Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Aszfaltsáv felfestések kamerakép alapú felismerése

Önálló laboratórium 1. zárójegyzőkönyv
2017/18. II. félév

Dobai Botond

III. évf, mérnökinformatikus szakos hallgató
BSc. Szoftverfejlesztés specializáció

Konzulensek:

Varga Róbert
(Méréstechnika és Információs Rendszerek Tanszék)

Hadházi Dániel
(Méréstechnika és Információs Rendszerek Tanszék)

Feladat:

Leírás

A feladatom közúti felfestések, illetve sávok felismerése volt konvolúciós hálók segítségével. A félév során ehhez próbáltam egy működő modellt megvalósítani. Mivel kevés a használható adathalmaz, ezért kétféle megközelítést is kipróbáltam, amihez találtam megfelelő adathalmazokat.

Az alap elképzelés az, hogy egy FCN (fully convolutional network) típusú háló segítségével valósítom meg a felfestések szegmentálását. Ez egy olyan hálóstruktúra, amelynek a bemeneti és a kimeneti oldalán is konvolúciós rétegek találhatóak, tehát jelen esetben egy RGB kép batch bemenetére a hozzájuk tartozó, képpontonként osztályozott felfestések bitmap képe lesz a kimenet. A saját sávot vagy ugyanezzel a szegmentáló hálóval szeretném meghatározni, vagy egy olyan konvolúciós hálóval, amelynek a kimeneti oldalán fully connected rétegek adják meg a két (jobb- és baloldali) másodfokú polinom együtthatóit.

Mivel a felfestések nagyon különbözőek lehetnek, így célszerűbbnek látom, ha kiszámítjuk mind a felfestéseket, mind a saját sávot, mind a felfestéseket, és ezek alapján osztályozzuk a saját sáv két oldalához legközelebb eső vonalak típusát.

Fejlesztői környezet

A labor során Python nyelvű kódokkal dolgoztam, amelynek előnye egyszerűségében és jó támogatottságában rejlik. A fejlesztés során PyCharm IDE-t használtam. A jól használható felület mellett tartalmaz egy Python-csomagkezelőt, amely megkönnyítette a rendszer konfigurálását.

A következő fontosabb Python-könyvtárakat használtam a megvalósítás során:

- Numpy: egy elengedhetetlen modul, amely több dimenziós tömbük egyszerű inicializálását, kezelését, transzformációját teszi lehetőv.
- Az OpenCV egy igen széleskörű eszköztárat nyújt. Alapvetően képek kezelésére, illetve különböző képfeldolgozási eljárások meghívására használtam.
- Kerast használtam a neurális hálómodell eléréséhez, tanításához. Ez egy absztrakciós réteget nyújt különböző gépi tanulásos könyvtárakhoz. Én a Tensorflow-t használtam, mint backend.

A tanításhoz a Tensorflow CUDA-alapú hardveres gyorsítását használtam.

A következő konfigurációt használtam a futtatáshoz:

- Nvidia GTX 1050Ti videokártya, 4GB VRAM
- Intel Xeon 1230v2 processzor
- 8GB rendszermemória
- Antergos Linux

Tanítási halmazok

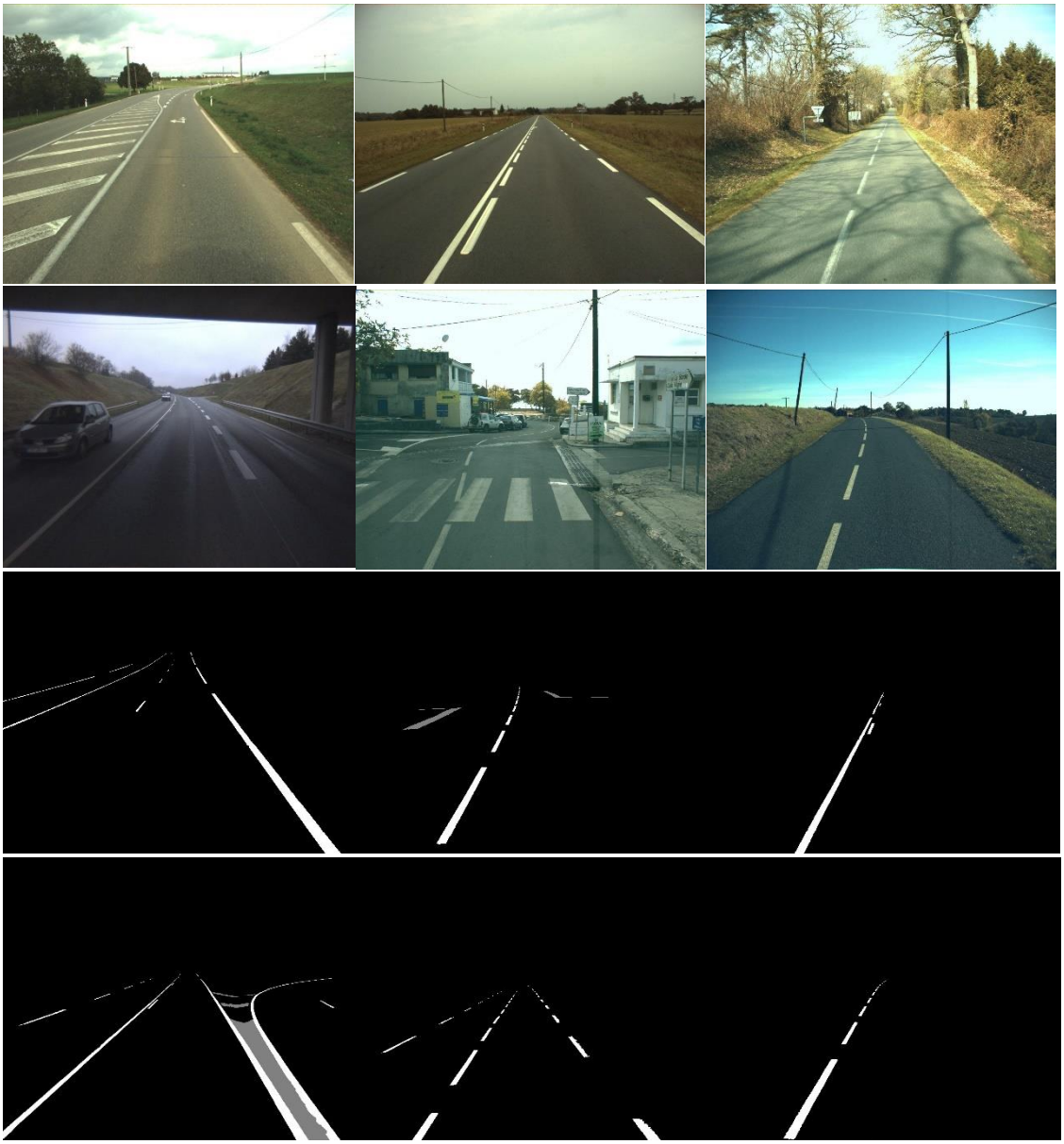
A legsarkalatosabb pont az volt a munkám során, hogy kevés adatbázis állt rendelkezésre rendes felcímkézéssel, így ehhez kellett igazodni. Kétféle osztályozást használnak a kinézett adatbázisok: az egyik fajta az, ahol a felfestések jelölve vannak. Mivel egy ilyet kézzel időigényes felcímkézni, így nem sok ilyet találtam. Nagyobb, géppel szegmentált adathalmazokat nem találtam.

A másik típus az, ahol csak az útfelület, és a saját sáv van megjelölve. Ebből a típusból már bővebb körben lehet válogatni.

Végül négy adathalmazt választottam ki a munkához kapcsolódóan:

ROad MArkings

http://perso.lcpc.fr/tarel.jean-philippe/bdd/index.html



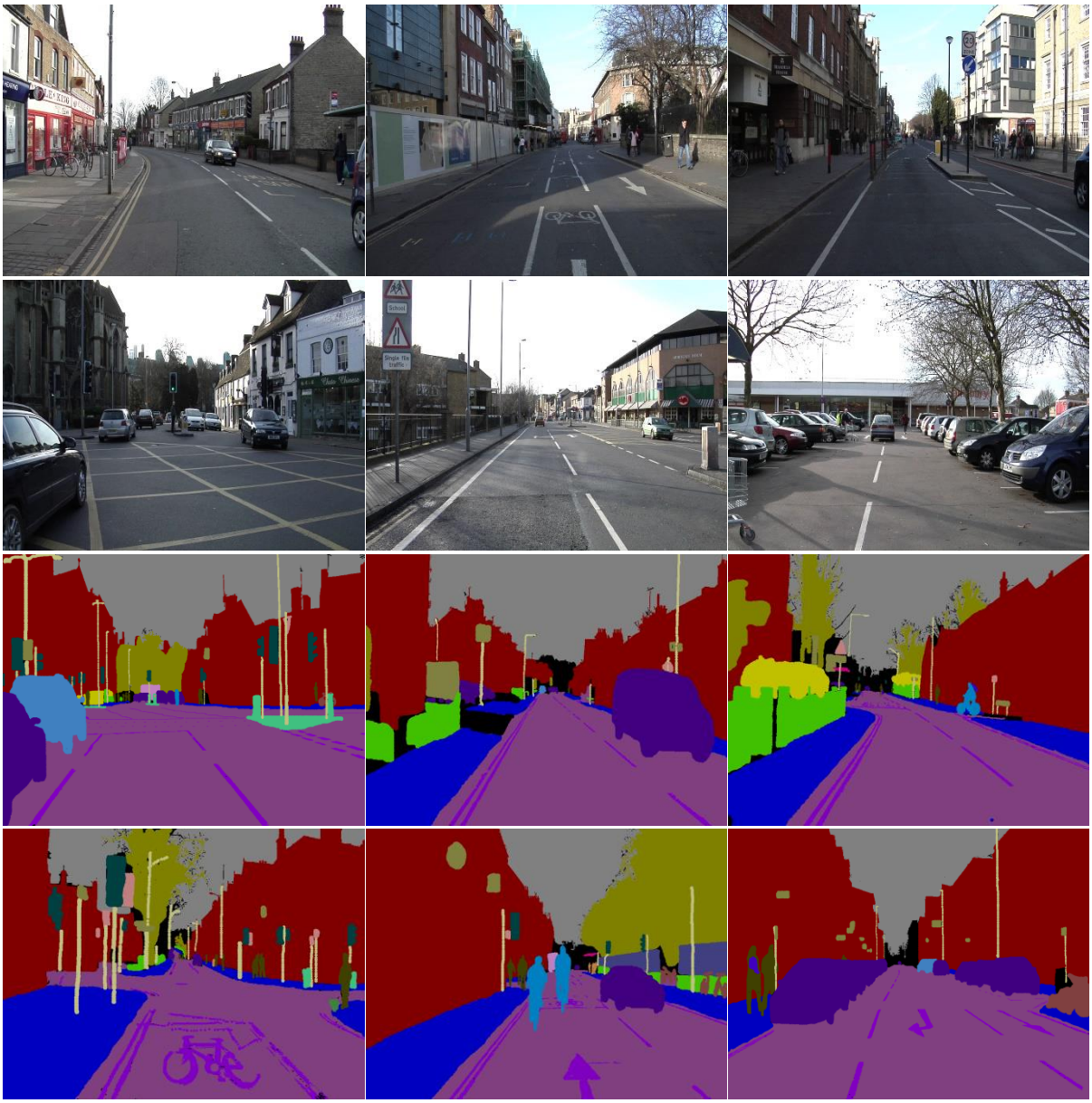
Ez egy 116 egyedi képből álló adatbázis, amely közúti fotókat tartalmaz különböző típusú felfestésekkel. A címkézés itt egy képre illeszkedő bitmap. Külön vannak ösztályozva a sávtartáshoz szükséges vonalak, és az egyéb jelölésre szolgáló felfestések.

CamVid

http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

https://github.com/mostafaizz/camvid

Ennél az adatbázisnál 701 egyedi kép áll rendelkezésünkre városi és városon kívüli környezetből vegyesen. A címkézés itt egy bitmap kép, több osztállyal, amelyeket különböző színek jelölnek. Ebből a projektben kizárólag a felfestést használtam fel.



MLND-capstone

https://github.com/mvirgo/MLND-Capstone/blob/master/README.md

Ez 1978 egyedi képből, és az ebből készült módosulatokból áll (~12-14 ezer kép összesen). Ez jelentősen különbözik az előzőktől, hiszen itt a felfestések helyett a saját sáv van jelölve. Mivel

eredetileg ez egy OpenCV segítségével, polinomillesztéssel készült dataset, így a meglévő címkékből viszonylag egyszerűen lehetett jobb-baloldali, határoló másodfokú polinomokat generálni.

Erre az adathalmazra egy nyilvános projekt részeként találtam rá, amelyből részben a modelleket is felhasználtam.

(KITTI)
http://www.cvlibs.net/datasets/kitti/eval_road.php

Ennél az adatbázisnál a saját sáv, illetve az úttest van kijelölve. A jelenlegi projektemben nem szerepel, de felhasználható lehetne ehhez a projekthez a jövőben.

Modellek

Felhasználás

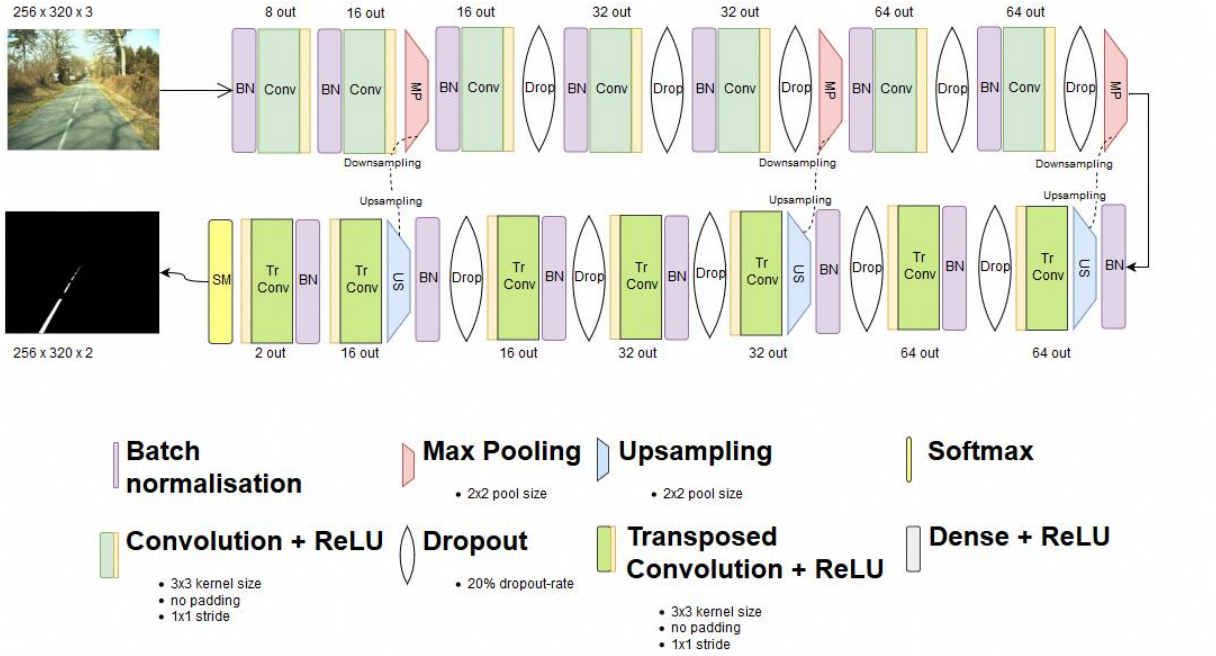
Kétféle hálóval dolgoztam a flélv során, ezek három tanítási fázisban voltak használva:

I. Az FCN modellt feltanítjuk a CamVid, majd a ROMA adathalmaz segítségével. Ez alapján később a vonalakat típus szerint osztályozni, illetve a súlyokat esetlegesen fel lehet használni a saját sáv felismerését végző modellekhez is.
II. Szintén az FCN modellt használjuk a saját sáv felismeréséhez az MLND-Capstone adathalmazának segítségével. Itt megvizsgáltam, hogy van-e pozitív hatása, ha a felfestésen tanult modell súlyaival inicializálunk.
III. Egy másik hálóval polinomos becslést adunk ugyanarra a problémára, mint a II-esben. Ez a fajta kimenet talán kevesebb utófeldolgozást igényelne, egyszerűbb kezelni, illetve nagyobb tanítóhalmaz áll rendelkezésünkre (pl MLND, KITTY, Cityscapes).

Egy jó megközelítés lehet, hogy az egyik modellel kijelöljük a vezető saját sávjának határait, a másikkal szegmentált felfestések képét pedig a határoló vonalak típusának meghatározásához használjuk fel.

Modellek felépítése

A feladathoz kétféle típusú neurális hálót használtam fel:



1. ábra Szegmentáló modell

Az első modellem egy szegmentáló háló, amelyet az I. és II. számú felhasználási módhoz szántam. Ez a háló konvolúciós rétegekből, és a feature-ök skálázására szolgáló rétegekből épül fel. A háló első része („encoder") végzi a kimenet számításához szükséges, magasabb szintű feature-ök tömörítését. Az encoder részben konvolúciós műveletekkel hozunk létre feature mapeket, és ezeket tömörítjük max pooling rétegek segítségével. A háló második része („decoder") a feleslegesnek vélt információtól megszűrt feature-ökből próbál az elvártnak megfelelő kimenetet adni. Megfigyelhető, hogy a két oldal bizonyos értelemben inverze egymásnak, azaz a tömörítő rétegek felskálázó párt, a konvolúciós rétegek transzponált konvolúciós párt kapnak.

A háló paraméterezése és rétegei az MLND-Capstone projektből származnak, kisebb átalakításokkal. A projekt szerzője egy SegNet nevű architektúrát vette alapul, úgyhogy én is átnéztem a tanulmányt. A hálót annyiban módosítottam, hogy több 'Batch Normalization' réteget használtam fel, illetve szürkeárnyalatos kimenet helyett egy bináris osztályozó kimenetet kapott.

A rétegek

A *MaxPooling* rétegek felezik a képméretet 2x2-es régiónkénti maximumkiválasztás segítségével.

Az *UpSampling* rétegek megkettőzik a képméretet a mezők interpolált felskálázásával.

A Convolution rétegek 3x3-as súlymátrixokkal (konvolúciós kernel) hajtanak végre konvolúciós műveleteket a bemenetükön. Itt úgy származtatunk kimeneti értékeket, hogy csúszó-ablakosan ráillesztjük a kernelt a bemeneti mátrixra, majd lépésenként vesszük a

részmátrixok súlyozott összegét. Ahogy a háló mélyül, úgy a kimeneti feature-mapek száma nő. Aktivációs függvényként ReLU -t használ.

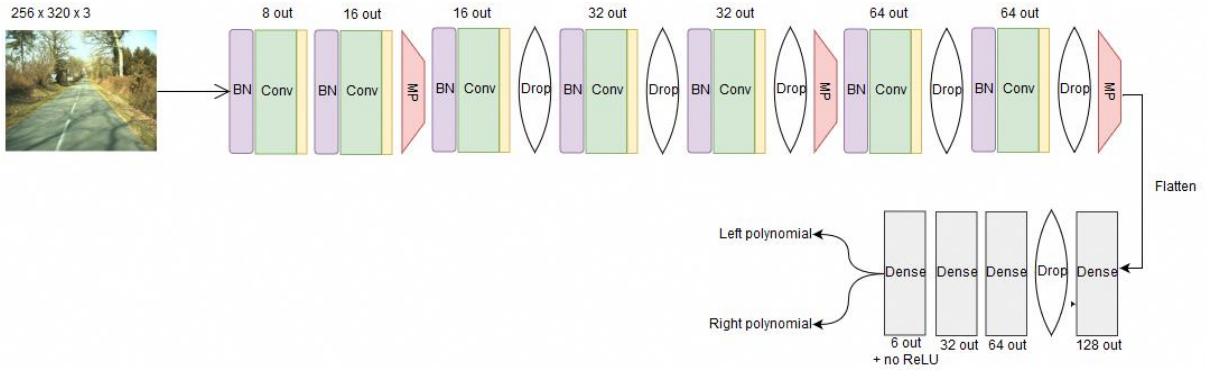
A *TransposedConvolution* rétegek transzponált konvolúciót hajtanak végre 3x3-as kernelekkel. 1x1 stride és padding nélkül ('valid' padding) ez egy normál konvolúciós műveletet jelent úgy paddingelve, hogy a kimenet mérete egyezzen a vele párban lévő Convolution réteg bemeneti méretével. Aktivációs függvényként ReLU -t használ.

A *Batch Normalization* rétegeknek az a funkciójuk, hogy a rétegek bemeneteit a batch-enként vett átlag és szórás segítségével normalizálja. Ezzel csökkenti az esetleges, feature-ök közti nagyságrendi különbségeket, illetve elősegítheti a modellünk jobb általánosítóképességét.

A *Dropout* rétegek véletlenszerűen eldobják az őt megelőző réteg kimeneteinek 20%-át. Ez a túltanulás esélyének csökkentése miatt szükséges lépés, illetve minimálisan a számítási igényt is csökkenti a szélesebb rétegek előnyeinek megtartása mellett.

A Softmax aktivációs függvény eredménye alapján osztályozzuk a kimeneti réteg pixeleit (nem felfestés ←→ felfestés).

A hiba-visszaterjesztéshez Adam optimizert használtam categorical crossentropy hibafüggvénnyel.



2. ábra Polinomszámító modell

A második számú modellem első része egyezik a korábbi encoder részével, a második része viszont fully connected rétegekből áll. A kimenete 6 szám lesz – a bal- és jobboldali, sávot határoló, másodfokú polinomok együtthatói. A fully connected rétegek a kimenet kivételével ReLU aktivációs függvényt használnak.

A háló tanításához itt is Adam optimizert használtam, mean absolute error (átlagos abszolút különbség) hibafügvénnyel.

Tanítás

Előkészületek

1. Az adathalmazokat le kellet tölteni, mappákba rendezni, átnevezni. Bizonyos esetekben (gyorsabb inicializálás futtatáskor, vagy memóriahiány) szerializálni kellett őket az átalakítások után. A CamVid címkék esetén el kell távolítani a labelekből a számunkra

felesleges jelöléseket. Az MLND címkékből OpenCV segítségével polinomegyütthatókat kellett készíteni. Ugyanennél az adathalmaznál több, külön szerializált részre kellett bontani az adathalmazt, hogy elférjen a memóriában.
2. A képeket be kellett olvasni, át kellett méretezni, labelek esetén át kellett őket alakítani úgy, hogy az rgb színcsatornák helyett one-hot osztályozási vektorokat tartalmazzon minden egyes pixelre.
3. A modellt be kel tölteni tanítások előtt. Ha vannak előretanított súlyok, azokat a megfelelő rétegekbe be kell tölteni a modell összeállítása után.
4. A minták szaporításához, illetve az általánosabb jellegű tanuláshoz képgenerátort kellett készíteni, amely véletlenszerűen eltolja a színcsatornákat, tükrözi a képet horizontálisan, illetve +-5° eltéréssel forgatja a képeket a tanítási ciklus során. MLND esetén ezek már megvannak szerializált formában, úgyhogy itt a memóriába való szakaszos betöltést kellett csak megoldani.
5. A bemenetek ki kell értékelni a betanított modellekkel, az eredeti képpel össze kell kombinálni az eredmény képét, hogy szemléletes eredményt is kapjunk.
6. Kellett csinálni egy egyszerű scriptet, amely ffmpeg használatával képkockákat vág ki egy videóból, amit független mintaként fel lehet használni a teszteléshez.
7. A betanított modelleken a validáció elvégzése, ellenőrzése manuálisan. Keresztvalidáció, illetve független teszthalmazon végzett manuális validáció elvégzése.
8. Az eredmények alapján esetleges további tanítások elvégzése.

Tanítás menete

Első lépésben a CamVid adatbázison tanítottam a szegmentáló hálót. Az adatok 15%-a alkotta a validációs halmazt, így nagyjából 600 képen zajlott maga a tanítás. A képek minden tanítás során véletlenszerűen lettek vízszintesen tükrözve, elforgatva, illetve a színcsatornák is el lettek véletlenszerűen tologatva. A tanítás 20-as batchmérettel zajlott, ez már egy elég jó kompromisszumnak tűnt. Összesen 500 epochon át tanult. Valószínű, hogy még lehetne rajta finomhangolni további tanítási ciklusokkal, de mivel ez egy from-scratch tanított modell egy viszonylag kis tanítóhalmazzal, így időigényes munka elkerülni a túltanulást.

Mivel nagyon aránytalan a két osztály (felfestés : nem felfestés), így a hatékonyabb tanulás érdekében érdemes súlyozni a hiba-visszaterjesztésnél a két osztályt. Mivel a Keras ezt nem támogatja ilyen dimenziójú kimenetekre, így ezt csak saját hibafüggvény implementálásával lehetne megoldani. Készítettem egy ilyen hibafüggvényt, de jelenleg problémás a működése.

Második lépésben a ROad MArkings adatbázison tanítottam a már előzőleg feltanított szegmentáló modellt. Ez egy kisebb adatbázis, a tanító halmazban nagyjából 100 kép marad. Annyiban más ez az adathalmaz, hogy itt a kifejezetten sávtartást szolgáló vonalak el vannak különítve az egyéb felfestésektől (zebra, forgalomtól elzárt terület, stb), így ez egy kevésbé általános tanítóhalmaz, jobban hozzá lehet illeszteni a feladathoz.

Itt is azonos paraméterekkel végeztem a tanítást (Adam optimizer, categorical_crossentropy hibafüggvény, 20-as batch size, 500 tanítási ciklus).

A tanítást elvégeztem teszt gyanánt úgy is, hogy a modell első felében befagyasztottam a súlyokat, azaz nem változtak a tanítás során. Ezt a módszert gyakran szokták alkalmazni olyan esetekben, ahol több, vagy akár többféle adathalmazon akarunk tanítani egy hálót. Ilyenkor a

magasabb szintű rétegekről azt feltételezzük, hogy az előállított feature-ök mindkét tanító halmaz esetén hasznosak. Ha az adathalmazunk jelentősen kisebb, mint az eredeti tanítóhalmaz, akkor ez csökkenti a túltanulást.

A harmadik lépésben átalakítottam az MLND-capstone adatbázis címkéit polinomegyütthatókká, és ezekkel szerettem volna a fully connected réteges hálót betanítani. Itt a fő nehézséget talán az jelenti, hogy az együtthatók között több nagyságrendnyi különbség van. Mivel a változóknak nincs túl nagy szórása, így ezt lehet kezelni például egy egyszerű egy szintre szorzással. Itt kétszeres batchmérettel lehetett dolgozni (40 kép) a kisméretű címkézés miatt (együtthatók bitmapek helyett).

A polinombecslő modellel nem sikerült eddig értékelhető eredményt kapnom.

Negyedik lépésben szintén az MLND-Capstone adathalmaz képei alapján szerettem volna tanítani a szegmentáló hálót, viszont itt már a saját sávok felismerése a cél. A dokumentáció leadásáig még csak egy nagyon rövid teszttanítást hajtottam rajta végre, de bíztató volt az eredmény.

Eredmények

Keresztvalidálás

adathalmaz\ modell	Camvid	Roma-1	Roma-2	Roma-3
CamVid (loss)	0.020887863453266757 TP:0.8867602078419811% FN:0.6459246941332547% FP:0.1785220739976415% TN:98.28879302402711%	0.06709001072735157 TP:0.4784119803950472% FN:1.0542729215801887% FP:0.2443248820754717% TN:98.222990215949493%	0.07408484661916517 TP:0.1926062242040094 4% FN:1.340078677771226 4% FP:2.5642481390035377% TN:95.90306695902122%	0.07631155511118332 TP:0.2821321307488207 6% FN:1.250552771226415% FP:0.0725627395341981 2% TN:98.39475235849056%
Roma (loss)	0.010452094694806 47 TP:0.6848144531 25% FN:0.2531602647569444% FP:0.3997802734375% TN:98.66224500868056%	0.005773875086257855 TP:0.8233642578125% FN:0.1146104600 6944445% FP:0.5002170138888888% TN:98.56180826822917%	0.033497163818942174 TP:0.4863145616319444% FN:0.45166015625% FP:3.1477864583333335% TN:95.91423882378473%	0.010750207015209727 TP:0.654296875% FN:0.2836778428819444% FP:0.2433946397 5694445% TN:98.81863064236111%
Roma teljes (loss)	0.011896133454966134 TP:0.8323722050107759% FN:0.2611988988415 9486% FP:0.3145415207435345% TN:98.5918873754041%	-	-	-

Jelmagyarázat:

Camvid – csak a CamViden tanított modell
Roma-1 – CamViden, és ROMA adathalmazon tanított modell, 10^-3 tanítási tényező
Roma-2 – CamViden, és ROMA adathalmazon tanított modell, 10^-4 tanítási tényező
Roma-3 – CamViden, és ROMA adathalmazon tanított modell, befagyasztott encoder résszel tanítva

megj.: a teljes ROMA adathalmazon is tesztelve lett az egyik modell, mivel sosem láthatta azt. A többi modell csak annak a validáló halmazán lett tesztelve.

TP: felfestés találat, TN: nem felfestés találat, FP: felfestés hibás találat, **FN**: nem felfestés hibás találat

Tanulságok:

Első sorban a loss, a true positive és a false negatív értékek fontosak számunkra. Ez alapján elmondható, hogy a ’Camvid’ és a ’Roma-1’ modellek teljesítettek a legjobban.

A befagyasztott modellsúlyos tanítás ugyan működött, de a tesztek során nem szerepelt kiemelkedően.

Manuális tesztelés

Három különböző, internetes videókból kivágott képkockasorral végeztem egy manuális tesztelést. Városi környezet, országút és autópálya is szerepelt a képeken (a harmadik videó linkjét nem találtam):

https://www.youtube.com/watch?v=Juym1p10TdQ (1.)

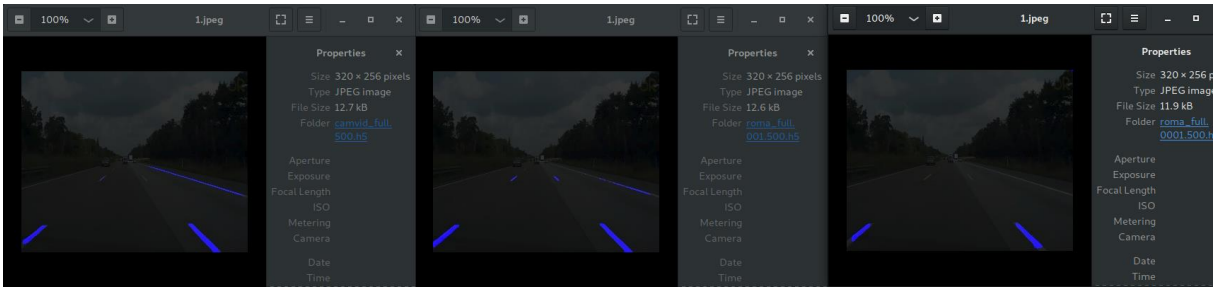
https://www.youtube.com/watch?v=uX6xSb6v6Pc&t=13199s (3.)

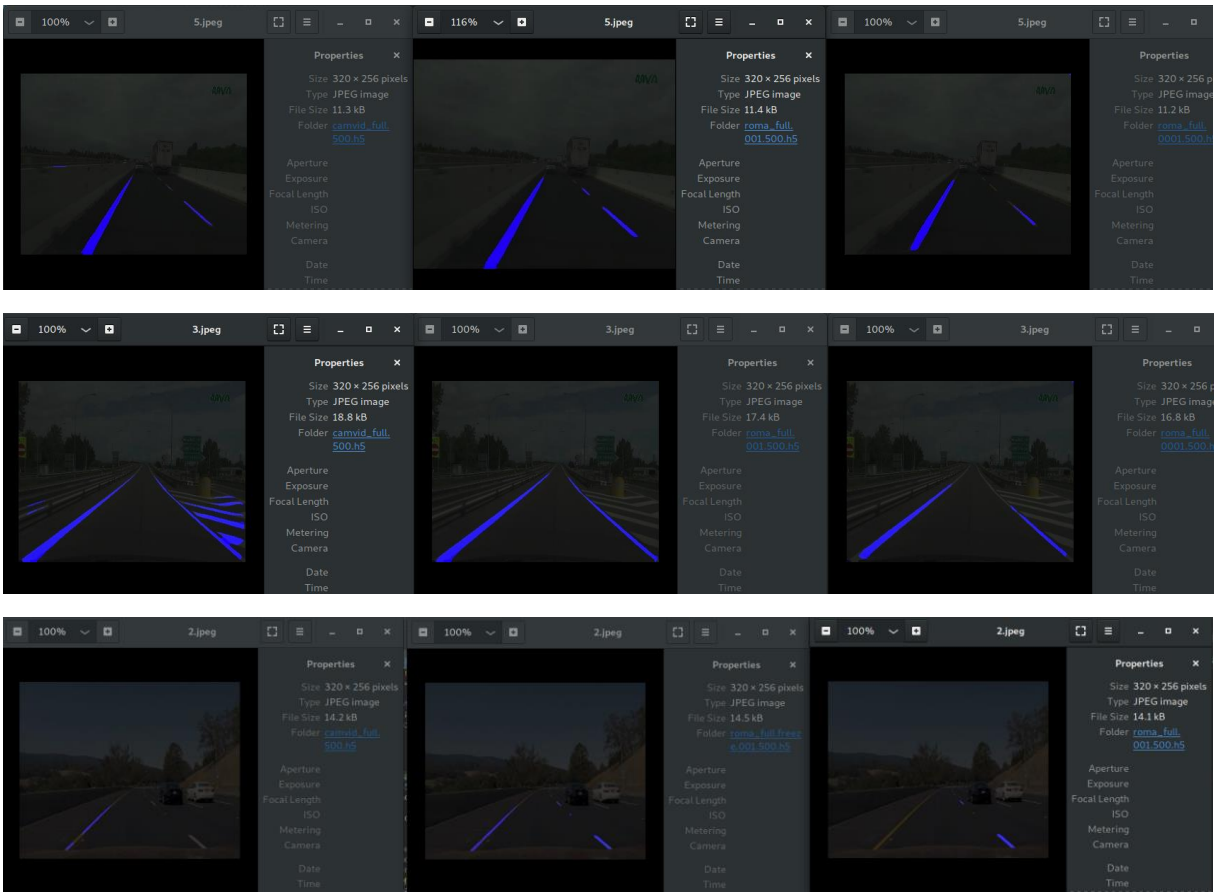
A tesztképeken átlagosan pozitívnak jelölt pixelek száma (TP + FP):

Camvid	Roma-1	Roma-2	Roma-3
1247,142857	1547,952381	1106,380952	1459,71429
170,7	96,8	69,8	239,4
792,49	927,01	546,21	795,91

Az eredmények nagyjából egyeznek a szemmel látható tapasztalatokkal. Alapvetően a Roma-1 teljesített a legjobban általános terepen. A Camvid modell rosszabbul teljesített, viszont könnyebben felismer bizonyos speciális felfestéseket, amiket (pl. sárga felfestések, a szokásosnál jóval vastagabb vonalak, kereszt/átlós irányú felfestések). Ezt a tulajdonságát örökölte a Roma-3-as modell is, ami valamivel rosszabb eredményt hoz, mint a Roma-1, viszont ezekkel a speciális felfestésekkel szignifikánsan jobban boldogul. Ezt mutatja a táblázat 2. sora is, ami egy nagyon világos képsor, kopottszürke aszfalttal, sárga szélső vonallal.

Néhány, a fontosabb tapasztalatokat jellemző összehasonlítás:





Sávfelismerő: transfer-learning teszt

Ugyan még nincs rendes tanítás a sávfelismerő modellhez, de 300 képen lefuttattam néhány epochnyi teszttanítás, hogy lássam, rátanul-e, Ennek a kísérletnek a lényege az, hogy van-e érdemi hatással a tanításra, ha egy hasonló – felfestések detektálása -, de nem azonos feladattal előretanított súlyokkal inicializálunk tanítás előtt.

A megfigyelés az volt, hogy valamivel gyorsabban közelített az optimum felé, és sokkal kisebb kilengésekkel változott a loss értéke.

Egyéb

Továbbfejlesztési lehetőségek

- Tanítás befejezése
- Próbálkozás más adathalmazzal (KITTY? Cityscapes)
- Sáv vonalainak kijelölése, csoportosítása OpenCV segítségével (ez a feladatrész jelenleg erősen hiányos)

További felhasznált irodalom

http://www.deeplearningbook.org/

https://keras.io/

Kód elérhetősége

https://github.com/boti996/onlab-public

Tartalomjegyzék