SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 2284

# KONVOLUCIJSKI MODELI ZA OPTIČKI TOK

Antonio Anđelić

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 2284

# KONVOLUCIJSKI MODELI ZA OPTIČKI TOK

Antonio Anđelić

Zagreb, lipanj 2020.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 13. ožujka 2020.

# DIPLOMSKI ZADATAK br. 2284

| | |
|---|---|
| Pristupnik: | **Antonio Anđelić (0036492930)** |
| Studij: | Računarstvo |
| Profil: | Računarska znanost |
| Mentor: | prof. dr. sc. Siniša Šegvić |

Zadatak: **Konvolucijski modeli za optički tok**

Opis zadatka:

Procjenjivanje optičkog toka neriješen je problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolja rješenja tog problema postižu se dubokim konvolucijskim modelima. Ovaj rad razmatra nadzirane pristupe kod kojih svaki par slika iz skupa za učenje mora biti označen gustim poljem točnog optičkog toka. U okviru rada, potrebno je proučiti konvolucijske arhitekture za procjenu optičkog toka. Oblikovati odgovarajući model i naučiti ga na javno dostupnim skupovima. Procijeniti mogućnost izvedbe na ugradbenom računalnom sustavu. Validirati hiperparametre, prikazati i ocijeniti ostvarene rezultate te provesti usporedbu s rezultatima iz literature. Predložiti pravce budućeg razvoja. Radu priložiti izvorni kod razvijenih postupaka uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 30. lipnja 2020.

*Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na stručnim savjetima i ukazanom povjerenju.*

*Hvala roditeljima, bratu i sestrama na podršci.*

# SADRŽAJ

**1. Uvod** **1**

**2. Optički tok** **2**

2.1. Definicija optičkog toka . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2.2. Izobličavanje (*engl. warping*) . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3. Volumen cijene (*engl. cost volume*) . . . . . . . . . . . . . . . . . . . . 3

**3. Duboko učenje** **4**

3.1. Glavni dijelovi dubokih modela . . . . . . . . . . . . . . . . . . . . . 4

3.1.1. Konvolucijski sloj . . . . . . . . . . . . . . . . . . . . . . . . 7

3.1.2. Transponirani konvolucijski sloj . . . . . . . . . . . . . . . . 8

**4. PWCNet** **10**

4.1. Opis modela . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

**5. Skupovi podataka** **16**

5.1. Izlaz optičkog toka i vizualizacija . . . . . . . . . . . . . . . . . . . . 16

5.1.1. Vizualizacija . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

5.2. Flying Chairs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

5.3. KITTI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

**6. Eksperiment i rezultati** **19**

6.1. Mjera EPE (*engl. end-to-end point error*) . . . . . . . . . . . . . . . . 19

6.2. Inicijalno treniranje modela . . . . . . . . . . . . . . . . . . . . . . . . 20

6.3. Ugađanje (*engl. fine-tuning*) na skupu podataka KITTI . . . . . . . . 20

6.3.1. Evaluacija . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

6.4. Budući radovi i mogućnost za napredak . . . . . . . . . . . . . . . . . 23

**7. TensorRT i CUDA** **29**

7.1. CUDA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

7.1.1. Stvaranje dretvi . . . . . . . . . . . . . . . . . . . . . . . . 31

7.2. TensorRT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

7.2.1. Optimizacija modela . . . . . . . . . . . . . . . . . . . . . . 32

7.2.2. Komponente aplikacije za izgradnju stroja za zaključivanje . . 33

7.2.3. TensorRT i parseri . . . . . . . . . . . . . . . . . . . . . . . 34

**8. Implementacija PWCNet modela u TensorRT-u** **37**

8.1. Učitavanje Tensorflow težina . . . . . . . . . . . . . . . . . . . . . 37

8.2. Definiranje modela . . . . . . . . . . . . . . . . . . . . . . . . . . 38

8.2.1. Aktivacijska funkcija . . . . . . . . . . . . . . . . . . . . . . 39

8.2.2. Konvolucijski slojevi . . . . . . . . . . . . . . . . . . . . . . 39

8.2.3. Slojevi transponirane konvolucije . . . . . . . . . . . . . . . . 40

8.2.4. Sloj volumena cijene . . . . . . . . . . . . . . . . . . . . . . 40

8.2.5. Sloj izobličavanja . . . . . . . . . . . . . . . . . . . . . . . . 41

8.3. Rezultati . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42

8.4. Diskusija . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

**9. Zaključak** **46**

**Literatura** **48**

# 1. Uvod

Računalni vid široko je područje koje se bavi problematikom izlučivanja raznih informacija sa slike i videa. Kroz povijest, razvili su se mnogi algoritmi koji rješavaju različite probleme računalnog vida.

Jedan od takvih problema je optički tok. Optički tok je metoda procjene i prikaza kretanja objekata na temelju dvije ulazne slike. Optički tok uz samu detekciju kretanja pomaže pri detekciji i praćenju objekta, navigaciji i mnogim drugim sličnim problemima. Zbog njegovih mogućih primjena, postoje razni algoritmi koji pokušavaju riješiti problem optičkog toka.

Kao i za ostale probleme računalnog vida, trenutno dominiraju algoritmi s područja dubokog učenja. Duboko učenje je mnoge probleme računalnog vida s lakoćom rješilo, kao što su klasifikacija i semantička segmentacija slike. Postoje razni modeli koji su naučili procjenjivati optički tok. Glavni nedostatak većine modela je korištenje klasičnih algoritama optičkog toka za velik broj komponenti. Uz to, modeli su često preveliki. Optički tok zadaje velike probleme pri definiranju rješenja koje je prihvatljive točnosti, dovoljno brzo i s malim memorijskim zauzećem.

PWCNet je model opisan u radu Sun et al. (2017). Oni su htjeli ponuditi rješenje za optički tok koje postiže dobru točnost, ali koje je u isto vrijeme dovoljno jednostavno i brzo. Cilj ovog rada je ostvariti model definiran u njihovom radu.

Detekcija smjera i brzine kretanja određenih dijelova slike vrlo lako pronalazi primjenu u autonomnim vozilima. Autonomna vozila koriste duboko učenje kako bi riješili mnoge probleme zbog čega su razvijena radni okviri za lakše definiranje modela prilagođenih za ugradbena računala. Također, kao cilj ovog rada je istražiti mogućnost pokretanja PWCNet modela na ugradbenim računalima.

# 2. Optički tok

Kako bi se mogli baviti ostvarenim rješenjima u ovom radu, potrebno je definirati problem koji pokušavamo riješiti.

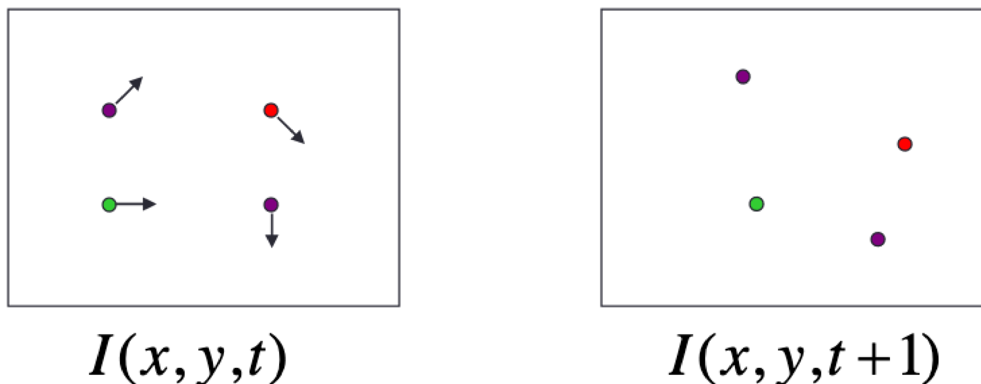
Probleme računalnog vida možemo podijeliti po domenama u kojima se razmatraju. Klasifikacija i segmentacija su primjeri problema koje pronalazimo u prostornoj domeni, na temelju jedne slike pokušavamo razaznati što slika prikazuje ili što svaki piksel slike prikazuje. Međutim, želimo razviti algoritme koji mogu donositi zaključke na temelju više slika. Drugim riječima, želimo riješiti probleme specifične za videa koji su samo skup slika u određenom redoslijedu. Video uvodi vremensku dimenziju u slikama. Optički tok je upravo metoda koja na temelju dviju slike, odvojene određenim vremenskim intervalima, pokušava donijeti određene zaključke.

## 2.1. Definicija optičkog toka

Glavni zadatak optičkog toka je procjena pomaka na temelju dviju slika. Za svaki piksel pokušavamo odrediti njegov pomak u vremenu. Vrlo bitna pretpostavka kod definicije optičkog toka je očuvanje svjetline. Svjetlina piksela kojeg promatramo je konstantna jer optički tok pokušava procijeniti kretanje svjetline piksela između dviju slika.

Optički tok pretpostavlja da vrijedi $I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$. Piksel s određenim karakteristikama u vremenu $t$ postoji i u vremenu $t + \Delta t$, ali na različitom mjestu što je označeno s pomakom $(\Delta x, \Delta y)$.

Očuvanje svjetline je pretpostavka koja razlikuje optički tok od polja pomaka. Primjer toga je rotacija crne kugle kojoj je svjetlina svakog piksela jednaka. Polje pomaka će prikazati pomak dok optički tok neće detektirati pomak. Međutim, translaciju takve kugle koja miruje polje pomaka neće prepoznati dok optički tok hoće jer detektira pomak piksela određenih karakteristika (Wu).




**Slika 2.1:** Prikaz optičkog toka. **Izvor:** `https://www.cc.gatech.edu/~afb/classes/CS4495-Fall2014/slides/CS4495-OpticFlow.pdf`

## 2.2. Izobličavanje (*engl. warping*)

Izlazi optičkog toka su vektori koji prikazuju kretanje piksela između dviju slika. Zbog toga, moguće je transformirati jednu od tih sliku primjenjujući dobivene rezultate. Takva transformacija se koristi u velikoj mjeri u rješenjima koja koriste duboko učenje jer možemo tražiti odstupanja između ulazne slike i izlaza navedene transformacije primijenjene na drugoj slici.

## 2.3. Volumen cijene (*engl. cost volume*)

Optički tok pokušava procijeniti pomak određenog piksela. Jedan od koraka je i pronalazak takvog piksela, a za to nam je potrebna i procjena odstupanja. Volumen cijene je upravo oblik kvantificiranja odstupanja asocijacijom određenog piksela iz prve ulazne slike s pikselima u drugoj slici. Takvu operaciju je u mnogim algoritmima potrebno primijeniti na cijeloj slici što predstavlja značajni procesorski napor, a time i usporenje cijelog algoritma.

# 3. Duboko učenje

Dio umjetne inteligencije koji u zadnje vrijeme privlači pozornost mnogih znanstvenika je strojno učenje. Strojno učenje je skup metoda koje, uz pravilnu definiciju i uz pomoć velikih skupova podataka, pronalaze rješenja za mnoge probleme. Međutim, podaci koji su nama zanimljivi sadrže mnoštvo informacija. Popularni algoritmi strojnog učenja s lakoćom su rješavali mnoge probleme, ali za određene probleme je potrebna analiza puno veće količine informacija. Duboko učenje je dio strojnog učenja koji na zanimljiv način analizira veliku količinu informacija, kao što su pikseli slike, i donosi određene zaključke. Glavna karakteristika dubokog učenja je podjela složenijih problema na više manjih, jednostavnijih problema.

## 3.1. Glavni dijelovi dubokih modela

Duboki modeli su temeljeni na neuronskim mrežama, pogotovo na konvolucijskim neuronskim mrežama. Zbog toga su najbitniji dijelovi dubokih modela i dijelovi neuronskih mreža.

#### Slojevi

Duboki modeli se sastoji od velikog broja parametarskih funkcija koji se često nazivaju i slojevi. Najosnovniji modeli se sastoje od primjene određenih funkcija na ulaz i prosljeđivanja izlaza sljedećem sloju.

**Potpuno povezani slojevi** su primjeri slojeva koji se često pojavljuju u dubokim modelima. To su slojevi gdje se za svaki izlaz sloja koriste svi ulazi. Potpuno povezani sloj se može definirati kao

$$\begin{aligned} f(x; W, b) &= \sigma(Wx + b) \\ \sigma(S)_i &= \sigma(S_i) \end{aligned} \tag{3.1}$$

Najčešći su konvolucijski slojevi, a njegova obrnuta operacija, transponirana konvolucija, prisutna je u puno manjoj mjeri. Oba sloja će biti detaljnije objašnjenja u nastavku.

#### Aktivacijska funkcija

Kako bi se ostvarila nelinearnost, slojevi sadrže u sebi aktivacijske funkcije. Aktivacijske funkcije se primjenjuju na svaki izlaz, a primjer takve funkcije bila bi sigmoidalna funkcija koja je definirana kao $\sigma(x) = \frac{1}{1+e^{-x}}$.

**Zglobnica** je aktivacijska funkcija koja trenutno dominira na području dubokog učenja. Zglobnica (*engl. ReLU*) je funkcija koja na svaki ulaz primjenjuje jednostavnu operaciju $f(x) = max(0, x)$. Drugim riječima propušta ulaze koji su veći od 0. Zbog njene jednostavnosti, jako brzo se računa, ali i ubrzava konvergenciju stohastičkog gradijenta. Postoje razne varijacije zglobnice. Jedna od njih je propusna zglobnica (*engl. leaky ReLU*) koja dopušta manji, pozitivni gradijent kad je ulaz manji od 0, formalno definirano kao

$$f(x) = \begin{cases} x & \text{ako } x > 0, \\ \alpha x & \text{inače} \end{cases} \tag{3.2}$$

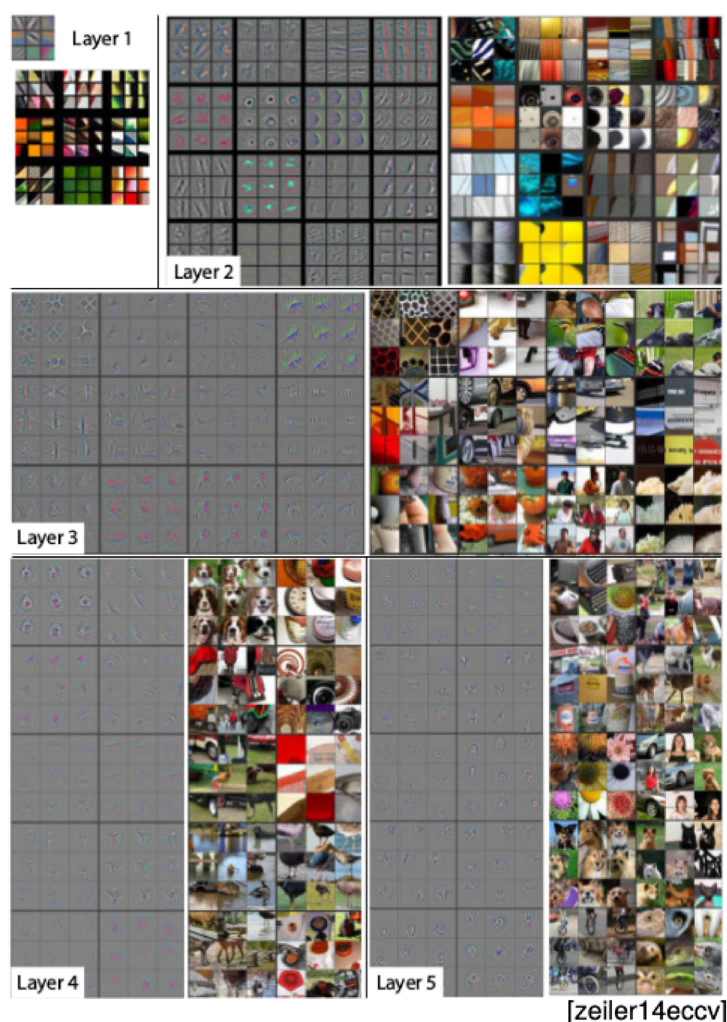
Prikaz zglobnice i njene varijacije je na slici 3.2.

#### Funkcija gubitka

Kako bi procijenili uspješnost dubokih modela, potrebno je definirati funkciju koja bi odredila odstupanje dobivenih rezultata od željenih rezultata. Funkcija gubitka na temelju izlaza modela i željenih rezultata kao izlaz daje broj kojim možemo procijeniti koliko naš model dobro radi na određenim podacima. Minimiziranje funkcije gubitka je najčešće i cilj dubokih modela.

#### Regularizacija

U dubokom učenju je bitno dobiti model koji radi dobro na primjerima koje nije vidio za vrijeme učenja. Time model pokazuje moć generalizacije. Drugim riječima, model ne sadrži rješenje specifično za skup za treniranje, nego je sposobno dati prihvatljivo rješenje i za slične ulaze.

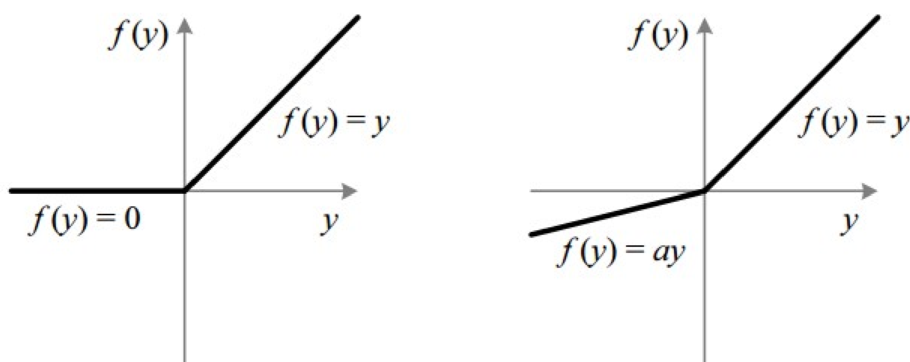



**Slika 3.1:** Prikaz rada algoritma dubokog učenja. Na slici je vidljivo kako model u prvom sloju pronalazi najjednostavnije uzorke koje u kasnijim slojevima spaja u složenije. **Izvor:** `http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf`

**Prenaučenost** je pojam koji definira upravo suprotno ponašanje, a tome su duboki modeli izrazito skloni. Postoje mnogi načini kako bi se izbjegla prenaučenost, a primjer toga je dodavanje funkciji gubitka funkciju regularizacije.

**L1 i L2** su primjeri takvih funkcija. Funkcija gubitka poprima oblik $L(X, Y) + \lambda N(w)$. Funkcija N predstavlja jednu od funkcija regularizacije, u ovom slučaju *L1* i *L2* norme prikazane formulama:

$$
\begin{aligned}
L1 &= \sum_{j=1} |w| \\
L2 &= \sum_{j=1} w_j^2
\end{aligned}
\tag{3.3}
$$




**Slika 3.2:** Prikaz zglobnice (**lijevo**) i propusne zglobnice (**desno**). **Izvor:** https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

, gdje $w$ predstavlja težine modela, a $\lambda$ hiperparametar.

### 3.1.1. Konvolucijski sloj

Vjerojatno najbitniji sloj u dubokim modelima je konvolucijski sloj jer je njegov zadatak upravo ono čemu modeli uglavnom i teže, pronalazak određenih uzoraka u ulazu. Konvolucijski slojevi se sastoje od jedne ili više jezgri koji predstavljaju težine posložene u tenzor trećeg reda, što je ujedno i oblik ulaza konvolucijskog sloja. Ulaz i težine se sastoje od dvije prostorne i jedne "semantičke" dimenzije, a izlaz konvolucijskog sloja nazivamo mapom značajki.

U strojnom učenju, konvolucija zapravo predstavlja operaciju unakrsne korelacije:

$$h(t) = (W * X)(t) = \int_{D(W)} W(\tau)X(t+\tau)d\tau \tag{3.4}$$

Iz jednadžbe je vidljivo da jezgru $W$ možemo koristiti za ekstrakciju lokalnih značajki iz signala $X$, upravo ono što želimo dobiti s konvolucijskim slojem.

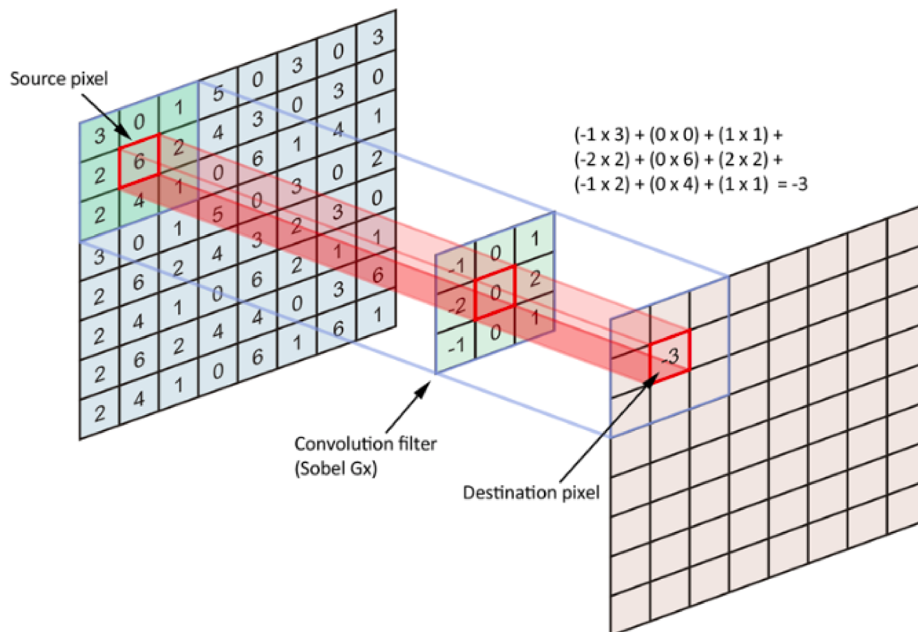
Konvolucijski slojevi su definirani jednostavnim množenjem elemenata jezgre s elementima dijela slike koji je jednake veličine kao i jezgra, a na kraju je moguće dodavanje istog broja svakom elementu što predstavlja pristranost (*engl. bias*). Vidimo da su transformacije lokalne, pikseli izlaza ovise samo o lokalnom susjedstvu piksela ulaza. Primjer jednog koraka je na slici 3.3.

Bitno svojstvo konvolucijskih slojeva je malen broj težina čemu pomaže i odabir manjih jezgri koje su često puno manje od dimenzija ulaza.

Kako su dimenzije jezgre često manje od ulaza, jezgre prolaze kroz cijeli ulaz s određenim korakom i slažu mapu značajki. Međutim, određene kombinacije dimen-

zija ulaza, dimenzija jezgre i koraka jezgre mogu dovest do nepravilnog prolaza jer nam u bar jednoj prostornoj dimenziji preostane broj piksela koji je manji od prikladne dimenzije jezgre. Postoje dva načina rješavanja takvog problema, odbacivanjem navedenih piksela ili dodavanjem dopune kako bi i zadnje mapiranje postalo ispravno.

Uz korak i dimenzije jezgre, možemo definirati i dilataciju. Dilatacijom definiramo razmak između težina.




**Slika 3.3:** Korak konvolucijskog sloja. Jezgra se primjenjuje na dio slike. **Izvor:** `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`

### 3.1.2. Transponirani konvolucijski sloj

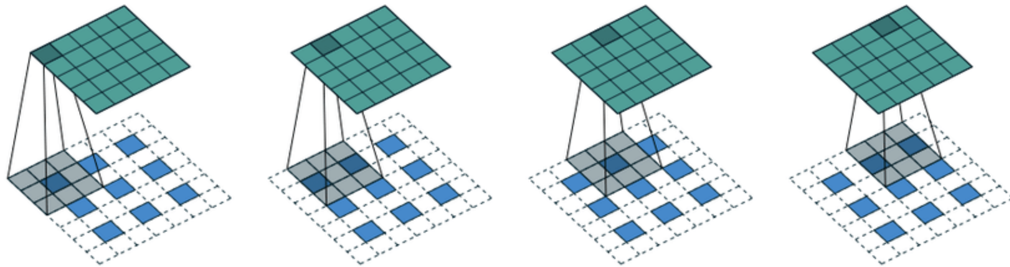
Postoje situacije kada ne želimo tražiti značajke na slici, nego obaviti suprotnu operaciju. Drugim riječima htjeli bi provesti naduzorkovanje.

Naduzorkovanje se može postići interpolacijom i drugim algoritmima gdje je potrebna intervencija ljudi. S transponiranom konvolucijom možemo prepustiti računalu učenje potrebnih transformacija.

Iako se transponirana konvolucija često naziva i dekonvolucija, ne predstavlja istu operaciju kao kod procesiranja signala i slike. U takvim slučajevima dekonvolucija predstavlja jednostavno operaciju koja poništava konvoluciju.

Transponirana konvolucija se sastoji od jednakih dijelova kao i konvolucija, koraka jezgre, dimenzije jezgre i dilatacije.

Kako bi iz ulaza dobili izlaz veće dimenzije potrebno je dodati dopunu i obavljanjem jednakih operacija kao kod konvolucije dobivamo izlaz, jezgra prolazi kroz ulaz određenim korakom. Primjer transponirane konvolucije možemo vidjeti na slici 3.4.



**Slika 3.4:** Primjer koraka transponirane konvolucije. **Izvor:** `https://www.researchgate.net/figure/Figure-B3-The-transposed-convolution-operation-also-called-deconvolution-performing_fig31_324783775`

# 4. PWCNet

Metode dubokog učenja su se pokazale uspješnima na mnogim problemima vezanim za sliku pa tako i na problemu procjene optičkog toka. Dosadašnje metode optičkog toka sastojale su se u velikoj mjeri od kompleksnih optimizacijskih problema što uzrokuje presporo izvođenje za aplikacije u pravom vremenu. Konvolucijski duboki modeli pokušavaju riješiti taj problem.

Mogućnost detektiranja optičkog toka dubokim modelima dokazali su Fischer et al. u radu Fischer et al. (2015) gdje su predložili dvije arhitekture bazirane na U-Net-u, FlowNetS i FlowNetC.

Par godina kasnije Ilg et al. su slaganjem FlowNetC i FlowNetS modela dobili jedan veći model, nazvan FlowNet2, koji postiže odlične rezultate i s puno većom brzinom. Međutim, veći modeli su skloniji prenaučenosti pa se podmreže FlowNet2 moraju trenirati sekvencijalno, a njegov memorijski otisak od 640MB nije prikladan za mobilne i ugradbene uređaje.

SPyNet definiran u radu Ranjan i Black (2016) pokušava riješiti probleme memorije koristeći prostornu piramidalnu mrežu (*engl. spatial pyramid network*) i transformira drugu sliku koristeći izračunati optički tok što su dvije tehnike prisutne i u klasičnim metodama procjene optičkog toka. Kako se detektiraju manji pokreti, potrebna je i manja mreža za samu detekciju. Iako je model postigao bolje rezultate od FlowNetC, rezultati su bili lošiji od FlowNet2 i FlowNetS. Time je pokazano kako manjim modelom dobivamo i slabiju točnost.

PWCNet, koji je opisan u Sun et al. (2017), je dokaz da ipak ne mora postojati kompromis između veličine modela i njegove točnosti. Pametnim korištenjem klasičnih metoda optičkog toka i konvolucijskih slojeva može se dobiti model koji u isto vrijeme postiže odličnu točnost, zauzima malo memorije i u isto vrijeme procesira slike u razumnom vremenu.

## 4.1. Opis modela

Kako je navedeno ranije u radu, cilj PWCNet-a je ostvariti točnost na razini najboljih modela, ali i smanjiti brzinu izvođenja. Cilj pokušavaju ostvariti korištenjem metoda dubokog učenja, kao što su konvolucijski slojevi, u što većoj mjeri.

Na samom početku, iz ulaznih slika se ne stvara fiksna piramida, nego piramida značajki koja se sastoji od slobodnih parametara. Razlog tome je različitost između ulaznih slika zbog promjene u svjetlini ili sjene. Izobličavanje prisutno u klasičnim metodama koristi se kao jedan od slojeva modela kako bi se procijenile veće kretnje. Volumen cijene se također pojavljuje kao sloj jer je diskriminativnija reprezentacija optičkog toka od ulazne slike, a na temelju njegovog izlaza procjenjuje se optički tok slojevima tipičnih za konvolucijske modele. Kako sloj izobličavanja i volumena cijene nemaju parametre za učenje, veličina modela je manja.

Učestala praksa u metodama optičkog toka je procesiranje izlaza koristeći kontekstualne informacije. PWCNet je odlučio i taj dio zamijeniti slojevima konvolucijskih modela.

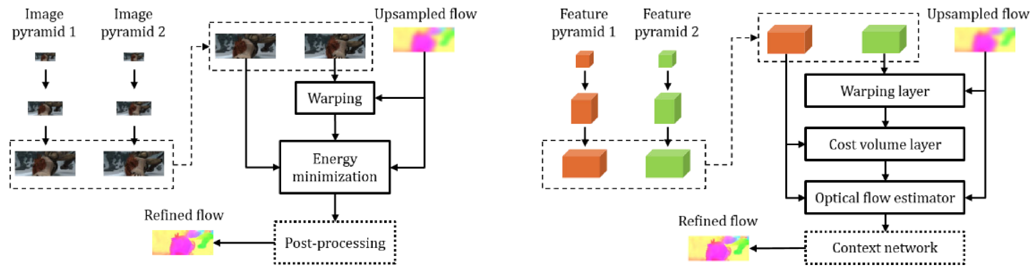
U teoriji, izobličavanje, volumen cijene i slojevi konvolucijskih modela su jeftiniji za izračunati od provođenja optimizacijskih postupaka kao što su minimizacija energije zbog čega bi PWCNet trebao u cijelosti biti brži.

Na slici 4.1 vidimo usporedbu klasičnih metoda optičkog toka s PWCNet-om na jednoj razini piramide. PWCNet, nakon što izračuna piramidu značajki, za svaki sloj prolazi kroz jednake korake. Prvo izobličava značajke druge slike koristeći naduzorkovani tok, izračunava volumen cijene što kasnije procesira konvolucijskim slojevima. Procesiranje i kontekstualna mreža su opcionalni u oba načina.

Važno je za napomenut da se sva naduzorkovanja izračunavaju s transponiranim konvolucijskim slojevima. Primjenjivanjem i učenjem transponiranih konvolucijskih slojeva dobivamo efikasan način naduzorkovanja. Također, jedina aktivacijska funkcija koja se koristi u modelu je propusna zglobnica.

#### Ekstraktor piramide značajki

Reprezentaciju značajke označit ćemo sa $c$. Na temelju dviju ulaznih slika, $I_1$ i $I_2$, generira se piramida s $L$ razina gdje su donja, nulta razina ulazne slike, $c_t^0 = I_t$. Kako bi se generirala reprezentacija značajki na $l$-tom sloju, $c_t^l$, koriste se konvolucijski slojevi kako bi se poduzorkovale značajke $l-1$-tog sloja piramide, $c_t^{l-1}$, s faktorom 2. Od prvog do šestog sloja piramide broj jezgri je 16, 32, 64, 96, 128.




**Slika 4.1:** Usporedba klasičnih metoda za izračun optičkog toka i PWCNet-a. **Lijevo:** Prikaz izračuna optičkog toka na jednoj razini piramide kroisteći minimizaciju energije. **Desno:** Piramida značajki i izračun toka za jednu razinu piramide značajki definiran u PWCNet-u. Postprocesiranje i kontekstualna mreža su opcionalne kod oba pristupa. **Izvor:** Sun et al. (2017).

### Sloj izobličavanja

Na $l$-toj razini izobličavamo značajke druge slike prema prvoj slici koristeći naduzorkovani tok iz $l+1$-te razine:

$$c_w^l(x) = c_2^l(x + up_2(w^{l+1})(x)), \tag{4.1}$$

gdje $x$ predstavlja indeks piksela, a $up_2(w^{l+1})$ ima vrijednost 0 na najvišoj razini. Za operaciju izobličavanja koristi se bilinearna interpolacija. Korisno svojstvo izobličavanja je to što može kompenzirati za određena geometrijska iskrivljenja i postaviti određene dijelove slike u točan omjer za netranslacijske pokrete.

### Sloj volumena cijene

Volumen cijene se računa na temelju značajki i sprema cijenu asociranja piksela s pripadnim pikselom na sljedećoj slici. Cijena asociranja je definirana kao korelacija između značajki prve slike i izobličenih značajki druge slike:

$$cv^l(x_1, x_2) = \frac{1}{N}(c_1^l(x_1))^T c_w^l(x_2), \tag{4.2}$$

gdje je T operacija transponiranja, a N duljina stupčanog vektora $c_1^l(x_1)$. Zbog korištenja piramide s L-razina, trebamo izračunati samo djelomičan volumen cijene s ograničenim rasponom od $d$ piksela, drugim riječima $|x_1 - x_2|_\infty \leq d$. Pokret od jednog piksela na najvišoj razini predstavlja pokret od $2^{L-1}$ na slikama pune rezolucije. Iz tih razloga možemo postaviti malen $d$. Dimenzije 3D volumena cijene su $d^2 \times H^l \times W^l$, gdje su $H^l$ i $W^l$ dimenzije $l$-te razine piramide.

### Procjenitelj optičkog toka

Jedan od jednostavnijih dijelova koji se sastoji samo od konvolucijskih slojeva je upravo procjenitelj optičkog toka. Na temelju volumena cijene, značajki prve slike i naduzorkovanog optičkog toka izračunava se izlaz $w^l$ na $l$-toj razini. Broj jezgri u svakom konvolucijskom sloju ovisno o razini su redom 128, 128, 96, 64 i 32. Procjenitelji na različitim razinama ne dijele parametre. Postupak procjenjivanja se ponavlja do željene razine, $l_0$.

Procjenitelj se može ojačati gusto povezanim slojevima (*engl. DenseNet*) opisanih u Huang et al. (2016). Gusto povezani slojevi definiraju ulaz kao spojeni izlaz i ulaz njegovog prethodnog sloja. DenseNet modeli sadrže više direktnih veza nego klasični slojevi što vodi do poboljšanja u klasifikaciji slike.

### Kontekstualna mreža

Klasične metode za procjenu optičkog toka često koriste kontekstualne informacije kako bi procesirali svoje izlaze. PWCNet i taj dio zamjenjuje s manjom podmrežom nazvanom kontekstualna mreža. Kontekstualna mreža efikasno povećava receptivno polje svakog izlaza na željenoj razini piramide. Kao ulaz prima procijenjeni optički tok i značajke predzadnjeg sloja iz procjenitelja toka, a kao izlaz daje dorađeni tok.

Podmreža je jednostavna unaprijedna konvolucijska mreža. Sastoji se od 7 konvolucijskih slojeva dimenzija $3 \times 3$. Svaki sloj sadrži različitu dilatacijsku konstantu. Konstante su redom definirane s vrijednostima 1, 2, 4, 8, 16, 1 i 1. S većom dilatacijom efikasno povećavamo receptivno polje svakog izlaza.

### Funkcija gubitka

Definiramo $\Theta$ kao skup svih slobodnih parametara modela što uključuje ekstraktor piramide značajki i procjenitelje optičkog toka za svaku razinu piramide. Neka $w^l_\Theta$ predstavlja tok na $l$-toj razini piramide predviđenog od strane modela, a $w^l_{GT}$ točan optički tok. Koristi se jednak gubitak na više razina predloženog u Fischer et al. (2015):

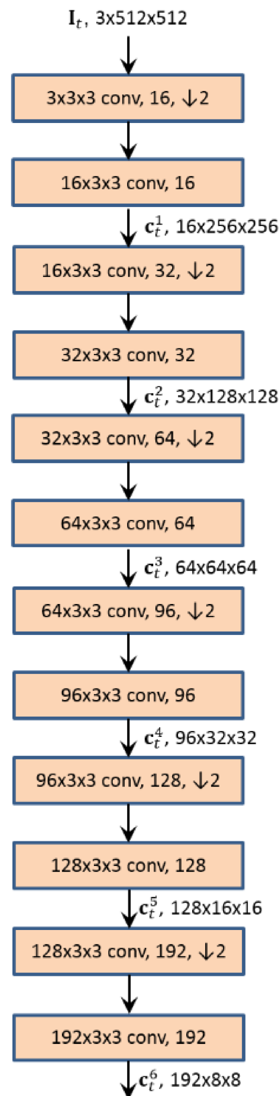
$$L(\Theta) = \sum_{l=l_0}^{L} \alpha_l \sum_x |w^l_\Theta(x) - w^l_{GT}(x)|_2 + \gamma|\Theta|_2, \tag{4.3}$$

gdje $|\cdot|_2$ izračunava L2 normu vektora, a drugi član regularizira parametre modela.

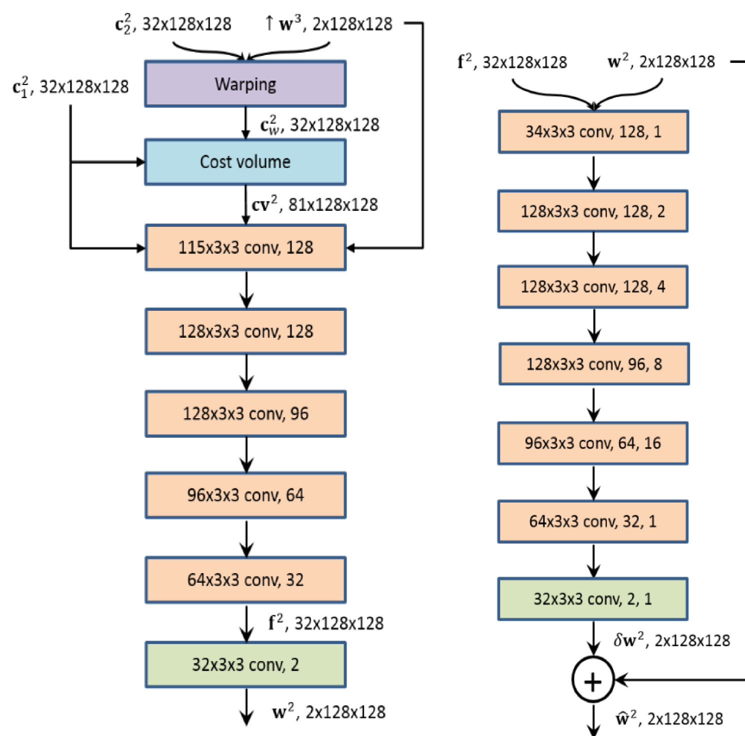
Za ugađanje (*engl. fine-tuning*) koristi se sljedeći gubitak:

$$L(\Theta) = \sum_{l=l_0}^{L} \alpha_l \sum_x (|w^l_\Theta(x) - w^l_{GT}(x)| + \epsilon)^q + \gamma|\Theta|_2, \tag{4.4}$$

gdje $|\cdot|$ predstavlja L1 normu, $q < 1$ manje kažnjava odstupajuće vrijednosti, a $\epsilon$ je manja konstanta.




**Slika 4.2:** Prikaz grafa podmreže za ekstrakciju piramide značajki. **Izvor:** Sun et al. (2017)




**Slika 4.3:** Prikaz grafa podmreža. Lijevo: podmreža za procjenu optičkog toka. Desno: kontekstualna mreža. **Izvor:** Sun et al. (2017)

# 5. Skupovi podataka

Kod metoda dubokog učenja podaci su izrazito bitna komponenta. Što su model i problem kompleksniji, potrebna je sve veća količina podataka. Uz samo količinu, bitno je da su podaci i ispravno označeni. Klasifikacija je primjer problema koji je ljudima relativno jednostavan jer je slici potrebno pridružiti odgovarajuću klasu. Iako je primjer jednostavan, greške su i dalje moguće. Optički tok predstavlja izrazito velik problem kod označavanja. U slučaju segmentacije označavamo svaki piksel, što samo po sebi predstavlja veliki izazov. Kod optičkog toka potrebno je za svaki piksel odrediti još veći broj informacija.

Iz navedenih razloga ne postoji velik broj skupova podataka za optički tok. Za većinu skupova podataka za optički tok vrijedi jedno zanimljivo svojstvo, izgenerirani su računalom. Ako sami izgeneriramo podatke, možemo biti u potpunosti sigurni u njihovu točnost. Drugim riječima, za svaki računalno izgenerirani pomak piksela znamo točan smjer u svim dimenzijama i brzinu gibanja potrebnih za optički tok.

Također, za optički tok je specifično da kao ulaz zahtjeva dvije slike jer se pomak traži upravo na temelju razlike između parova.

## 5.1. Izlaz optičkog toka i vizualizacija

Optičkim tokom želimo označiti pomak pa su nam za svaki piksel potrebne dvije informacije za pomak u obje dimenzije, intenzitet i smjer. Stoga, označeni podaci su dimenzija $H \times W \times 2$, gdje $H$ i $W$ predstavljaju visinu i širinu ulaznih parova slika, a zadnja dimenzija predstavlja pomak piksela.

### 5.1.1. Vizualizacija

Kako bi podatke vizualizirali, koristimo HSV prostor boja. HSV prostor boja se sastoji od 3 komponente, ton boje, zasićenje boje i svjetlina. Ton je predstavljen kutom, a zasićenost vrijednošću od 0 do 100. Možemo primijetiti kako je taj prostor boja

intuitivan za prikaz optičkog toka. Iz tog razloga prebacujemo izlaz optičkog toka iz kartezijevog koordinatnog sustava u polarni koordinatni sustav iz čega dobijemo kut i duljinu što pridružujemo tonu boju i zasićenju boje. Svjetlinu postavljamo na najveću vrijednost

## 5.2. Flying Chairs

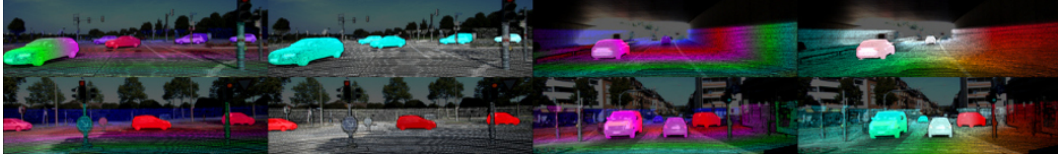
Flying Chairs skup podataka je primjer skupa podataka koji je računalno izgeneriran. Sastoji se od 22872 parova slika i pripadajućih optičkih tokova. Slike prikazuju generirane 3D modele stolaca koji se miču ispred slučajno odabranih pozadina sa sustava Flickr. Skup podataka je nastao za potrebe treniranja modela definiranog u Fischer et al. (2015). Ostali skupovi podataka sadržavali su izrazito malen broj podataka što nije slučaj za Flying Chairs skup podataka. Primjer podataka Flying Chairs skupa podataka možemo vidjeti na slici 5.1.

**Slika 5.1:** Primjer dva podatka iz Flying Chairs skupa podataka. Prve dvije slike predstavljaju ulaze, a zadnje slika optički tok označen na prvoj slici. **Izvor:** Fischer et al. (2015)

## 5.3. KITTI

KITTI je niz skupova podataka koji se mogu koristiti za razvijanje rješenja za razne probleme kao što su stereo, optički tok, vizualna odometrija, 3D detekcija objekata i 3D praćenje. Za njihov razvoj su zaslužni Menze et al. koji opisuju skup podataka u svoja dva rada, Menze et al. (2015) i Menze et al. (2018). Ono što je bitno kod

svih KITTI skupova podataka je to da su podaci snimljeni u vožnji, a točni podaci su dobiveni koristeći laserski skener (lidar). Kao što je navedeno ranije, generirani podaci dominiraju među skupovima podataka za optički tok što ovdje nije slučaj. Primjer oznaka iz njihovog skupa podataka možemo vidjeti na slici 5.2. KITTI skupovi se često koristi u mjerenju točnosti modela (*engl. benchmarking*).



**Slika 5.2:** Primjer oznaka za optički tok iz KITTI skupa podataka. **Izvor:** `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=flow`

# 6. Eksperiment i rezultati

Jedan od ciljeva ovog rada bio je implementirati model PWCNet definiranog u radu Sun et al. (2017) i opisanog u poglavlju 4. Kao što je ranije navedeno, kod procjenitelja optičkog toka moguće je koristiti gusto povezane slojeve. Također, kontekstualna mreža se može pozivati na svakoj razini piramide. Zbog jednostavnosti, implementirao sam PWCNet bez gusto povezanih slojeva i time smanjio broj parametara s 9M na 4M parametara, a kontekstualna mreža se poziva samo na zadnjoj razini piramide. Kako je navedeno u radu, koristio sam piramidu značajki sa 7 razina, a prostor pretrage $d$ je bio 4.

## 6.1. Mjera EPE (*engl. end-to-end point error*)

Postoje mnoge mjere uz pomoć kojih mjerimo uspješnost rada modela. Kod optičkog toka najpopularnija je mjera EPE (end-to-end point error). Dobije se kao euklidska udaljenost između dva vektora, u našem slučaju između izlaza implementiranog modela i stvarnog stanja. Grafički prikaz EPE mjere može se vidjeti na slici 6.1. EPE mjera je definirana jednadžbom:

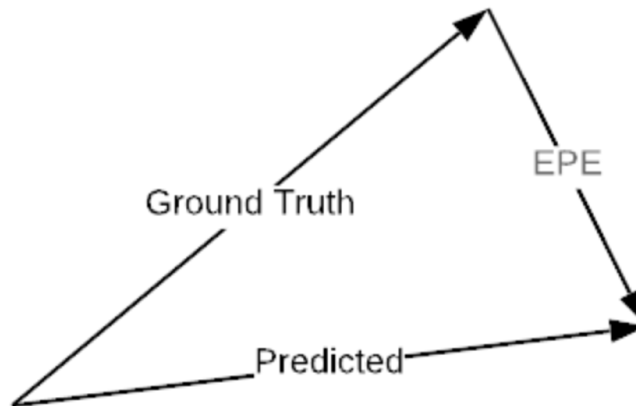
$$EPE = \sqrt{(u - u_{GT})^2 + (v - v_{GT})^2} \tag{6.1}$$

, gdje je $(u_{GT}, v_{GT})$ točan optički tok za neki piksel, a $(u, v)$ procjenjeni optički tok.

Uz EPE, često se koristi i srednja kutna greška (*engl. average angular error*) koja računa razliku kutu između točnog i procjenjenog vektora toka definirana jednadžbom:

$$AAE = cos^{-1}(\hat{c} \cdot \hat{e}) \tag{6.2}$$

, gdje je $\hat{c}$ normiliziran vektor točnog optičkog toka, a $\hat{e}$ normalizirani vektor procjenjenog optičkog toka.




**Slika 6.1:** Grafički prikaz EPE mjere. **Izvor:** `https://medium.com/swlh/what-is-optical-flow-and-why-does-it-matter-in-deep-learning-b3278bb205b5`

## 6.2. Inicijalno treniranje modela

Kako model ima slučajno inicijalizirane parametre bilo je potrebno trenirati model na nekom većem skupu podataka. Iz tog razloga, model je prvo treniran na Flying Chairs skupu podataka. 97% podataka korišteno je za treniranje, a ostalih 3% za validaciju.

Težine gubitaka za svaku razinu definirane su kao u radu, $\alpha_6 = 0.32$, $\alpha_5 = 0.08$, $\alpha_4 = 0.02$, $\alpha_3 = 0.01$ i $\alpha_2 = 0.005$. Regularizacijski faktor je postavljen na $\gamma = 0.0004$. Za optimizaciju treniranja korišten je Adam sa $\epsilon = 10^{-8}$. Kao gubitak korištena je jednadžba 4.3. Stopa učenja je inicijalno postavljena na $10^{-4}$. Nakon $30000$ koraka smanjena je na $5 \times 10^{-5}$, a nakon sljedećih $30000$ koraka na $10^{-5}$. Veličina grupe je bila 8. Na slikama za treniranje nisu primijenjene transformacije.

Cilj treniranja je bio pripremiti model za treniranje na manjim, kompleksnijim skupovima. Iz tog razloga, model je treniran sve dok nismo spustili srednji EPE na validacijskom skupu na razumnu razinu. Nakon $100000$ koraka dobiven je srednji EPE od $2.73$. U tablici 6.1 možemo vidjeti usporedbu dobivene srednje EPE vrijednosti s ostalim modelima koji su ostvarili dobre rezultate.

## 6.3. Ugađanje (*engl. fine-tuning*) na skupu podataka KITTI

Nakon što sam dobio model naučen na Flying Chairs skupu podataka, model je dotreniran na skupu podataka KITTI. Skup podataka KITTI predstavlja kompleksniji i

| Model | EPE na Flying Chairs skupu podataka |
|---|---|
| EpicFlow | 2.94 |
| DeepFlow | 3.53 |
| FlowNetS | 2.71 |
| FlowNetC | 2.19 |
| FlowNet 2.0 | 1.78 |
| SpyNet | 2.63 |
| PWCNet | 2.00 |
| PWCNet (bez DenseNet-a) | 2.06 |
| PWCNet (moja implementacija na skupu za validaciju) | 2.73 |
| PWCNet (moja implementacija na skupu za treniranja) | 2.31 |

**Tablica 6.1:** Usporedba srednjih EPE mjera najboljih modela i vlastite implementacije.

osjetljiviji skup podataka koji se sastoji od 200 slika s označenim optičkim tokom. 170 slika je bilo korišteno za samo treniranje modela, a 30 za validaciju. Zbog jednostavnosti korištene su oznake samo za piksele koji se mogu pronaći na obje slike.

Težine gubitaka za svaku razinu jednake su kao u prethodnom slučaju. Isto vrijedi i za regularizacijski faktor. Za razliku od početnog treniranja, parametar $q$ je postavljen na $0.4$, a $\epsilon$ na $0.01$. Stopa učenja je na početku postavljena na $10^{-5}$, što je svakih $100000$ koraka smanjeno redom na $5 \times 10^{-6}$, $10^{-6}$, $5 \times 10^{-7}$. Veličina grupe je bila 8. Na slikama za treniranje nisu primijenjene transformacije.

Model je treniran na skupu za treniranje $400000$ koraka. Nakon treniranja na skupu za treniranje, dodan je i validacijski skup i model je treniran još $100000$ koraka na spojenom skupu za treniranje i validaciju.

Nakon cjelokupnog treniranja na KITTI skupu podataka postignuti su rezultati prikazani u tablici 6.2.

Grafički prikazi izlaza modela na testnom skupu vidljivi su za pojedine primjere na slikama 6.2, 6.3, 6.4, 6.5. Možemo primijetiti kako model izrazito dobro pronalazi kretanje vozila na slikama. Svaka slika u KITTI skupu podataka snimljena je za vrijeme vožnje pa su time i vozila najčešći objekti. Međutim, ako je više vozila na maloj udaljenosti, model teže pronalazi parove piksela. Uz to, bitna je okolina. Primjeri koji su snimljeni u gradu sadrže više različitih objekata na slici što predstavlja velik problem modelu. Možemo primijetiti kako model uspješno detektira statične pozadine čemu je vjerojatno pomogao i Flying Chairs skup podataka koji se sastoji samo od statičkih

pozadina.

| Model | EPE na KITTI skupu podataka |
|---|---|
| FlowNet 2.0 | 2.30 |
| PWCNet | 2.16 |
| PWCNet (moja implementacija) | 10.35 |

**Tablica 6.2:** Usporedba srednjih EPE mjera najboljih modela i vlastite implementacije (svi navedeni rezultati su dobiveni na skupu korišten za učenje)

### 6.3.1. Evaluacija

Kako bismo dobili detaljniji uvid u performanse modela pokrenuo sam službenu evaluaciju autora skupa podataka KITTI. Zbog nemogućnnosti dobivanja točnih oznaka za skup za testiranje, pokrenuo sam evaluaciju na skupu za treniranje.

Jedan od podataka koji kao izlaz generira evaluacija je postotak piksela za koji je predviđeni optički tok pogrešan za različite stupnjeve odstupanja. Rezultati su prikazani u tablici 6.3. Iako grafički rezultati za pojedine slike izgledaju dobro, na temelju pogrešnih piksela možemo primijetiti kako dosta piksela odstupa za veći broj piksela od točnog.

| Odstupanje | Postotak pogrešnih piksela |
|---|---|
| $\leq 1$ piksel | 0.70 |
| $\leq 2$ piksela | 0.63 |
| $\leq 3$ piksela | 0.58 |
| $\leq 4$ piksela | 0.54 |
| $\leq 5$ piksela | 0.50 |

**Tablica 6.3:** Postotak pogrešnih piksela predviđenog optičkog toka za različita dopuštena odstupanja.

Na slikama 6.6 i 6.7 vidimo grafički prikaz pogrešaka. Što je piksel bliži bijeloj boji to je greška veća. Crvena boja označava piksele koji su vidljivi samo na prvoj slici. Slike grešaka potvrđuju prethodni zaključak. Vidimo kako su vozila često tamnije označena jer su greške za njih relativno male, ali za kompleksnije pozadine primjećujemo veliku količinu bijelih piksela.

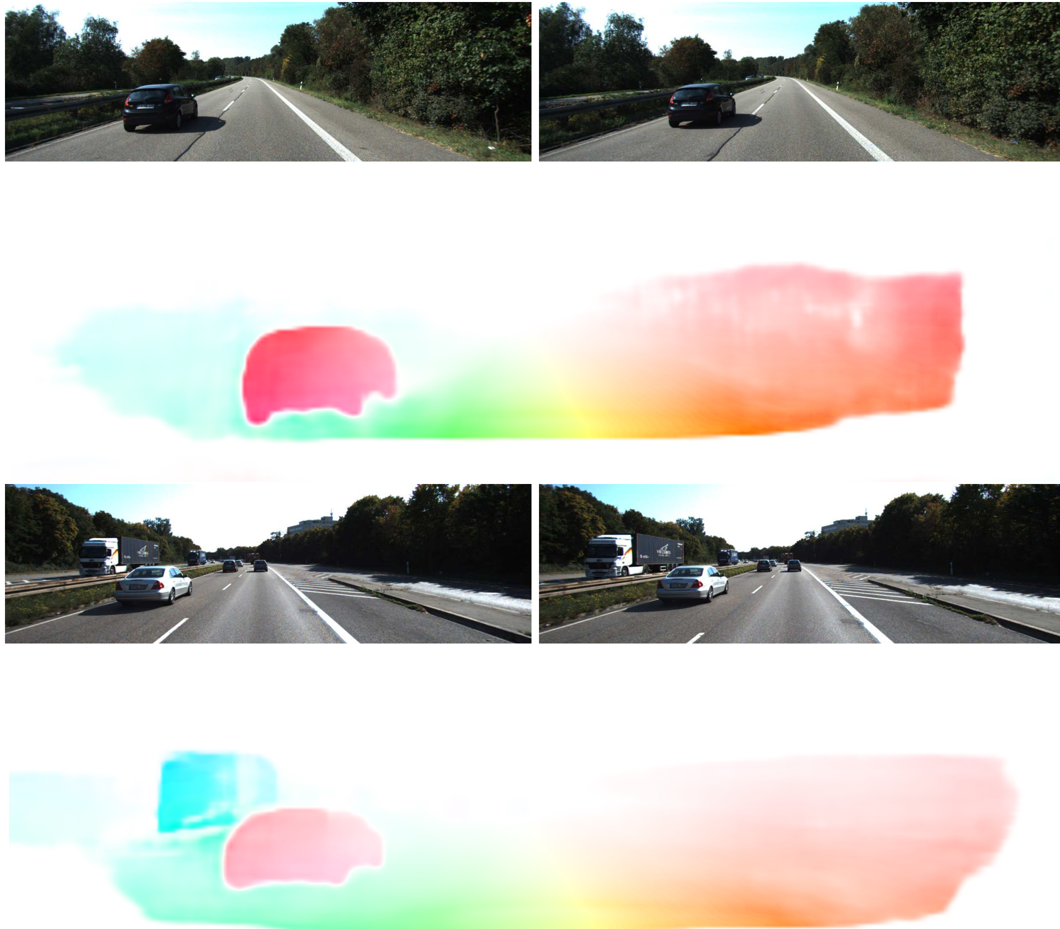
**Slika 6.2:** Grafički prikaz izlaza modela na skupu za testiranje.

## 6.4. Budući radovi i mogućnost za napredak

Rezultati pokazuju kompleksnost problema detektiranja optičkog toka. Model pokušava locirati izrazito male pomake za svaki piksel. Do neke mjere uspijeva na pikselima vozila, najviše na slikama gdje vozilo odudara od svoje pozadine, ali probleme mu stvaraju pomaci velikog broja sličnih piksela.

Rezultati evaluacije su dobiveni na skupu za učenje što znači da bi se bolji rezultati mogli ostvariti dužim učenjem. Međutim, prilikom učenja na skupu podataka KITTI primjetio sam značajno usporenje pada gubitka. Iz tog razloga, upitno je koliko bi duže učenje na navedenom skupu pomoglo. Bitno je napomenuti da kod prvotnog učenja na skupu podataka Flying Chairs i ugađanja na skupu podataka KITTI nije primjenjen nijedan oblik transformacije što zasigurno pridonosi lošijim rezultatima.

Rezultate možemo poboljšati i boljim modelom. Trenirani model nije koristio gusto povezane slojeve što povećava kompleksnost modela. U radu Sun et al. (2017)

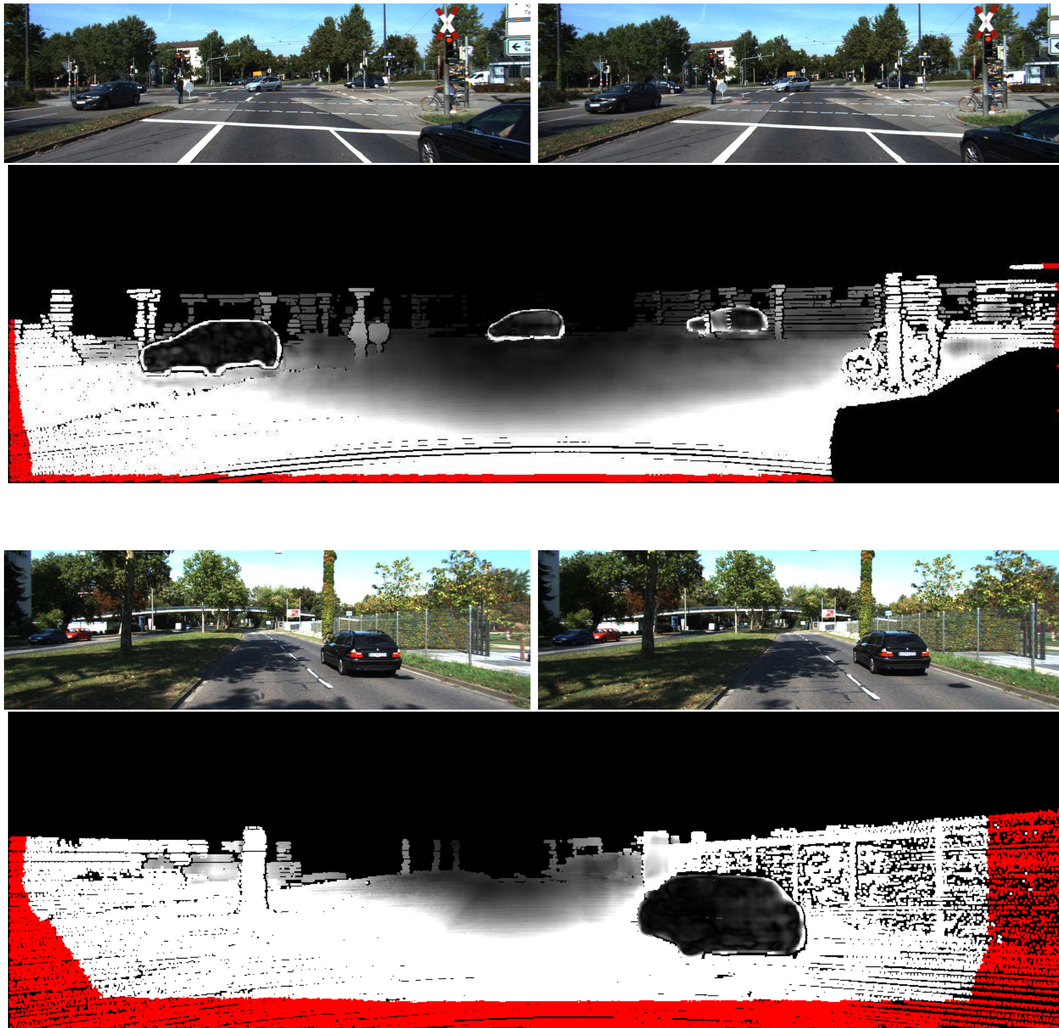


**Slika 6.3:** Grafički prikaz izlaza modela na skupu za testiranje.

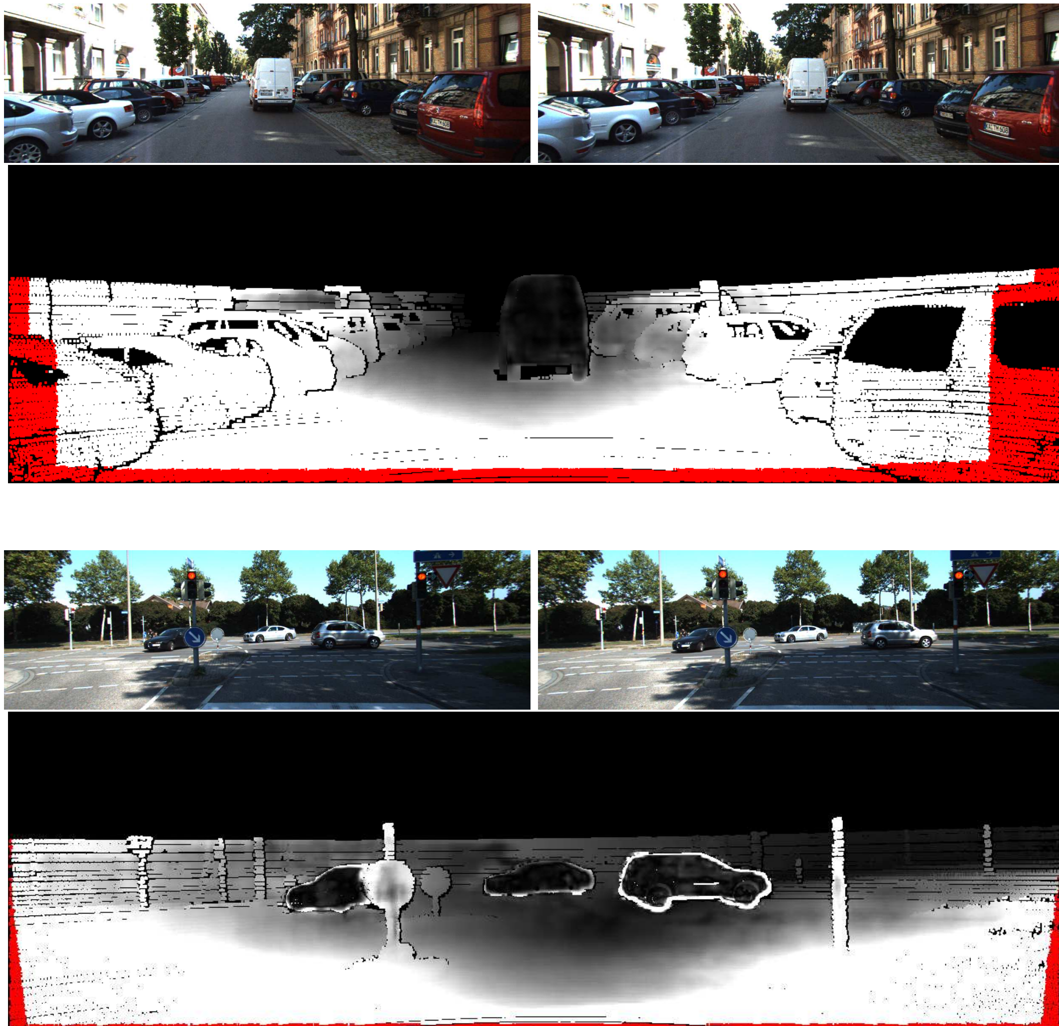
je navedeno kako takvi modeli postižu bolje rezultate. Uz to, kompleksnost modela moguće je povećati i rezidualnim vezama što bi predstavljalo pozivanja kontekstualne mreže za izlaz svake razine.

**Slika 6.4:** Grafički prikaz izlaza modela na skupu za testiranje.

**Slika 6.5:** Grafički prikaz izlaza modela na skupu za testiranje. Možemo primjetiti lošije rezultati kod slika s kompleksnijim pozadinama i autima koji nisu jasno vidljivi.



**Slika 6.6:** Grafički prikaz grešaka u izlazima modela.



**Slika 6.7:** Grafički prikaz grešaka u izlazima modela.

# 7. TensorRT i CUDA

Duboki modeli se sastoje od izrazito velikog broja operacija koje sadrže ogroman broj parametara. Većina operacija su relativno jednostavne za izvođenje, ali postoji određen broj podataka koje procesorska jedinica može procesirati u isto vrijeme. Ta brojka za CPU nije velika i zbog toga je izvođenje modela na takvim uređajima često sporo.

U nekim računalima i mobilnim uređajima nalazi se i grafička procesorska jedinica (GPU). GPU često ima jednu svrhu, a to je obavljanje brojnih operacija vezanih za grafički prikaz. Takve operacije su jednostavne, ali u jednom trenutku se primjenjuju na ogromnom broju podataka.

Možemo primijetiti sličnosti između grafičkog prikaza i dubokih modela. Oba rješenja se sastoje od ogromnog broja manjih, jednostavnih matematičkih operacija. Kod algoritama korištenih za grafički prikaz želimo primijeniti određenu funkciju na što veći broj piksela dok kod dubokih modela želimo izračunati izlaze za što veći broj izlaza. Iz tog razloga, duboko učenje u velikoj mjeri koristi GPU kao glavnu procesorsku jedinicu.

## 7.1. CUDA

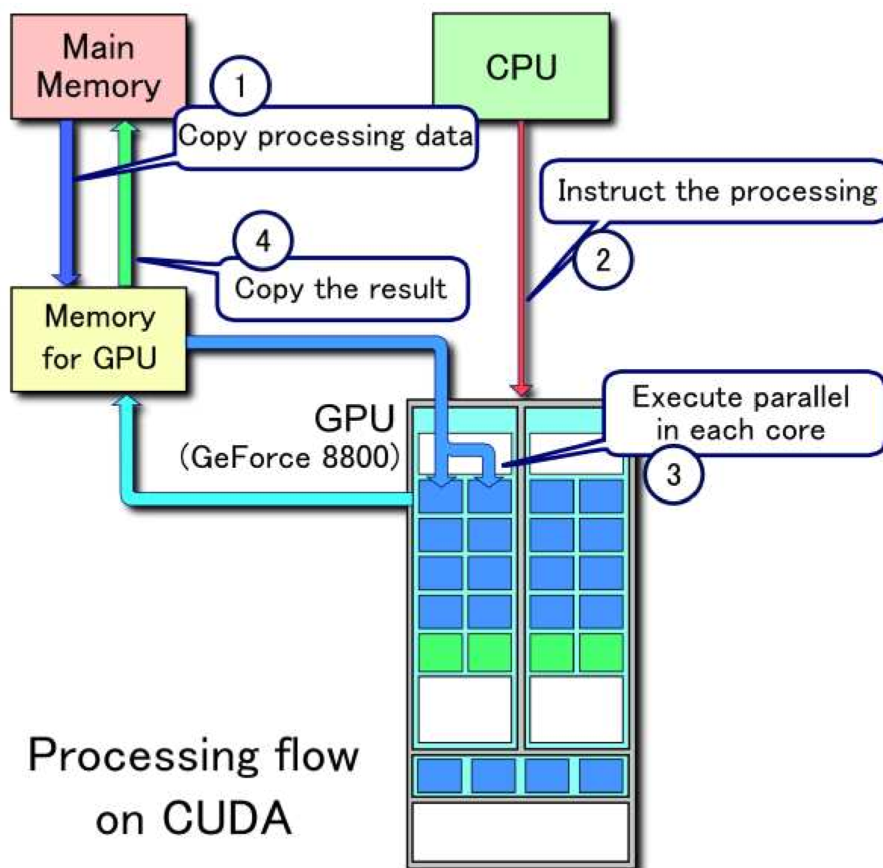
Često je zahtjevno razviti program koji ostvaruje potpuni potencijal određenih uređaja, pogotovo kompleksnijih uređaja kao što je GPU. Kako bi olakšala razvoj na svojim GPU uređajima, Nvidia razvija platformu i sučelje za paralelno računarstvo nazvanog CUDA (*engl. Compute Unifed Device Architecture*). CUDA pokušava omogućiti inženjerima korištenje grafičkih procesorskih jedinica u proizvoljne svrhe. Time su se pokušali udaljiti od specijalizirane upotrebe GPU-ova.

CUDA-u je moguće koristiti u više programskih jezika, ali najpopularniji su zasigurno C i C++. Postoje razne biblioteke koje definiraju česte operacije na GPU-ovima što uvelike olakšava budući razvoj.

Međutim, CUDA-u nije moguće koristiti na svim GPU uređajima. Moguće ih koristiti samo na GPU-ovima razvijenim od strane Nvidije. Uz to, GPU mora podržavati

CUDA-u što je navedeno za svakih njihov GPU. Međutim, moguće je pretvoriti kod napisan koristeći CUDA-u u neki drugi format koji podržava veći broj GPU uređaja. Primjer toga je HIP okruženje koje omogućava pisanje i prilagodbu koda za velik broj GPU uređaja razvijenih od strane Nvidije i AMD-a.

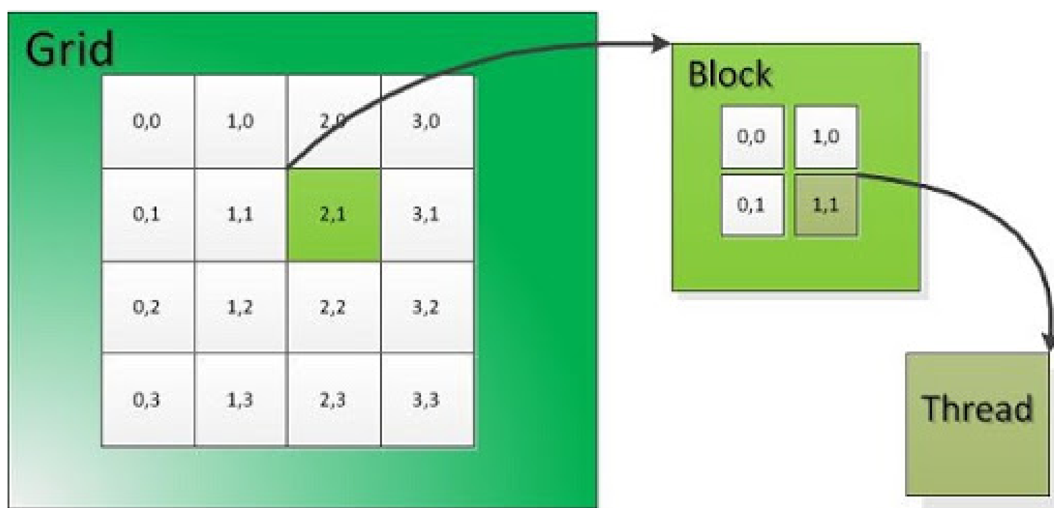
Na slici 7.1 prikazan je primjer rada aplikacije koja koristi CUDA platformu. Možemo primijetiti vrlo bitan detalj, a to je kopiranje iz glavne, radne memorije u memoriju GPU uređaja i obrnuto. Iako GPU omogućava izrazito brzo izvođenja velikog broja operacija, njegova namjena nije za ostale, kompleksnije operacije. Iz tih razloga veliki dio operacija se i dalje obavlja na CPU uređajima, a operacije koje su jednostavne i potrebno je izvesti na ogromnom broju podataka šalju se na GPU. Takvo slanje podataka predstavlja usko grlo u takvim aplikacijama.




**Slika 7.1:** Prikaz rada koristeći CUDA-u. **Izvor:** `https://en.wikipedia.org/wiki/CUDA`

### 7.1.1. Stvaranje dretvi

CUDA zahtjeva specifičan način procesiranja podataka gdje se pokušava sa što većim brojem dretvi obraditi veći broj podataka. Za svaku funkciju koja se poziva na CUDA uređaju može se definirati broj dretvi. Dretve se grupiraju u blokove, a blokovi se grupiraju u mrežu (*engl. grid*). Kao i ulazi koji se obrađuju na CUDA uređajima mreža i blokovi se mogu definirati u više dimenzija (najviše u 3 dimenzije). Na slici 7.2 je prikazana takva struktura.




**Slika 7.2:** Prikaz strukturiranja CUDA dretvi. Dretve se smještaju u blokove koji se smještaju u mrežu blokova. **Izvor:** `http://15418.courses.cs.cmu.edu/spring2013/article/11`

Unutar CUDA funkcije svaka dretva mora znati koji podatak obrađuje. Iz tog razloga, unutar takvih funkcija dostupni su indeksi za mrežu, blokove i dretvu koji jedinstveno lociraju dretvu u kojoj se trenutno nalazi funkcija. Indeksi mreža, blokova i dretvi mogu biti definirani s više dimenzija, ovisno kako su pojedini dijelovi definirani. Na primjer, ako su dretve smještene u strukturu definiranoj s dvije dimenzije, indeks određene dretve unutar bloka definiran je s dvije dimenzije, x i y. Isto vrijedi za blokove i mreže. Dimenzije struktura u kojima su smještene mreže, blokovi i dretve međusobno su neovisne. Tako možemo definirati mrežu i blokove kao jednodimenzionalne strukture, a dretve kao dvodimenzionalne. Slika 7.3 prikazuje stvaranje dretvi u više dimenzija.

## 7.2. TensorRT

TensorRT je skup alata za razvoj dubokih modela visokih performansi na Nvidijinim grafičkim procesorskim jedinicama. Trenutno podržava dva programska jezika, C++ i Python. Postoje razni radni okviri za treniranje dubokih modela. Međutim, takvi radni okviri ne stavljaju fokus na optimizaciju samog zaključivanja.

Postoji velika potreba za dubokim učenjem u raznim industrijama. Kako potreba raste, nastaje sve veći naglasak na točnost i performanse samih modela. Mnoge industrije zahtijevaju pokretanje aplikacija na ugradbenim uređajima ili sličnim manjim, specijaliziranijim uređajima. Postoje aplikacije koje imaju ogroman broj zahtjeva vezanih za sigurnost i u takvim aplikacijama je manje kašnjenje (*engl. latency*) i veći protok (*engl. throughput*) izuzetno bitan. TensorRT je rješenje za lakše pripremanje istreniranih modela za produkciju upravo za takve slučajeve.

TensorRT nije zamjena za radne okvire za treniranje kao što su TensorFlow, Caffe i PyTorch, nego upravo suprotno. Jedan od njegovih ciljeva je dobra suradnja s takvim radnim okvirima. Iz tog razloga, postoji velika podrška u obliku alata za izuzetno lagano prebacivanje istreniranih modela u popularnijim radnim okvirima u optimizirane modele, prilagođene raznim Nvidia uređajima.

Na slici 7.4 možemo vidjeti vizualizaciju samog procesa izgradnje optimiziranog stroja za zaključivanje. Proces se sastoji od 3 bitne faze:

- definiranje ulaznog modela
- optimiziranje modela
- izgradnja stroja za zaključivanje

### 7.2.1. Optimizacija modela

Nakon što definiramo model, potrebno je provesti postupak optimizacije. Kao izlaz dobivamo optimizirani stroj (*engl. engine*) za zaključivanje. Takav stroj je zasebna optimizirana cjelina koja je ekvivalentna ulaznom modelu. Iz tog razloga, prethodne korake više nije potrebno ponavljati, a stroj koji smo dobili kao izlaz možemo koristiti u produkcijskom okruženju.

Kod izgradnje stroja provode se razne operacije optimizacije koje mogu biti i specifične za određene uređaje. Primjer takvih optimizacija su:

- Izbacivanje slojeva čiji se izlazi ne koriste
- Izbacivanje operacija koje se ne izvode

– Spajanje konvolucije i primjene aktivacijskih funkcija
– Spajanje operacija s dovoljno sličnih parametara i jednakim ulaznim tenzorima
– Stapanje slojeva spajanja (*engl. concatenation layer*) povezivanjem izlaza sloja s točnom eventualnom destinacijom

Za vrijeme izgradnje slojeva također može doći do prilagođavanja preciznosti težina. Na slici 7.5 vidimo primjer ispisa za vrijeme optimizacije modela kod izgradnje stroja.

### 7.2.2. Komponente aplikacije za izgradnju stroja za zaključivanje

TensorRT sadrži biblioteku za lakše definiranje i izgradnju optimiziranih strojeva za zaključivanje. Takva aplikacija, kojoj je cilj proizvesti opitmizirani stroj za zaključivanje, sastoji se od određenih komponenti.

**Definicija mreže (*engl. Network Definition*)** Kako bismo izgenerirali optimizirani stroj za zaključivanje, potrebno je definirati model. Moguće je definirati ulazne i izlazne tenzore, ali i određene slojeve koje je moguće konfigurirati kroz definirano sučelje. Uz to, moguće je definirati vlastite operacije kroz poseban tip sloja, sloja dodatka (*engl. Plugin layer*). Kroz takve slojeve moguće je definirati operacije koje nisu direktno podržane od strane TensorRT biblioteke. Važno je za napomenuti da je moguće koristiti razne CUDA operacije u navedenim slojevima.

**Konfiguracija izgraditelja (*engl. Builder Configuration*)** Detalji vezani za izgradnju stroja sadržani su u konfiguraciji izgraditelja. Preko konfiguracije moguće je definirati optimizacijske profile, najveću veličinu radnog prostora, najmanja prihvatljiva preciznost i ostale slične dijelove izgradnje stroja.

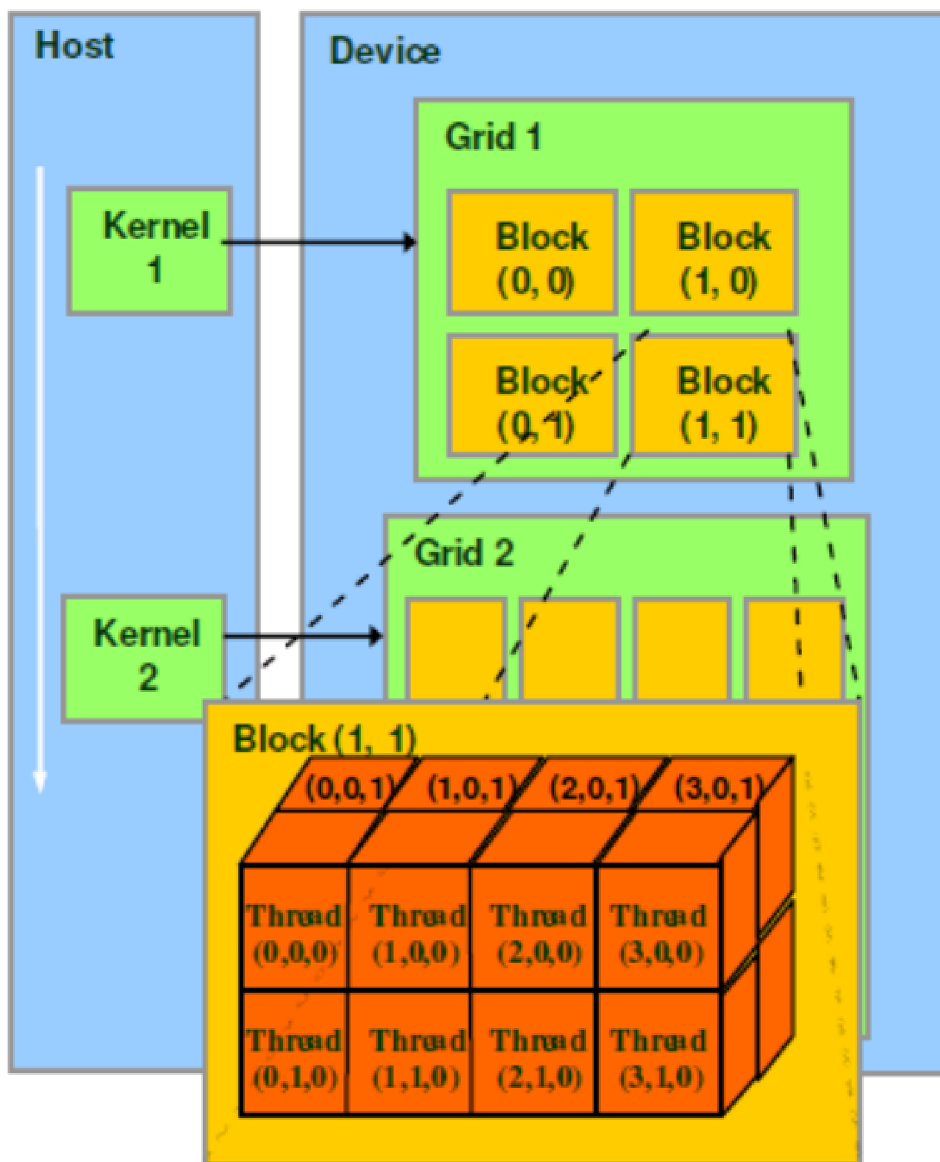
**Izgraditelj (*engl. Builder*)** Izgraditelj na temelju definicije mreže i konfiguracije stvara optimizirani stroj za zaključivanje.

**Stroj za zaključivanje (*engl. Inference Engine*)** Kao izlaz aplikacije dobivamo stroj za zaključivanje koji je optimizirani ekvivalent definiranom modelu. Podržava sinkrono i asinkrono izvođenje i profiliranje. Jedan stroj može sadržavati više konteksta izvođenja, što omogućava korištenje jednog skupa treniranih parametara kroz više izvođenja. Drugim riječima, omogućeno je efikasno izvođenje zaključivanja na više grupa podataka.

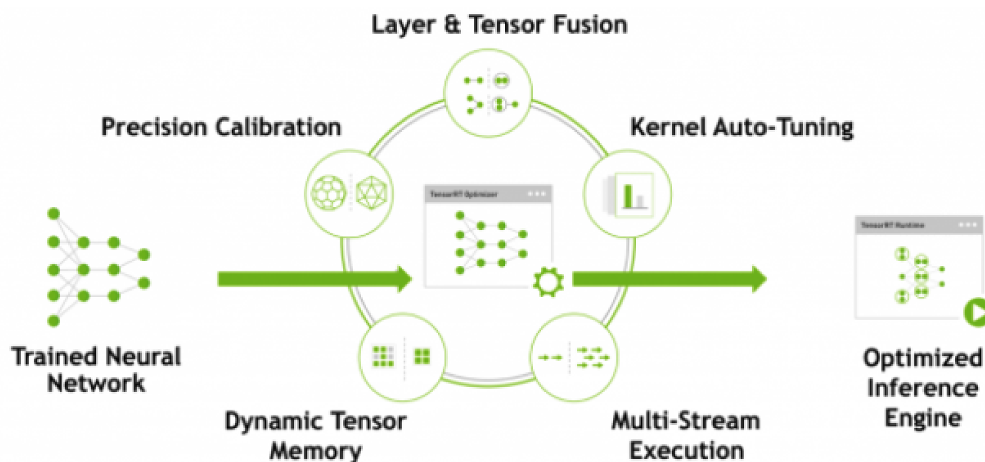
### 7.2.3. TensorRT i parseri

TensorRT olakšava prebacivanje modela istreniranih u određenim radnim okvirima definiranjem parsera. Parseri automatski učitavaju mrežu u određenom formatu, parsiraju i prevode u oblik prepoznatljiv TensorRT-u. Najpoznatiji parseri su Caffe Parser, UFF Parser i ONNX Parser.

UFF parser je namijenjen za TensorFlow grafove. Proces se sastoji od zamrzavanja TensorFlow grafa i prebacivanje u UFF format koji se učitava i parsira u TensorRT aplikaciji. ONNX Parser učitava i parsira ONNX format. ONNX je format namijenjen za modele strojnog učenja. Postoji podrška za razne radne okvire i generiranje njihovih modela u takav format. Iz tog razloga, ONNX format može se smatrati univerzalnim međukorakom za generiranje TensorRT modela. Međutim, njegova moć se oslanja na jakoj podršci za svaki radni okvir. Radni okviri su vjerojatno implementacijski različita. Iz tog razloga, potrebno je definirati zasebne alate za pretvaranje iz izlaznih formata svakog radnog okvira. Uz to, redovito izlaze novije verzije radnih okvira. Kako bi ONNX i dalje pravilno radio, potrebno je nadograđivati takve alate za pretvaranje u ONNX format.




**Slika 7.3:** Prikaz definiranja dretvi koristeći višedimenzionalne strukture. **Izvor:** `https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf`




**Slika 7.4:** Prikaz TensorRT procesa izgradnje stroja za zaključivanje. U sredini slike prikazan je proces optimiziranja modela koji kao ulaz prima prethodno definirani model. Nakon optimizacije izgrađuje se stroj za zaključivanje što je ujedno i izlaz cijelog procesa. **Izvor:** `https://developer.nvidia.com/tensorrt`

```
[05/01/2020-04:44:35] [I] [TRT] ---------------- Timing dc_conv6(14)
[05/01/2020-04:44:35] [I] [TRT] Tactic 1363534230700867617 time 0.105472
[05/01/2020-04:44:35] [I] [TRT] Tactic 1642270411037877776 time 0.080896
[05/01/2020-04:44:35] [I] [TRT] Tactic 3146172331490511787 time 0.111616
[05/01/2020-04:44:35] [I] [TRT] Tactic 3528302785056538033 time 0.115712
[05/01/2020-04:44:35] [I] [TRT] Tactic 5443600094180187792 time 0.09216
[05/01/2020-04:44:35] [I] [TRT] Tactic 5552354567368947361 time 0.089088
[05/01/2020-04:44:35] [I] [TRT] Tactic 5824828673459742858 time 0.104448
[05/01/2020-04:44:35] [I] [TRT] Tactic -6618588952828687390 time 0.103424
[05/01/2020-04:44:35] [I] [TRT] Tactic -6362554771847758902 time 0.089088
[05/01/2020-04:44:35] [I] [TRT] Tactic -2701242286872672544 time 0.106496
[05/01/2020-04:44:35] [I] [TRT] Tactic -2535759802710599445 time 0.079872
[05/01/2020-04:44:35] [I] [TRT] Tactic -675401754313066228 time 0.086016
[05/01/2020-04:44:35] [I] [TRT]
[05/01/2020-04:44:35] [I] [TRT] ---------------- Timing dc_conv6(1)
[05/01/2020-04:44:35] [I] [TRT] Tactic 0 time 0.177152
```

**Slika 7.5:** Primjer ispisa u koraku optimizacije modela kod izgradnje stroja

# 8. Implementacija PWCNet modela u TensorRT-u

U svrhu ovog rada za treniranje PWCNet modela koristio sam Tensorflow 2 radni okvir. Kako bih isprobao mogućnosti implementacije navedenog modela
u TensorRT-u koristio sam Nvidia DRIVE PX 2 računalnu platformu koja se sastoji od Tegra X2 čipova i Nvidijinih GPU-ova iz Pascal generacije. Na platformi je instalirana TensorRT biblioteka verzije 3.0.4. Važno je za napomenuti da navedena verzija TensorRT-a ne sadrži mnoge dijelove dubokih modela koje se pojavljuju u PWCNet modelu, a implementirani su u kasnijim verzijama TensorRT-a.

Kao što je navedeno ranije u radu, postoje mnogi parseri koji parsiraju modele zapisanih u određenom formatu. Međutim, odlučio sam se za vlastitu implementaciju modela zbog boljeg upoznavanja sa samim TensorRT alatima i CUDA bibliotekama. Uz to, prebacivanje u UFF format nije službeno podržano od strane Tensorflowa s novom verzijom (2.0) koju sam koristio za treniranje modela. Definirao sam aplikaciju u kojoj je u potpunosti definiran PWCNet model, a kao izlaz aplikacija dobiva se optimizirani stroj za zaključivanje.

## 8.1. Učitavanje Tensorflow težina

Prvi problem koji je bilo potrebno riješiti kod implementacije modela u TensorRT-u je učitavanje težina naučenih u Tensorflow 2 radnom okviru. Spremanje trenutnih težina Tensorflow modela koji su implementirani koristeći Keras sučelje ostvarujemo pozivom `save_weights` metode. Kao argument metodi šaljemo ime datoteke s ekstenzijom `.h5` kako bi težine spremili u HDF5 format.

HDF5 je format koji je specijaliziran za veće količine podataka. Međutim, htjeli bismo što jednostavniji format koji možemo lagano učitati i parsirati u TensorRT aplikaciji. Težine spremljene u HDF5 formatu sam učitao koristeći Python biblioteku `h5py`. Težine sam zapisao slijedno, a svaki skup težina sam zapisao u vlastitom format. Svaki

skup težina je definiran redom s imenom, tipom težina, broj težina i pojedinačne vrijednosti težina zapisane u heksadecimalnom formatu. Oblik težine je identičan za oba radna okvira s iznimkom jezgri konvolucijskog sloja. U Tensorflowu jezgre su spremljene u HWCO formatu, gdje H predstavlja visinu jezgre, W širinu jezgre, C dubinu ulaza, a O dubinu izlaza. TensorRT očekuje težine za konvolucijske slojeve u OCHW pa je bilo potrebno transponirati težine u Tensorflow formatu prije spremanja za TensorRT aplikaciju.

U TensorRT aplikaciji na početku svake izgradnje stroja za zaključivanje poziva se učitavanje težina iz vlastitog definiranog formata. Težine se spremaju u `std::unordered_map<std::string, nvinfer1::Weights>` gdje ključ predstavlja ime skupa težina.

## 8.2. Definiranje modela

Prilikom definiranja modela bilo je potrebno analizirati model kako bi se locirale komponente za koje je potrebna vlastita implementacija. Za PWCNet model bilo je potrebno definirati funkciju koja računa volumen cijene i sloj izobličavanja. Također, nedostajala je implementacija propusne zglobnice. Svaki dio koji je definiran kao novi sloj definiran je kao `IPlugin` koji je dio TensorRT biblioteke.

Mnoge komponente sam definirao koristeći CUDA biblioteku. Kao što je navedeno ranije u radu, CUDA obrađuje veći broj podataka stvaranjem većeg broja dretvi strukturiranih u mreže i blokove. Često broj dretvi ne može pokriti sve podatke koje obrađujem pa je potrebno prilikom završetka obrade jednog podatka prijeći na prvi slobodni. Petlje prisutne u klasičnom programiranju nisu prikladne za GPU uređaje. U aplikaciji za GPU uređaje možemo pronaći poseban oblik petlji nazvanog *Grid-Stride* petlje. Takvu petlju sam definirao macro naredbom:

```
#define CUDA_KERNEL_LOOP( i, n ) \
    for ( int i = blockIdx.x * blockDim.x + threadIdx.x; \
          i < ( n ); \
          i += blockDim.x * gridDim.x )
```

U petlji možemo primijetiti kako se trenutni indeks podataka koji se obrađuje u određenoj dretvi označen varijablom `i` izračunava na temelju trenutnih indeksa blokova i dretvi. Nakon što obrađivanje završi prelazi se na sljedeći podatak koji se dobije uvećavanjem trenutnog indeksa za ukupan broj dretvi koji dobijemo iz dimenzija strukture koja sadrži dretvi.

### 8.2.1. Aktivacijska funkcija

Jedina aktivacijska funkcija koja se koristi u PWCNet modelu je propusna zglobnica. Korištena verzija TensorRT-a ne sadrži definiciju takve aktivacijske funkcije pa je bila potrebna vlastita implementacija. Propusna zglobnica je jednostavnija operacija koja na ulazu prima samo jedan podatak pa je i sama definicija takve funkcije jednostavna, a u potpunosti se izvodi na GPU uređaju. Na slici 8.1 je prikazana implementacija propusne zglobnice prilagođena CUDA uređajima. Kako bi tako definiranu funkciju mogli u definiciji modela, napisao sam sloj koji na sve svoje ulaze primjenjuje propusnu zglobnicu.

```
__global__ void LeakyRelu( const int n, const float* in, float* out, float alpha )
{
    CUDA_KERNEL_LOOP(index, n)
    {
        out[ index ] = in[ index ] < 0.0f ? in[ index ] * alpha : in[ index ];
    }
}
```

**Slika 8.1:** Isječak koda koji sadrži implementaciju propusne zglobnice.

### 8.2.2. Konvolucijski slojevi

Za konvolucijske slojeve postoji implementirano sučelje `IConvolutionLayer` pa za njih nije bilo potrebno definirati vlastitu implementaciju. Međutim, implementacija sloja u korištenoj verziji TensorRT-a sadrži određena ograničenja. Najbitnije ograničenje je definiranje dopune. Dopuna može biti samo simetrična što nije garantirano za slojeve prisutnih u PWCNet modelu. Iz tog razloga, bilo je potrebno definirati funkciju koja računa dopunu za širinu i visinu. Ulaz se dopunjava prije svakog konvolucijskog sloja koristeći `IPaddingLayer`.

Naravno, potrebno je izračunati dopunu koja je identična onoj primijenjenoj u Tensorflow radnom okviru za parametar `SAME`. Iz Tensorflow implementacije koja je dostupna javno pronašao sam način na koji računaju dopunu za navedeni parametar. Na slici 8.2 je prikazana funkcija koja računa dopune za ulaze konvolucijskog sloja na temelju zadanih parametara. Funkcija na temelju dimenzije ulaza, veličine koraka, dimenzije jezgre i dilatacije računa dopunu za obje prostorne dimenzije.

```
std::pair< nvinfer1::DimsHW, nvinfer1::DimsHW > getPadding
(
    nvinfer1::Dims const& inputDimension,
    nvinfer1::DimsHW const& strides,
    nvinfer1::DimsHW const& kernelDimension,
    nvinfer1::DimsHW const& dilation
)
{
    assert( inputDimension.nbDims == 3 );
    auto const calculatePadding = [ & ]( int size, int stride, int kernelSize, int dilation  ) noexcept -> int
    {
        int effectiveFilterSize = ( kernelSize - 1 ) * dilation + 1;

        int outSize = ( size + stride - 1 ) / stride;

        int totalPadding = ( outSize - 1 ) * stride + effectiveFilterSize - size;

        totalPadding = totalPadding > 0 ? totalPadding : 0;

        return totalPadding;
    };

    int totalHeightPadding = calculatePadding( inputDimension.d[ 1 ], strides.h(), kernelDimension.h(), dilation.h() );
    int totalWidthPadding = calculatePadding( inputDimension.d[ 2 ], strides.w(), kernelDimension.w(), dilation.w() );

    int preHeightPadding = totalHeightPadding / 2;
    int preWidthPadding = totalWidthPadding / 2;

    return
    {
        nvinfer1::DimsHW{ preHeightPadding, preWidthPadding },
        nvinfer1::DimsHW{ totalHeightPadding - preHeightPadding, totalWidthPadding - preWidthPadding }
    };
}
```

**Slika 8.2:** Isječak koda u kojem se računa dopuna za širinu i visinu.

### 8.2.3. Slojevi transponirane konvolucije

Slojeve transponirane konvolucije nije bilo potrebno definirati jer u TensorRT-u postoji implementacija. Sloj je nazvan `IDeconvolutionLayer` i sadrži identična ograničenja kao konvolucijski sloj. Međutim, za transponirane konvolucije nije bilo potrebno definirati funkciju koja računa dopunu nego se primjenjuje dopuna od jednog piksela simetrično na visinu i širinu.

### 8.2.4. Sloj volumena cijene

Volumen cijene je uz sloj izobličavanja dio kojeg je bilo potrebno ručno implementirati. Ne sadrži parametre, a za određene dijelove kao što su dopuna i spajanje postoje implementacije. Komponenta koja računa volumen cijene je zahtijevala vlastitu implementaciju. Za svaki piksel se odrađuju identične operacije. Prva operacija je dohvaćanja dijela tenzora (*engl. slice*). U korištenoj verziji TensorRT-a ne postoje implementacije kojom se može dobiti dio tenzora. Na slici 8.3 prikazana je implementacija *slice* operacije. Ostatak izračuna se sastoji od jednostavnog množenja tenzora i smanjivanjem (*engl. reduce*) broja elemenata za dimenziju dubine srednjom vrijednošću. Obje operacije postoje u cuDNN biblioteci. cuDNN je biblioteka koja sadrži implemen-

taciju primitiva prisutnih u dubokim modelima prilagođenih GPU uređajima. Iz tog razloga, ostatak implementacije se svodi na pravilno korištenje dviju cuDNN funkcija, `cudnnOpTensor` i `cudnnReduceTensor` što je prikazano u isječku na slici 8.4.

```
void getSlice
(
    const float* input,
    float* output,
    nvinfer1::DimsCHW const & inputDims,
    nvinfer1::DimsCHW const & sliceStart,
    nvinfer1::DimsCHW const & sliceSize,
    int batchSize,
    cudaStream_t stream
)
{
    nvinfer1::Dims strides = getStrides( { 4, { batchSize, inputDims.c(), inputDims.h(), inputDims.w() } } );
    int const batchOffset = strides.d[ 0 ];
    int const channelOffset = strides.d[ 1 ];
    int const rowOffset = strides.d[ 2 ];

    const float* batchStart{ input };
    float* outputStart{ output };
    for ( int batch = 0; batch < batchSize; ++batch )
    {
        const float* channelStart{ batchStart };
        for ( int channel = 0; channel < sliceSize.c(); ++channel )
        {
            const float* rowStart{ channelStart + sliceStart.h() * rowOffset };
            for ( int row = 0; row < sliceSize.h(); ++row )
            {
                CHECK( cudaMemcpyAsync
                (
                    outputStart,
                    rowStart + sliceStart.w(),
                    sliceSize.w() * sizeof(float),
                   cudaMemcpyDeviceToDevice,
                    stream
                ))
                rowStart += rowOffset;
                outputStart += sliceSize.w();
            }
            channelStart += channelOffset;
        }
        batchStart += batchOffset;
    }
}
```

**Slika 8.3:** Isječak koda s implementacijom operacije izrezivanja (*engl. slice*)

### 8.2.5. Sloj izobličavanja

Zasigurno najkompleksniji sloj za implementirati je bio sloj izobličavanja. U modelu je bilo potrebno implementirati funkciju izobličavanja čija bi funkcionalnost trebala biti identična odgovarajućoj funkciji u okviru Tensorflow. Sloj prvo poziva funkciju *meshgrid* s dimenzijama ulaza kao argument. Od dobivene rešetke oduzima se izračunati tok kako bi dobili novu lokaciju za svaki piksel. Kako je tok definiran s realnom vrijednošću, velike su šanse da nećemo dobiti cijeli broj. Iz tog razloga potrebno je primijeniti bilinearnu interpolaciju na 4 okolna piksela. Vrijednosti koje izlaze izvan

```
CHECK( cudnnOpTensor( mCudnn, mMultiplyDesc,
    &Consts::kOne, mMultiplyOperandDesc, pdst,
    &Consts::kOne, mMultiplyOperandDesc, inputs[ 0 ],
    &Consts::kZero, mMultiplyOperandDesc, pdst ) )
CHECK( cudnnReduceTensor( mCudnn, mMeanDesc, nullptr, 0, pdst + inputSize, inputSize * sizeof( float ),
                          &Consts::kOne, mMultiplyOperandDesc, pdst,
                          &Consts::kZero, mOutputDesc, *outputs ) )
```

**Slika 8.4:** Isječak koda s ostatakom izračuna volumena cijene.

okvira slike postavljaju se na rubne vrijednosti. Sve dijelove je bilo potrebno ručno implementirati.

Implementacija *meshgrid* funkcije je vrlo jednostavna. Međutim, sastoji se od velikog broja kopiranja podataka sa *host* uređaja na GPU uređaj što predstavlja usko grlo u implementaciji.

Bilinearna interpolacija je kompleksna funkcija i može se podijeliti na ključne dijelove. Prvo je potrebno izračunati pomoćne varijable `floor`, `ceil` i `alpha` za svaki kanal ulaza što je prikazano na slici 8.5. Funkcije koje se koriste za izračun varijabli sam implementirao koristeći CUDA biblioteku kako bi se mogle koristiti na GPU uređaju.

Nakon što smo izračunali pomoćne varijable, možemo ih iskoristiti za izračunavanje koordinata. Na slici 8.6 je prikazan dio s izračunom koordinata. Važno je za napomenuti da se kopiraju dobivene koordinate s GPU uređaja na *host* uređaj jer se kod dohvaćanja koriste njihove vrijednosti na *host* uređaju.

Zadnji korak je dohvaćanje za svaki kanal piksele koji su definirani koordinatama i primjena operacija prikazanih na slici 8.7. Dohvaćanje koordinata se svodi na jednostavno kopiranje podataka na uređaju na temelju dobivenih koordinata. Na dohvaćenim podatacima primjenjuju se operacije koje interpoliraju dobivene točke. Funkcije koje obavljaju te operacije također sam ručno definirao koristeći CUDA biblioteku zbog jednostavnijeg korištenja takvih naredbi. Alternativa vlastitoj implementaciji je bila korištenje cuDNN biblioteke.

## 8.3. Rezultati

Kod sam izgradio i uspješno pokrenuo na Drive PX2 uređaju. Svi rezultati su izmjereni i dobiveni na navedenom uređaju.

Samo prevođenje i povezivanje koda definirano je uz pomoć alata CMake. Kako bi dobili izvršnu datoteku potrebno je pokrenuti CMake generiranje s dostupnim generatorom i pozvati naredbu `cmake` s `build` parametrom i direktorijem koji sadrži

```
auto const getAlpha = [ & ]( int const dimensionSize, float* input, float* alpha, float* floor, float* ceil )
{
    float max = dimensionSize - 2.f;
    float min = 0.f;

    clipTensor( channelSize, input, floor, min, max );
    roundToIntTensor( channelSize, floor, floor );
    addValueToTensor( channelSize, floor, ceil, 1.f );

    subtractTensors( channelSize, input, floor, alpha );
    clipTensor( channelSize, alpha, alpha, 0.f, 1.f );
};

getAlpha( mFlowDimensions.h(), batchedGrid, alpha0, floor0, ceil0 );
getAlpha( mFlowDimensions.w(), batchedGrid + channelSize, alpha1, floor1, ceil1 );
```

**Slika 8.5:** Isječak koda bilinearne interpolacije s izračunom pomoćnih varijabli

izgenerirane datoteke za izgradnju.

Težine modela je potrebno pretvoriti u format opisan u 8.1. U slučaju alternativnog formata potrebne je izmijeniti implementaciju `loadWeights` metode u klasi `PWCNet`.

Ulazi se definiraju u metodi `processInput` u klasi `PWCNet`. Modelu je potrebno definirati dva ulaza jednakih dimenzija.

Pomoću klase `PWCNetParams` moguće je definirati određene konfiguracije kao što su naziv datoteke s veličinima, dimnenzije ulaza, broj razina piramide.

Pokretanjem modela s identičnim težinama i ulazima dobivenih su identični rezultati s greškom reda veličine $10^{-5}$ koja je nastala vjerojatno zbog različitog prikaza varijabli.

Na slikama dimenzije $64 \times 64$ vrijeme izvođenja je bilo $3442$ms što je približno $3.5$s.

## 8.4. Diskusija

Zbog veće mjere korištenja metoda dubokog učenja, nije problematično implementirati PWCNet model za ugradbene uređaje. Metode koje su specifične za model su jednostavne i lako ih je ručno implementirati, a u novijim verzijama TensorRT-a mnoge metode koje je bilo potrebno implementirati u korištenoj verziji nije potrebno implementirati. Time je sama implementacija još jednostavnija. Uz to, moguće je koristiti parsere čime bi se implementacija PWCNet modela svela na prebacivanje u zadani format, a metode koje je potrebno implementirati ovise u velikoj mjeri o verziji TensorRT alata koji se koriste.

Vrijeme je izrazito veliko i mjerenjima je vidljivo da najviše na vrijeme utječe ve-

```
float* linearCoordinates{ bottomRight + channelSize };
auto const gather = [ & ]( const float* rowIndex, const float* columnIndex, float* outputCoordinates )
{
    float imageWidth = mImageDimensions.w();
    CHECK( cudnnOpTensor
    (
        mCudnn,
        mFlatAddOp,
        &imageWidth,
        mFlowChannelFlatDesc,
        rowIndex,
        &Consts::kOne,
        mFlowChannelFlatDesc,
        columnIndex,
        &Consts::kZero,
        mFlowChannelFlatDesc,
        linearCoordinates
    ))

    cudaMemcpyAsync
    (
        outputCoordinates,
        linearCoordinates,
        channelSize * sizeof( float ),
        cudaMemcpyDeviceToHost,
        stream
    );
};
gather( floor0, floor1, topLeftCoordinates );
gather( floor0, ceil1, topRightCoordinates );
gather( ceil0 , floor1, bottomLeftCoordinates );
gather( ceil0, ceil1, bottomRightCoordinates );
```

**Slika 8.6:** Isječak koda bilinearne interpolacije s izračunom koordinata.

lik broj kopiranja s *host* uređaja na GPU uređaj prisutnih u izračunu volumena cijene i u sloju izobličavanja. Takva kopiranja su izrazito skupa i potrebno je definirati implementacije koja u što većoj mjeri izbjegavaju kopiranje između *host* uređaja i GPU uređaja. Iz tog razloga je potrebno razviti optimalnije rješenje za navedena dva sloja koja maksimalno smanjuju broj kopiranja i pravilno koriste puni potencijal GPU uređaja. Slojevi su definirani u `CostVolumeLayer` i `ImageWarpLayer` klasama. Izračun volumena cijena se sastoji od primjene operacija množenja i smanjivanja dimenzija na određene dijelove ulaza. Potencijalno ubrzanje bi se moglo ostvariti eliminiranjem dohvaćanja dijelova (*engl. slice*) jer se ta operacija sastoji od kopiranja s GPU uređaja na *host* uređaj. Kod sloja izobličavanja potrebno je smanjiti broj poziva kopiranja kod stvaranje rešetki i računanja koordinata. Za ostale slojeve sam koristio već gotove implementacije dostupne u TensorRT 3.0.2. Ponovna implementacija tih slojeva sigurno nije potrebna.

Optimalan broj dretvi je izrazito kompleksan problem koji ovisi i o mogućnostima samoga uređaja. Za svaku funkciju koju smo definirali uz pomoć CUDA biblioteke možemo definirati broj dretvi i blokova. Kao što je ranije navedeno, broj dretvi je mo-

```
float* interpTop{ batchedGrid };
float* interpBottom{ batchedGrid + channelSize };
auto interpolateChannel = [ & ]( const float* channel, float* output )
{
    gatherFromChannel( channel, topLeftCoordinates, channelSize, topLeft, stream );
    gatherFromChannel( channel, topRightCoordinates, channelSize, topRight, stream );
    gatherFromChannel( channel, bottomLeftCoordinates, channelSize, bottomLeft, stream );
    gatherFromChannel( channel, bottomRightCoordinates, channelSize, bottomRight, stream );

    //              interp_top = alphas[1] * (top_right - top_left) + top_left
    subtractTensors( channelSize, topRight, topLeft, interpTop );
    multiplyAddTensors( channelSize, alpha1, interpTop, topLeft, interpTop );

    //              interp_bottom = alphas[1] * (bottom_right - bottom_left) + bottom_left
    subtractTensors( channelSize, bottomRight, bottomLeft, interpBottom );
    multiplyAddTensors( channelSize, alpha1, interpBottom, bottomLeft, interpBottom );

    //              interp_top = alphas[1] * (top_right - top_left) + top_left
    subtractTensors( channelSize, interpBottom, interpTop, output );
    multiplyAddTensors( channelSize, alpha0, output, interpTop, output );
};
```

**Slika 8.7:** Isječak koda bilinearne interpolacije s dohvaćanjem piksela na temelju izračunatih koordinata.

guće definirati u više dimenzija. Time indeksiranje i dohvaćanje podataka u određenoj dretvi postaje kompliciranije. Iz tih razloga, za sve vlastite implementacije korišten je fiksni broj dretvi definiran u jednoj dimenziji, a broj blokova se računao na temelju veličine ulaznih podataka i broja dretvi po bloku koristeći formulu $\frac{N+T-1}{T}$, gdje $N$ predstavlja broj ulaza, a $T$ broj dretvi. Veću brzinu izvođenja je sigurno moguće ostvariti korištenjem većeg broja dretvi. Za propusnu zglobnicu sam morao napisati vlastitu implementaciju pa iz tog razloga su dimenzije dretvi i blokova definirane na ranije opisan način. Implementacija propusne zglobnice se koristi u skoro svakom sloju modela pa bi veći broj dretvi za tu funkciju ubrzalo izvođenje. Optimalne dimenzije struktura koje sadrže blokove i dretve potrebno je pronaći na temelju karakteristika uređaja na kojem se pokreće kod, ali i na temelju većeg broja mjerenja.

# 9. Zaključak

Područje računalnog vida sastoji se od mnogih problema koje se u zadnje vrijeme pokušavaju riješiti metodama dubokog učenja. Problemi se razlikuju po kompleksnosti i prikladnosti metoda dubokog učenja za njihovo rješavanje.

Procjena optičkog toka je izrazito kompleksan problem koji pokušava procijeniti pomak svakog piksela pojedinačno. Takav problem zahtjeva i kompleksniji model koji može naučiti raspoznavati i najmanje promjene između para slika. Prijašnje metode za optički tok sastojale su se od zahtjevnih algoritama i operacija. Iako su postizali dobre rezultate, njihova brzina izvođenja nije uvijek bila zadovoljavajuća. Implementacijom i učenjem modela za procjenu optičkog toka možemo primijetiti kako se mnogi takvi algoritmi mogu zamijeniti tehnikama dubokog učenja. Takvi modeli se i dalje sastoje od velikog broja operacija, ali su glavne sastavnice modela sastavljene od izrazito jednostavnih operacija koje se s lakoćom mogu optimizirati na raznim procesorskim jedinicama, pogotovo na grafičkim. Neke dijelove kao što su izračun volumena cijene nije moguće u potpunosti zamijeniti dubokim učenjem, ali možemo iskoristiti svojstva dubokih modela i time smanjiti kompleksnost klasičnih metoda.

PWCNet model pokazuje kako je moguće procijeniti optički tok koristeći u velikoj mjeri metode dubokog učenja, a da se u isto vrijeme postigne i zadovoljavajuće vrijeme izvođenja. Međutim, možemo primijetiti i veliku ovisnost modela o količini podataka. Za vrijeme učenja na velikom skupu podataka kao što je FlyingChairs s jednostavnijom verzijom modela uspjeli smo dobiti dobre rezultate. U slučaju puno manjeg skupa podataka, kao što je KITTI skup podataka, dobili smo lošije rezultate. Taj problem bi se mogao riješiti s kompleksnijim modelom, ali onda dobivamo model koji sadrži veći broj parametara što zahtjeva više memorije.

Kako se model sastoji uglavnom od metoda dubokog učenja, s lakoćom se mreža može prevesti koristeći TensorRT skup alata. Time je mrežu moguće optimizirati i pokrenuti na GPU uređajima. Model koristi za izračun volumena cijene i izobličavanje klasične algoritme za procjenu optičkog toka. Iz tog razloga te metode je potrebno ručno implementirati koristeći cudNN i CUDA biblioteke. Metode same po sebi nisu

komplicirane i vrlo je jednostavno dobiti točnu implementaciju. Međutim, napisati optimizirane verzije takvih metoda nije jednostavno i zahtjeva detaljna proučavanja i mjerenja.

Na temelju ostvarenih rezultata možemo vidjeti prednosti dubokog učenja na području računalnog vida. S modelom koji se većinom sastoji od jednostavnih operacija mogu se ostvariti dobri rezultati što se tiče točnosti, ali i brzine izvođenja. Uz to, otvara nam se mogućnost lakog prilagođavanja modela za GPU uređaje što, uz pravilnu implementaciju, predstavlja značajno ubrzanje bez većeg truda. Samo manji dio modela i dalje koristi klasične algoritme za procjenu optičkog toka koje se potencijalno mogu zamijeniti, kao i ostali dijelovi, s metodama dubokog učenja. Međutim, implementacija klasičnih algoritama i dalje utječe na samu brzinu izvođenja. Iako sam uspio dobiti jednake izlaze za iste težine na GPU uređaju, brzina izvođenja od 3.5s nije ni blizu optimalne brzine. Razlog tome je neefikasna implementacija klasičnih algoritama kao što su volumen cijene i izobličavanje.

PWCNet je pokazao moć dubokog učenja na području računalnog vida. S pametnom arhitekturom možemo riješiti izrazito kompleksne probleme. Iako neke dijelove nije moguće zamijeniti u potpunosti metodama dubokog učenja, možemo iskoristiti i prilagoditi klasične algoritme u našem modelu. Duboko učenje često postiže bolje rezultate, ali klasični algoritmi su proizvod mnogih znanstvenih istraživanja. Takvi algoritmi su maksimalno prilagođeni problemu koji pokušavaju riješiti. PWCNet je pokazao da se klasični algoritmi i dalje mogu iskoristiti u modelima i time postići rezultate koje nadilaze zasebno korištenje dubokog učenja i klasičnih algoritama.

# LITERATURA

Aaron Bobick. *CS 4495 Computer VisionMotion and Optic Flow*. URL https://www.cc.gatech.edu/~afb/classes/CS4495-Fall2014/slides/CS4495-OpticFlow.pdf.

Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, i Thomas Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015. URL http://arxiv.org/abs/1504.06852.

Mark Gituma. *What is Optical Flow and why does it matter in deep learning*. URL https://medium.com/swlh/what-is-optical-flow-and-why-does-it-matter-in-deep-learning-b3278bb205b5.

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

Mark Harris. *CUDA Pro Tip: Write Flexible Kernels with Grid-Stride Loops*, 2013. URL https://devblogs.nvidia.com/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/.

Gao Huang, Zhuang Liu, i Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, i Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016. URL http://arxiv.org/abs/1612.01925.

Josip Krapac i Siniša Šegvić. Konvolucijski modeli. 2020. URL http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf.

Moritz Menze, Christian Heipke, i Andreas Geiger. Joint 3d estimation of vehicles and scene flow. U *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

Moritz Menze, Christian Heipke, i Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

Peter O'Donovan. Optical flow: Techniques and applications. 2005. URL https://www.dgp.toronto.edu/~donovan/stabilization/opticalflow.pdf.

Anurag Ranjan i Michael J. Black. Optical flow estimation using a spatial pyramid network. *CoRR*, abs/1611.00850, 2016. URL http://arxiv.org/abs/1611.00850.

Igor Smolković. Analiza i vrednovanje odabranihizvedbenih detalja postupaka optičkog toka, 2014. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/smolkovic14ms.pdf.

Deqing Sun, Xiaodong Yang, Ming-Yu Liu, i Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. *CoRR*, abs/1709.02371, 2017. URL http://arxiv.org/abs/1709.02371.

Ying Wu. Optical flow and motion analysis. URL http://users.eecs.northwestern.edu/~yingwu/teaching/EECS432/Notes/optical_flow.pdf.

**Konvolucijski modeli za optički tok**


**Sažetak**

Procjenjivanje optičkog toka neriješen je problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolja rješenja tog problema dobivaju se dubokim konvolucijskim modelima. Ovaj rad razmatra nadzirane pristupe kod kojih svaki par slika iz skupa za učenje mora biti označena gustim poljem točnog optičkog toka.

U okviru rada, proučene su konvolucijske arhitekture za procjenu optičkog toka. Oblikovan je odgovarajući model i naučen na javno dostupnim skupovima. Mogućnost izvedbe na ugradbenom računalnom sustavu je procijenjena i opisana u sklopu rada. Hiperparametri su validirani, prikazani uz ostvarene rezultate koji su uspoređeni s rezultatima iz literature. Predložen je pravac budućeg razvoja.

Uz rad je priložen izvorni kod razvijenih postupaka uz potrebna objašnjenja i dokumentaciju.



**Ključne riječi:** optički tok,duboko učenje,PWCNet,TensorRT,CUDA,konvolucijski modeli,duboki modeli,računalni vid

**Convolutional models for optical flow**

**Abstract**

Optical flow estimation is an unsolved problem in computer vision with many interesting applications. Lately, the best results are achieved using deep convolutional models. This paper analyses approaches using supervised learning where each pair of images from training set has correctly annotated optical flow.

In this paper, different convolutional models for optical flow estimation are described. One of those models was implemented and trained on publicly available datasets. Possibility of implementing that model on an embedded device was researched and described. Hyperparameters were validated and presented alongside the achieved results that were also compared to the results from the literature. Suggestion for future work is also described.

**Keywords:** optical flow,deep learning,PWCNet,TensorRT,CUDA,convolutional models, deep models,computer vision