SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1730

Procjenjivanje nesigurnosti predikcija diskriminativnih konvolucijskih modela primjenom suparničkog učenja

Josip Šarić

Zagreb, lipanj 2018.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

U posljednje vrijeme metode dubokog učenja pokazale su se nadmoćnima nad klasičnim metodama u rješavanju narazličitijeg skupa problema računalnog vida. Važna osobina sustava prilikom primjene dubokih modela je mogućnost procjene nesigurnosti predikcija. Uobičajeni način za procjenu nesigurnosti je iskoristiti izlaze modela iz funkcije softmax koji glume vjerojatnosnu distribuciju po predviđanim razredima. To se pokazalo problematično u više slučajeva i nedovoljno precizno. Suparnički modeli sa svojom nenadziranom prirodom i mogućnosti učenja nad neoznačenim podacima uspješno su primijenjeni i donijeli su poboljšanja u različitim problemima računalnog vida.

Cilj ovoga rada bio je proučiti i opisati postojeće pristupe za procjenu nesigurnosti predikcija, kao i istražiti i implementirati postupke procjene nesigurnosti dubokih konvolucijskih modela primjenom nenadziranog suparničkog učenja. Dodatno, bilo je potrebno implementirati i evaluirati ove postupke na konkretnom zadatku semantičke segmentacije te ocijeniti i usporediti rezultate sa uobičajenim pristupima.

Ostatak ovog rada strukturiran je na sljedeći način. U drugom poglavlju dan je uvod u ideju strojnog učenja i opisane su tri osnovne paradigme učenja. Nakon toga u trećem poglavlju opisani su osnovni pojmovi dubokog učenja, arhitektura ResNet i osnove suparničkog učenja. Slijedi poglavlje u kojemu je opisan problem primjene funkcije softmax kao detektora nesigurnosti, dan pregled dosadašnjih metoda za procjenu nesigurnosti predikcija i opisana moguća primjena kroz suparničko učenje za problem semantičke segmentacije. U petom poglavlju dani su detalji vezani uz implementaciju i točno su opisane korištene arhitekture generatora i diskriminatora kao osnovnih cjelina suparničkog modela. Evaluacija i rezultati eksperimenata opisani su u šestom poglavlju. Na kraju rada dan je zaključak.

2. Strojno Učenje

Strojno učenje je grana računarske znanosti koja se bavi proučavanjem sposobnosti učenja računala iz podataka. Ethem Alpaydin ga je opisao kao programiranje računala na način da optimiziraju neki kriterij uspješnosti temeljem podatkovnih primjera ili prethodnog iskustva [2]. Postoji model koji je definiran do na neki parametar i učenje je zapravo izvršavanje računalnog programa koji optimira te parametre modela koristeći se podacima ili prethodnim iskustvom. Modeli mogu biti prediktivni, oni koji rade neka predviđanja o budućnosti, ili deskriptivni koji bolje opisuju i pomažu pri razumijevanju podataka. Ideja je da model tijekom učenja stekne sposobnost generalizacije i povjereni mu zadatak obavlja dobro na primjerima odnosno podacima koje nije vidio. U strojnom učenju poznate su tri osnovne paradigme učenja:nadzirano učenje, nenadzirano učenje te potporno učenje.

2.1. Paradigme učenja

2.1.1. Nadzirano učenje

Kao motivacijski primjer promotrimo zadatak određivanja rukom pisane znamenke koja je prikazana na grayscale slici veličine $28 \times 28 = 784$ piksela [3]. Cilj je stvoriti program koji će na ulazu dobiti sliku koju možemo predstaviti vektorom od 784 realna broja, a na izlazu dati znamenku predstavljenu ulaznom slikom. Ovo je netrivijalan problem. Mogao bi se rješavati stvaranjem nekih pravila ili primjenom heuristika, ali to bi nas dovelo do ogromnog broja složenih pravila i iznimki, a u praksi, zapravo daje loše rezultate. Puno prirodniji (i lakši) način za rješavanje ovoga problema bio bi primjenom strojnog učenja. Prvi korak bio bi prikupiti skup primjera/slika takav da je svakom primjeru pridružena oznaka. U našem slučaju svakoj slici bi bila pridružena znamenka koja je predstavljena na slici. Te oznake određuje čovjek, kojemu je ovaj zadatak zapravo lagan. U općenitom slučaju takav skup zovemo skupom za učenje (treniranje) i ako imamo ukupno N primjera možemo ga označiti sa $D = \left\{(\boldsymbol{x}^i, y^i)\right\}_{i=1}^{N}$

[24]. Svaki $\boldsymbol{x}^i$ je primjer predstavljen vektorom značajki dimenzije n (u našem slučaju 784), a y^i oznaka toga primjera (znamenka). Napomenimo kako u općenitom slučaju oznaka ne mora biti skalar, nego vektor što znači da za jedan primjer moramo odrediti više oznaka/razreda. U nadziranom učenju dakle, model u fazi učenja dobiva skup primjera s oznakama. Još jedno ime je „učenje s učiteljem“ jer možemo zamisliti da postoji učitelj kao izvor istine i nepokolebljivog znanja, koji modelu prilikom učenja za svaki primjer kaže stvarnu odnosno željenu oznaku. Naš motivacijski primjer klasičan je primjer klasifikacije, u kojem se za svaki primjer određuje razred/klasa kojem on pripada. Drugi problem je problem regresije, kod koje se za svaki primjer predviđa neka kontinuirana vrijednost. Razlika je dakle u tome je li oznaka nominalna (diskretna) ili kontinuirana vrijednost. Klasičan primjer regresije je algoritam linearne regresije.

2.1.2. Nenadzirano učenje

Kod nenadziranog učenja ne postoji više učitelj kao izvor istine, nego samo skup primjera sa nekim brojem značajki, ali bez oznaka. Možemo ga zapisati kao: $D = \left\{\boldsymbol{x}^i\right\}_{i=1}^{N}$ Tri tipična zadatka nenadziranog učenja su grupiranje (eng. *clustering*) podataka, otkrivanje novih vrijednosti ili vrijednosti koje odskaču (eng. *novelty/outlier detection*) i smanjenje dimenzionalnosti (engl. *dimensionality reduction*) [24].

2.1.3. Potporno učenje

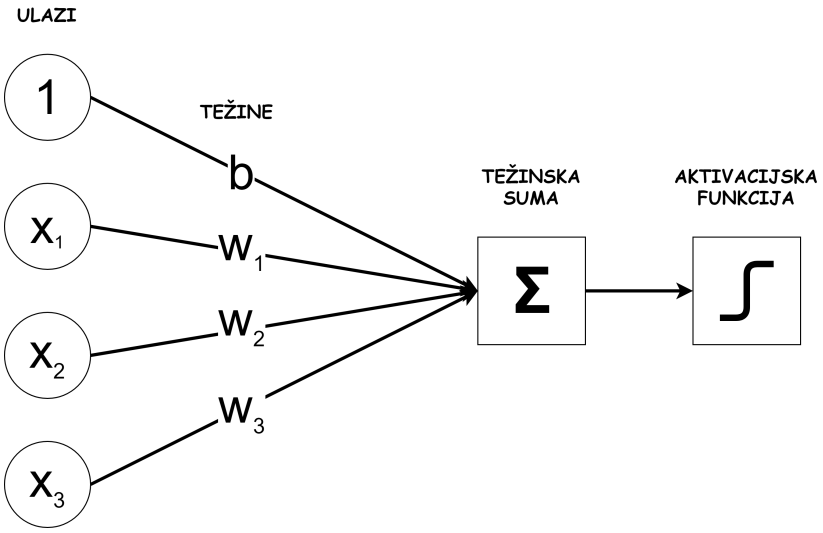
U potpornom učenju (eng. *reinforcement learning*) [2] učenik je agent koji donosi odluke u okolini od koje za svaku svoju odluku ili akciju dobije nagradu ili kaznu u pokušaju da riješi problem. Razlika u odnosu na nadzirano učenje je u tome što ne postoji učitelj koji govori što točno treba raditi, nego povratna informacija govori kako smo dobro napravili „do sada“ ili „u prošlosti“. Ovakvo učenje još se naziva i učenje kritikom, a ono nikada ne informira unaprijed. Uobičajeno je da se agent modelira Markovljevim procesom, a rješenje problema je zapravo sekvenca akcija koje agent mora napraviti.

3. Duboko učenje

Duboko učenje je grana strojnog učenja temeljena na učenju složenih reprezentacija različitih vrsta podataka. Metode dubokog učenja drastično su unaprijedile rezultate u prepoznavanju govora, obradi prirodnog jezika, detekciji i prepoznavanju objekata na slici i mnogim drugim problemima do tada rješavanim klasičnim metodama umjetne inteligencije. To su metode učenja reprezentacije podataka koja je kompozitne odnosno hijerarhijske prirode sa više razina apstrakcije. Reprezentacija je obično naučena kroz niz modula od kojih svaki radi jednu nelinearnu transformaciju podataka, počevši od sirovih podataka preko niza skrivenih reprezentacija do krajnje razine apstrakcije koja je prikladna za određeni zadatak, primjerice klasifikaciju. Kompozicijom takvih transformacija moguće je naučiti veoma kompleksne funkcije. Ako promatramo zadatak klasifikacije slika, prva reprezentacija obično pokazuju prisutnost različito orijentiranih i lokaliziranih rubova. Druga reprezentacija predstavlja kombinacije tih rubova u različitim aranžmanima, a sljedeća bi mogla detektirati jednostavni dio nekog objekta. Kasnije reprezentacije mogu predstavljati i prisutnost cijelog objekta kao kombinacije prethodno detektiranih dijelova. Na tome primjeru jasno je vidljiva mogućnost učenja složenih kompozitnih struktura s naglaskom na učenju, a ne dizajniranju tih značajki od strane ljudskih eksperata.

3.1. Višeslojni perceptron

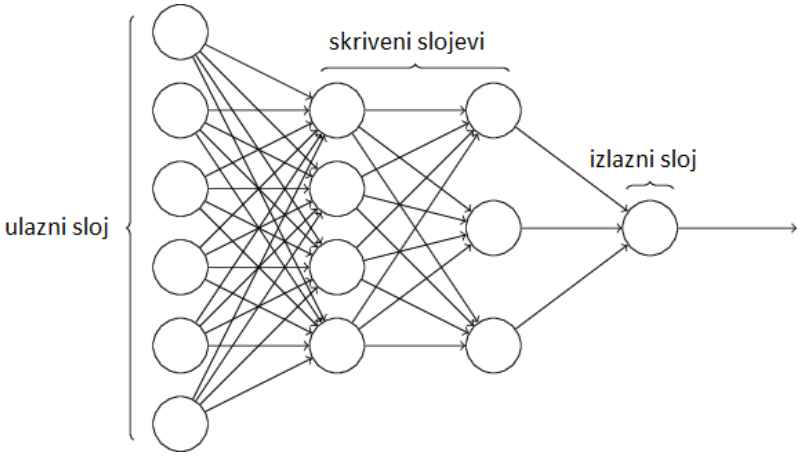
Osnovni model u dubokom učenju je višeslojni perceptron ili neuronska mreža, koja je inspirirana neuroznanosti i radom ljudskoga mozga. Klasična neuronska mreža se sastoji od međusobno povezanih grupa osnovnih procesnih jedinica zvanih neuronima. Najjednostavniji model neurona prikazan je na slici 3.1. Svakom ulazu je dakle pridružena njegova težina (parametar koji se uči) koja ga množi, nakon čega se ti umnošci sumiraju skupa sa pomakom, a konačan izlaz se dobiva aktivacijom te sume. Mogli bismo dakle izlaz jednoga neurona opisati kao: $y = f(\boldsymbol{w}^T \boldsymbol{x} + b)$ gdje je $f(x)$ aktivacijska funkcija o čemu će više riječi biti kasnije. Uobičajeno je neurone grupirati u slojeve,



Slika 3.1: model neurona

a neuronsku mrežu predstaviti nizom slojeva. U unaprijednoj neuronskoj mreži protok informacija događa se u samo jednom smjeru, dakle izlaz trenutnog sloja ovisi samo o izlazima prethodnih slojeva. Ako izlaz sloja ovisi o svim izlazima prethodnoga sloja, onda govorimo o potpuno povezanom sloju.

Transformaciju koju obavlja potpuno povezani sloj možemo iskazati funkcijom $f(W\boldsymbol{x} + \boldsymbol{b})$, gdje je W matrica svih težina u sloju, a $\boldsymbol{b}$ vektor pomaka. Jedan redak matrice W predstavlja težine, a jedna komponenta vektora b pomak jednog neurona. Problem potpuno povezanog sloja je što uslijed visokodimenzionalnih reprezentacija unosi ogroman broj parametara koje treba naučiti i često nepotrebnu složenost u cjelokupni model. Unaprijedni potpuno povezani modeli važni su više iz povijesnih razloga, nego njihove današnje praktične primijene



Slika 3.2: unaprijedna neuronska mreža

3.2. Učenje dubokih modela

Pod pojmom učenja neuronskih mreža uobičajeno podrazumijevamo optimizaciju neke kriterijske funkcije ovisne o parametrima neuronske mreže $\boldsymbol{\Theta}$. Kriterijsku funkciju zovemo i funkcijom gubitka koju računamo nad skupom za učenje:

$$J(\boldsymbol{\Theta}) = E_{(\boldsymbol{x},y)\sim p_{data}}[L(f(\boldsymbol{x};\boldsymbol{\Theta}), y)] = \frac{1}{N}\sum_{i=1}^{N} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\Theta}), y^{(i)}) \tag{3.1}$$

gdje $f(\boldsymbol{x};\boldsymbol{\Theta})$ predstavlja funkciju koju obavlja neuronska mreža, a L gubitak izračunat nad jednim primjerom.

U neuronskim mrežama najčešće se radi o gradijentnoj optimizaciji skalarne funkcije više varijabli. Gradijent takve funkcije bit će vektor koji će u sebi sadržavati sve parcijalne derivacije. Označimo funkciju koju optimiramo sa f prema Taylorovom razvoju prvog reda vrijedi:

$$f(\boldsymbol{x} + \boldsymbol{\Delta x}) = f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{\Delta x} \tag{3.2}$$

Ako sada za pomak $\boldsymbol{\Delta x}$ uvrstimo negativan gradijent pomnožen sa hiperparametrom ϵ dobivamo:

$$\boldsymbol{\Delta x} = -\epsilon \cdot \boldsymbol{\nabla} f(\boldsymbol{x}) \tag{3.3}$$

$$f(\boldsymbol{x} + \boldsymbol{\Delta x}) = f(\boldsymbol{x}) - \epsilon \cdot \boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{\nabla} f(\boldsymbol{x}) \tag{3.4}$$

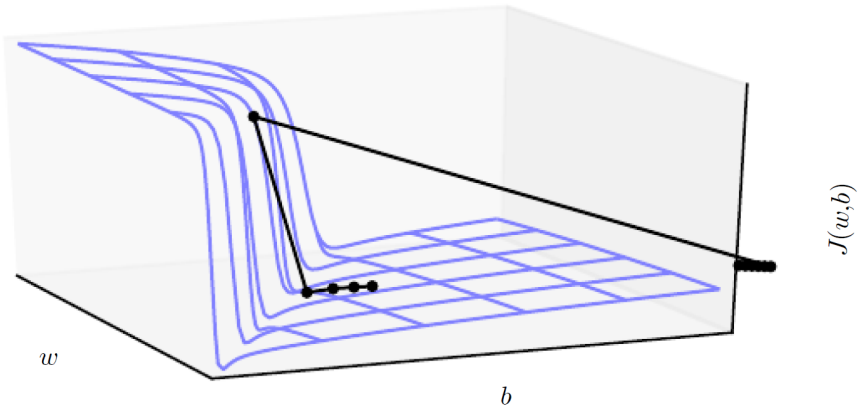
Vrijednost funkcije f bi očigledno trebala opadati. Jednostavno intuitivno objašnjenje gradijentnog spusta uz poznavanje činjenice da gradijent funkcije pokazuje u smjeru najvećeg rasta funkcije je to da se s ciljem minimizacije onda želimo kretati u suprotnom smjeru. Hiperparametar ϵ u kontekstu neuronskih mreža zovemo stopom učenja. Pokazat će se kao bitan i osjetljiv hiperparametar jer u slučaju postavljanja prevelike vrijednosti optimizacija može potpuno divergirati, a u slučaju premale pomaci će biti mali i optimizacija će biti spora s velikom vjerojatnosti zapinjanja u nekome lokalnom optimumu. Za uspješnu optimizaciju potrebno je pronaći neku zlatnu sredinu, za što nažalost ne postoji egzaktan postupak, nego se obično radi metodom pokušaja i pogreške.

Prilikom optimizacije funkcije gubitka neuronske mreže, gradijente bi trebalo izračunati za sve primjere iz skupa za treniranje kojih može biti jako puno. Takav izračun je precizan, ali računalno skup i obično se zbog toga ne koristi. Dijametralno suprotno od toga bilo bi računati gradijent i ažurirati parametre za svaki primjer posebno. Mane toga pristupa su sljedeće: onemogućuje kvalitetno iskorištavanje hardwareske paralelizacije i gradijent je neprecizan(slaba aproksimacija jer minimiziramo gubitak nad

cijelim skupom za učenje). Uobičajeno je danas učenje nad mini grupama (eng. *mini-batch*). Umjesto za jedan primjer, računa se prosječni gradijent nad manjom skupinom primjera i nakon toga se vrši ažuriranje. Gradijenti su precizniji što je veličina mini grupe veća, ali se također pokazalo da učenje sa minigrupom ima i dobre regularizacijske osobine jer u ažuriranja unosi mali šum koji može pomoći prilikom izbjegavanja lokalnog optimuma.

Naime, cilj svake optimizacije je pronaći globalni optimum, međutim za veliki broj funkcija je to jako teško. Funkcija gubitka kod neuronskih mreža je nekonveksna funkcija u visokodimenzionalnom prostoru sa puno lokalnih minimuma. Rezultat postupka učenja najčešće i nije točka globalnog minimuma, nego neka točka koja je dovoljno dobra i zadovoljava naše kriterije.

Lokalni optimumi nisu jedini problem za optimizaciju funkcije gubitka. Prema [10] puno se češće točke sedla. U njihovoj lokalnoj okolini neke točke imaju veću vrijednost, a neke manju. Možemo zamisliti kao da je to lokalni maksimum po jednom presjeku te funkcije, a ujedno i lokalni minimum po nekom drugom presjeku. Gradijent u tim točkama je jednak nuli i moguće je da će optimizacijski postupak tu zapeti jer su zbog maloga gradijenta pomaci također neznatni ili nepostojeći. Još veći problem od sedla predstavljaju velike široke regije sa konstantnom vrijednosti funkcije. U takvim područjima gradijent je također jednak nuli, ali za razliku od sedla ovdje ne postoji šansa da će postupak "napipati" nizbrdicu koja bi mu pomogla da napravi pomak. Problem nisu samo nepostojeći gradijenti, nego i magnitudom jako veliki gradijenti. To susrećemo kod pojave takozvanih litica, naglih i velikih padova vrijednosti funkcije kao što je prikazano na slici 3.3. Veliki gradijent mogao bi nas navesti na veliki korak kojim bismo potencijalno mogli vratiti optimizacijski postupak na početak kao da smo zaboravili sve što smo do toga trenutka naučili. Spomenimo još i pro-



Slika 3.3: ilustracija pojave litice prilikom optimizacije funkcije gubitka [10]

blem nesklada lokalne i globalne strukture funkcije gubitke. To je slučaj u kojem nas lokalna informacija iz trenutne točke navodi na pomake koji su potpuno suprotni od smjera globalnoga minimuma.

Kako bi se nadišli svi(neki) od ovih problema razvijeni su razni algoritmi učenja bazirani na gradijentnoj optimizaciji. Najjednostavniji takav algoritam poznat je pod imenom stohastički gradijentni spust. Njegov pseudokod opisan je u nastavku.

Algoritam 1: Stohastički gradijentni spust

Ulaz: Stopa učenja ϵ

Ulaz: Početni parametri $\boldsymbol{\Theta}$

$k \leftarrow 1$ **dok** *nije ispunjen uvjet zaustavljanja* **radi**

uzorkuj mini-grupu od m primjera $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ sa odgovarajućim oznakama $\boldsymbol{y}^{(i)}$

izračunaj procjenu gradijenta: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\Theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}), y^{(i)})$

napravi ažuriranje: $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \epsilon \hat{\boldsymbol{g}}$

$k \leftarrow k + 1$

kraj

U posljednje vrijeme najboljim se pokazao algoritam Adam (od eng. *adaptive moments*)[18]. U odnosu na gradijentni spust ova metoda ima dva nova svojstva: korištenje momenta i adaptivna stopa učenja. Ideja momenta je iskoristiti informaciju sadržanu u gradijentima iz prošlosti kako bi se smanjio šum prilikom trenutnog ažuriranja i dodatno usmjerila optimizacija ka globalnom optimumu. Ime moment dolazi iz fizike. To se radi na način da se u jednu varijablu akumuliraju gradijenti iz prošlosti, ali u svakom koraku pomnoženi sa faktorom propadanja. Ta varijabla često se naziva i brzina jer pokazuje globalni smjer kretanja parametara. Eksponencijalnim propadanjem postiže se efekt zaboravljanja jako starih gradijenata i pridavanja veće važnosti novijim. Ideja adaptivne stope učenje jeste umjesto korištenja jedne stope učenja za sve parametre mreže, svakome parametru dodijeliti njegovu stopu učenja koja se prilagođava tijekom postupka učenja. To se radi na način da se u za svaki parametar posebno akumuliraju kvadrati gradijenta uz korištenje iste ideje eksponencijalnog propadanja kao kod momenta. Ta akumulirana varijabla se prilikom ažuriranja koristi kako bi se skalirao globalno zadani hiperparametar stope učenja. Konkretna implementacija ovih metoda u algoritmu Adam može se vidjeti u pseudokodu koji slijedi.

Algoritam 2: ADAM [18]

Ulaz: Stopa učenja ϵ

Ulaz: Eksponencijalne mjere propadanja za procjenu momenta, $\rho_1, \rho_2 \in [0, 1)$.(Predložene vrijednosti su: 0.9 i 0.999)

Ulaz: Mala konstanta δ korištena za numeričku stabilizaciju (predložena vrijednost:10^{-8})

Ulaz: Početni parametri $\boldsymbol{\Theta}$

Inicijaliziraj vremenski korak $t = 0$

Inicijaliziraj prvu i drugu varijablu za momente $\boldsymbol{s} = \mathbf{0}, \boldsymbol{r} = \mathbf{0}$

dok *nije ispunjen uvjet zaustavljanja* **radi**

- uzorkuj mini-grupu od m primjera $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ sa odgovarajućim oznakama $\boldsymbol{y}^{(i)}$
- izračunaj procjenu gradijenta: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\Theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}), y^{(i)})$
- $t \leftarrow t + 1$
- ažuriraj prvi pristrani moment: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
- ažuriraj drugi pristrani moment: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
- korigiraj pristranost prvog momenta: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
- korigiraj pristranost drugog momenta: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
- izračunaj ažuriranje: $\Delta\boldsymbol{\Theta} \leftarrow -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$
- napravi ažuriranje: $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} + \Delta\boldsymbol{\Theta}$

kraj

3.3. Duboki konvolucijski modeli

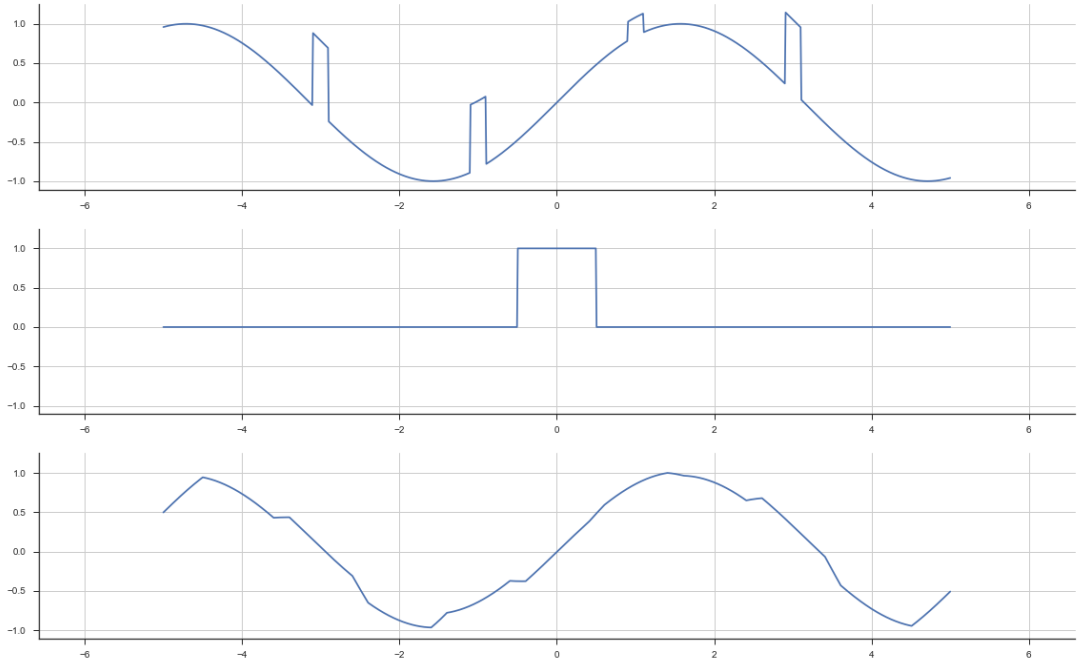
Za uspjeh dubokih modela u rješavanju problema računalnog vida posebno su zaslužni konvolucijski modeli. To su modeli koji imaju barem jedan konvolucijski sloj. U nastavku bit će posebno razmatran konvolucijski sloj i drugi slojevi koji se često koriste u dubokim konvolucijskim modelima, kao i posebna skupina modela sa takozvanim preskočnim vezama.

3.3.1. Konvolucijski sloj

Konvolucija u području signala i sustava, već je dugo i dobro poznat pojam. To je matematička operacija nad dvije funkcije, čiji rezultat prikazuje količinu preklapanja između njih dok se jedna pomiče preko druge. Definirana je sljedećim izrazom:

$$f(t) * g(t) = \int_I f(\tau)g(t-\tau)d\tau \tag{3.5}$$

Konvoluciju možemo vizualizirati na način da zamislimo da se graf jedne funkcije giba konstantnom brzinom duž osi apscisa, a u svakom trenutku izračunavamo površinu presjeka što odgovara iznosu njihove konvolucije. U području signala i sustava, često jedna funkcija odgovara nekome signalu u vremenu, a druga filtru kojim želimo nad signalom napraviti neku transformaciju. Na slici je na prvom grafu sinusoidni signal



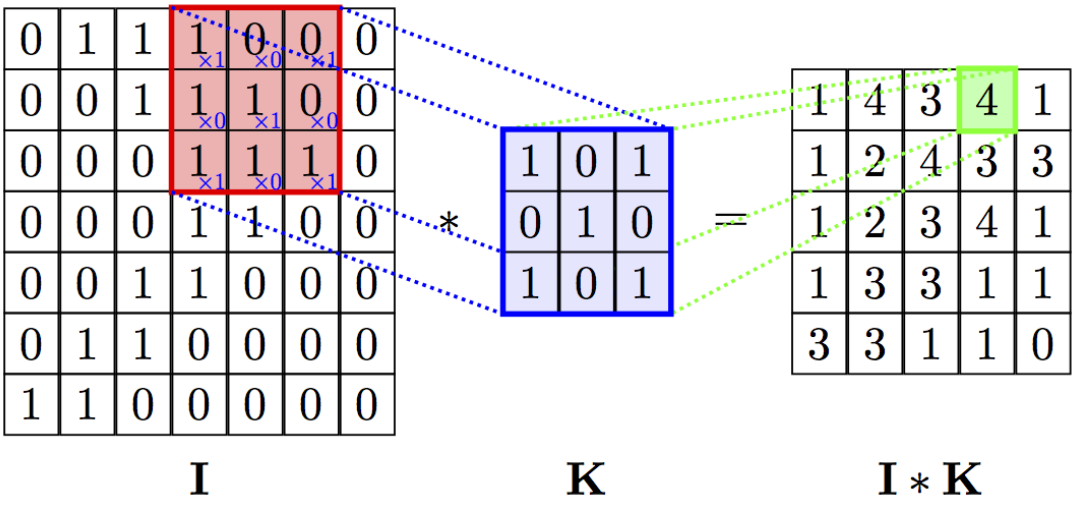
Slika 3.4: konvolucija signala

kojemu je na nekim dijelovima dodan šum, a na drugom grafu je funkcija vrata koja će služiti kao filter. Konačno, na trećem grafu prikazan je rezultat konvolucije ove dvije funkcije. Može se primijetiti kako je rezultat zapravo izravnati signal. Razlog tomu možemo pronaći u tome što konvolucija u svaku točku donosi i informaciju iz njene lokalne okoline. Veličinu okoline određuje širina, a težinski utjecaj okoline vrijednost filtra.

U kontekstu dubokog učenja, a posebno u obradi slike i računalnom vidu zanimljivija nam je diskretna dvodimenzionalna konvolucija. Definirana je sljedećim izrazom:

$$I * K(i;j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.6}$$

Sa I je označen dvodimenzionalni ulaz u konvoluciju koji može biti slika sa svojim



Slika 3.5: ilustracija 2D konvolucije [23]

prostornim dimenzijama(širinom i visinom), a sa K filter odnosno jezgra (eng. *kernel*) koja je obično manjih dimenzija od ulaza. Izlaz konvolucije je mapa značajki, koja nastaje na način da se jezgra pomiče preko cijelog ulaza. Na svakoj poziciji svaki prekriveni ulaz množi odgovarajuću težinu jezgre, nakon čega se ti umnošci sumiraju i tako nastaje jedan izlaz u mapi značajki. Kako bi izračunali cijelu mapu značajki, potrebno je provesti račun za svaku moguću poziciju jezgre u ulazu. Na slici 3.4 prikazan je primjer 2D konvolucije, a posebno je istaknut izračun za četvrtu poziciju jezgre (jezgra se uobičajeno pomiče počevši od gornjeg lijevog kuta prema desno do kraja, nakon čega se pomiče ponovno otpočetka, ali spuštena za jedan redak). U dubokim modelima jezgra je uobičajeno kvadratna, dimenzija KxK gdje je K najčešće neparni broj(obično 3, 5 ili 7). Primijetimo kako ovakva konvolucija smanjuje dimenzionalnost

ulaznog podatka. Postoji varijanta konvolucije koja zadržava dimenzije ulaza, a to se postiže na način da se oko ulaza nadopuni neka konstanta(obično nula) nakon čega se provodi isti postupak, tada govorimo o konvoluciji s nadopunjavanjem (eng. *padding*). U primjeru sa slike pomak jezgre jednak je jedan, ali on može biti proizvoljan. Sa većim pomakom S (eng. *stride*) dimenzija izlaza se smanjuje i to S puta. Konačno dimenzije izlaza možemo računati prema sljedećoj formuli:

$$O = \frac{D - K + 2P}{S} + 1 \tag{3.7}$$

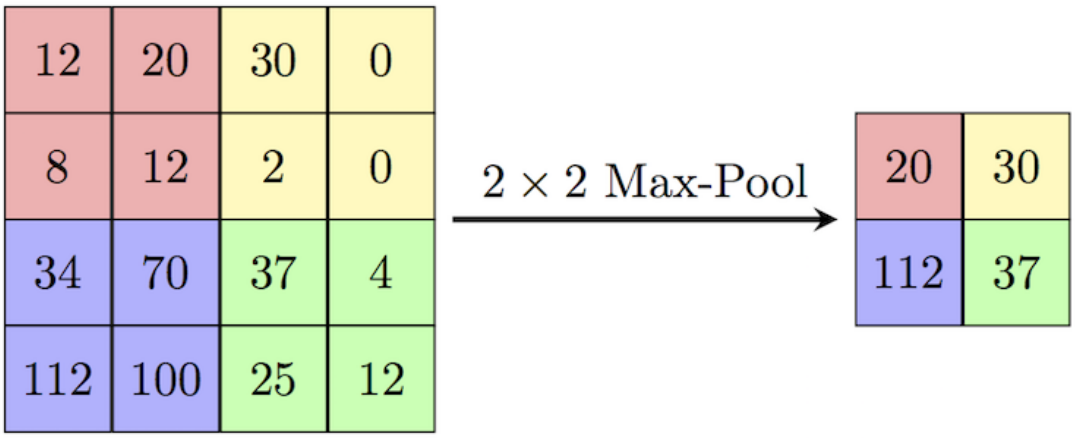
gdje je O dimenzija izlaza (širina ili visina), a D odgovarajuća dimenzija ulaza.

Slike koje obrađujemo najčešće nisu crno-bijele slike sa samo jednim kanalom, nego **RGB** slike koje imaju tri kanala: crveni (eng. ***r**ed*), zeleni (eng. ***g**reen*) i plavi (eng. ***b**lue*). To znači da je svaki piksel predstavljen sa uređenom trojkom koja označava vrijednost svakoga od kanala. Slike sa više kanala možemo predstaviti 3D tenzorima dimenzija HxWxC (visina, širina i broj kanala ili dubina). Također, uobičajeno je da unutar jednog konvolucijskog sloja radimo konvoluciju sa više različitih jezgara kojima stvaramo više različitih mapi značajki. Takve mape se tada slažu po dubini čime nastaje 3D tenzor. Treća dimenzija se često naziva i semantičkom, jer je nastala konvolucijom sa različitim jezgrama od kojih svaka poželjno ima odziv na neku drugu semantički različitu značajku. Ako je ulaz u konvoluciju 3D tenzor, onda je i jezgra trodimenzionalna KxKxD gdje D odgovara dubini ulaznog tenzora, a K veličini jezgre. Jezgra se i dalje pomiče po prostornim dimenzijama (visini i širini), a množenje i zbrajanje se obavlja analogno čime nastaje jedna mapa značajki.

Ukupan broj parametara konvolucijskog sloja možemo izračunati sljedećim izrazom $F \cdot (K \cdot K \cdot D + 1)$, gdje je F broj konvolucijskih jezgri, K dimenzija jezgre, D dubina ulaznog tenzora, a jedinica potječe od pomaka koji se dodaje za svaku mapu značajki posebno. Konvolucija u odnosu na potpuno povezani sloj ima puno manje parametara i modelira samo lokalne interakcije. To znači da jedan "piksel" izlazne mape značajki ovisi samo o njegovom lokalnom susjedstvu u ulaznoj mapi značajki. Još jedno svojstvo konvolucije je ekvivarijantnost na pomak, što znači da ako se ulaz pomakne jednako se pomiče i izlaz. Ako promatramo jednu mapu značajki možemo vidjeti kako nam ona daje informaciju o tome gdje se točno u ulazu pojavljuje detektirana značajka. Iako je to važna informacija, često je dovoljno znati pojavljuje li se neka značajka ili ne. To je glavna motivacija za uvođenje nove podvrste konvolucijskih slojeva koji se zovu slojevi sažimanja.

Slojevi sažimanja obavljaju poduzorkovanje primijenom jedne od mogućih funkcija na obično disjunktne regije ulazne mape značajki. Sažimanje možemo vizualizirati

na način da se dvodimenzionalno okno pomiče po ulaznoj mapi značajki i iz svih obuhvaćenih značajki se računa jedna izlazna značajka. Pomak okna je najčešće takav da nema preklapanja i za svaku mapu značajki posebno. Funkcija koja se primjenjuje računa neki statistički pokazatelj ulaznih značajki. Najviše se primjenjuje funkcija maksimalne vrijednosti, a manje se koriste funkcije srednje vrijednosti, L2 norme i dr. Na slici 3.6 prikazano je sažimanje maksimalnom vrijednosti i to sa pomakom i oknom veličine 2.



Slika 3.6: primjer sažimanja maksimalnom vrijednosti [23]

Slojevi sažimanja značajno smanjuju prostorne dimenzije (ne i semantičke) skrivenih reprezentacija, a samim time i računalni trošak. Izlazne mape značajki konvolucijskog sloja sadrže informaciju o lokaciji značajki u slici, a slojevi sažimanja daju modelu svojstvo invarijatnosti na lokalne pomake tih značajki. Napomenimo još kako se nerijetko ovakvi slojevi zamjenjuju sa pravim konvolucijskim slojevima sa pomakom jednakim veličini jezgre.

3.3.2. Aktivacijske funkcije

Na izlazu konvolucijskih i potpuno povezanih slojeva primjenjuje se aktivacijska funkcija. Njena zadaća je unošenje nelinearnosti u model, a bez nje model bi mogao naučiti samo linearne granice među podacima. Pokažimo to na sljedećem primjeru mreže s dva potpuno povezana sloja te matricama težina W_1, W_2 i pomacima $\boldsymbol{b_1}, \boldsymbol{b_2}$. Transformaciju koja obavlja takva mreža možemo napisati ovako:

$$\boldsymbol{h} = W_1\boldsymbol{x} + \boldsymbol{b_1} \tag{3.8}$$

$$\boldsymbol{y} = W_2\boldsymbol{h} + \boldsymbol{b_2} \tag{3.9}$$

Ako sada u drugu jednadžbu uvrstimo prvu dobijemo:

$$\boldsymbol{y} = W_2(W_1\boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2} \tag{3.10}$$

$$\boldsymbol{y} = W_2W_1\boldsymbol{x} + W_2\boldsymbol{b_1} + \boldsymbol{b_2} \tag{3.11}$$

Ako napravimo sljedeću zamjenu $W' = W_2W_1$ i $\boldsymbol{b'} = W_2\boldsymbol{b_1} + \boldsymbol{b_2}$ dobijemo novu mrežu koja obavlja jednaku funkciju, ali sa samo jednim slojem odnosno jednom linearnom transformacijom:

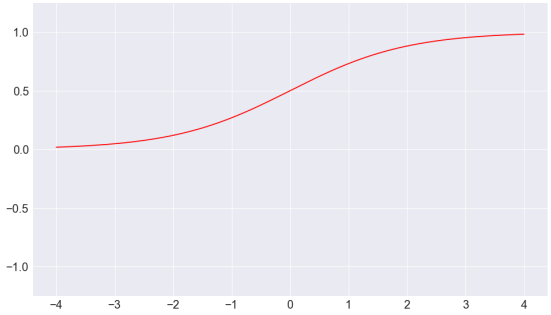
$$\boldsymbol{y} = W'\boldsymbol{x} + \boldsymbol{b'} \tag{3.12}$$

Nizanje slojeva, dakle, bez aktivacijske funkcije na izlazu svakoga sloja nema smisla. U dubokim modelima najčešće se koriste sljedeće aktivacijske funkcije: sigmoida, tangens hiperbolni, zglobnica i njen generalizirani oblik.

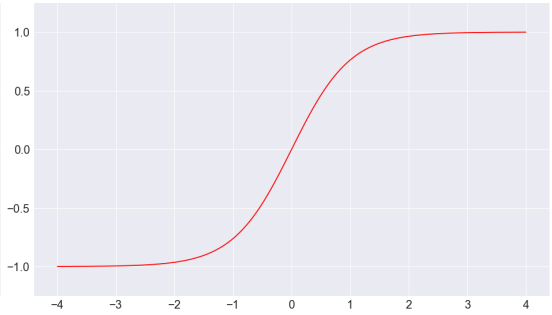
Graf sigmoidalne aktivacijske funkcije prikazan je na slici 3.7, a definirana je sljedećim izrazom:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.13}$$

Njena domena je cijeli skup realnih brojeva $\mathbb{R}$, a kodomena interval $(0, 1)$. Ona dakle



Slika 3.7: sigmoida



Slika 3.8: tangens hiperbolni

uzima realni broj i "gnječi" ga na interval od 0 do 1. Kroz povijest se često koristila jer dobro imitira okidanje neurona od potpuno neaktivnog stanja (0), do potpuno aktivnog stanja (1). U posljednje vrijeme se rijetko koristi u dubokim unaprijednim mrežama, a nešto češće u modelima sa povratnim vezama i nekim generativnim modelima. Za to postoje dva razloga: problem nestajećih gradijenata i izlaz koji nije centriran oko nule. Problem nestajećih gradijenata(eng. *vanishing gradient problem*) javlja se kada se izlaz sigmoide nalazi u stanju potpunog zasićenja (0 ili 1) jer je u tome području gradijent jednak ili jako blizu nule. Kako lokalni gradijent množi gradijent po izlazima koji se unatražno propagira kroz ostatak mreže, to onemogućuje učenje dijela mreže prije zasićenog neurona. Neuron u zasićenju tako guši signal gradijenta i ne propušta

ga prilikom unatražnog prolaza. Također, ako se koristi sigmoida kao aktivacijska funkcija treba biti oprezan prilikom inicijalizacije težina. Ako su težine po magnitudi velike to može dovesti neuron u zasićenje što će odmah na početku postupka sabotirati učenje. Izlaz sigmoide nije centriran oko nule, što znači da će neuroni koji slijede uvijek primati pozitivan ulaz, a to može utjecati na dinamiku učenja. To svakako je problem, ali puno manji od problema nestajućeg gradijenta.

Nad izlazima posljednjeg sloja dubokog modela, primjenjuje se općenitiji oblik sigmoide poznat kao softmax. Funkcija softmax vektor realnih vrijednosti mapira na interval $(0, 1)$ i to na način da je suma svih komponenti vektora jednaka 1. Izlaz softmaxa na neki način "glumi" vjerojatnosni izlaz modela. Formula za izračun jedne komponente je sljedeća:

$$f(x) = softmax(x_j) = \frac{\exp x_j}{\sum_i \exp x_i} \tag{3.14}$$

Problem necentriranosti izlaza rješava aktivacijska funkcija tangens hiperbolni, ali još uvijek zadržava problem nestajućeg gradijenta. To je zapravo skalirana sigmoida na način da je kodomena sada interval $(-1, 1)$. Njen graf prikazan je na slici 3.8, a računamo je prema formuli:

$$f(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.15}$$

Kao i sigmoida zbog nepropuštanja gradijenta u stanju zasićenja rijetko se koristi kod dubokih unaprijednih modela, ali se njena primjena može naći kod modela sa povratnim vezama.

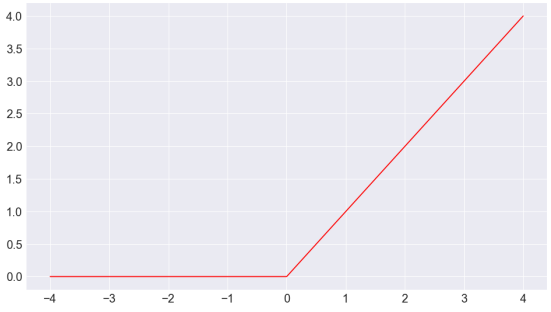
U dubokim unaprijednim modelima najčešće se koristi zglobnica (eng. *rectified linear unit, ReLU*) kao aktivacijska funkcija. Zglobnica je za razliku od prethdno razmatranih aktivacijskih funkcija jednostavna i jeftina operacija za izračunati, ona sve aktivacije koje su manje od nule postavi na nulu. Graf je prikazan na slici 3.9, a formulu možemo zapisati na sljedeći način:

$$f(x) = ReLU(x) = max(0, x) \tag{3.16}$$

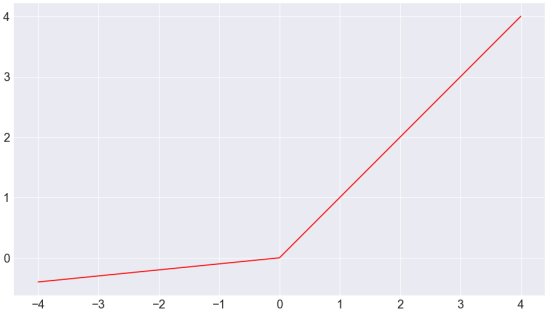
Osim što je računski jeftinija operacija, pokazalo se da značajno ubrzava učenje i konvergenciju prilikom primjene stohastičkog gradijentnog spusta. Vjeruje se da je to zbog svoje linearne prirode i izostanka zasićenja. Gradijent u području $x > 0$ iznosi 1, a inače 0. Nažalost prilikom korištenja neurona sa zglobnicom, primijetio je se problem umirućih neurona. To je slučaj u kojem za bilo koji ulaz doveden u neuron, izlaz je jednak nuli. Možemo zamisliti slučaj u kojem je naučen veliki negativni pomak,

kojeg različiti ulazi ne mogu "pobijediti". Problem je što takav neuron uopće ne propušta gradijent i zbog toga nema načina da se oporavi. Taj problem donekle rješava sestrinska aktivacijska funkcija pod imenom LeakyReLU. Ona naime nema prag kojim negativne vrijednosti postavlja na nulu, nego ih množi sa nekom malom konstantom (npr. 0.01). Graf te funkcije prikazan je na slici 3.10, a formulu ako paramater α ograničimo na $\alpha \in (0, 1)$ možemo pisati kao:

$$f(x) = LeakyReLU(x) = max(\alpha x, x) \tag{3.17}$$



Slika 3.9: ReLU



Slika 3.10: LeakyReLU

3.3.3. Normalizacija nad grupom

Učenje dubokih neuronskih mreža komplicira činjenica da se distribucija ulaza svakoga sloja tijekom učenja mijenja kako se mijenjaju parametri prethodnih slojeva[14]. To usporava postupak učenja, zahtijeva manju stopu učenja i pažljivu inicijalizaciju parametara, a učenje dubokih modela sa aktivacijskim funkcijama sa zasićenjem čini gotovo nemogućim. Normalizacija nad grupom[14] algoritam je koji rješava navedene probleme, a ideju učenja i arhitekturu dubokih modela mijenja minimalno. U originalnom članku autori su pokazali kako je primjenom ovoga algoritma za učenje složenog klasifikacijskog modela bilo potrebno 14 puta manje koraka, a također su uočili i poboljšanje u rezultatima. Normalizacija nad grupom ima i regularizacijski efekt na model, što smanjuje potrebu za primjenom drugih regularizacijskih metoda.

Uvjet za primijenu ovoga algoritma je učenje nad mini-grupama. Normalizacija nad grupom svaki ulaz u sloj (individualno na razini skalara) transformira u distribuciju sa srednjom vrijednosti 0 i varijancom 1 obzirom na vrijednosti toga ulaza u trenutnoj mini-grupi. Što je mini-grupa veća to će izračunate statistike biti preciznije, analogno kao i gradijent. Ponekad smo zbog veličine modela i memorijske ograničenosti hardwarea prisiljeni model trenirati sa manjim mini-grupama (npr. samo 2 primjera). U

tim slučajevima moguće je da će treniranje biti nestabilno zbog nemogućnosti izračuna preciznijih statistika.

U nastavku dan je pseudokod algoritma.

Algoritam 3: Normalizacija po grupama [14]

Ulaz: Vrijednosti neke aktivacije x u mini grupi $G = \{x_1, \ldots, x_m\}$

Izlaz: Normalizirane aktivacije $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$; `// srednja vrijednost mini-grupe`

$\sigma_G^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_G)^2$; `// varijanca mini-grupe`

$\hat{x}_i \leftarrow \frac{x_i - \mu_G}{\sqrt{\sigma_G^2 + \epsilon}}$; `// normalizacija`

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$; `// skaliranje i pomak`

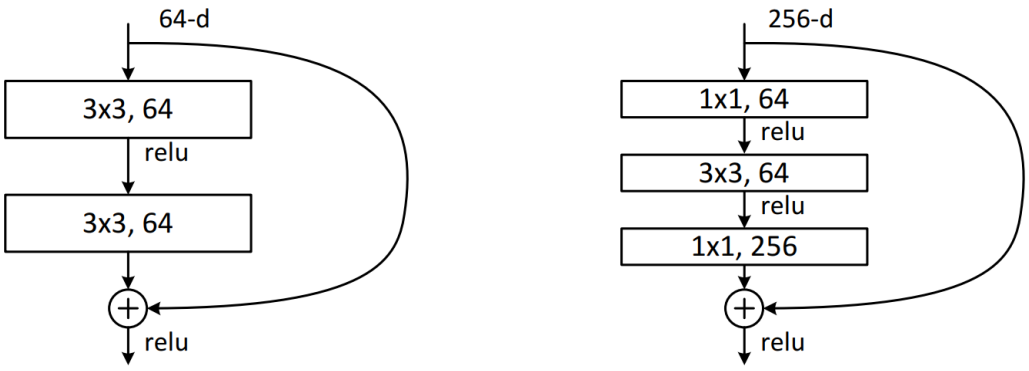
Nakon uobičajene normalizacije srednjom vrijednosti i varijancom uzorka, normalizirane vrijednosti se skaliraju množenjem parametrom γ i pomiču dodavanjem parametra β. To su parametri koji se uče kao i ostali parametri modela. Taj korak modelu omogućuje i učenje neke druge distribucije skrivenih značajki ako je to potrebno. Gradijente sloja koji obavlja ovu transformaciju moguće je izračunati pravilom ulančavanja, što omogućava uobičajeno učenje modela algoritmom propagacije pogreške unatrag. Ako se normalizacija primjenjuje na konvolucijski sloj, tada se statistike računaju za sve značajke jedne mape značajki zajedno.

U fazi eksploatacije modela nije moguće računati statistike jer na ulazu imamo samo jedan primjer, a ne mini-grupe sa m primjera. Zato je potrebno prilagoditi algoritam tako da se u fazi eksploatacije modela koriste statistike koje dobro opisuju značajke izračunate iz uzoraka stvarne distribucije podataka. Jedno rješenje je da se nakon učenja izračunaju srednje vrijednosti i varijance svih značajki uzimajući u obzir cijeli skup za treniranje, a drugo je da se u fazi učenja cijelo vrijeme računa pomična srednja vrijednost i varijanca za sve značajke (eng. *moving average*).

3.3.4. Arhitektura ResNet

Arhitektura ResNet[11] je konvolucijska arhitektura koja je omogućila učenje značajno dubljih modela nego što je do tada bio slučaj. Zasnovana je na gradivnom bloku koji može naučiti rezidualno mapiranje $F(x) := H(x) - x$, a omogućuje i bolji protok gradijenata u fazi učenja.

Shema gradivnog bloka prikazana je na slici 3.11: lijevo se nalazi gradivni blok koji se koristi kod plitkih rezidualnih mreža (do 30 slojeva), a desno modificirani blok



Slika 3.11: rezidualni blok [11]

za treniranje jako dubokih mreža. Zajednička ideja kod oba bloka je u preskočnim vezama koje ulaze prvog konvolucijskog sloja pribrajaju izlazima posljednjeg, nakon čega stoji aktivacijska funkcija. Preskočne veze ne unose nove parametre u model, ne povećavaju računalni trošak, a mreža se i dalje može učiti algoritmom propagacije pogreške unatrag. Model pomoću njih može lagano naučiti i funkciju identiteta, ako je to optimalno. Zbrajanje se radi po elementima, a ako ne odgovaraju dimenzije tenzora, tenzor koji dolazi preskočnom vezom se modificira nadopunjavanjem ili konvolucijom sa jezgrom veličine 1 i potrebnim pomakom. Kod jako dubokih modela zbog bojazni od prevelikog broja parametara, rezidualni gradivni blok je modificiran na način da se prvo provodi 1x1 konvolucija koja smanjuje broj mapa značajki, a onda 3x3 konvolucija. Na kraju bloka obavlja se još jedna 1x1 konvolucija koja vraća ulazni broj mapa značajki.

Autori su pokazali da obični izravnati modeli kojima su dodane preskočne veze postižu značajno bolje rezultate i brže konvergiraju. Danas se ResNet arhitektura često koristi kao ekstraktor značajki za različite složenije arhitekture koje rješavaju razne probleme iz područja računalnog vida. Poznate su mreže ResNet-50, ResNet-101 i ResNet-152 gdje brojka označava broj konvolucijskih slojeva, a dostupni su i modeli sa predtreniranim težinama na ImageNet skupu podataka.

3.4. Generativni suparnički modeli

Generativni suparnički modeli (eng. *Generative Adversarial Nets, GAN*) [9] pripadaju skupini generativnih modela koji ne definiraju funkciju gustoće vjerojatnosti $p_{model}(x)$ eksplicitno, nego pružaju način treniranja modela u kojima imaju samo indirektnu interakciju sa funkcijom gustoće vjerojatnosti (uobičajeno kroz uzorkovanje). S druge

strane postoje modeli koji eksplicitno definiraju distribuciju $p_{model}(x; \Theta)$, a tijekom treniranja cilj je pronaći parametre Θ koji maksimiziraju izglednost podataka iz skupa za treniranje. Model GAN dizajniran je kako bi izbjegao poznate nedostatke drugih generativnih modela, a ima sljedeće prednosti:

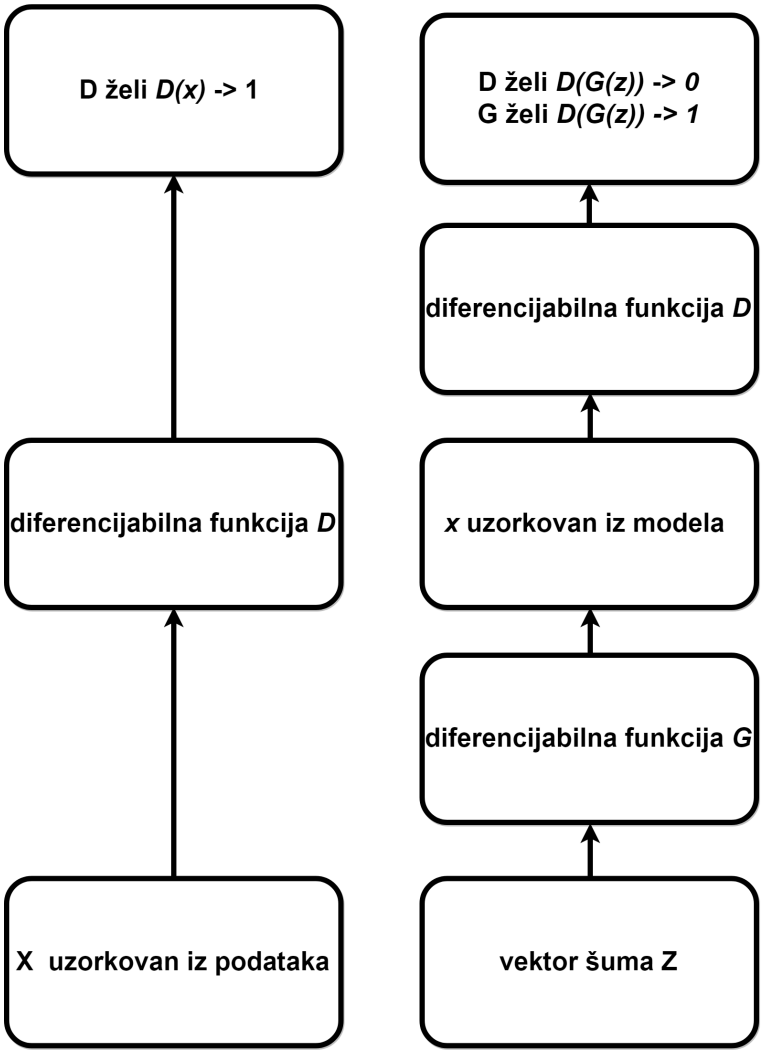
- može paralelno generirati uzorke,
- generatorska funkcija ima jako slabe restrikcije,
- nema potrebu za Markovljevim lancima i
- primjećeno je kako stvara kvalitetnije i realnije uzorke od drugih generativnih modela [8].

Pored navedenih prednosti primijećen je problem nestabilnog treniranja generativnih suparničkih modela. Izbjegavanje toga slučaja najveći je izazov prilikom razvoja modela koji pripadaju ovoj skupini.

Osnovna ideja modela GAN je suprotstaviti dva igrača, generator i diskriminator, u međusobnoj igri. Zadatak generatora je stvarati uzorke koji bi trebali dolaziti iz iste distribucije kao uzorci iz skupa za treniranje. Zadatak diskriminatora je razlikovati generirane uzorke od stvarnih uzoraka iz skupa za treniranje. Natjecanje prisiljava oba igrača da unaprjeđuju svoje metode sve do trenutka kada se generirani uzorci ne mogu razlikovati od stvarnih.

Generator je diferencijabilna funkcija $G(\boldsymbol{z}; \boldsymbol{\Theta}_G)$ koja na ulazu prima vektor šuma $\boldsymbol{z}$ uzorkovan iz neke jednostavne fiksirane distribucije $p_{\boldsymbol{z}}(\boldsymbol{z})$, a izlaz je vektor $\boldsymbol{x}$ uzorkovan iz implicitno definirane distribucije p_g. Generator je obično višeslojna neuronska mreža parametrizirana s $\boldsymbol{\Theta}_G$ učena algoritmom propagacije pogreške unatrag kako bi zavarao diskriminator. Konačan cilj učenja je maksimalno približiti distribuciju p_g distribuciji ulaznih podataka p_{data}. Napomenimo još kako slučajni vektor šuma ne mora nužno biti ulaz u prvi sloj neuronske mreže nego u bilo koji drugi. Diskriminator je također diferencijabilna funkcija $D(\boldsymbol{x}; \boldsymbol{\Theta}_D)$, obično višeslojna neuronska mreža, koja na ulazu prima uzorak $\boldsymbol{x}$ iz distribucije podataka p_{data} ili modela p_g, a na izlazu daje 1 ili 0. Njegov cilj je naučiti razlikovati ulazne podatke i za sve stvarne podatke na izlaz postaviti 1, a za podatke uzorkovane modelom generatora 0. Na slici 3.12 prikazan je dijagram koji opisuje postupak učenja generativnih suparničkih modela.

Oba igrača minimiziraju funkciju gubitka koja je parametrizirana i s parametrima drugog igrača, a na koje nemaju direktan utjecaj. Zato je ovaj problem prirodnije opisati kao igru, a ne kao optimizacijski problem. Rješenje igre je Nashov ekvilibrij, odnosno uređeni par $(\boldsymbol{\Theta}_G, \boldsymbol{\Theta}_D)$ koji je lokalni minimum funkcije gubitka generatora J^G po $\boldsymbol{\Theta}_G$ i lokalni minimum funkcije gubitka diskriminatora J^D po $\boldsymbol{\Theta}_D$. Ova igra



Slika 3.12: dijagram modela GAN(prema [8])

može se promatrati kao minimax igra sa funkcijom vrijednosti:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z}[\log(1 - D(G(z)))] \quad (3.18)$$

Učenje generativnih suparničkih modela obavlja se sa dva istovremena postupka gradijentnog spusta. U svakome koraku uzorkuju se dvije mini-grupe: jedna iz skupa za učenje, a druga iz jednostavne latentne distribucije p_z. Iz druge mini-grupe generator će stvoriti mini-grupu sintetičkih primjera jednakih dimenzija kao mini-grupa iz skupa za učenje. Tada se dva koraka gradijentnog spusta obavljaju istovremeno: jedan koji minimizira gubitak diskriminatora J^D obzirom na parametre Θ_D i drugi koji minimizira gubitak generatora J^G obzirom na parametre Θ_G. Različite su mogućnosti prilikom izbora funkcije gubitka. U najjednostavnijoj varijanti koja slijedi iz teorijske

postavke problema kao igre slijedi da je:

$$J^D(\boldsymbol{\Theta}_G, \boldsymbol{\Theta}_D) = -\frac{1}{2}E_{x\sim p_{data}(x)}[\log D(x)] - \frac{1}{2}E_{z\sim p_z}[\log(1 - D(G(z))) \tag{3.19}$$

$$J^G(\boldsymbol{\Theta}_G, \boldsymbol{\Theta}_D) = -J^D(\boldsymbol{\Theta}_G, \boldsymbol{\Theta}_D) \tag{3.20}$$

što zapravo odgovara gubitku unakrsne entropije. Problem s ovako definiranim funkcijama gubitka je taj što su pri početku učenja generirani primjeri jako loši i diskriminatoru nije teško razlikovati generirane od primjera iz skupa za učenje. Zato će gradijent izraza $\log(1 - D(G(z)))$ biti u zasićenju i neće donositi dovoljno informacije za učenje do generatora. Jednostavno rješenje je da se generator trenira na način da maksimizira vjerojatnost da će diskriminator njegove generirane primjere označiti kao realne, tada je funkcija gubitka generatora:

$$J^G(\boldsymbol{\Theta}_G, \boldsymbol{\Theta}_D) = -E_{z\sim p_z}[\log(D(G(z)))] \tag{3.21}$$

što donosi mnogo snažnije gradijente pri početku učenja.

3.4.1. Primjene suparničkog učenja

Ideja suparničkog učenja predstavljena kroz generativne suparničke modele potaknula je istraživanje generativnih modela i iskorištavanje mogućnosti nenadziranog ili polunadziranog učenja na različitim problemima. U nastavku dan je kratki opis nekoliko znanstvenih radova koji su primijenili suparničko učenje na zadatke učenja skrivenih reprezentacija i generiranja realnih primjera, strojnog prevođenja slike, povećanja rezolucije slike i semantičke segmentacije.

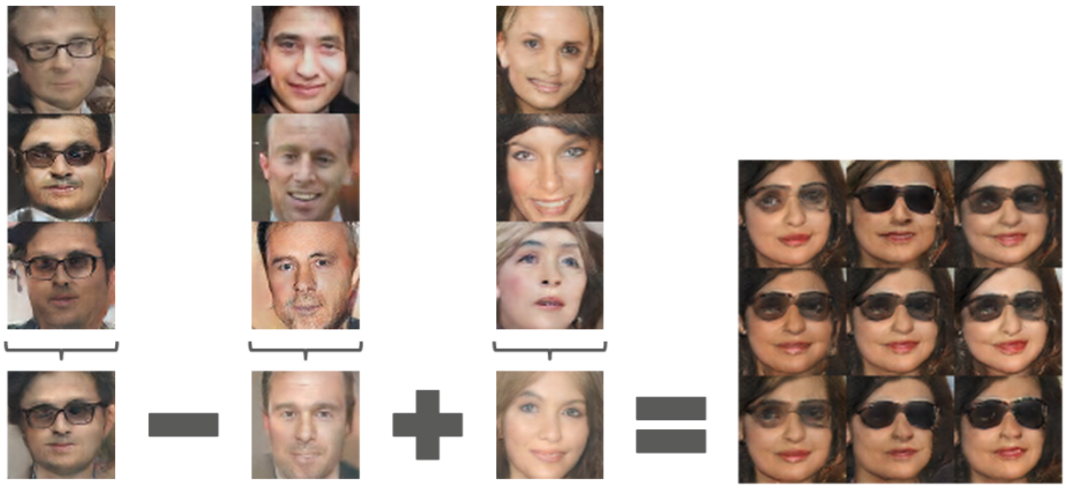
Nenadzirano učenje skrivenih reprezentacija poznati je problem računalnog vida. Cilj je iz neoznačenih slika, koje su lako dostupne, naučiti reprezentaciju slike koja je manje dimenzionalnosti i pogodna za primjenu drugih algoritama koji rješavaju razne zadatke, a često imaju problema sa visokom dimenzionalnosti podataka. Duboki konvolucijski generativni suparnički modeli(eng. *deep convolutional generative adversarial nets, DCGAN*)[22] pokazali su da mogu generirati realne primjere i uz to naučiti reprezentaciju podataka koja je pogodna za dalju primjenu klasičnih klasifikacijskih metoda strojnog učenja. Autori su također iznijeli niz uputa koje stabiliziraju postupak učenja konvolucijskih generativnih suparničkih modela:

- slojeve sažimanja zamijeniti sa konvolucijama s pomakom,
- koristiti normalizaciju po grupama u generatoru i diskriminatoru,
- izbaciti potpuno povezane slojeve u dubljim arhitekturama,

– u generatoru koristiti aktivacijsku funkciju ReLU, a na posljednjem sloju tangens hiperbolni te

– koristiti LeakyReLU u svim slojevima diskriminatora.

Osim značajnog poboljšanja kvalitete skrivenih reprezentacija naučenih modelom DCGAN u odnosu na druge generativne metode, pokazani su i drugi zanimljivi rezultati. Pokazano je kako generirane semantički slične slike također imaju i slične reprezentacije u prostoru latentnih značajki. Štoviše, prijelazi iz jedne reprezentacije u drugu su glatki bez naglih promjena u prostoru generirane slike. U jednom eksperimentu autori su pronašli skrivene vektore koji predstavljaju muškarca sa naočalama, od njega oduzeli vektor muškarca i dodali na vektor žene bez naočala. Rezultat je prikazivao sliku žene sa naočalama. Eksperiment je prikazan na slici 3.13.



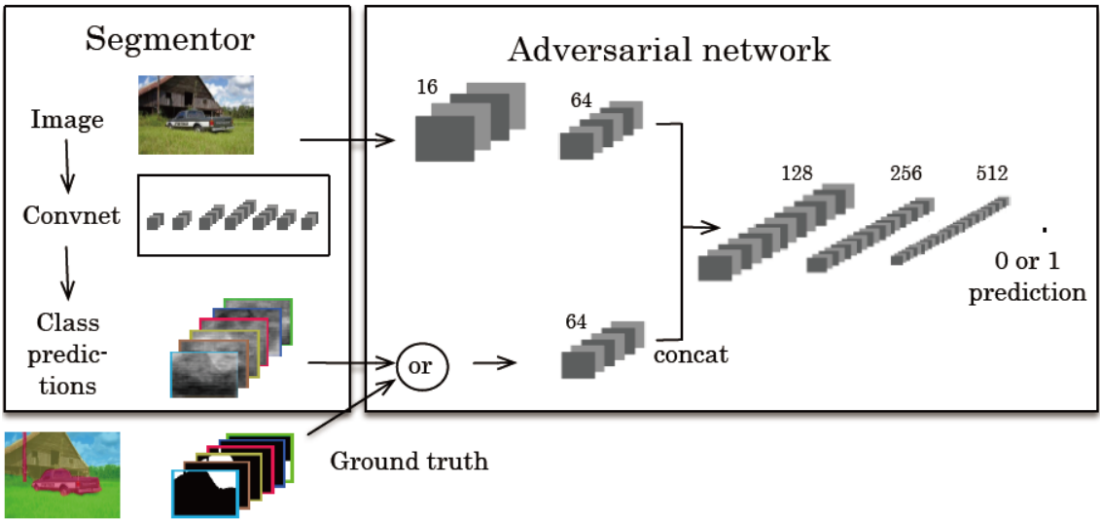
Slika 3.13: eksperiment sa aritmetikom vektora u latentnom prostoru [22]

Uvjetni generativni suparnički modeli primijenjeni su na problem strojnog prevođenja slike [15]. Strojno prevođenje slike je jednostavno mapiranje scene iz jedne reprezentacije u drugu. Autori su model trenirali na različitim problemima prevođenja: iz segmentacijskih mapa u sliku, iz grayscale reprezentacije u RGB, iz skice rubova objekta u objekt sa teksturom, iz scene uslikane noću u scenu uslikanu danju i dr. U ovome slučaju generator na ulazu ne prima slučajni vektor šuma, nego sliku u izvornoj reprezentaciji(time je "uvjetovan"), a na izlazu daje sliku u ciljanoj reprezentaciji. Diskriminator na ulazu prima obje slike i pokušava razlikovati stvarne parove od generiranih. Za ovakvo učenje potrebno je imati skup podataka koji sadrži obje reprezentacije istih scena, primjerice i skicu i obojanu sliku sa teksturom. Ovaj pristup problemu strojnog prevođenja slike izgleda obećavajući s velikim potencijalom za buduća istraživanja.

Slika 3.14: strojno prevođenje iz slike mape u satelitsku snimku (ulaz, izlaz, stvarni izlaz)[15]

Jako izazovan problem u računalnom vidu je stvaranje slika visoke rezolucija(HR) iz slika niske rezolucije(LR), poznat kao super rezolucija. Model SRGAN [19] temeljen na generativnim suparničkim modelima postigao je izniman uspjeh u rješavanju ovoga problema. Autori su trenirali generativni suparnički model na način da je generator na ulazu primao slike niske rezolucije, a na izlazu generirao slike visoke rezolucije. Zadatak diskriminatora je razlikovati stvarne od generiranih slika visoke rezolucije. Autori su za treniranja ovoga modela koristili skup slika visoke rezolucije iz kojeg su metodama poduzorkovanja stvarali slike niske rezolucije te na taj način dobili parove (LR, HR). Srednja ocjena kvalitete slika nastalih ovom metodom bliže su ocjenama originalnih slika visoke rezolucije od bilo koje do tada poznate metode.

Jedan od najzahtjevnijih zadataka računalnog vida je semantička segmentacija. Cilj semantičke segmentacije je razumijeti sliku na razini piksela na način da se za svaki piksel odredi razred kojem on pripada. Za računalo to je najbliže čovjekovom razumijevanju slike. Prvu primijenu suparničkog učenja na problem semantičke segmentacije napravila je Pauline Luc sa suradnicima [20]. U njihovom sustavu generator je zapravo model za semantičku segmentaciju koji na ulazu prima sliku, a na izlazu daje semantičku mapu i to sa vjerojatnosnom distribucijom po klasama za svaki piksel. Diskriminator na ulazu prima sliku i korespodentnu semantičku mapu, izlaz iz generatora ili oznake iz skupa za treniranje, koje na izlazu treba razlikovati. Skica ovoga sustava prikazana je na slici 3.15. Gubitak generatora sastoji se iz dvije komponente. Prva komponenta je gubitak unakrsne entropije između predviđene i označene segmentacijske mape, a druga komponenta je gubitak binarne unakrsne entropije na izlazu diskriminatora. Ideja je da diskriminator prenese dodatnu informaciju o kvaliteti predviđene semantičke mape i tako tjera generator da stvara semantičke mape koje su bliže oznakama iz skupa za treniranje. Pokazalo se da ta informacija zaista pomaže prilikom treniranja generatora i podiže njegove mjere uspješnosti za par postotnih poena.



Slika 3.15: skica sustava za semantičku segmentaciju korištenjem suparničkog učenja[20]

4. Procjenjivanje nesigurnosti diskriminativnih modela

Poželjno svojstvo svih prediktivnih modela je to da na izlazu uz same predikcije daju i informaciju o pouzdanosti. U po život kritičnim sustavima kvaliteta te informacije je ključna i može biti glavni razlog za izostanak adaptacije modela strojnog učenja. Primjerice u medicini, ne bismo htjeli da model cijelo vrijeme daje dijagnoze sa visokom pouzdanosti, čak i onda kada je potrebna ljudska intervencija. Na izlazima dubokih modela uobičajeno se koristi aktivacijska funkcija softmax koja normalizira izlaze i pretvara ih u vjerojatnosnu distribuciju. Međutim, taj izlaz često je slabo koreliran sa stvarnom pouzdanosti predikcija. Poznato je da modeli čak i u slučajevima kada daju netočne predikcije, još uvijek imaju visoku pouzdanost. Možemo zamisliti da generalno duboki modeli imaju neopravdani višak samopouzdanja.

Procjenjivanje pouzdanosti odnosno nesigurnosti poznat je problem u području dubokog učenja i vrlo aktualna istraživačka tema. U nastavku dan je pregled nekoliko znanstvenih radova, a posebno se razmatra procjenjivanje nesigurnosti pomoću suparničkog učenja što je ujedno i tema ovoga rada.

4.1. Pregled područja

Problematika softmaxa kao indikatora nesigurnosti i detekcija netočnih i primjera izvan distribucije razmatrana je u [12]. Autori smatraju da problem leži u prirodi funkcije softmax koja je kontinuirani aproksimator indikatorske funkcije. Vjerojatnosti se računaju koristeći brzorastuću eksponencijalnu funkciju i male promjene u ulazima (tkz. logiti) mogu značajno utjecati na vjerojatnosni izlaz. Pokazali su kako višeslojni perceptron naučen kao klasifikator na MNIST skupu podataka za ulaze uzorkovane iz gaussovog šuma u prosjeku ima pouzdanost od 91%. Isti klasifikator za netočno klasificirane primjere iz MNIST skupa podataka u prosjeku daje predikcije s pouzdanošću od 86%. Autori su također na ulaz postavili primjere koji dolaze iz nekih drugih sku-

pova podataka, prosječna pouzdanost u različitim kombinacijama je bila iznad 75%. To je jasan pokazatelj kako se izlaz iz softmaxa ne može uzeti kao izlaz pouzdanosti ili detektor nesigurnosti. Ipak, razlika u pouzdanostima za primjere iz i van distribucije podataka skupa za učenje je postojala i okaraketirizana je kao dovoljna za detekciju.

Detekcija primjera van distribucije skupa za učenje zadatak je koreliran sa zadatkom određivanja pouzdanosti predikcija. Za očekivati je da će pouzdanost za ulaze koji su različti od onih koje je model vidio u postupku učenja biti značajno niža. Ta razlika je dostatna za detekciju primjera van distribucije. Dovoljno je dakle naučiti model čije su pouzdanosti realne i inuitivno lako interpretabilne. Jedan od načina je dodati posebnu granu koja za klasifikacijske modele ima skalarni izlaz c aktiviran sigmoidalnom funkcijom koji predstavlja pouzdanost modela [6]. Model sa takvim izlazima možemo zapisati na sljedeći način:

$$p, c = f(x, \Theta) \quad p_i, c \in [0, 1], \sum_i p_i = 1. \tag{4.1}$$

Standardni softmax izlaz mreže se onda modificira na sljedeći način:

$$p'_i = c \cdot p_i + (1 - c) \cdot y_i \tag{4.2}$$

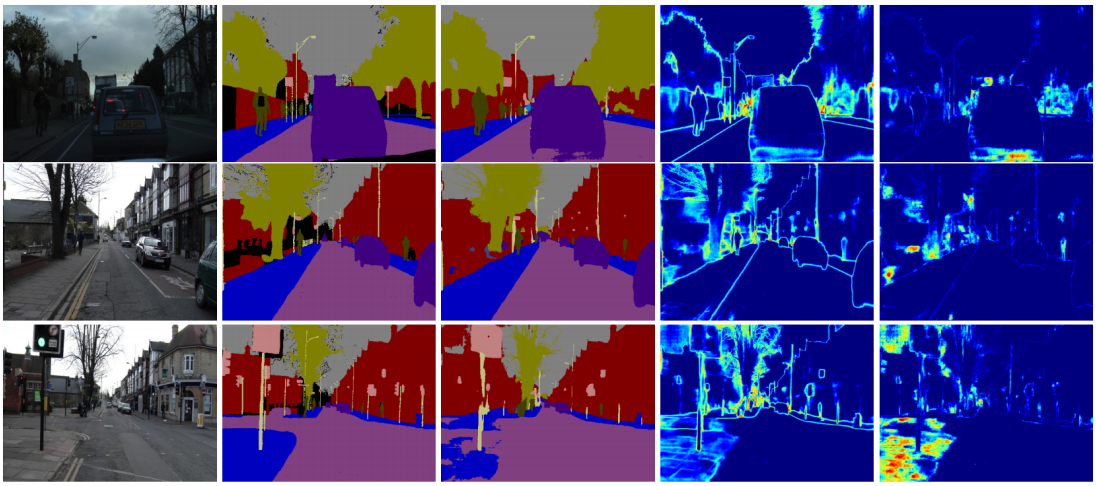
Ova ideja je inspirirana situacijom pisanja testa u kojem student ima pravo na korištenje pomoći/savjeta (eng. *hint*). Međutim, za svaku pomoć koju student iskoristi, također dobije i neku malu kaznu. Dobra strategija za studenta bi bila prvo odgovoriti na pitanja u čije odgovore je jako siguran, a za odgovore u koje nije siguran iskoristiti pomoć. Na kraju testa pouzdanost u njihove odgovore moguće je aproksimirati koristeći broj pomoći koje su zatražili na svakome pitanju. Ta ideja kroz posebnu granu koja estimira pouzdanost u svoje predikcija prenesena je i na duboke modele. Ako je model siguran u svoje predikcije izlaz c bi trebao biti blizu 1. Kada je model sve nesigurniji, pouzdanost c bi trebala težiti u 0, njegov izlaz p' bit će sve bliži *one-hot* oznaci za taj primjer i gubitak će biti manji. U takvoj situaciji model bi mogao minimizirati gubitak na način da c uvijek drži blizu 0 i uvijek se služi sa danom pomoći. Zato se uobičajenom gubitku unakrsne entropije $\mathcal{L}_{ce} = -\sum_i y_i \log p_i$ dodaje izraz $\mathcal{L}_c = -\log c$, koji će tjerati model da bude sigurniji u svoje predikcije i manje koristi pomoć. Konačan gubitak dobije se težinskim zbrajanjem ove dvije komponente:

$$\mathcal{L} = \mathcal{L}_{ce} + \lambda \cdot \mathcal{L}_c \tag{4.3}$$

Hiperparametar λ regulira cijenu pomoći i tijekom treniranja se mijenja kako bi se gubitak $\mathcal{L}_c$ zadržao blizu parametra koji opisuje dostupni budžet za korištenje pomoći β

i na taj način zadržao realne i interpretabilne izlaze pouzdanosti c. Autori su pokazali kako se primjeri van distribucije mogu lagano detektirati samo postavljanjem praga δ: ako je $c > \delta$ onda je primjer iz distribucije ulaznog skupa podataka, a u suprotnom primjer potiče iz neke druge distribucije. Također, na primjerima gdje je više šuma i slika nije toliko oštra, pouzdanost modela se očekivano smanjuje. Pokazalo se da je ideja smislena i postiže intuitivne rezultate. Autori u budućem radu planiraju primjenu na složenije zadatke kao što je semantička segmentacija i isprobavanje složenijih koncepata "dobivanja pomoći".

Bayesovo duboko učenje nudi još jedno moguće rješenje[17] za problem estimacije pouzdanosti modela. Oni govore o dvije nepouzdanosti/nesigurnosti koje je moguće modelirati: aleatorička i epistemička. Aleatorička nesigurnost je ona koja postoji zbog prirode podataka i šuma u njima. To je šum nastao primjerice zbog nepreciznih senzorskih očitanja ili gibanja kamere. Ona se ne može objasniti dodatnim podacima. Epistemička nesigurnost je ona koja postoji zbog neispravnih parametara modela. Ona je objašnjiva ako je skup podataka dovoljno velik i još se naziva i nesigurnost modela. Na slici 4.1 je vidljivo kako se aleatorička nesigurnost očituje na rubovima objekata i objektima koji se nalaze u daljini. Epistemička nesigurnost može se primijetiti na zathjevnim pikselima gdje model nema dovoljno znanja kako bi napravio ispravnu segmentaciju. Epistemička nesigurnost se u Bayesovim neuronskim mrežama mode-



Slika 4.1: na slikama je redom prikazano: ulazna slika, označena semantička mapa, predviđena semantička mapa, aleatorička nesigurnost i epistemička nesigrunost [17]

.

lira na način da se postavi neka apriorna distribucija nad težinama mreže, primjerice Gaussova: $\boldsymbol{W} \sim \mathcal{N}(0, 1)$. Cilj je onda uz dani skup podataka $\boldsymbol{X}, \boldsymbol{Y}$ izračunati aposteriornu vjerojatnost težina $p(\boldsymbol{W}; \boldsymbol{X}, \boldsymbol{Y})$. To je nemoguće izračunati i zato se često ta

distribucija aproksimira sa nekom parametriziranom distribucijom i tada se optimiziraju zapravo parametri te nove distribucije, umjesto originalnih parametara neuronske mreže. Aleatorička nesigurnost se modelira na način da se aproksimira parametar šuma u podacima σ i on za razliku od klasičnih pristupa nije fiksiran i različit je za svaki podatak. Autori su pokazali kako modeliranje ovih nesigurnosti podiže cjelokupne performanse modela. Također zaključuju kako je aleatorička nesigurnost posebno važna u situacijama sa velikim skupovima podataka kada je epistemička nesigurnost objašnjena podacima i u primjenama u realnom vremenu. Epistemička nesigurnost je važna u aplikacijama za koje je ključna sigurnost jer omogućuje prepoznavanje podataka i situacija različitih od onih u skupu za učenje.

4.2. Procjena nesigurnosti suparničkim učenjem

Ideja procjene nesigurnosti suparničkim učenjem temeljena je na sustavu suparničkog učenja za semantičku segmentaciju [13]. Razlika u odnosu na prethodno spomenuti rad Pauline Luc[20] na istu temu je u tome što diskriminator sada pokušava razlikovati je li semantička mapa stvorena od strane čovjeka ili modela na razini piksela. Generator je model semantičke segmentacije (u radu konkretno temeljen na modelu DeepLab-V2 [4]) koji na ulazu prima RGB sliku dimenzija HxWx3, a na izlazu svaki piksel klasificira u jednu od klasa i time stvara vjerojatnosnu semantičku mapu dimenzija HxWxC (H - visina slike, W - širina slike, C - broj klasa/razreda). Diskriminator na ulazu prima semantičku mapu dimenzija HxWxC, a na izlazu generira vjerojatnosnu mapu dimenzija HxWx1 u kojoj je za svaki piksel zapisana vjerojatnost da on dolazi iz stvarne semantičke mape označene od strane ljudskog anotatora. Cilj diskriminatora je razlikovati stvarne od umjetno generiranih semantičkih mapa. Ulaz koji prima od strane generatora je izlaz iz softmaxa pa to već je vjerojatnosna distribucija po svim klasama za svaki piksel. Kada je ulaz anotirana semantička mapa ona je pretvorena u distribuciju na način da je za stvarnu klasu vjerojatnost jednaka 1, a za sve ostale klase 0 i tako za svaki piksel. To je poznato i kao jednojedinični kod (eng. *one-hot encoding*). Generator i diskriminator su potpuno konvolucijski modeli tako da ih je moguće primijeniti na slike varijabilnih veličina. Jedna od prednosti ovakvog sustava je i ta što je moguće i polunadzirano učenje, a to je posebno važno kada imamo veliki skup slika od kojih je samo manji dio anotiran.

Gubitak generatora sastoji se iz tri komponente:

$$\mathcal{L}_G = \mathcal{L}_{ce} + \lambda_{adv}\mathcal{L}_{adv} + \lambda_{semi}\mathcal{L}_{semi} \tag{4.4}$$

Prva komponenta gubitka $\mathcal{L}_{ce}$ je unakrsna entropija koja mjeri razliku između anotiranih semantičkih mapa i izlaza iz funkcije softmax primijenjene nad logitima generatora. Računa se na sljedeći način:

$$\mathcal{L}_{ce} = -\sum_{h,w}\sum_{c\in C} \boldsymbol{Y}_n^{(h,w,c)} \log\left(S(\boldsymbol{X}_n)^{(h,w,c)}\right) \tag{4.5}$$

gdje $\boldsymbol{Y}_n^{(h,w,c)}$ predstavlja c-tu komponentu jednojediničnog koda za piksel na poziciji h, w u n-tom primjeru u skupu podataka, a $S(\boldsymbol{X}_n)^{(h,w,c)}$ c-tu komponentu softmax izlaza iz generatora za piksel na poziciji h, w.

Druga komponenta gubitka $\mathcal{L}_{adv}$ jest suparnički gubitak kojime zapravo generator pokušava zavarati diskriminator i maksimizirati vjerojatnost da diskriminator izlaze generatora ocijeni kao stvarne/realne. Definirana je kako slijedi:

$$\mathcal{L}_{adv} = -\sum_{h,w} \log(D(S(\boldsymbol{X}_n))^{(h,w)}) \tag{4.6}$$

gdje D označava izlaz diskriminatora, odnosno vjerojatnost da je piksel realan (potječe iz skupa anotiranih oznaka). Ova komponenta gubitka tjera generator da stvara semantičke mape koje su što sličnije anotiranim. Diskriminator bi mogao iz anotiranih oznaka naučiti oblike nekih objekata, primjerice da su automobilske gume obično okrugle ili da su rubovi tv-a ravne linije. Baš to mogao bi mu biti indikator da prepozna iz koje distribucije dolaze pikseli, a tu informaciju bi generator iskoristio kako bi popravio regije u kojima ga je diskriminator "prepoznao" i time stvorio što realniju semantičku mapu.

Treća komponenta gubitka $\mathcal{L}_{semi}$ je gubitak polunadziranog učenja i on se primjenjuje samo kod neoznačenih podataka. Računa se prema sljedećoj formuli:

$$\mathcal{L}_{semi} = -\sum_{h,w}\sum_{c\in C} I(\log(D(S(\boldsymbol{X}_n))^{(h,w)}) > T_{semi}) \log\left(S(\boldsymbol{X}_n)^{(h,w,c)}\right) \tag{4.7}$$

gdje je $I(x)$ indikatorska funkcija, a T_{semi} je prag kojim se kontrolira osjetljivost postupka učenja nad neoznačenim podacima. Ovaj gubitak se može promatrati kao maskirani gubitak unakrsne entropije. Ideja je da diskriminatorova vjerojatnosna mapa detektira regije u kojima su izlazi generatora dovoljno blizu ciljanim izlazima. Na taj način diskriminator kao da govori generatoru u kojim regijama je napravio kvalitetne predikcije i da baš tamo bude još sigurniji. U praksi autori preporučuju da se prag T_{semi} postavi na vrijednost između 0.1 i 0.3. Ovaj gubitak ne primjenjuje se pri početku postupka učenja jer su tada diskriminatorove predikcije još uvijek nekvalitetne i slabo razlikuju kvalitetne oznake od pogrešnih. Takav signal mogao bi samo unositi

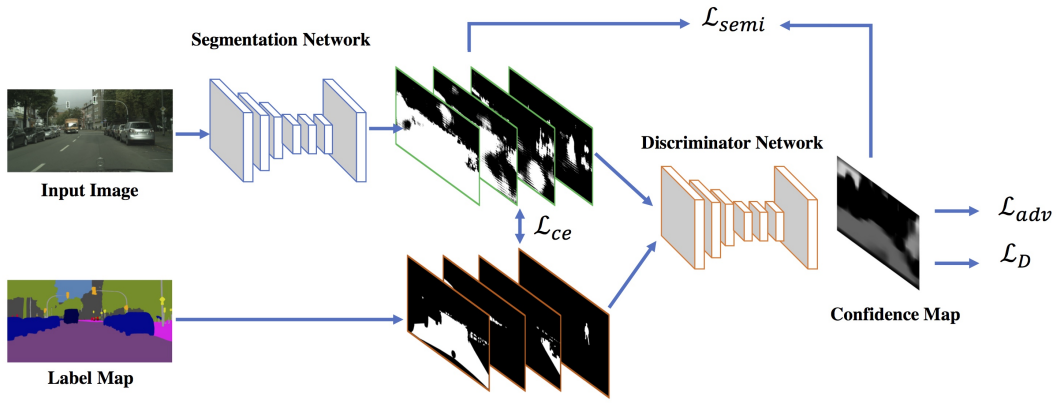
šum prilikom učenja generatora. Dakle, kvalitetniji diskriminator u kasnijim fazama učenja donosi kvalitetniji signal za učenje generatora.

Suparnički gubitak i gubitak polunadziranog učenja otežani su sa hiperparametrima λ_{adv} i λ_{semi}. Tipične vrijednosti su oko 0.01, a oni omogućuju segmentacijskom modelu da većinu signala za učenje prima iz označenih podataka, a preostale dvije komponente služe samo kao svojevrsna dopuna.

Gubitak diskriminatora je gubitak binarne unakrsne entropije sumiran preko svih piksela:

$$\mathcal{L}_D = -\sum_{h,w}(1-y_n)\log(1-D(S(\boldsymbol{X}_n))^{(h,w)}) + y_n \log(D((\boldsymbol{Y}_n))^{(h,w)}) \tag{4.8}$$

gdje je $\boldsymbol{Y}_n$ anotirana semantička mapa pretvorena u jednojedinični kod, a $y_n = 1$ ako je ulaz uzorkovan iz skupa anotiranih oznaka te $y_n = 0$ ako je ulaz uzorkovan iz segmentacijskog modela. Kod ovakvih sustava postoji bojazan da će diskriminator lako naučiti razlikovati oznake, jer anotirane oznake dolaze izrazito u jednojediničnom kodu. Rješenje za to jeste uvesti šum na način da se dio vjerojatnosti skine sa točno označene klase i raspodijeli na preostale. Autori u ovome radu nisu primijetili spomenuti problem i promijene u ponašanju diskriminatora.



Slika 4.2: skica sustava za suparničko učenje modela za semantičku segmentaciju[13]

Eksperimenti su pokazali kako dodana komponenta suparničkog gubitka podiže performanse modela za oko 2 postotna poena, a ako se doda i komponenta gubitka nenadziranog učenja to donosi dodatni 1 postotni poen.

Motivacija za ovaj diplomski rad jeste iskoristiti ovu shemu suparničkog učenja za procjenu nesigurnosti modela semantičke segmentacije. Problem primijene softmaxa kao izlaza pouzdanosti postoji i kod segmentacijskih modela. Česta je pojava da je pouzdanost za unutrašnjost nekoga segmenta jako visoka, a tek na tankim rubovima

nešto niža, čak i kada je taj segment krivo klasificiran. Diskriminator bi kvalitetne predikcije teško trebao razlikovati i za takve piksele izlazi bi trebali biti bliže 1 jer su slični stvarnim oznakama. One segmente u kojima su predikcije generatora šumovite i pogrešne diskriminator bi trebao prepoznati i njegovi izlazi bi trebali težiti 0 i označiti ih kao lažne. Takvo ponašanje diskriminatora odgovara željenom ponašanju modela sa procjenom pouzdanosti. Izlaz diskriminatora može se iskoristiti kao pouzdanost predikcija modela i detektor regija u kojima je generator netočan odnosno nesiguran.

5. Implementacija

U okviru ovoga rada implementiran je suparnički model za semantičku segmentaciju koji na izlazu diskriminatora daje procjenu nesigurnosti predikcija na razini piksela. Za implementaciju korišten je programski jezik Python verzije 3.5.2. uz podršku raznih biblioteka.

TensorFlow [1] je biblioteka otvorenog koda izvorno razvijena od strane Google Braina za brze i složene numeričke računske operacije. Izračun je u TensorFlowu opisan pomoću računskog grafa. To je usmjereni graf u kojem čvorovi predstavljaju matematičke operacije, a bridovima putuju podaci predstavljeni kao tenzori varijabilnih dimenzija. Korisnici graf zadaju preko programskog koda (sučelje postoji za jezike C++ ili Python), a sama biblioteka je napisana u programskom jeziku C++. Biblioteka pruža mogućnost izvođenja na različitim platformama uz male ili nikakve preinake koda, a posebno je zanimljiva mogućnost izvođenja na grafičkim karticama (eng. *graphics processing unit*, *GPU*). Trenutno su podržane samo grafičke kartice proizvođača Nvidia zbog njihove biblioteke CUDA koja omogućuje efikasno paralelno izvođenje operacija. To je ključna stvar za primjenu u dubokom učenju jer se velika većina transformacija koje obavlja model može prikazati pomoću matričnog množenja čije je paralelno izvođenje na grafičkim karticama maksimalno optimirano. TensorFlow pruža mogućnost izračuna gradijenata pomoću metode automatske diferencijacije, što znači da je prilikom definiranja dubokog modela dovoljno samo napisati kod za izračun prolaza unaprijed. Biblioteka se danas većinom primjenjuje u područjima strojnog i dubokog učenja uz programski jezik Python zbog jednostavnosti.

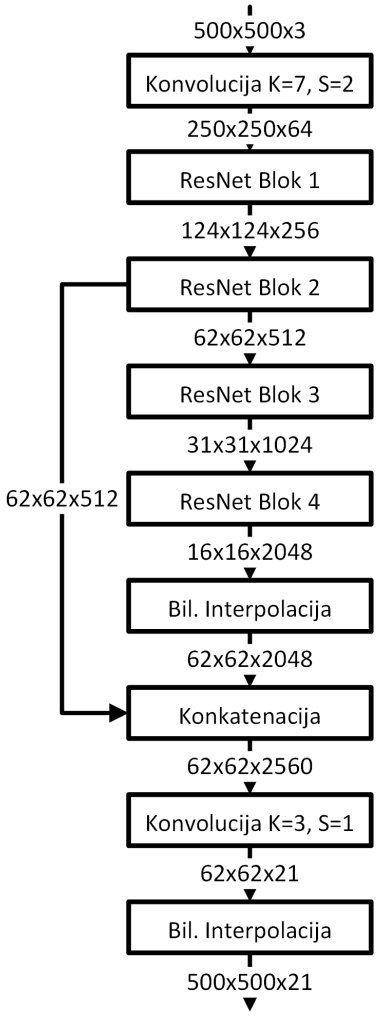
Keras [5] je biblioteka otvorenog koda za rad sa neuronskim mrežama koja pruža sučelje koje je jednu razinu apstrakcije iznad biblioteka za automatsku diferencijaciju kao što je TensorFlow. Keras zapravo pruža mogućnost izbora pozadinske biblioteke između sljedećih: TensorFlow, Microsoft Cognitive Toolkit, Theano, ili MXNet. Cilj ove biblioteke je stvoriti sučelje koje će biti jednostavnije i bliže korisniku što bi omogućilo brzo prototipiranje i iteriranje između modela.

U ovome radu korištena je biblioteka Keras sa TensorFlow pozadinom. Za mani-

pulaciju slikama korištena je biblioteka PIL [21]. Dodatno korištena je i biblioteka NumPy za razne numeričke izračune i manipulacije tenzorima van samog dubokog modela te Matplotlib za vizualizaciju, obje kao dio Scipy paketa [16].

5.1. Arhitektura generatora i diskriminatora

U modelu generatora kao kostur modela korištena je arhitektura ResNet-50 sa predtreniranim težinama na podatkovnom skupu Imagenet i odsječenim klasifikacijskim dijelom. Arhitektura je prikazana dijagramom na slici 5.1 gdje su u pravokutnicima navedene operacije odnosno moduli, a na bridovima dimenzije ulaznih/izlaznih tenzora. Kod konvolucijskih modula K označava veličinu kvadratne jezgre, a S pomak. Na ulazu je slika visine i širine 500x500 piksela sa tri kanala i prvo se konvolucijom

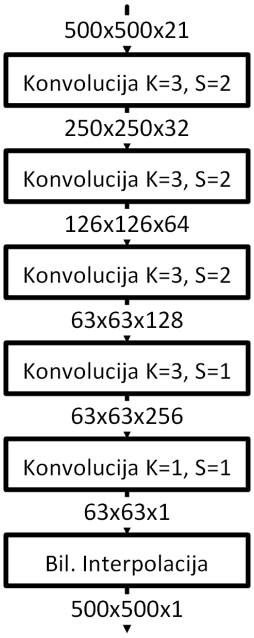


Slika 5.1: dijagram modela generatora

sa 64 jezgre veličine 7 i korakom 2 smanjuju prostorne dimenzije na 250x250 piksela.

Nakon toga slijede 4 ResNet bloka od kojih svaki polovi prostornu veličinu slike i udvostručuje broj kanala odnosno dubinu tenzora. ResNet blokovi sastoje se od više modificiranih osnovnih rezidualnih blokova(3, 4, 6 i 3 redom kako se pojavljuju). Prvi blok smanjuje veličinu slike koristeći sažimanje najvećom vrijednosti, a ostali konvoluciju sa pomakom 2 u svome prvom gradivnom bloku. Ukupno ResNet-50 prostorne dimenzije smanji 32 puta i taj izlazni tenzor potrebno je povećati na veličinu ulaza kako bi se dobile guste predikcije za semantičku segmentaciju. To se radi na način da se prvo izlaz iz posljednjeg bloka bilinearnom interpolacijom uveća na dimenzije izlaza drugog bloka, a nakon toga se ta dva izlaza konkateniraju po dubini. Nad tim tenzorom se tada provodi konvolucija sa brojem jezgara jednakim broju razreda i nakon toga ponovno bilinearnom interpolacijom dolazimo do veličine slike jednake veličini ulaza. Konkatenacija ulaza iz drugog bloka motivirana je gubitkom prostornih informacija u procesu poduzorkovanja. Na taj način se prostorne informacije restauriraju i imaju veći utjecaj na konačne guste predikcije. Kao aktivacijska funkcija korišten je ReLu, a korištena je i normalizacija po grupi.

Arhitektura diskriminatora odgovara diskriminatoru kojeg su koristili Radford i suradnici [22]. To je jednostavna konvolucijska arhitektura sa ukupno 5 konvolucijskih slojeva. Dijagram arhitekture prikazan je na slici 5.2. Prva tri konvolucijska sloja



Slika 5.2: dijagram modela diskriminatora

imaju veličinu jezgre 3 i pomak 2, pa smanjuju prostorne dimenzije ulaznog tenzora ukupno 8 puta. Slijedi još jedna konvolucija sa ukupno 256 jezgara, a nakon toga još

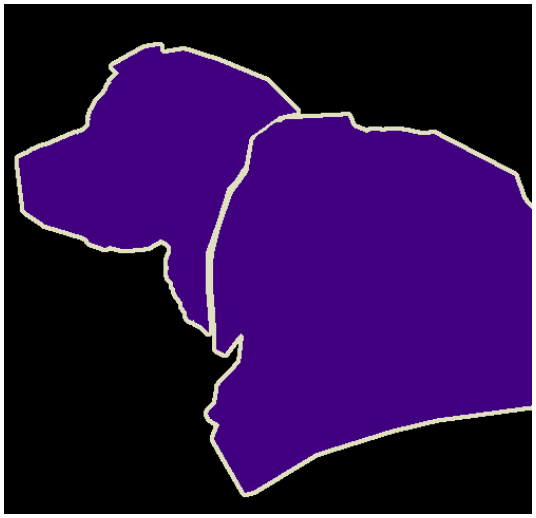
jedna konvolucija sa jednom jezgrom veličine 1, kako bi se dubina tenzora dovela na 1. Nakon toga se bilinearnom interpolacijom povećava slika na ulazne dimenzije. Kao aktivacijska funkcija korišten je LeakyReLu, a regularizacija je napravljena regularizacijom ispuštanjem (eng. *dropout*) sa vjerojatnošću 0.25. U radu su korištene verzije diskriminatora sa i bez normalizacije po grupi.

6. Evaluacija

6.1. Skup podataka Pascal VOC2012

Svi eksperimenti uključujući i postupak učenja i validacije modela provedeni su nad skupom podataka Pascal VOC2012 [7]. To je skup podataka koji osim anotacija za semantičku segmentaciju, sadrži anotacije i za klasifikaciju i detekciju objekata na slici. Za semantičku segmentaciju nudi anotacije na razini instance ili objekta. U ovome radu korištene su anotacije na razini objekta. To znači da je cilj apstrahirati sve instance nekoga objekta u jedan razred, primjerice ako ima više ljudi na slici model bi za svakoga trebao dati istu predikciju "čovjek". Pascal ukupno sadrži više od 10000 slika, ali su za semantičku segmentaciju anotirane 1464 slike u skupu za učenje i 1449 slika u skupu za validaciju. Oznake skupa za testiranje nisu javno dostupne, pa se u ovome radu ne koriste. Na slikama su označeni objekti u 20 različitih razreda, neki od

Slika 6.1: slika iz skupa Pascal VOC2012[7]



Slika 6.2: odgovarajuća seg. mapa

njih su: čovjek, avion, automobil, pas, stol i dr. Dodatni razred predstavlja pozadina, a postoji i tkz. "void" razred koji se nalazi na rubovima objekata i predstavlja nesigurnu/nepreciznu anotaciju. Nad pikselima koji su označeni kao "void" ne računa se

gubitak i prilikom evaluacije i izračuna metrika oni se ignoriraju. Slike su različitih dimenzija s tim da je u pravilu uvijek širina ili visina jednaka 500 piksela. Iako su modeli korišteni u ovome radu potpuno konvolucijski i mogu raditi sa svim dimenzijama slika, prilikom treniranja potrebno je stvoriti grupe podataka koje su konstantne veličine zbog efikasne iskoristivosti izračuna na grafičkim karticama. Jedno moguće rješenje je učiti nad kvadratnim uzorcima slučajno odrezanim iz slika, a ono koje je korišteno je nadopuniti sliku sa pikselima jednakim vrijednosti prosječnog piksela u skupu za treniranje. Kao oznaka za nadopunjene piksele postavljena je oznaka "void".

6.2. Metrike

U binarnoj klasifikaciji ako uspoređujemo stvarne oznake sa predikcijama modela moguća su ukupno četiri slučaja:

- TP (eng. *true positives*) - broj primjera koje je model ispravno klasificirao kao pozitivne,
- FP (eng. *false positives*) - broj primjera koje je model neispravno klasificirao kao pozitivne,
- FN (eng. *false negatives*) - broj primjera koje je model propustio klasificirati kao pozitivne,
- TN (eng. *true negatives*) - broj primjera koje je model ispravno klasificirao kao negativne.

Sažeto se mogu prikazati matricom zabune (eng. *confusion matrix*) koja se i sama ponekad koristi kao mjera vrednovanja. Međutim, češće se iz nje računaju druge mjere koje će biti opisane u nastavku.

Tablica 6.1: Matrica Zabune

		Oznake	
		1	0
Predikcije	1	TP	FP
	0	FN	TN

Točnost (eng. *accuracy*) je mjera koja računa udio ispravno klasificiranih primjera u cijelom skupu primjera. Problem ove mjere je što u slučaju neuravnotežene zastup-

ljenosti razreda nije indikativna.

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

Preciznost (eng. *precision*) je mjera koja računa udio ispravno pozitivno klasificiranih primjera u svim pozitivno klasificiranim primjerima. Ova mjera najčešće se koristi u kombinaciji sa odzivom.

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

Odziv (eng. *recall*) je mjera koja računa udio ispravno pozitivno klasificiranih primjera u skupu svih pozitivnih primjera.

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

Površina ispod krivulje preciznost-odziv (eng. *area under precision recall curve*, AUPR) je mjera koja je indikativna i u slučajevima nebalansiranih razreda. Moguće ju je računati za modele koji na svome izlazu za svaki razred daju vrijednost kao mjeru pripadnosti primjera tome razredu. Nije nužno da izlaz bude normaliziran ili vjerojatnosna distribucija. Točke krivulje preciznost odziv određene su izračunom preciznosti i odziva postavljanjem različitih pragova na izlazu modela čime se određuje predviđeni razred. Idealan model ima visoki odziv i visoku preciznost, što znači da je točan u svojim predikcijama i da ne propušta pozitivne primjere. Najčešće se između te dvije osobine modela mora birati i visoka preciznost znači niži odziv i obrnuto.

Srednji Jaccardov Indeks (eng. *mean Intersection Over Union*, mIOU) je standardna evaluacijska mjera za modele semantičke segmentacije. Jaccardov indeks računa omjer između presjeka i unije predikcija i oznaka. Srednji Jaccardov Indeks dobije se tako da se izračuna prosjek Jaccardovih indeksa izračunatih za svaki razred posebno.

$$IOU = \frac{TP}{TP + FN + FP} \tag{6.4}$$

$$mIOU = \frac{1}{|C|} \sum_{C} IOU_C \tag{6.5}$$

6.3. Rezultati

U nastavku će biti prikazani rezultati obavljenih eksperimenata.

Jedan od ciljeva ovoga diplomskog rada bio je proučiti utjecaj suparničkog učenja na rezultate modela za semantičku segmentaciju. U tablici 6.2 prikazani su rezultati

treniranja modela sa utjecajem i bez utjecaja suparničkog gubitka. Prvi model je treniran sa hiperparametrom $\lambda_{adv} = 0.0$ što znači da nije imao komponentu suparničkog gubitka. Drugi i treći model su modeli kojima je dodana komponenta suparničkog gubitka sa težinom $\lambda_{adv} = 0.1$. Razlika je u tome što je modelu SSGAN komponenta suparničkog gubitka bila uključena od samog početka učenja, a modelu SSGAN - continued tek nakon što je završeno treniranje bez suparničkog gubitka. Očekivano u posljednjem slučaju je naučeni diskriminator mogao pružiti kvalitetniji signal za učenje i rezultati to pokazuju. Uključivanje komponente suparničkog gubitka pokazalo je da donosi $1 - 2$ postotna poena u mjeri mIOU, s manjim utjecajem na mjeru točnosti piksela. Mjera točnosti piksela u ovome slučaju nije indikativna jer je na slikama dominantan razred pozadine, pa je realnija mjera srednji Jaccardov indeks. U tablici 6.3

Tablica 6.2: Rezultati evaluacije modela SSGAN na skupu za validaciju PASCAL VOC2012

Model	Acc	mIOU
SSGAN - $\lambda_{adv} = 0.0$	0.865	0.475
SSGAN - $\lambda_{adv} = 0.1$	**0.871**	0.487
SSGAN - continued - $\lambda_{adv} = 0.1$	0.867	**0.494**

prikazan je Jaccardov indeks po razredima za najbolji model. Očekivano zbog prirode skupa podataka model najbolje segmentira pozadinu, a najtežim razredom se pokazala stolica.

Tablica 6.3: IOU po razredima

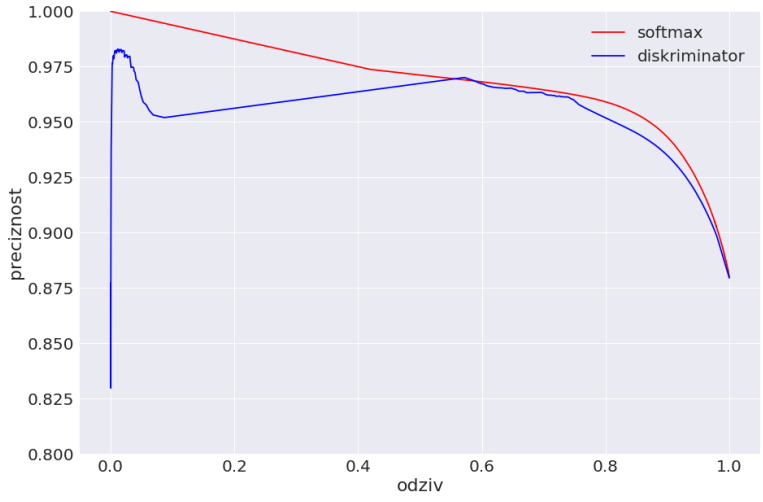
	pozadina	avion	bicikl	ptica	brod	boca	bus	auto	mačka	stolica	govedo	stol	pas	konj	motocikl	čovjek	biljka	ovca	kauč	vlak	tv
SSGAN - continued	0.88	0.65	0.43	0.55	0.39	0.50	0.60	0.61	0.59	0.14	0.44	0.31	0.50	0.39	0.58	0.66	0.30	0.50	0.28	0.52	0.45

Glavni cilj rada bio je usporediti izlaze diskriminatora kao izlaze pouzdanosti segmentacijskog modela na razini piksela sa izlazima softmaxa u istoj službi. Diskriminator i generator trenirani su paralelno. Diskriminator je u svakoj iteraciji postupka učenja bio ažuriran sa dvije minigrupe pomiješanih stvarnih oznaka i predikcija. Dodatno su oznake modificirane na način da je se pikselima označenima kao "void" promijenila oznaka u njima po euklidskoj udaljenosti najbliži razred. Razlog za to je što se u predikcijama modela razred "void" uopće ne pojavljuje i diskriminatoru bi tada bilo trivijalno razlikovati predikcije od oznaka gledajući samo postojanje piksela sa oznakom "void".

Kako bi usporedili izlaze pouzdanosti možemo zamisliti da je cilj u oba slučaja zapravo binarna klasifikacija između piksela koje je segmentacijski model ispravno klasificirao (1) i piksela koje je model netočno klasificirao (0). Iz softmaxa je kao pouzdanost da je piksel ispravno klasificiran preuzeta maksimalna vrijednost po svim razredima, a kod diskriminatora je to zapravo vjerojatnost da taj piksel potječe iz distribucije stvarnih oznaka. Kao evaluacijska mjera promatrana je površina ispod krivulje preciznost-odziv. Na slici 6.3 prikazane se krivulje preciznost-odziv za izlaze softmaxa i diskriminatora, a na tablici 6.4 površine ispod krivulje preciznost-odziv. Boljim detektorom nesigurnosti odnosno netočno klasificiranih piksela pokazao se softmax i to za oko 0.5 postotnih poena gledajući mjeru AUPR. Razlog za to može se tražiti u tome

Tablica 6.4: Rezultati evaluacije izlaza pouzdanosti predikcija za izlaze softmaxa i diskriminatora na skupu za validaciju PASCAL VOC2012

Model	AUPR
SSGAN - continued - softmax	0.963
SSGAN - continued - diskriminator	0.958



Slika 6.3: krivulja preciznost odziv

što diskriminator nije izravno treniran za zadatak detekcije nesigurnosti, i njegovi izlazi su na neki način ovdje prisilno/pogrešno iskorišteni. Primjerice, diskriminator je treniran da ruši "pouzdanost" čak i za piksele koje je generator ispravno klasificirao.

Sukladno tome, diskriminator ako modelu da visoku pouzdanost na ispravnim pikselima bit će jako kažnjen.

Na slikama 6.4a i 6.5a prikazani su primjer dobre i loše predikcije generatora i odgovarajuće mape pouzdanosti diskriminatora, a na slikama 6.4b i 6.5b prateće oznake i pouzdanosti diskriminatora u tome slučaju. Kod segmentacijskih mapa svaki je razred prikazan drugom bojom, a na desnom kraju slike se nalazi odgovarajuća legenda. Pouzdanosti se kreću od potpuno crne boje (pouzdanost jednaka 0.0) do potpuno bijele boje(pouzdanost jednaka 1.0).

(a) predikcije

(b) oznake

Slika 6.4: primjer dobre segmentacije modela i izlaza diskriminatora kao pouzdanosti

(a) predikcije

(b) oznake

Slika 6.5: primjer loše segmentacije modela i izlaza diskriminatora kao pouzdanosti

Očigledno je kako najveći signal diskriminatoru pomoću kojega razlikuje oznake

od predikcija modela dolazi sa rubova objekata. Rubovi oznaka su pravilniji sa čišćim linijama i u tome uskom području diskriminator ima visoku pouzdanost da ti pikseli dolaze iz skupa stvarnih oznaka. Analogno rubove predikcija također prepoznaje i ima nisku pouzdanost u te piksele i smatra da model griješi. Pozadinu model dosta kvalitetno segmentira i to je najteže područje za razlikovati po diskriminatora. Slično vrijedi i za unutrašnjost dovoljno velikih objekata, izgleda kao da je diskriminatoru to dovoljno blizu stvarnih oznaka. Ukratko se može zaključiti kako je diskriminator naučio da kada model griješi, onda griješi na rubovima objekata i tu mu daje nisku pouzdanost. Ovakav izlaz nije dobar izlaz pouzdanosti jer diskriminator i za rubove koji su dobro segmentirani daje nisku pouzdanost, a očito da i za pozadinu koja je u većini slučajeva kvalitetno segmentirana ne daje dovoljno visoku pouzdanost.

7. Zaključak

Zadatak ovoga diplomskog rada bio je proučiti i opisati postojeće pristupe za procjenu nesigurnosti predikcija, kao i mogućnost primjene nenadziranog suparničkog učenja na ovaj problem. Bilo je potrebno i uhodati postupak procjene nesigurnosti pomoću suparničkog učenja za zadatak semantičke segmentacije i evaluirati model na nekom od standardnih skupova podataka.

Implementiran je sustav kod kojega diskriminator na ulazu prima semantičku mapu nastalu predikcijama generatora ili uzorkovanu iz skupa oznaka i pokušava ih razlikovati na razini piksela. Generator je uobičajeni model semantičke segmentacije temeljen na potpuno konvolucijskoj arhitekturi ResNet-50. Izlazi takvoga diskriminatora iskorišteni su kao procjena nesigurnosti predikcija modela za semantičku segmentaciju. Model je treniran i evaluiran na skupu podataka Pascal VOC 2012. Pokazalo se kako komponenta suparničkog gubitka kao posljedica težnje segmentacijskog modela da zavara diskriminator pomaže i podiže evaluacijsku metriku semantičke segmentacije mIOU za $1 - 2$ postotna poena. Diskriminator kao procjenitelj nesigurnosti se pokazao lošijim od standardnog izlaza iz softmaxa. Rubovi objekata su se pokazali kao ključan signal za učenje diskriminatoru u njegovom nastojanju da razlikuje stvarne oznake od onih nastalih predikcijom generatora. Stoga je na rubovima objekata generalno pokazivao značajno manju pouzdanost u predikcije modela nego u unutrašnjosti objekata. Budući pravac razvoja mogao bi iskoristiti ideju za procjenu nesigurnosti inspiriranu mogućnosti studenta za uzimanjem pomoći prilikom rješavanja testa uz primjenu suparničkog učenja. Također, bilo bi vrijedno testirati procjenu nesigurnosti pomoću suparničkog učenja na drugim ispitnim skupovima i drugim zadacima računalnog vida.

LITERATURA

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, i Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, drugo izdanju, 2010. ISBN 026201243X, 9780262012430.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, i Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[5] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[6] Terrance DeVries i Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, i A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[8] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, i Yoshua Bengio. Generative adversarial nets. U *Advances in neural information processing systems*, stranice 2672–2680, 2014.

[10] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] K. He, X. Zhang, S. Ren, i J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[12] Dan Hendrycks i Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[13] Wei-Chih Hung, Yi-Hsuan Tsai, Yan-Ting Liou, Yen-Yu Lin, i Ming-Hsuan Yang. Adversarial learning for semi-supervised semantic segmentation. *arXiv preprint arXiv:1802.07934*, 2018.

[14] S. Ioffe i C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*, Veljača 2015.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, i Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[16] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`.

[17] Alex Kendall i Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? U *Advances in Neural Information Processing Systems*, stranice 5580–5590, 2017.

[18] Diederik P Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[20] Pauline Luc, Camille Couprie, Soumith Chintala, i Jakob Verbeek. Semantic segmentation using adversarial networks. *arXiv preprint arXiv:1611.08408*, 2016.

[21] Fredrik Lundh et al. PIL: Python imaging library. URL `https://pillow.readthedocs.io/`.

[22] Alec Radford, Luke Metz, i Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[23] Petar Veličković. Deep learning for complete beginners: convolutional neural networks with keras, 2017. URL `https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/`.

[24] Jan Šnajder i Bojana Dalbelo-Bašić. *Strojno Učenje*. 2012.

Procjenjivanje nesigurnosti predikcija diskriminativnih konvolucijskih modela primjenom suparničkog učenja

Sažetak

Procjena nesigurnosti predikcija važna je osobina sustava dubokog učenja. U ovome radu dan je pregled postojećih postupaka za procjenu nesigurnosti i predložen je takav postupak zasnovan na suparničkom učenju. Implementiran je model suparničkog učenja kod kojeg je generator model semantičke segmentacije zasnovan na arhitekturi ResNet-50, a diskriminator na svome izlazu daje gustu mapu procjene nesigurnosti predikcija generatora. Pokazan je utjecaj suparničkog gubitka na performanse modela u zadatku semantičke segmentacije. Uspoređene su i evaluirane procjene nesigurnosti funkcijom softmax i kao izlaz diskriminatora u suparničkom učenju.

Ključne riječi: duboko učenje, semantička segmentacija, procjena nesigurnosti, duboki konvolucijski modeli, suparničko učenje

Estimating prediction uncertainty of discriminative convolutional models with adversarial learning

Abstract

Estimating prediction uncertainty is important component of deep learning systems. In this thesis, an overview of existing methods for uncertainty estimation is given and one based on adversarial learning is proposed. Adversarial learning model is implemented in which generator is semantic segmentation model based on ResNet-50 architecture and discriminators output is uncertainty estimation map at pixel level of generators predictions. An impact of adversarial loss on performances of semantic segmentation model was shown. Uncertainty estimation using softmax function and discriminators output in adversarial learning were compared and evaluated.

Keywords: deep learning, semantic segmentation, uncertainty estimation, deep convolutional models, adversarial learning