korištene biblioteke i modeli

4.1. Biblioteka OpenCV

OpenCV [16] besplatna je biblioteka za programske jezike C++, Python i C koja sadrži korisne metode i objekte za područje računalnog vida i strojnog učenja. Neke od funkcionalnosti koje smo koristili u radu, a OpenCV nudi, su: Houghova transformacija, detekcija ruba primjenom Canny algoritma, pronalazak kontura te morfološke operacije.

4.2. Biblioteka Albumenations

Biblioteka Albumentations [17] biblioteka je programskog jezika Python koja služi za brzo augmentiranje slika. Albumentations nudi širok izbor intuitivnih transformacija pomoću čijih se kompozicija može postići proizvoljna augmentacija slike.

Sintaksa biblioteke je jednostavna što čini često naporne zadatke vrlo jednostavnim. Dio je PyTorch [18] ekosustava što znači da je kompatibilna s bibliotekama poput PyTorcha [18] i TensorFlowa [19].

4.3. Biblioteka PyTorch

PyTorch [18] biblioteka je za strojno učenje temeljena na biblioteci Torch koju je originalno razvila Meta AI. Pruža dvije klase jako bitne za strojno učenje: `torch.Tensor` i `torch.nn` (neuronske mreže). Biblioteka ima velik ekosistem što znači da je kompatibilna s velikim brojem drugih bitnih biblioteka. U sklopu biblioteke ponuđeni su i razni modeli poput konvolucijskog ResNet50 modela (*residual network*) s 50 slojeva.

4.4. Biblioteka scikit-learn

Scikit-learn [14] također je biblioteka strojnog učenja za programski jezik Python. U ovoj biblioteci ponuđene su razne metode za analizu podataka poput: klasifikacije, regresije, grupiranja, redukcije dimenzionalnosti, odabira modela i predprocesiranja.

5. Programsko rješenje

U sklopu programskog rješenja, konstruirali smo vlastiti skup podataka, izmjerili smo metrike performansi na testnom skupu podataka te rekonstruirali nekoliko ploča za svaki od korištenih modela. Korišteni modeli uključuju: jednostavni konvolucijski model treniran metričkim treniranjem (prikazan kodom 8..1), model ResNet50 fino ugođen metričkim treniranjem, model ResNet50 predtreniran na skupu podataka ImageNet-1k i fino ugođen metričkim treniranjem, model DINOv2-small fino ugođen metričkim treniranjem, model DINOv2-base fino ugođen metričkim treniranjem i kombinaciju mađarskog algoritma i algoritma k-sredina koji za ugrađivanje koristi model DINOv2-base. Za treniranje modela korištena je grafička kartica NVIDIA RTX A4500 koristeći biblioteku za brzo matrično množenje CUDA [20].

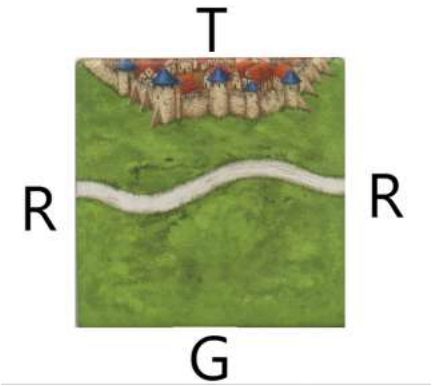
5.1. Skup podataka

U području računalnog vida koristimo razne tehnike strojnog i dubokog učenja te tehnike obrade slika. Kao i u ostalim područjima umjetne inteligencije, prenaučavanje je jedan od najvećih problema i u računalnom vidu. Do prenaučavanja dolazi kada model nauči savršeno klasificirati podatke iz skupa za učenje, a na ostalim skupovima podataka postiže loše rezultate. U sprječavanju prenaučavanja, veliku ulogu igra kakav skup podataka koristimo te kako smo te podatke prikupili.

5.1.1. Označavanje

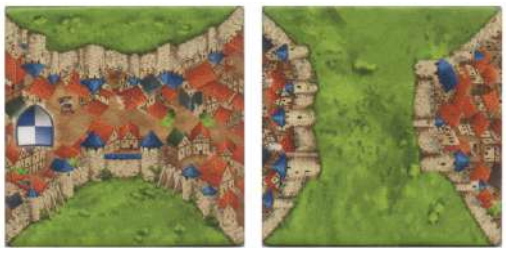
Kako ne postoji javno dostupan skup podataka koji bi bio koristan, skup podataka generirali smo augmentiranjem skeniranih pločica. Svakoj pločici pridodijeljena je klasa koja je definirana njenim rubovima krenuvši od lijevog suprotno od kazaljke na satu sa sljedećom notacijom: T : Grad, R : Cesta, G : Polje kao prikazano na slici 5.1. Originalna igra

sadrži 72 pločice među kojima je jedina razlika u sitnim detaljima, za potrebe ovog rada nije potrebno razlikovati takve pločice. Iznimka ovome je klasa TGTG gdje je potrebno razlikovati klase između one gdje grad dijeli i gdje grad ne dijeli polje. Dvije posebne klase gdje postoji prolaz za polje u sredini označavamo s TGTG_ i GTGT_. Kako bi bilo lakše koristiti klase, prevedene su u broj na način da je oznaka pretvorena u ternarni broj gdje se znakovi mapiraju na način G : 0, R : 1, T : 2. Klasa GTRG bi se označavala s ternarnim brojem $0210_3 = 19_{10}$. Klase TGTG_ i GTGT_ se klasificiraju u posebne klase 81 i 82.



Slika 5.1. Primjer pločice koja bi bila označena kao RGRT01

Slika 5.2. Primjer više neaugmentiranih slika za klasu GGGT, za naš zadatak nije potrebno razlikovati ove pločice.



Slika 5.3. Primjer gdje je potrebno razlikovati klase zbog različitog načina na koji ove dvije pločice dijele. Kada ne bi razlikovali ovaj poseban slučaj, obje pločice bile bi klasificirane kao klasa TGTG.

5.1.2. Augmentacija

Za sve korake augmentacije slike, koristili smo biblioteku Albumentations [17]. Augmentirana slika dobivena je nasumičnom primjenom jednom ili više sljedećih operacija:

- Rotacija za $\theta \in [-5, 5]$ radijana
- Izrezivanje na način da ako je izrezan dio slike van granica, nadopunjava se reflaktiranom slikom
- Povećanje / smanjenje slike
- Promjena svjetline i kontrasta
- Dodavanjem zamućenja

Za svaku od 72 neaugmentirane slike, generiramo sliku rotiranu za 90, 180 i 270 stupnjeva. Na svaku ovakvu sliku primjenjujemo augmentaciju 60 puta. Na ovaj način iz samo 72 skenirane slike dobivamo skup podataka koji sadrži $72 * 60 * 4 = 17280$ slika.

Postoji ukupno $3^4 = 81$ mogućih kombinacija rubova pločice, a dodatne dvije klase 81 i 82 povećavaju taj broj na 83. Postoje i klase koje jednostavno ne postoje u igri, takvih klasa je 26: 5, 7, 11, 15, 17, 19, 21, 23, 25, 29, 33, 35, 45, 47, 50, 51, 55, 57, 59, 61, 63, 65, 69, 70, 73, 75. Iz ovoga slijedi da je ukupan broj mogućih klasa 57. Augmentirane slike imaju dimenzije 256×256 piksela.

5.1.3. Particioniranje

Svaku sliku iz augmentiranog skupa podataka nasumično smo podijelili u jedan od skupova za treniranje, evaluaciju i testiranje s vjerojatnostima od 0.7, 0.15 i 0.15 tim redom. Ovom nasumičnom podjelom dobili smo skup podataka za treniranje koji sadrži 12084 slike, skup podataka za evaluaciju koji sadrži 2553 slike i skup podataka za testiranje koji sadrži 2643 slike.

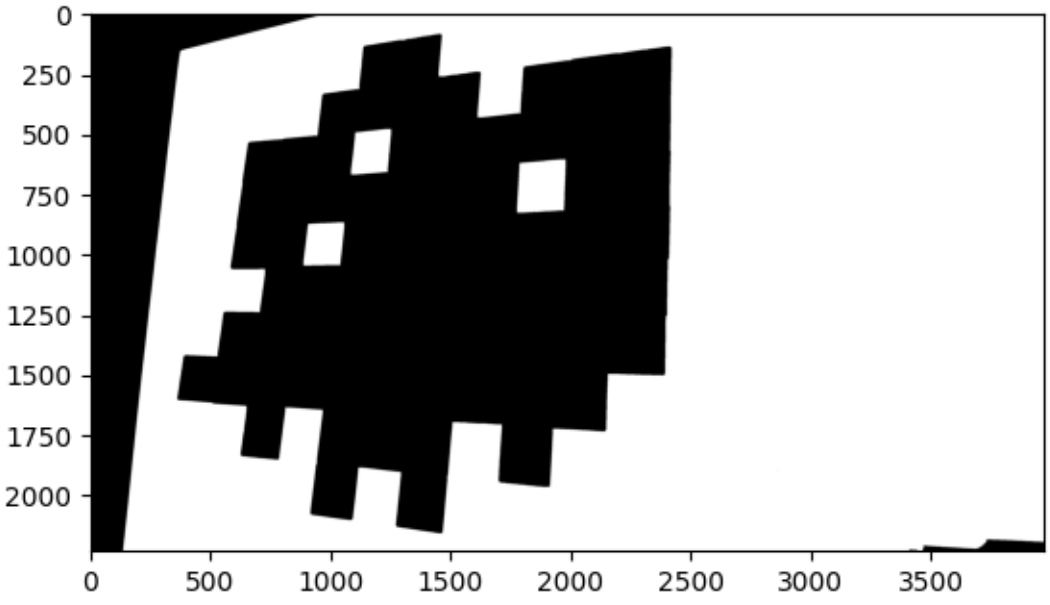
5.2. Segmentacija završne ploče

Kako bi mogli odrediti gdje se koja pločica nalazi, potrebno je završnu ploču segmentirati na zasebne pločice te odrediti njihov međusobni položaj. Ovo ćemo postići pronalaskom

konture završne ploče, projekcijom te rekonstrukcijom mreže pločica. Ovaj pristup pretpostavlja da će kontura završne ploče imati oblik iz kojeg se mogu dobiti i visina i širina pojedine pločice. Na primjer, neće imati oblik savršenog pravokutnika. Dodatno, pretpostavljamo da je pozadina bijela kako bi se olakšalo razlikovanje šarene ploče od pozadine.

5.2.1. Uklanjanje pozadine

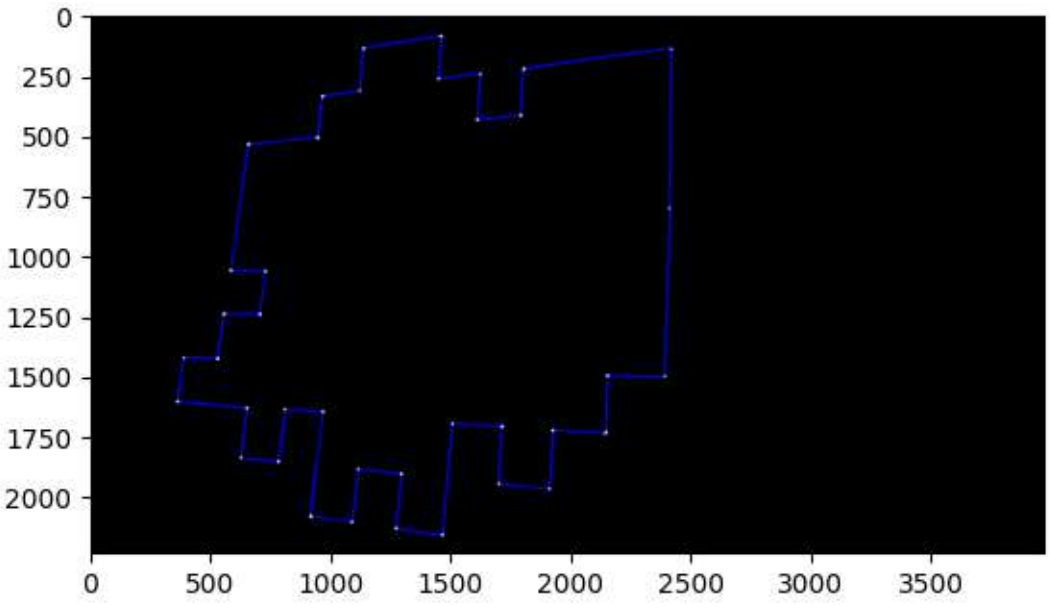
Konturu smo pronašli tako što smo prvo uklonili pozadinu na temelju HSV vrijednosti (*Hue, Saturation, Value*). Prvo smo kao pozadinu označili sve piksele koji imaju vrijednost zasićenosti S (engl. *saturation*) veću od 127. Kao pozadinu također označavamo i sve piksele koji imaju vrijednost V (engl. *value*, interpretira se kao svjetlina) manju od 127. Piksele koje smo označili kao pozadinu bojimo u bijelo, a sve ostalo bojimo u crno. Dodatno smo primjenili morfološku transformaciju [21] erozije popraćenu dilatacijom kako bi se pokušalo zacrniti sve bijele piksele na mjestima gdje bi trebala biti ploča. Ova morfološka transformacija koje eroziju popraćuje dilatacijom naziva se otvaranje (engl. *opening*).



Slika 5.4. Primjer uklonjene pozadine na temelju HSV vrijednosti. Pozadina je označena bijelim pikselima, a sve što želimo zadržati crnim.

5.2.2. Pronalazak konture ploče

Kontura je bilo koja linija koja spaja sve kontinuirane točke nekog obruba s istom bojom ili jačinom. Za ovaj korak koristili smo metodu `cv2.findContours` biblioteke OpenCV. Prije same metode potrebno je sliku pretvoriti u sivu sliku (engl. *grayscale*) (svaki piksel je opisan samo jačinom) što slika 5.4. već zadovoljava. Algoritam koji metoda `cv2.findContours` koristi opisan je u radu [22]. Jednom kada smo pronašli sve konture, uzimamo onu koja je površinom najveća te ju iscrtavamo, u našem slučaju to je kontura same ploče. Na takvom crtežu primjenjujemo Houghovu transformaciju kako bismo pronašli grublju konturu ploče za koju je trivijalno odrediti kutove. Kuteve smo odredili kao sjecišta susjednih Houghovih linija. Pomoću dobivenih linija moguće je pronaći i granični okvir (engl. *bounding box*) - kao sjecište pravaca najgornje i najdonje horizontalne Houghove linije te pravaca najlijevije i najdesnije vertikalne Houghove linije.



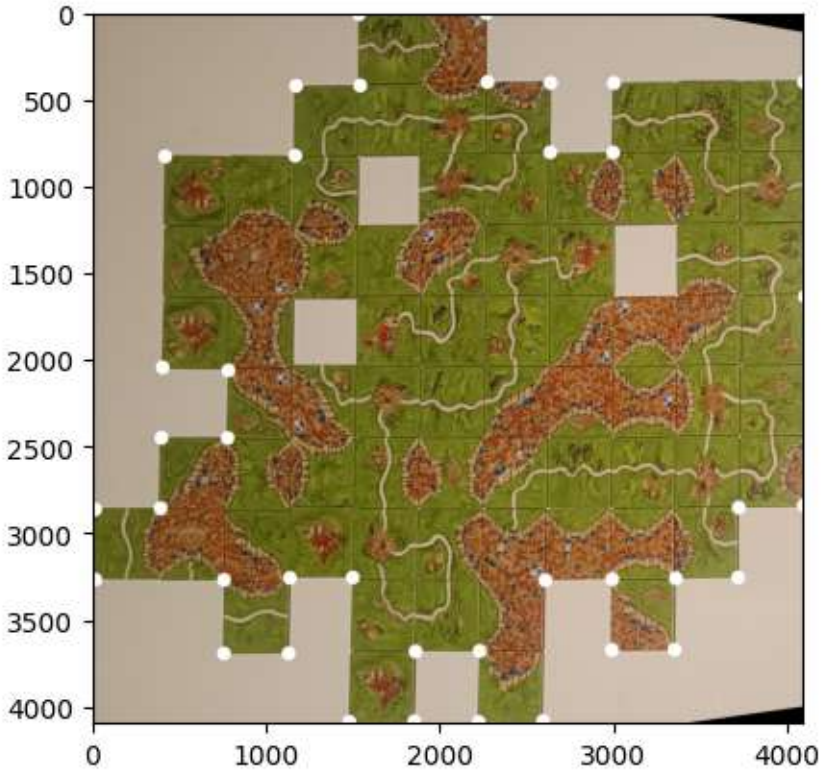
Slika 5.5. Primjer konture ploče dobivene Houghovom transformacijom. Bijele točke predstavljaju kutove konture, a definirane su kao sjecišta susjednih Houghovih linija.

5.2.3. Projekcsijska transformacija

Projekcijsku transformaciju proveli smo korištenjem metode `cv2.warpPerspective` i `cv2.getPerspectiveTransform`. Metoda `cv2.getPerspectiveTransform` na ulaz prima dva para uređenih četvorki točaka, a vraća matricu koja transformira prvu četvorku točaka u drugu. Metoda `cv2.warpPerspective` prima sliku i transformacijsku matricu te primjenjuje transformaciju na svaki piksel.

U programskom rješenju, matrica transformacije je definirana kao matrica koja preslikava točke graničnog okvira u kvadrat fiksne veličine.

Matricu transformacije primijenili smo na originalnu sliku i na kutove konture. Projekcijsku transformaciju koristimo i kasnije u radu kako bismo izrezali pojedine pločice.



Slika 5.6. Primjer slike projicirane na kvadrat s označenim kutovima.

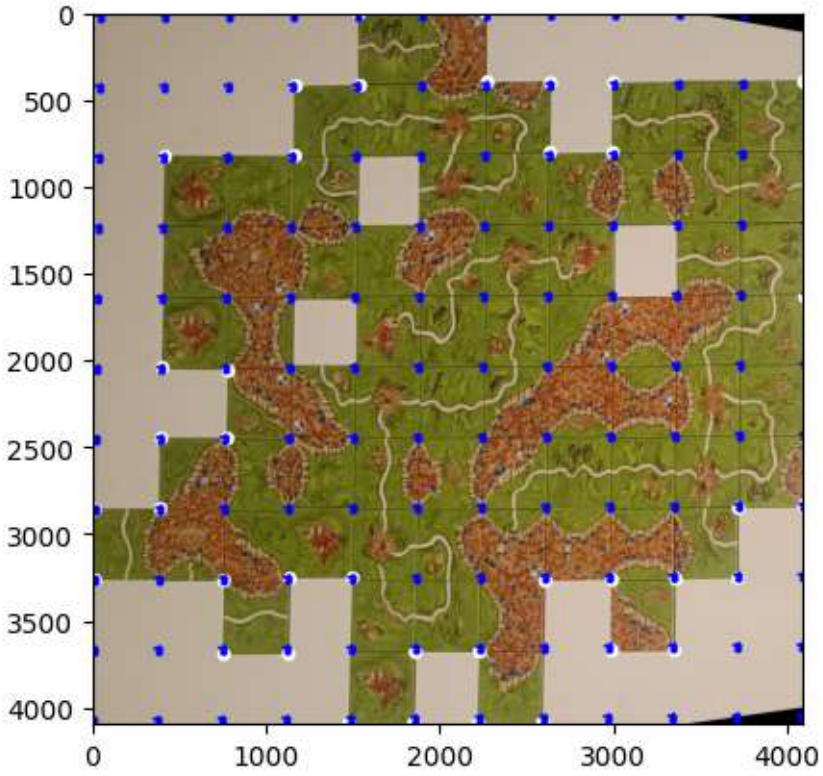
5.2.4. Rekonstrukcija mreže

Rekonstrukciju mreže proveli smo na temelju međusobnog položaja transformiranih kutova konture. Prvo je bilo potrebno odrediti svaku liniju koja spaja neke dvije točke u istom retku ili stupcu. Ovo smo postigli uzimanjem samo onih linija čiji se kut nagiba pravca nalazi unutar $\frac{\pi}{50}$ od savršeno vertikalne ili horizontalne linije. Kao nagib mreže uzeli smo aritmetičku sredinu svih nagiba ovakvih linija.

Određivanje širine pločice postigli smo tako da smo sve duljine horizontalnih linija grupirali ako su dovoljno bliske po vrijednosti. Zatim smo za vrijednost svake grupe uzeli srednju vrijednost svih duljina unutar nje. Na ovaj način smo dobili skup od nekoliko potencijalnih kandidata za širinu pločice. Za očekivati je bilo da će među kandidatima biti ponuđena tražena širina kao i neki od njenih višekratnika. Zbog ovoga smo kao širinu

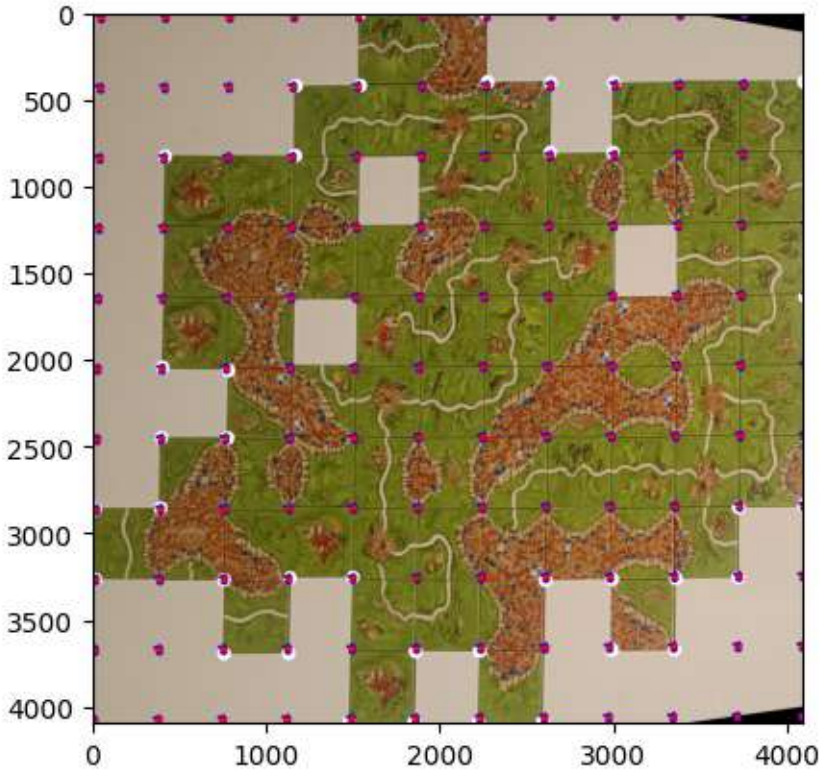
pločice koja se koristi dalje u programskom rješenju uzeli minimalnu vrijednost kandidata takvu da je većina drugih kandidata veće vrijednosti njen višekratnik [1]. Drugim riječima, kao širinu ne želimo uzeti neku vrijednost koja nema svoje višekratnike među kandidatima jer se oslanjamo na to da bi u idealnom slučaju sve ponuđene vrijednosti trebale biti višekratnici najmanje. Postupak određivanja visine i širine polja u mreži je analogan.

Nakon određivanja visine i širine pojedinih pločica, za svaki rub konture P generirali smo novu točku $P'(i,j) = (P.x + i * w, P.y + j * h)$ za $\forall i, j \in N, P'(i,j) \in D_x \times D_y$ gdje su w i h širina i visina pojedine pločice, a D_x i D_y predodređeni intervali za x i y koordinatu u kojima se mogu generirati točke. Takve točke se grupiraju po udaljenosti te se za konačnu mrežu uzima srednja vrijednost svake grupe točaka, prikazano na slici 5.8.



Slika 5.7. Primjer grupa točaka, svaka plava točka zapravo je skupina više malih plavih točaka gdje je svaka nastala translacijom jednog od kutova konture.

[1]Kako se radi o decimalnim vrijednostima, omjer $\frac{Wj}{Wi}, j > i$ mora biti unutar 0.15 od najbližeg cijelog broja



Slika 5.8. Primjer rekonstruirane mreže, crvene točke predstavljaju kutove četverokuta mreže, a dobivene su kao srednja vrijednost grupe plavih točaka.

5.2.5. Izrezivanje pločica

Na ploči 5.8. smo svaki minimalni četverokut izrezali i projicirali na kvadrat. Kako ne bismo nepotrebno spremali pločice koje pripadaju pozadini, pločice čije središte [2] ima dovoljno malu varijancu nismo spremali. Zbog projekcije ponekad može doći do stvaranja potpuno crnih piksela pa se pločice koje u svom središtu imaju udio jako crno/bijelih piksela veći od 0.05 uklanjaju.

5.3. Treniranje metričkog ugrađivanja

Za treniranje metričkog ugrađivanja koristili smo pet modela: ResNet50 model bez predtreniranja, fino ugođeni ResNet50 model predtreniran na skupu podataka ImageNet-1k, dva fino ugođene DINOv2 modela (*small* i *base*)i jednostavni konvolucijski model za metričko ugrađivanje opisan kodom 8..1 Korištena funkcija pogreške za metričko učenje je funkcija trojnog gubitka. Kao izlaz, različiti modeli ugrađuju slike u vektorske prostore različitih dimenzija:

[2]Središte pločice je kvadrat koji ima duljinu stranice dva puta manju od duljine pločice

- ResNet50-ImageNet-1k ugrađuje u 64-dimenzionalni vektorski prostor
- ResNet50 ugrađuje u 32-dimenzionalni vektorski prostor
- jednostavni konvolucijski model ugrađuje u 256-dimenzionalni vektorski prostor
- DINOv2-small ugrađuje u 64-dimenzionalni vektorski prostor
- DINOv2-base ugrađuje u 64-dimenzionalni vektorski prostor

Za stopu učenja uzeli smo vrijednost od 10^{-5} osim za jednostavni konvolucijski model i model ResNet50-ImageNet-1k gdje smo uzeli vrijednosti od 10^{-4}. Treniranje je trajalo 11 epoha za svaki model te su bilježeni podaci o iznosu funkcije gubitka na skupu za treniranje i testiranje. Funkcija pogreške trojnog gubitka izračunata je kao srednja vrijednost.

$$loss = \frac{ReLU(d(a,p) - d(a,n) + 1)}{\sum ReLU(d(a,p) - d(a,n) + 1)} \tag{5.1}$$

gdje je d(x, y) = ||x, y||, a varijable a, p i n su redom: sidro, pozitivni primjerak i negativni primjerak.

5.3.1. Fino ugađaje modela

Fino smo ugađali modele ResNet50 i DINOv2. Ugađanje smo postigli dodavanjem linearnih slojeva na izlaz te odmrzavanjem [3] težina samih modela.

Za fino ugađanje ResNet50 modela dodali smo dva linearna sloja između kojih se nalazi ReLU zglobnica. Na izlazu prvog i ulazu drugog sloja prenosi se vektor s 512 dimenzija.

Za fino ugađanje oba DINOv2 modela odmrznuli smo težine zadnja dva skrivena sloja, a na izlaz modela dodana su dva linearna sloja. Prvi linearni sloj ima 512 izlaznih dimenzija za model *small* i 1024 za model *base* koje se normaliziraju te prolaze kroz ReLU zglobnicu. Drugi sloj ima prikladan broj ulaznih dimenzija i 64 izlazne značajke koje se normaliziraju. Prilikom finog ugađanja, treniraju se težine ova dva linearna sloja.

[3] Odmrznute težine su one težine koje će biti promijenjene tijekom procesa strojnog učenja, zamrznute težine zadržavaju vrijednost dobivenu predtreniranjem

5.3.2. Jednostavan konvolucijski model

Jednostavan konvolucijski model sadrži tri konvolucijska sloja gdje se svaki sastoji od 2d normalizacije, ReLU zglobnice i dvodimenzionalne konvolucije. Između slojeva provodi se sažimanje, a nakon trećeg sloja provodi se globalno sažimanje. Model smo opisali kodom 8..1

5.4. Treniranje algoritma grupiranja

Treniranje algoritma grupiranja proveli smo primjenom algoritma k sredina nad ugrađivanjima koje su izvedene koristeći model DINOv2-base. Rezultat algoritma grupiranja je grupiranje skupa za treniranje u 57[4] grupa, nakon toga, koristeći mađarski algoritam, svakoj grupi dodijelili smo jedinstvenu oznaku klase.

5.4.1. Grupiranje primjenom algoritma k-sredina

Za izdvajanje značajki, koristili smo model DINOv2-base koji svakoj slici pridodijeljuje vektor od 768 dimenzija, a za grupiranje koristili smo klasu `KMeans` i njenu metodu `KMeans.fit_predict` biblioteke scikit-learn [14]. Algoritam u pozadini ove specifične metode je Lloydov algoritam koji smo opisali u uvodu. Pomoću biblioteke UMAP [23], moguće je vizualizirati rezultate algoritma grupiranja nad skupom podataka za treniranje, prikazano na slici 5.9.

5.4.2. Označavanje grupa Mađarskim algoritmom

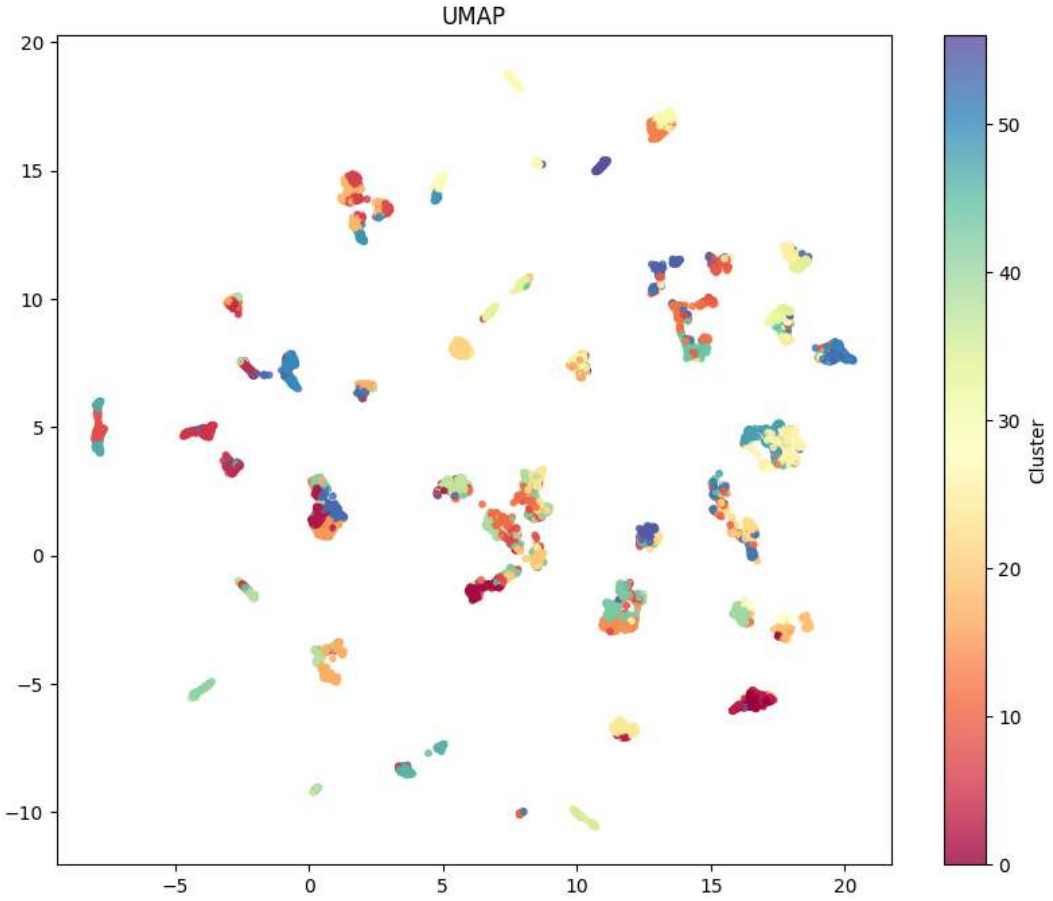
Kako bi svakoj grupi mogli pridodijeliti jedinstvenu klasu koristeći Mađarski algoritam, potrebno je definirati matricu cijena M. Matrica M definirana je kao

$$m_{i,j} = \begin{cases} (1 - f_j(i))^3 & \text{ako } f_j(i) \neq 0, \\ 100000 & \text{inače} \end{cases} \tag{5.2}$$

$f_j(i)$ predstavlja koliki udio klase j se nalazi u grupi i.

Na ovaj način, rezultat Mađarskog algoritma će biti potpuno sparivanje grupa i klasa na način da je suma cijena bridova, koje se temelje na matrici cijena, minimalna. Ma-

[4] Ukupno postoji 57 mogućih klasa za pločice igre Carcassonne



Slika 5.9. UMAP dijagram koji prikazuje projekciju ugrađivanja svih podataka na 2D plohu nakon grupiranja algoritmom k-sredina. Ista boja predstavlja pripadnike iste grupe. Iz slike je vidljivo da su pripadnici iste grupe nakon algoritma grupirani međusobno blizu.

tricu cijena dizajnirali smo tako da nagrađuje grupe niskom cijenom ako sadrže velik udio ukupnog pojavljivanja neke klase. Posebnim uvjetom pokušavamo spriječiti da se grupe spajaju s onim klasama koje one ne sadrže. Koristi se implementacija Mađarskog algoritma biblioteke SciPy [24].

6. Rezultati i rasprava

Mjerenje uspješnosti proveli smo na skupu podataka za testiranje nakon provedenog treniranja, također smo prikazali i rekonstrukciju ploče za sliku za različite modele 2.2.b

Kako bismo mogli usporediti uspješnost modela, moramo prvo definirati metrike po kojima se uspješnost uspoređuje. Za izračun različitih mjera uspješnosti često svaki podatak moramo podijeliti u jednu od 4 skupine: pozitivan ispravan (engl. *true positive*), negativan ispravan (engl. *true negative*), pozitivan neispravan (engl. *false positive*) i negativan neispravan (engl. *false negative*). Kako koncepti pozitivan i negativan imaju smisla samo u kontekstu binarne klasifikacije, u našem zadatku višeklasne klasifikacije metrike računamo pojedinačno za svaku klasu (gdje ta klasa predstavlja pozitivnu klasu, dok su sve druge klase negativne) te na kraju uzimamo njihovu aritmetičku sredinu. Uspoređene metrike su točnost, preciznost, odziv i F1-metrika. Točnost (engl. *accuracy*) je definirana kao udio ispravnih pozitivnih među svim podacima. Predstavlja koliko često model točno klasificira neki podatak.

$$accuracy = \frac{TP}{TP + TN + FP + FN} \tag{6.1}$$

Preciznost (engl. *precision*) definiramo kao udio točno klasificiranih pozitivnih podataka među svim podacima koji su klasificirani kao pozitivna klasa.

$$precision = \frac{TP}{TP + FP} \tag{6.2}$$

Odziv (engl. *recall*) definiramo kao udio točno klasificiranih pozitivnih podataka među svim podacima koje bismo trebali klasificirati kao pozitivnu klasu.

$$recall = \frac{TP}{TP + FN} \tag{6.3}$$

Metrike odziv i preciznost daju nam dodatan uvid u ponašanje modela i pogotovo su važne u nebalansiranim skupovima podataka gdje je točnost velika za modele koji na svoj izlaz skoro isključivo daju samo najčešću klasu.

F1-metrika (engl. *F1-metric*) objedinjuje metrike preciznosti i odziva, a računa se kao harmonična sredina te dvije metrike.

$$F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = \frac{2TP}{2TP + FP + FN} \tag{6.4}$$

6.1. Usporedba uspješnosti

Tablica 6.1. Usporedba uspješnosti za različite modele na skupu podataka za treniranje

Model	Točnost	Preciznost	Odziv	F1
Jednostavni konvolucijski model	0.555	0.598	0.571	0.567
ResNet50	0.513	0.502	0.492	0.482
ResNet50-ImageNet-1k	0.998	0.998	0.998	0.998
DINOv2-small	0.996	0.995	0.995	0.995
DINOv2-base	0.993	0.991	0.991	0.991
k-sredina	0.416	0.434	0.409	0.394

Tablica 6.2. Usporedba uspješnosti za različite modele na skupu podataka za testiranje

Model	Točnost	Preciznost	Odziv	F1
Jednostavni konvolucijski model	0.553	0.589	0.572	0.564
ResNet50	0.510	0.501	0.505	0.485
ResNet50-ImageNet-1k	0.979	0.997	0.997	0.997
DINOv2-small	0.993	0.991	0.992	0.991
DINOv2-base	0.991	0.990	0.991	0.990
k-sredina	0.406	0.430	0.409	0.390

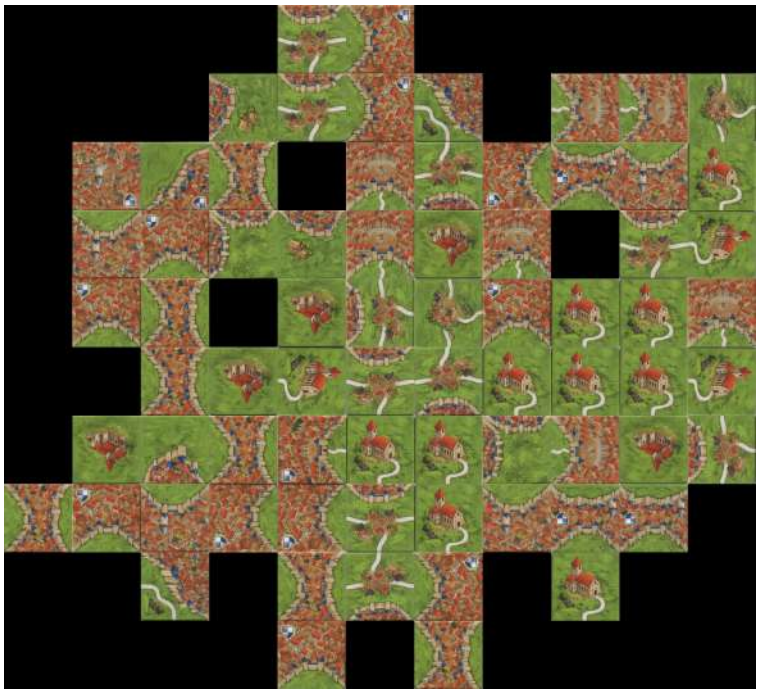
Vidimo da najbolju ukupnu točnost imaju DINOv2 i ResNet50-ImageNet-1k, što je očekivano jer su trenirani kompleksnijim modelima i imaju puno više parametara. Najgoru ukupnu točnost ima algoritam k-sredina, ovo vjerojatno proizlazi iz toga što algoritam pridodjeljuje klase skupinama koje sadrže jako mali udio te klase u pokušaju da minimalizira cijenu sparivanja. U ovom slučaju, svaka slika koja pripada takvoj klasi skoro sigurno neće biti točno klasificirana jer grupa koja sadrži najvećim udjelom tu klasu ima drugačiju oznaku klase. Ovo se dešava često za klase poput 3, 9, 27 i 81 (razne rotacije samostana s cestom, prikazano na slici 2.1.b), koje su dovoljno slične s obzirom na rotaciju

pa se prilikom grupiranja, grupiraju sve u istu grupu što rezultira grupom koja ima velik udio većeg broja klasa. Iznenađujuće je što DINOv2-small i DINOv2-base imaju skoro identične mjere uspješnosti. Iz rezultata također možemo uočiti da je F1 metrika sličnog iznosa točnosti, ovo implicira da modeli efikasno klasificiraju različite klase usprkos nebalansiranom skupu podataka.

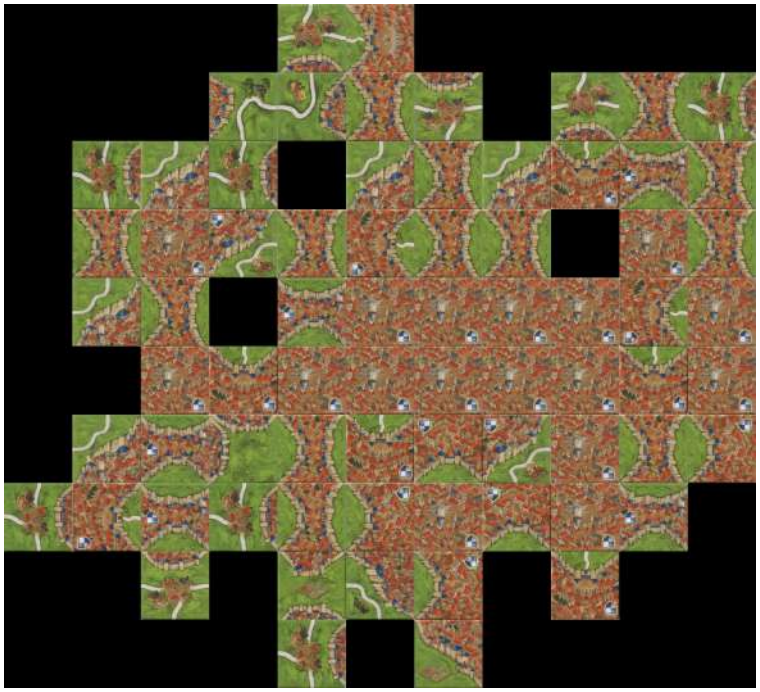
6.2. Rekonstruirane ploče

Iz rekonstrukcije slika, možemo primijetiti da algoritam k-sredina (slika 6.7.) radi najgore. Ulaskom u detalje, vidimo da ploča rekonstruirana tim algoritmom puno rjeđe od ostalih modela prepoznaje točnu klasu pločice, ali izgleda dobro iz daljine. Do ovoga vjerojatno dolazi zbog prije navedenog problema s grupama koje imaju veliki udio ukupnog broja više klasa. Jednostavni konvolucijski model (slika 6.2.) i model ResNet50 (slika 6.3.) imaju sličnu uspješnost, ali nakon rekonstrukcije njihove ploče ne liče nalik originalnoj ploči. Do ovoga moguće dolazi jer su modeli jednostavni i nisu dobro uspjeli naučiti bitne značajke. Modeli ResNet50-ImageNet-1k (slika 6.4.) i oba DINOv2 modela (slike 6.6. i 6.5.) skoro savršeno rekonstruiraju ploču. Zanimljivo je kako se model koji koristi algoritam k-sredina bolje prilagodio prelasku s augmentiranog skupa podataka na stvarne slike pločica. Ovo možemo vidjeti po znatno većoj uspješnosti rekonstrukcije u usporedbi s manjom uspješnosti na skupu podataka za testiranje.

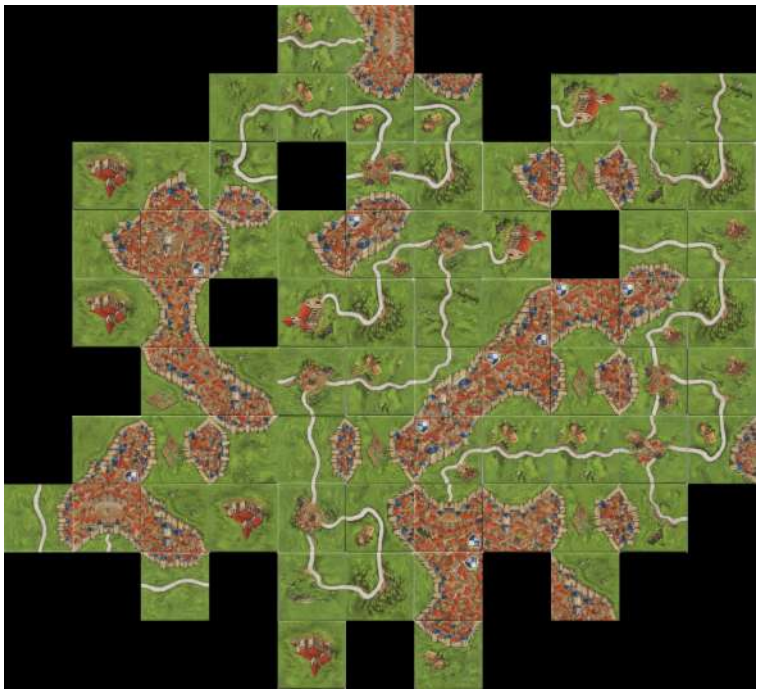
Slika 6.1. Slika za koju je provedena rekonstrukcija ploče za razčličite modele.



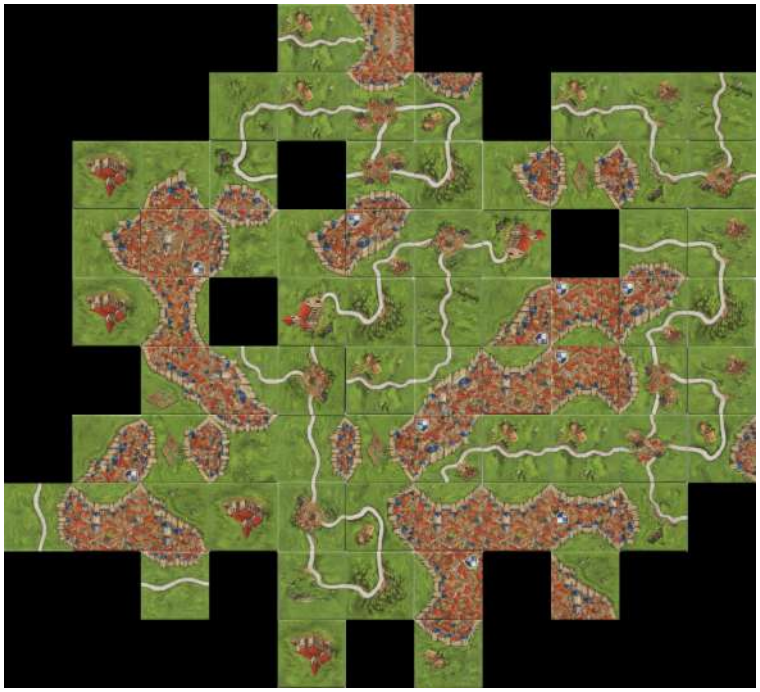
Slika 6.2. Primjer rekonstrukcije ploče koristeći jednostavan konvolucijski model, točno su rekonstruirana samo 5 polja što je 6.9% točnost rekonstrukcije.



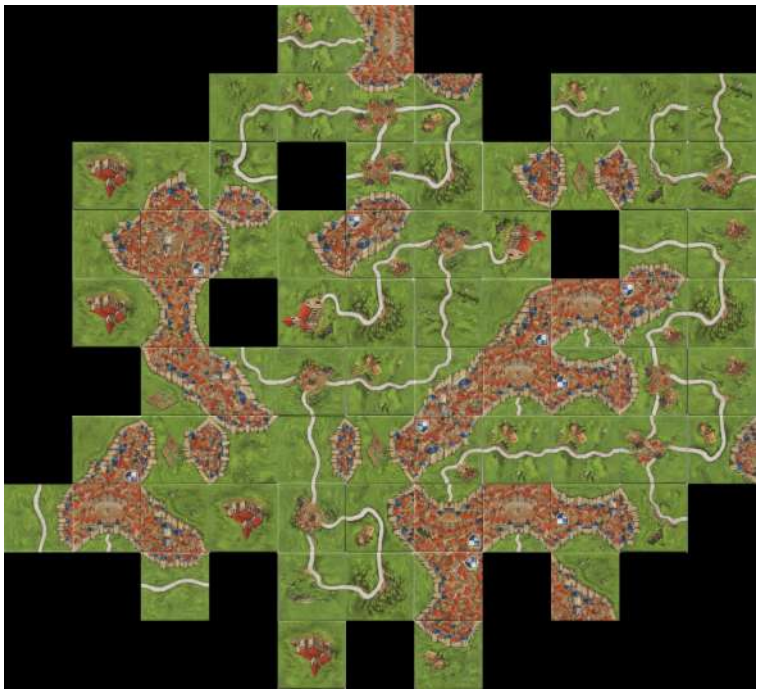
Slika 6.3. Primjer rekonstrukcije ploče koristeći ResNet50 model, točno su rekonstruirana samo 5 polja ka oi u slučaju jednostavnog modela što je ponovno 6.9% točnost rekonstrukcije.



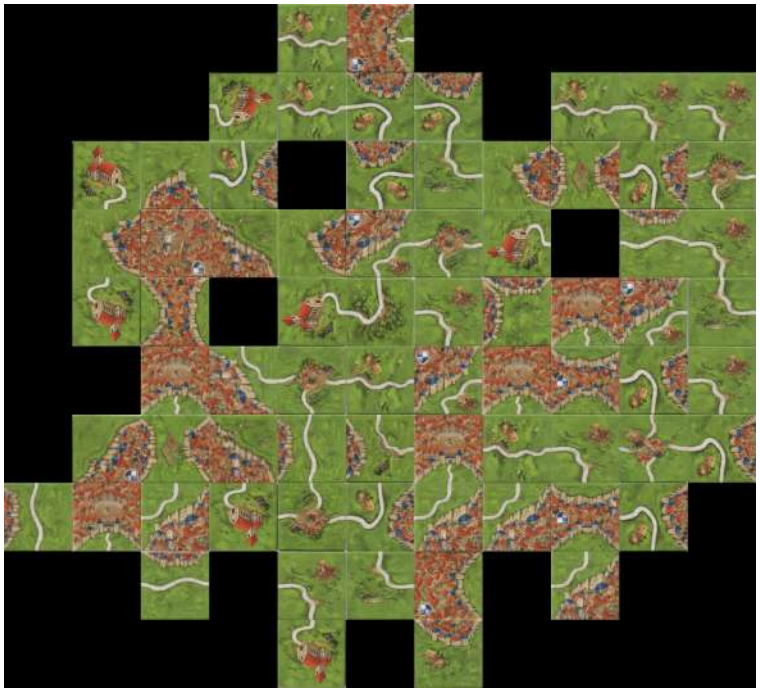
Slika 6.4. Primjer rekonstrukcije ploče koristeći predtrenirani ResNet50-ImageNet-1k model, točno je rekonstruirano 91.67% pločica.



Slika 6.5. Primjer rekonstrukcije ploče koristeći predtrenirani DINOv2-small model, točno je klasificirano 94.44% pločica.



Slika 6.6. Primjer rekonstrukcije ploče koristeći predtrenirani DINOv2-base model, točno je klasificirano 93.05% pločica.



Slika 6.7. Primjer rekonstrukcije ploče koristeći algoritam k-sredina, točno je rekonstruirano 43.05% pločica.

7. Zaključak

Tema završnog rada bila je rekonstrukcija ploče društvene igre Carcassonne primjenom različitih modela za klasifikaciju te kombinacijom algoritma k-sredina i Mađarskog algoritma.

Detaljno je objašnjen postupak kojim od slike ploče dolazimo do rekonstruirane slike. Različiti modeli su međusobno uspoređeni po točnosti, srednjoj preciznosti, srednjem odzivu i srednjoj F1 metrici. Objašnjeno je kako se od skeniranih pločica kreirao i particionirao skup podataka za učenje, validaciju i testiranje.

Opisan je i testiran jednostavan konvolucijski model baziran na metričkom ugrađivanju s trojnim gubitkom. Usporedbom s drugim navedenim modelima nadziranog učenja, ovaj model daje očekivano lošije rezultate što dolazi do izražaja u rekonstrukciji ploče 6.2.

Klasifikacija koristeći Mađarski algoritam i algoritam k-sredina ima manju točnost od drugih modela. Ovo je u radu obrazloženo time da sparivanje Mađarskim algoritmom ponekad spari klasu s grupom koja sadrži malen udio ukupnog pojavljivanja te klase. Umjesto Mađarskog algoritma vjerojatno bi bolji bio pristup gdje se grupa označava oznakom neke klase na temelju udaljenosti sredine od vektora značajki prototipa te klase.

Segmentacija ploče ima nekoliko pretpostavki poput bijele pozadine i određenog oblika ploče što smo mogli bolje napraviti drugačijim pristupom pronalaska mreže i uklanjanja pozadine.

Zadatak bi mogli dodatno proširiti učenjem na skupu podataka koji uključuje igraće figurice. U tom pravcu, mogli bi i trenirati model koji prepoznaje poziciju figurica te provodi automatsko bodovanje s obzirom na pravila i rekonstruiranu ploču. Prilikom

same rekonstrukcije postoje bitna pravila koja nismo uzeli u obzir, a mogla bi značajno utjecati na kvalitetu rekonstrukcije.

U rekonstrukciji, svaku pločicu klasificiramo zasebno i nezavisno od drugih pločica. U pravoj igri, svaka pločica mora biti smisleno postavljena s obzirom na svoje susjede (dijeljene stranice moraju biti istoga tipa) što znači da su same klase pločica zavisne sa svojim susjedima. U svakoj igri ukupan broj svake klase je konstantan, što je također nešto što nismo koristili, a moglo bi spriječiti rekonstrukcije poput one koju daje jednostavan konvolucijski model (slika 6.2.).

8. Korišteni kod

```
1
2  class _BNReluConv(nn.Sequential):
3      def __init__(self, num_maps_in, num_maps_out, k=3, bias=True):
4          super(_BNReluConv, self).__init__()
5          self.append(nn.BatchNorm2d(num_maps_in))
6          self.append(nn.ReLU())
7          self.append(nn.Conv2d(
8              num_maps_in,
9              num_maps_out,
10             k,
11             padding=k//2,
12             bias=bias)
13         )
14
15 class SimpleMetricEmbedding(nn.Module):
16     def __init__(self, input_channels, emb_size=32):
17         super().__init__()
18         self.emb_size = emb_size
19         self.conv1 = _BNReluConv(input_channels, emb_size)
20         self.conv2 = _BNReluConv(emb_size, emb_size)
21         self.conv3 = _BNReluConv(emb_size, emb_size)
22         self.pool = nn.MaxPool2d(kernel_size=3,
23             stride=4,
24             padding=1
25         )
26         self.global_pool = nn.AvgPool2d(9)
27
28     def get_features(self, img):
29         x = self.conv1(img)
30         x = self.pool(x)
```

```
        x = self.conv2(x)
        x = self.pool(x)
        x = self.conv3(x)
        x = self.global_pool(x)
        x = x.view(x.size(0), -1)
        return x

    def loss(self, anchor, positive, negative, identity=False):
        a_x = self.get_features(anchor)
        p_x = self.get_features(positive)
        n_x = self.get_features(negative)
        distP = F.pairwise_distance(a_x, p_x)
        distN = F.pairwise_distance(a_x, n_x)
        loss = F.relu(torch.max(
            distP - distN + 1,
            torch.tensor(0.0)
        )).mean()
        return loss

    def forward(self, img):
        features = self.get_features(img)
        return features
```

Listing 8..1: Jednostavan konvolucijski model

Literatura

[1] Dalbelo Bašić,B., Čupić,M., Šnajder,J. Umjetne neuronske mreže mreža, https://www.fer.unizg.hr/_download/repository/UI_12_UmjetneNeuronskeMreze[1].pdf, [mrežno; pristupljeno: Svibanj 2025.].

[2] Subašić,M. Detekcija objekata u slikama pomoću dubokih neuronskih mreža, https://www.fer.unizg.hr/_download/repository/Detekcija.pdf, [mrežno; pristupljeno: Svibanj 2025.].

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, i I. Polosukhin, "Attention is all you need", 2023. [Mrežno]. Adresa: https://arxiv.org/abs/1706.03762

[4] J. Devlin, M.-W. Chang, K. Lee, i K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", 2019. [Mrežno]. Adresa: https://arxiv.org/abs/1810.04805

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, i I. Sutskever, "Language models are unsupervised multitask learners", *OpenAI*, 2019., accessed: 2024-11-15. [Mrežno]. Adresa: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, i D. Amodei, "Language models are few-shot learners", 2020. [Mrežno]. Adresa: https://arxiv.org/abs/2005.14165

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, i N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale", 2021. [Mrežno]. Adresa: https://arxiv.org/abs/2010.11929

[8] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, i P. Bojanowski, "Dinov2: Learning robust visual features without supervision", 2024. [Mrežno]. Adresa: https://arxiv.org/abs/2304.07193

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, i B. Schiele, "The cityscapes dataset for semantic urban scene understanding", 2016. [Mrežno]. Adresa: https://arxiv.org/abs/1604.01685

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, i L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", u *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009., str. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[11] L. Ericsson, H. Gouk, C. C. Loy, i T. M. Hospedales, "Self-supervised representation learning: Introduction, advances, and challenges", *IEEE Signal Processing Magazine*, sv. 39, br. 3, str. 42–62, svibanj 2022. https://doi.org/10.1109/msp.2021.3134634

[12] Šegvić,S. Učenje sličnosti mreža, https://www.zemris.fer.hr/ ssegvic/du/du7metrics.pdf.

[13] D. MacKay, "Chapter 20: An example inference task: Clustering", u *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003., str. 284–292, accessed: [insert date of access]. [Mrežno]. Adresa: http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/284.292.pdf

[14] scikit-learn, https://scikit-learn.org/stable/, [mrežno; pristupljeno: Travanj 2025.].

[15] J. Munkres, “Algorithms for the assignment and transportation problems”, *Journal of the Society for Industrial and Applied Mathematics*, sv. 5, br. 1, str. 32–38, 1957. https://doi.org/10.1137/0105003

[16] OpenCV, https://pypi.org/project/opencv-python/, [mrežno; pristupljeno: Svibanj 2025.].

[17] Albumentations, https://albumentations.ai/, [mrežno; pristupljeno: Svibanj 2025.].

[18] PyTorch, https://pytorch.org/, [mrežno; pristupljeno: Svibanj 2025.].

[19] TensorFlow, https://www.tensorflow.org/, [mrežno; pristupljeno: Svibanj 2025.].

[20] R. S. Dehal, C. Munjal, A. A. Ansari, i A. S. Kushwaha, “Gpu computing revolution: Cuda”, u *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018., str. 197–201. https://doi.org/10.1109/ICACCCN.2018.8748495

[21] R. Srisha i A. Khan, “Morphological operations for image processing : Understanding and its applications”, 12 2013.

[22] S. Suzuki i K. be, “Topological structural analysis of digitized binary images by border following”, *Computer Vision, Graphics, and Image Processing*, sv. 30, br. 1, str. 32–46, 1985. https://doi.org/https://doi.org/10.1016/0734-189X(85)90016-7

[23] UMAP, https://umap-learn.readthedocs.io/en/latest/, [mrežno; pristupljeno: Travanj 2025.].

[24] SciPy linear-sum-assignment, https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html, [mrežno; pristupljeno: Travanj 2025.].

Sažetak

Prepoznavanje pločica društvene igre Carcassonne

Lovro Nidogon

Rad opisuje segmentaciju igraće ploče društvene igre Carcassonne na igraće pločice primjenom metoda računalnog vida. Opisan je način treniranja konvolucijskih modela zasnovanih na metričkom ugrađivanju. Uspoređene su metrike uspješnosti za različite fino ugođene i trenirane modele na augmentiranom skupu podataka. Opisan je način treniranja i klasifikacije modela koji kombinira algoritme k-sredina i Mađarski algoritam. Uspoređena je kvaliteta rekonstrukcije ploče za različite modele.

Ključne riječi: metričko učenje; algoritam k-sredina; konvolucijske neuronske mreže; računalni vid; augmentacija skupa podataka;

Abstract

Recognition of Carcassonne board game tiles

Lovro Nidogon

The paper describes the segmentation of the board of the board game Carcassonne into playing pieces using computer vision methods. A method for training convolutional models based on metric embedding is described. Performance metrics for different fine-tuned and trained models on an augmented dataset are compared. A method for training and classifying models that combines the k-means and Hungarian algorithms is described. The quality of the board reconstruction for different models is compared.

Keywords: metric learning; k-means algorithm; convolutional neural networks; computer vision; dataset augmentation;