SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Optimizacija dubokih modela na ugradbenom računalu

Stjepan Močilac
Voditelj: *Siniša Šegvić*

Zagreb, lipanj 2019.

SADRŽAJ

1. Uvod

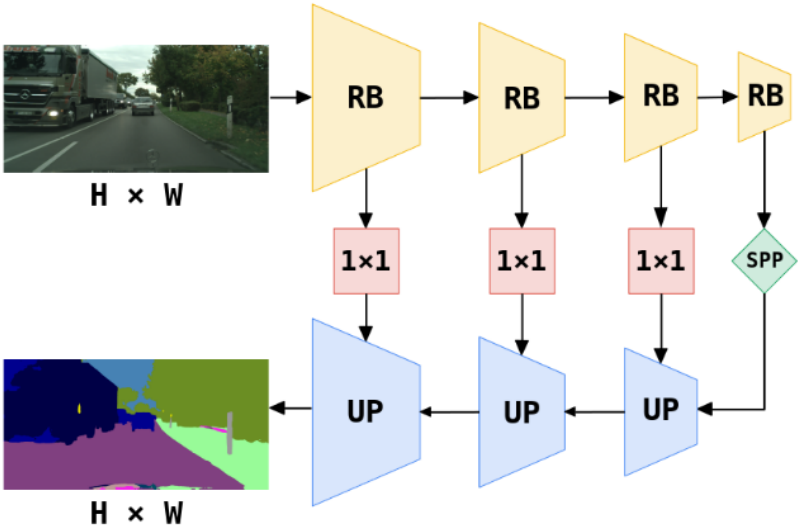
Pretraživanje prostora u cilju pronalaska optimalnih parametara dubokih modela obično se izvodi uz pomoć postojećih programskih okvira (npr. Tensorflow, Caffe, PyTorch, ...). Nakon postupka pretraživanja, potrebno je izgraditi i optimalni stroj za evaluaciju modela u cilju redukcije latencije, potrošnje snage i memorijskih resursa te u cilju maksimiziranja protoka podataka u jedinici vremena. Iako i prethodno navedeni programski okviri omogućuju vrlo efikasnu evaluaciju na grafičkim jedinicama, pokazalo se da postoje gubitci nastali pri postizanju generičnosti i fleksibilnosti. Time se javila potreba za alatom koji bi omogućio maksimalnu iskoristivost danih resursa. Aktualni paralelni programski model za optimizaciju evaluacije modela (eng. inference optimizer) TensorRT, razvijen u sklopu NVIDIA-e, omogućuje upravo takvu optimizaciju na grafičkim jedinicama određenih arhitektura. U sklopu ovog rada optimizirat će se duboki model za semantičku segmentaciju slike na ugradbenom računalu TX2 uz asistenciju alata TensorRT i platforme za paralelno programiranje CUDA-e.

TensorRT je programski okvir koji omogućuje optimalno izvođenje modela na specifičnom GPU (eng. Graphics Processing Unit). Kompatibilan je sa postojećim programskim okvirima (npr. Caffe2, TensorFlow, PyTorch). Konverzija modela u odgovarajući format, odnosno stroj za izvođenje (eng. inference engine) provodi se preko parsera integriranih u TensorRT. U ovom trenutku podržani su parseri za Caffe2, UFF (TensorFlow) te ONNX modele. Alat, uz navedenu kompatibilnost prema postojećim okvirima, omogućuju i ručno definiranje i unos slojeva te parametara. Caffe2 model također je moguće "nadopuniti", odnosno definirati slojeve koji nisu integrirani u TensorRT kao nadomjestak (eng. Plugin). Bitno je napomenuti da se konverzija modela u stroj za izvođenje provodi na način da se provede jedan unaprijedni prolaz (eng. forward pass), zapamte se slojevi, operacije i parametri (eng. stacking) te se iz tog rekonstruira arhitektura modela. Obzirom na navedeno, očigedno je da ne smije postojati ovisnost idućeg sloja o njegovom ulazi (uvjeti u ovisnosti o ulazu). Stoga, slojevi i operacije se ne smiju dinamički odabirati obzirom na ulazne vrijednosti.

2. Semantička segmentacija

Promatrati ćemo konkretni model za semantičku klasifikaciju, odnosno klasifikaciju slike na razini piksela. U tu svrhu iskoristit ćemo model predstavljen za CVPR 2019 [Orsic et al. (2019)]. Korištena metoda se temelji na tri gradivna bloka. Prvi je segmentacijski enkoder za koji se koristi predtrenirani ImageNet model. Autori preporučaju korištenje ResNet-18[He et al. (2015)] i MobileNet V2[Sandler et al. (2018)] arhitekture u tu svrhu. Idući gradivni blok je dekoder koji se koristi za naduzorkovanje značajki. U tu svrhu preporučen je jednostavan dekoder organiziran kao sekvenca modula za naduzorkovanje sa lateralnim vezama. Konačno, treći blok je pojednostavljeni modul za povećanje receptivnog polja iz PSPNet [Zhao et al. (2016)].

U okviru ovog rada razmatrat ćemo gore navedeni model koji za segmentacijski enkoder koristi ResNet-18. Enkoder daje mapu smanjene rezolucije semantički bogatih značajki ulazne slike. Dekoder služi za naduzorkovanje na početnu rezoluciju. Navedeni model koristi sekvencu dekodera sa lateralnim vezama (slika 2.1). Oni na ulazu dobivaju značajke niske rezolucije te lateralne značajke iz ranijeg sloja enkodera. Interno se značajke niske rezolucije bilinearnom interpolacijom podižu na rezoluciju lateralnih značajke te se te dvije mape zbroje (eng. element wise sumation). Konačno dobivena mapa prolazi kroz 3x3 konvoluciju. Zadnji modul spomenut u prethodnom paragrafu je modul za povećanje receptivnog polja. U tu svrhu koristi se SPP (eng. Spatial Pyramid Pooling) blok. [Orsic et al. (2019)]



Slika 2.1: Strukturalni dijagram predloženog single scale modela [Orsic et al. (2019)]

3. Optimizacija modela

Optimizacija modela u okviru ovog rada podrazumijeva maksimalno iskorištenje danih resursa u smislu vremenskog i prostornog zauzeća. Orijentirati ćemo se na programski okvir CUDA namjenjen programiranju NVIDIA grafičkih jedinica. Jezgra TensoRT-a [Ten] je C++ programska biblioteka namjenjena za poboljšavanje performansi na NVIDIA grafičkim procesnim jedinicama. TensorRT je kompatibilan sa većinom poznatih programskih okvira namjenjenih za duboko učenje (Caffe, Tensorflow, PyTorch i sličnim). Interno sadrži parsere za postojeće modele u formatu ONNX, Caffe i TensorFlow pa je potrebno napraviti konverziju modela u neki od tih formata. Također, u okviru TensorRT-a je omogućena programska definicija modela kroz Python i C++ aplikacijsko sučelje. Optimizacija mreže uspostavlja se kombiniranjem slojeva i optimizacijom odabira jezgri za poboljšanu latenciju, protok, potrošnju snage i konzumaciju memorije. Često se odabire izvođenje uz manju preciznost čime je moguće dodatno poboljšati navedene performanse.

TensorRT uzima definiciju mreže, izvodi optimizaciju za specifičnu platformu te generira stroj za izvođenje. Taj korak se zove faza izgradnje (eng. build phase). Ovaj korak može trajati dugo, ovisno o platformi na kojoj se izvodi te o kompleksnosti same arhitekture za koju se stroj generira. Upravo iz tog razloga, dobra praksa je napraviti serijalizaciju stroja te zapisati bajtove u datoteku. Bitna napomena je da je serijalizirani stroj direktno vezan za verziju TensorRT-a te za specifični GPU. Generalno, pri izgradnji stroja, provodi se nekoliko optimizacija:

- Eliminacija slojeva čiji se izlazi ne koriste
- Agregacija operacija sa dovoljno sličnim parametrima
- Konkatenacija i spajanje slojeva usmjeravanjem izlaza slojeva prema ispravnoj destinaciji
- Odabir najbržih jezgri obzirom na specifični GPU na način da se mreža evaluira nasumičnim podacima po slojevima. Gdje je moguće, izvršava se memorijska optimizacija i predformatiranje težina.

– Stapanje konvolucije, biasa i ReLU operacija

– Opcionalno, provodi se modifikacija preciznosti (FP16, INT8)

Dodatno, moguće je definirati vlastite slojeve implementirajući Plugin sučelje(eng. interface). Ključni dijelovi sučelja jezgre TensorRT-a su :

– INetworkDefinition omogućuje metode za definiranje mreže (ulazni i izlazni vektor, slojevi, kofiguracija tipa sloja). Svaki definirani sloj mora biti ili nativno podržan, ili implementirati sučelje Plugin.

– IBuilder omogućuje izgradnju optimiziranog stroja kroz definiciju mreže (NetworkDefinition). Definira se veličina grupe (eng. batch size) i veličina radnog prostora (eng. workspace size), te preciznost (INT8, FP16, FP32).

– ICudaEngine sučelje omogućuje izvršavanje optimiziranog stroja. Omogućuje sinhrono i asinhrono izvođenje, profiliranje, enumeraciju te povezivanje sa ulaznim i izlaznim podacima. Moguće je istovremeno izvršavanje više grupa (eng. multiple batches).

Dobra ideja je definirati minimalnu potrebnu implementaciju za ostvarivanje konačne optimizacije. U okviru ovoga rada orijentirati ćemo se na C++ sučelje radi podrške za ugradbeni uređaj TX2. Za početak, potrebno je ostvariti globalnu implementaciju sučelja ILogger. Minimalni primjer koji ostvaruje

```
class Logger : public ILogger
{
    nvinfer1::ILogger& getTRTLogger() {
        return *this;
    }
    void log(Severity severity, const char* msg) override {
        td::cout << msg << std::endl;
    }
} gLogger;
static Logger mLogger;

int main(){
    IBuilder* builder = createInferBuilder(gLogger.getTRTLogger());
    nvinfer1::INetworkDefinition* network = builder->createNetwork();
    auto parser = createParser(*network, mLogger.getTRTLogger());
    parseFile(parser, gLogger);
    set_builder_modes(builder); // FP32, batch=1
    ICudaEngine* engine = builder->buildCudaEngine(*network);
    trtModelStream = engine->serialize();
    destroy(engine, network, builder);
```

```
    IRuntime* runtime = createInferRuntime(mLogger.getTRTLogger());
    engine = runtime->deserializeCudaEngine(trtModelStream->data(),
            trtModelStream->size(), nullptr);
    IExecutionContext* context = engine2->createExecutionContext();
    doInference(*context, buffers, 1);
}
```

Globalna metoda prototipa createInferBuilder(ILogger) koristi se za izgradnju objekta tipa IBuilder te kao argument prima objekt klase Logger. U gornjem primjeru koristimo ONNX parser (unutar metode parseFile). Nakon parsiranja radimo serijalizaciju, odnosno dobivamo tok bajtova(eng. byte stream). Ako bismo htjeli učitati stroj direktno iz datoteke, mogli bismo preskočiti korake nakon parsiranja i odmah instancirati objekt tipa IRuntime te deserijalizirati stroj.

Implementacija alata u okviru ovoga rada može se podijeliti na nekoliko dijelova: konverzija modela u ONNX format, dohvat slike sa kamere na ugradbenom uređaju TX2, alate za mjerenje performansi u programskim okvirima TensorRT i PyTorch te alate za mjerenje točnosti modela.

Konverzija modela u ONNX format zasniva se na unaprijednom prolazu kroz mrežu. Redoslijed operacija i slojeva te parametri se pamte (eng. stacking) te se zapisuju u odgovarajućem formatu. Zaključak koji slijedi iz toga je da svi parametri modela moraju, u konačnici, biti konstantni. Jednostavna Python skripta koja opisuje konverziju našega modela dana je u nastavku.

```
use_bn = True
resnet = resnet18(pretrained=False,efficient=False,use_bn=use_bn)
model = SemsegModel(resnet, 19, use_bn=use_bn)
model.load_state_dict(torch.load('weights.pt'), strict=True)
model.eval()
model.to('cuda')
input_ = torch.ones([1,3,1024,2048]).to('cuda')
torch.onnx.export(model, input_, "swiftnet.onnx")
```

Dohvat slike sa kamere na ugradbenom uređaju TX2 služi za vizualizaciju segmentacije TensorRT modela. Korištena kamera je CSI (eng. Camera Sensor Interface) kamera. Alat za dohvaćanje takvih slika je GStreamer. Slike koje se dobivaju iz kamere su u NV12 formatu, pa je omogućena i konverzija slika u RGBA format unutar programskog okvira CUDA.

Alati za mjerenje performanse mogu se podijeliti na mjerenje performanse u PyTorch i TensorRT okruženju. Mjera performanse u PyTorchu pokazana je u Figure 5. u [Orsic et al. (2019)]. TensorRT performansa integrirana je u programsko riješenje u okviru

ovog rada. Sva mjerenja vezana za brzine izvođenja modela (eng. inference) uprosječena su nad 900 iteracija. Provedena mjerenja daju dvije bitne informacije, odnosno vrijeme potrebno za unaprijedni prolaz i vrijeme potrebno za CPU-GPU i GPU-CPU transfere. Njihova suma daje realnu procjenu brzine izvođenja.

Alati za mjere točnosti PyTorch modela odgovaraju onima u [Orsic et al. (2019)]. Za mjerenja TensorRT modela implementirana je međuprocesna komunikacija putem datoteke. Pojednostavljeno rečeno, dohvat, predobrada (eng. preprocessing) slike te izracun pogrešaka se i dalje izvode unutar Python okruženja, dok se za inference poziva C++ binarna datoteka.

4. Rezultati

U okviru rada napravljeno je više evaluacija u različitim okvirima te načinima rada. U ovom poglavlju držat ću se najrelevantnijih podataka za buduće primjene. Tablica 4.1 pokazuje brzinu evaluacije modela na ulaznoj rezoluciji 3x1024x2048. Prikazani rezultati su dobiveni nad TensorRT modelima. Model na GTX 1080i je u FP32 načinu rada, dok je onaj na TX2 u FP16 načinu rada.

Tablica 4.1: Rezultati zaključivanja sa i bez CPU-GPU transfera

GTX 1080i [3x1024x2048]	Vrijeme [ms]
Average inference with GPU-CPU and CPU-GPU transfer	24.39
Average inference only	19.95
TX2 [3x1024x2048]	**Vrijeme [ms]**
Average inference with GPU-CPU and CPU-GPU transfer	278.37
Average inference only	266.29

Tablica 4.2 pokazuje konačno ostvarenje početnog cilja - optimizaciju modela. Na GTX 1080i se jasno vidi ubrzanje od približno 1.7 puta, te je taj omjer približno očuvan po različitim rezolucijama. U donja dva retka vidi se napredak u FPS i na ugradbenom uređaju TX2. Ovdje je taj omjer značajno veći od 1.7, što je zasluga optimizacije u FP16 (eng. Floating Point 16). Također, očigledno je da taj omjer ostaje očuvan po različitim rezolucijama.

Tablica 4.3 pokazuje dobitak na brzini izvođenja ako se koristi FP16 na Jetson TX2 uređaju. Očigledno je da takva konverzija omogućuje značajan napredak u smislu brzine evaluacije.

Konačno, bitno je pokazati da model evaluiran na dva različita programska okvira daje identične rezultate točnosti. Tablica 4.4 prikazuje rezultate mIoU nad Cityscapes validacijskim datasetom (500 slika) modela u Pytorch-u i TensorRT-u. Može se zaključiti da smo, u slučaju da zadržimo FP32 način rada, u potpunosti sačuvali toč-

Tablica 4.2: Rezultati zaključivanja u FPS (eng. Frames Per Seconds)

	3x256x512	3x512x1024	3x1024x2048
GTX 1080i - PyTorch	226	92	25
GTX 1080i - TensorRT	384	143	41
Jetson TX2 - PyTorch	22.8	6.1	1.5
Jetson TX2 - TensorRT	45.3	13.7	3.6

Tablica 4.3: Rezultati TX2, FP16 vs FP32

	FP32	FP16
3x1024x2048	419.66 ms	278.37 ms
3x512x1024	105.7 ms	73.01 ms
3x256x512	36.02 ms	22.07 ms

nost, a napredovali u brzini izvođenja modela. Također, zanimljiv je i zadnji redak koji pokazuje da prelazak na FP16 rezultira gubitkom od samo 0.02%. Bitno je za napomenuti da je mjera mIoU 3% manja od originalno dobivene u [Orsic et al. (2019)]. Naime, TensorRT ne podržava bilinearnu interpolaciju u sloju za naduzorkovanje, pa je u okviru ovog rada za interpolaciju korištena metoda najbližeg susjeda (eng. nearest neighbour). Upravo ta promjena rezultira padom od navedenih 3%.

Tablica 4.4: Rezultati mjere točnost mIoU (eng. mean Intersection over Union)

	mIoU
GTX 1080i - PyTorch	72.42%
GTX 1080i - TensorRT FP32	72.42%
Jetson TX2 - PyTorch	72.42%
Jetson TX2 - TensorRT FP32	72.42%
Jetson TX2 - TensorRT FP16	72.40%

Izvorni kod dostupan je na `https://github.com/smocilac/swiftnet_inference`.

5. Zaključak

Optimizacija modela za integraciju donosi značajno poboljšanje performansi uz očuvanje točnosti. Razlog tomu je činjenica da se alati poput TensorRT-a mogu prilagoditi platformi na kojoj se nalaze te utilizirati prednosti specifičnog uređaja (u konkretnom slučaju GPU-a). Uz to, prijelaz sa FP32 na FP16 se pokazao prilično uspješnim, pa bi u budućnosti bilo interesantno modele evaluirati na grafičkim jedinicama koje podržavaju FP16 način rada.

U daljem radu interesantno bi bilo pokrenuti TensorRT model koji koristi bilinearnu interpolaciju. Također, kako je u radu kao enkoder korišten samo ResNet-18, zanimljiva informacija bi bila isprobati performanse različitih enkodera kao što je to predloženo u [Orsic et al. (2019)].

6. Literatura

Tensorrt developer guide. https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html. Accessed: 2019-05-16.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Marin Orsic, Ivan Kreso, Petra Bevandic, i Sinisa Segvic. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. *CoRR*, abs/1903.08469, 2019. URL http://arxiv.org/abs/1903.08469.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, i Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, i Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016. URL http://arxiv.org/abs/1612.01105.

7. Sažetak

Pretraživanje prostora u cilju pronalaska optimalnih parametara dubokih modela obično se izvodi uz pomoć postojećih programskih okvira (npr. Tensorflow, Caffe, PyTorch, ...). Nakon postupka pretraživanja, potrebno je izgraditi i optimalni stroj za evaluaciju modela u cilju redukcije latencije, potrošnje snage i memorijskih resursa te u cilju maksimiziranja protoka podataka u jedinici vremena. Iako i prethodno navedeni programski okviri omogućuju vrlo efikasnu evaluaciju na grafičkim jedinicama, pokazalo se da postoje gubitci nastali pri postizanju generičnosti i fleksibilnosti. Time se javila potreba za alatom koji bi omogućio maksimalnu iskoristivost danih resursa. Aktualni paralelni programski model za optimizaciju evaluacije modela (eng. inference optimizer) TensorRT, razvijen u sklopu NVIDIA-e, omogućuje upravo takvu optimizaciju na grafičkim jedinicama određenih arhitektura. U sklopu ovog rada optimizirali su se duboki model za semantičku segmentaciju slike na ugradbenom računalu TX2 uz asistenciju alata TensorRT i platforme za paralelno programiranje CUDA-e.

Pokazano je da optimizacija modela za integraciju donosi značajno poboljšanje performansi uz očuvanje točnosti. Razlog tomu je činjenica da se alati poput TensorRT-a mogu prilagoditi platformi na kojoj se nalaze te utilizirati prednosti specifičnog uređaja (u konkretnom slučaju GPU-a). Uz to, prijelaz sa FP32 na FP16 se pokazao prilično uspješnim, pa bi u budućnosti bilo interesantno modele evaluirati na grafičkim jedinicama koje podržavaju FP16 način rada.