Félig árnyékos rendszámtábla megtalálása képen

Kis Ádám

December 2019

1. Áttekintés

A feladat továbbra is félig árnyékos rendszámtáblák pozíciójának megtalálása egy képen. A feladatot nagyban nehezíti, hogy a képek különböző országban, különböző kamerákkal, különböző felbontással készültek, némely kép fekete-fehér, de a többség színes. Ezeket már az korábbi beadandóimban taglaltam, itt nem térnék ki rá bővebben.

2. Problémák

Ebben a félévben az alábbi problémákra kerestem megoldást:

- teljesítmény javítása
- pontosság javítása
- példahalmaz kiegyensúlyozása
- példahalmaz növelése
- eltűnő gradiens
- túlillesztés

3. Háló architektúra

A háló felépítését több fajtából állítottam össze:

- reziduális kapcsolatok a ResNet[4]-ből
- egy inception modulra hasonlító réteget a GoogleNet[8] inspirálta
- Region proposal-t az RNN[6] architektúra terjesztette el

A háló rétegei:

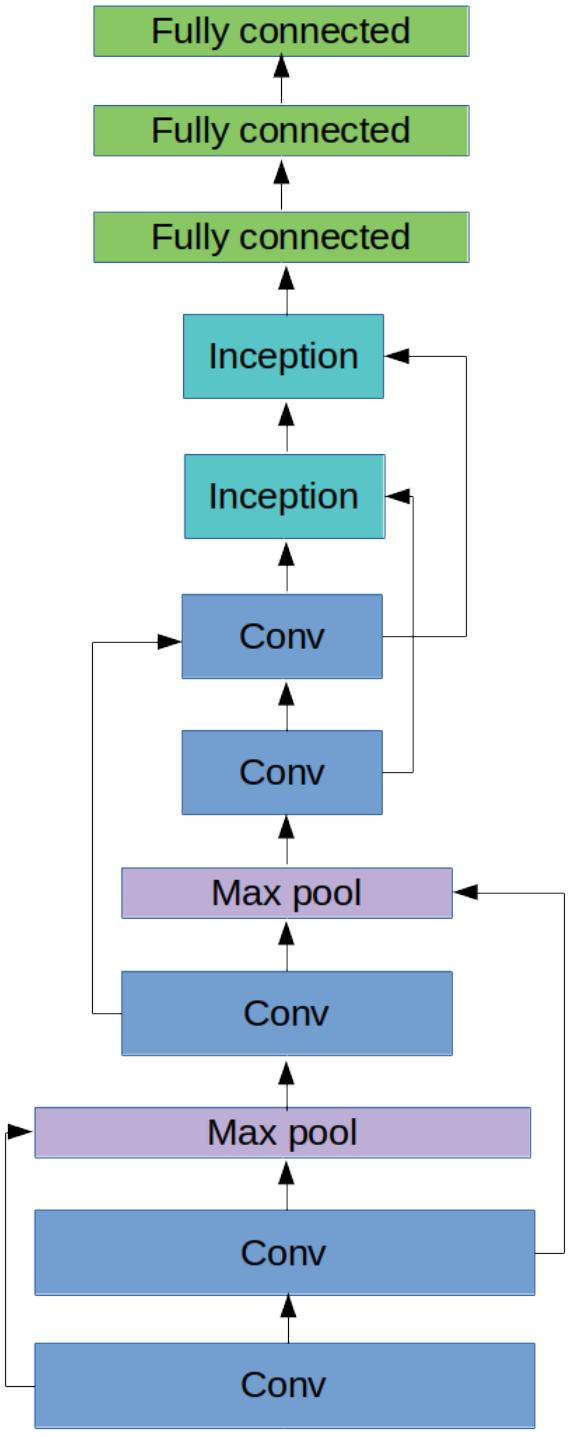
Réteg típusa	Mérete
Convolution	7x7x64
Convolution	3x3x64
Max pool	2x2
Convolution	3x3x128
Max pool	2x2
Convolution	3x3x256
Convolution	1x1x64
Inception	Lásd alább
Inception	Lásd alább
Fully connected	192
Fully connected	192
Fully connected	192

A reziduális kapcsolatok minden esetben pontosan egy réteget ugranak át és egyszerűen hozzá vannak fűzve a következő réteg kimenetéhez (ezzel szemben a ResNet[4] összeadja őket a ReLU előtt).

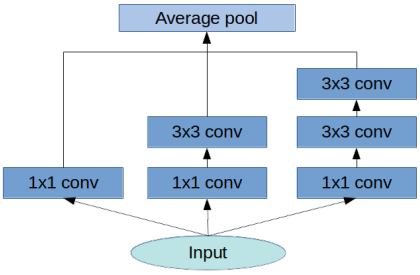
3.1. Inception modul

Bár a publikáció címe amiben bevezették "Going Deeper with Convolutions" (érdekesség: a cím a "need to go deeper", Inception című filmből vett képre (mémre) való utalás, a neve pedig arra utal, hogy ez gyakorlatilag háló a hálóban), az inception modul koncepciója inkább az, hogy a mélység helyett szélességet növeli. Ha egy réteg 3x3-as konvolúciót tartalmaz, ha mondjuk 5x5-ös kiterjedésű információra van szüksége, azt csak egy másik réteg segítségével tudja kinyerni. A háló mélyítése azonban "eltűnő gradiens" prolémát okoz. Erre megoldás, hogy egy rétegben miért ne lehetne 5x5-ös, 7x7-es konvolúció is?

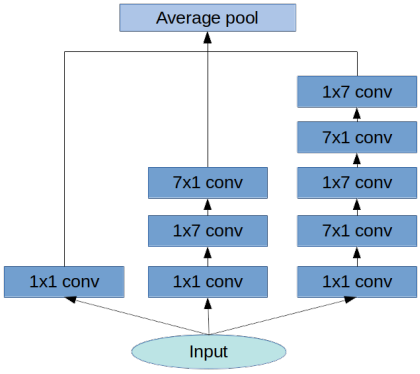
Az eredeti modultól itt is eltértem egy kicsit, remélve hogy csökkenthetem az erőforrásigényét a műveletnek. Az esetemben 7x7-es kernelek ki lettek cserélve 5x5-ösekre 2-es dilatációval.



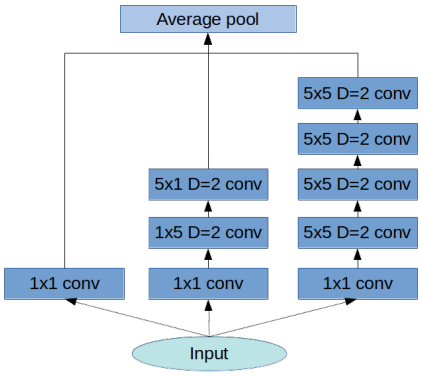
1. ábra. A háló 5 konvolúciós, 2 max pool, 2 inception és 3 teljesen összekötött réteget tartalmaz. Az oldalsó nyilak a reziduális kapcsolatokat jelölik.



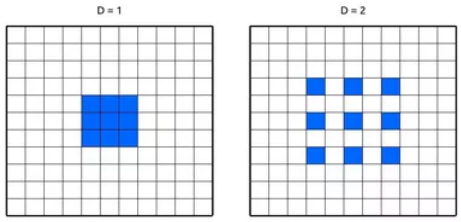
2. ábra. Inception A modul



3. ábra. Inception B modul. A 7x1-es és 1x7-es konvolúciók jóval könnyeben számolhatóak, mint a 7x7-esek, teljesítményük azonban nem marad el jelentősen.



4. ábra. Inception C modul - itt a 7x7-es konvolúció ki lett cserélve 5x5-ösre dilatációval (a kihagyás mértékét a plusz jel utáni számmal jelzem)



5. ábra. Dilated convolution. Esetemben az inception C modul jobb oldali ágában D=2, a többiben D=1 (azaz máshol nincs dilatáció).

4. Teljesítmény

A betanításhoz egy GeForce 1080Ti 11GB videókártyát használtam. Teljestményben nem sokban marad el a gépi tanuláshoz használt GPU-k teljesítményétől (ez annyira nem is fontos, legfeljebb kétszer annyi ideig tart a betanítás, bár ezzel természetesen limitálja egy szemeszter alatt kipróbálható háló architektúrák számát), a 11GB-os memória azonban komoly korlátot jelent a háló méretét tekintve. A szokásos objektum lokalizáló neurális hálózatok jellemzően minimum 32 GB-ot igényelnek.

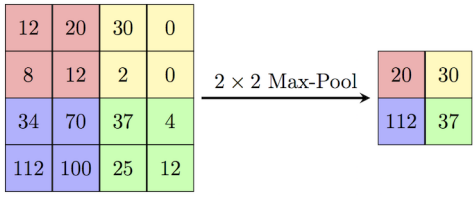
A problémát úgy oldottam fel, hogy a képet csúszóablakban dolgozom fel[2]. Mivel a rendszámtáblák szélesebben mint amilyen magasak (a pontos arányok országonként eltérőek lehetnek), ezért ennek méretét 300x150-esnek választottam. Hogy mindenképpen legyen olyan eset, amikor az egész rendszámtábla benne van, az ablakot 50 pixelenként mozgatom.

A nagy kép visszaállítása során több befoglaló téglalapot is készíthet ugyanarról a rendszámtábláról, ezt a korábbi félévben már ismertetett non-max suppression algoritmussal oldom fel. Esetemben ez csak a kimenet feljavítása után segít, lásd 11. fejezetet.

5. Pontosság javítása

A hálóm 2 maxpool réteget tartalmaz, egyet a 2 réteg után, egyet a harmadik után. Ezek úgy működnek, hogy 2x2-es darabokra bontják a képet, majd minden ilyen egységből csak a maximális értéket engedik tovább, ezzel negyedelve a kép méretét. Mivel az elvégzendő műveletek száma egyenesen arányos a kép méretével, ezért a használatával

mélyebb hálót lehet készíteni ugyanannyi erőforrás felhasználásával.



6. ábra. A max pooling x és y irányban is felezi a kép méretét. Kép forrása: https://computersciencewiki.org/

Ennek a megoldásnak azonban van egy hátránya: mivel csak a legnagyobb értéket küldi tovább, de annak pozícióját nem, elveszik valamennyi helyinformáció. Ez korlátot szab annak, hogy a háló mennyire tud maximálisan pontos lenni. Az én esetemben két réteg van belőle, azaz a háló maximum 4 pixel pontosságú eredményt tud adni. Erre megoldásnak azt találtam ki, hogy előrecsatolom minden neuron maximális kimenetét és annak pozícióját a teljesen összekötött rétegbe, ami ezt fel tudja használni a pontosításra. A betanulást megkönnyítendő minden pozíció 0 és 1 közé van normálva, így a különböző rétegből jövő pozícióértékek nagyjából átfedik egymást: a (0.5, 0.5) az első rétegben is ugyanazt jelenti mint az utolsóban (persze a felbontás más lehet).

Nincs pontos adatom arról, hogy ennek volt-e eredménye, kárt biztos nem tesz. Legrosszabb esetben a teljesen összekötött rétegek figyelmen kívül hagyják.

6. Példahalmaz kiegyensúlyozása

A csúszóablaknak van egy komoly hibája: csak kis részében lesz ténylegesen rendszámtábla. Ez két problémát is okoz:

- a háló túl sok "false negative" eredményt ad, mivel az eredmény valamennyire tükrözi a példahalmaz eloszlását
- lassabb lesz a betanulás, mert csak a példák kis hányadát tudja az objektum lokalizáció megtanulására használni

Ezekre megoldásként létrehoztam egy származtatott példahalmazt, ami a csúszóablakkal készített kisebb képekből áll, csak arányaiban sokkal több olyan példát tartalmaz, ahol rendszámtábla is megtalálható. Mivel a *sample augmentation* segítségével szinte végtelen sok (nyilván gyakorlatban véges, de hatalmas számú) ilyen kép generálható, időről időre újra kell generálni a példahalmazt, hogy a "sample augmentation" szerepe is érvényesüljön.

7. Eltűnő gradiens

A mély hálózatok egy nagy problémája, hogy a visszaterjesztett gradiens vagy eltűnik, vagy felrobban, azaz vagy túl kicsi vagy túl nagy lesz mire az alsóbb rétegekig elér. Erre már korábban is használtam a batch normalization[5] nevű réteget, de létezik egy másik technika, amit eredetileg a ResNet vezette be: a kapcsolatok kihagynak egy réteget. Ez a fajta kapcsolat megtalálható az agykéregben is. (Személyes, nem professzionális véleményem: a gépi tanulás közössége hajlamos cargo cult-ként kezelni mindent, ami az emberi agyban is megtalálható, lásd a mostanában divatból kiment local response normalization-t - persze ötletet meríteni nem hiba, de mindig érdemes meggondolni, hogy ez most valóban működik-e vagy csak mindenki a másikat másolja - persze eztnem mindig könnyű eldönteni.)

Ennek a megoldásnak több előnye is van:

- rétegek nem csak a közvetlenül alattuk lévő réteget használhatják fel bemenetként
- a gradiensnek van olyan útja, ahol kevesebb rétegen kell keresztül mennie, így kisebb esélye van eltűnni vagy felrobbanni

Itt én egy réteget kihagyó "skip connection"-öket használtam, így a visszaterjedő hiba potenciálisan akár feleakkora úton is el tud jutni az alsóbb rétegekbe.

Tapasztalatom szerint ez rengeteget segített. Sajnos nincs meg az erőforrásom (idő és hardware) a pontos összehasonlításhoz, így pontos adatot nem tudok közölni.

8. Túlillesztés

A háló betanítása során tapasztaltam, hogy a kiértékelő halmazon a háló teljesítménye nagyon távol állt a betanítóhalmazon látott eredménytől. Ez kis példahalmazon normálisnak mondható. Esetemben 499 példa állt rendelkezésemre, ami elég messze áll

a szokásos, sokszor több mint egy milliót tartalmazó adathalmazokhoz képest. Erre két megoldást használtam:

- dropout
- sample augmentation

8.1. Dropout

Ez a technika abból áll, hogy a neuronok kimeneti képének egy adott hányadát véletlenszerűen 0-ra állítjuk, ezzel gyakorlatilag alul-mintavételezzük a hálónkat. A megoldást Srivastava, Nitish, et al.[7] vezették be és azóta is alapvetőnek számít minden mélytanuló algoritmus implementálása során.

Gyakorlatban ez úgy néz ki, hogy generálunk egy, a neuron kimenetével megegyező kiterjedésű tenzort, aminek minden eleme 0 vagy 1 és ezzel elemenként összeszorozzuk a kimenetet. A nullák és egyesek arányát egy paraméter adja meg. Az én esetemben ez 0.5 a konvolúciós rétegekben, 0.3 a teljesen összekötöttekben (azaz 30% nullát tartalmaz) és 0 a kimenetiben.

8.2. Sample augmentation

A példahalmazt különböző módszerekkel gyakorlati szempontból növelni lehet. Például objektum lokalizáció esetében a kép eltoltja is érvényes bemenet lesz. Ezeket a technikákat már korábban ismertettem, most csak annyit tettem hozzá, hogy véletlenszerűen átméretezem a képet egy arányszám szerint, amit levágott normális eloszlásból húzok:

Minimum	0.7
Maximum	1.15
m	1
σ	0.2

Ezzel valamennyire változatosabb lesz a rendszámtáblák mérete enélkül, hogy a jellemző kiterjedést megváltoztatnám. Megjegyzés: sajnos ez valamennyi numerikus stabilitási hibát is okoz (a numpy multiply függvényét használtam).

9. A háló kimenete

A program két dolgot csinál:

- megmondja, hogy van-e a képen rendszámtábla - hiszen nem minden ablakban lesz
- ha van, visszaadja annak befoglaló téglalapját

Ennek megfelelően a kimenet egy 5 elemű tömb: $[x_1, y_1, x_2, y_2, c]$, a befoglaló téglalap bal felső, jobb alsó koordinátái és a háló bizonyossága, hogy a képen van-e rendszámtábla. A befoglaló téglalap pozíciója 0 és 1 közé van normalizálva, ahol a (0.0, 0.0) a bal felső, az (1.0, 1.0) a jobb alsó sarkot jelenti. A helyzet hibájához a szokásos L2 távolságot használom, az osztályozáshoz azonban ez nem elegendő.

9.1. Osztályozási hiba

A c-t szigmoid aktivációs függvény használatával szorítottam be a $[0, 1]$ intervallumba. Ennek képlete:

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

Deriváltja rendkívül egyszerűen számolható:

$$f'(x) = f(x)(1 - f(x)) \tag{2}$$

Ennek a problémája a "szigmoid szaturáció", azaz hogy a deriváltja 0.5-től eltérő x esetén alacsony, ami lassítja a háló betanítását. Erre megoldás, hogy a hibafüggvény a kereszt entrópiát használja:

Legyen c_n a háló kimenete és c_l a ground truth. Itt $c_l = 1$, ha van rendszám a képen, 0 ha nincs. Ekkor az l veszteség

$$l = -c_l \log c_n - (1 - c_l) \log(1 - c_n) \tag{3}$$

Illusztrációnak vegyünk egy példát: legyen az elvárt érték 1, a háló kimenete azonban 0.05. Ekkor $x = -2.94$. Ha a veszteséget a szokásos négyzetes hibával számolnánk, akkor az

$$l = (1 - f(x))^2 \tag{4}$$

x szerinti deriválja:

$$l' = 2(1 - f(x))f(x)(1 - f(x)) = 0.09025 \tag{5}$$

Legyen most $c_n = 0.5$. Ekkor a visszaterjesztett hiba 0.25. Látni lehet ez, hogy nagyobb eltérés esetén egyre kisebb, ami nem kívánatos tulajdonság. Ezt olja meg a kereszt entrópia. Előző példákat használva, ha $c_n = 0.05$

$$l' = -\frac{1}{f(x)}(1 - f(x))f(x) = 1 - f(x) = 0.95 \quad (6)$$

Ha $c_n = 0.5$ akkor pedig $l' = 0.5$.

Látható, hogy a kereszt-entrópia lineárissá teszi a deriváltat a szigmoid kimenete arányában, így a szigmoid szaturáció problémája eltűnik.

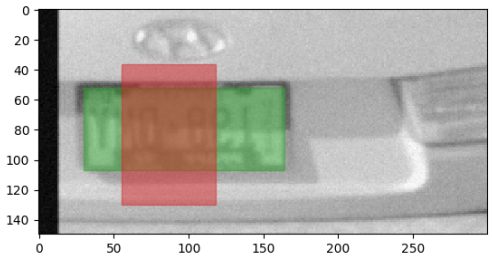
10. Keresési tér csökkentése

Az ablakozás hátránya, hogy jelentősen lassítja a kép feldolgozását. Egy 1024x768 kép 15x13 kisebbre bomlik le (átfedésekkel), így a hálót 195x kell lefuttatni egyetlen képen. Ez videók feldolgozásához sok. Az én gépem kb. 60 ablakot tud másodpercenként feldolgozni, azaz a fentebbi példát nagyjából 3mp alatt tudja kiértékelni. Ezen javítandó találták ki a "region proposal" [6] hálót. Ennek lényege, hogy az alsó rétegek ROI-kat (region of interest) vágnak ki a képből, és csak ezeket értékelik ki a felsőbb rétegek. Ezzel a megoldással a teljes képpel csak a háló egy része dolgozik, a többinek a jóval kisebb ROI-kon kell csak műveletet végeznie.

Az én esetemben ennek a megoldásnak egy nagy hátránya, hogy a ROI-t javasló és a felsőbb rétegnek egyszerre kell betanulnia, ami a GPU memóriájának limitációja miatt nem lehetséges. Megoldásnak azt alkalmaztam, hogy egy kisebb classificator minden ablakra megmondja, hogy van-e benne rendszámtábla és csak ezeket küldöm tovább a nagyobb hálónak. Ez a megoldás azonban számomra nem bizonyult sikeresnek, valószínűleg a fentebb említett probléma miatt.

11. Kimenet javítása

A kimenettel kapcsolatban azt figyeltem meg, hogy bár a rendszámtábla középpontját gyakran sikeresen meghatározza, annak kiterjedését már ritkábban találja el. Egy példa látható az 7 ábrán.

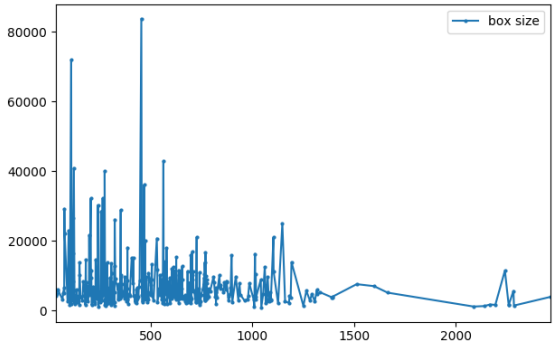


7. ábra. Egy példa arra, amikor a rendszámtábla közepét nagyjából jól ismerte fel, annak kiterjedését azonban nem.

Erre megoldás lehet, hogy egy közúti kamera esetében előre jól meghatározható, hogy a rendszámtábláknak a kép adott részén mekkorának kell lennie.

8. ábrán látható, hogy az általam használt adathalmazban bár összefüggés van az y koordináta és a befoglalótéglalap mérete között, a korreláció nem elég erős ahhoz, hogy ennek segítségével feljavítsam az eredményt. Gyakorlatban azonban ez a fajta hiba jól kiküszöbölhető.

A kimenet javításához még egy csalást használtam feltételezve, hogy a rendszámtábla mérete adott, a bemeneti példában használt rendszámtábla mérete alapján kijelölök egy régiót, amin belül csak a legnagyobb bizonyosságú kimenetét veszem a hálónak, es azt fogadom el. A régió mérete: rendszámtábla magassága x3, szélessége x2.



8. ábra. A rendszámtábla mérete (pixelben) és a vertikális pozíció kapcsolata. Pearson korrelációs együttható: -0.1318

12. Eredmények

Metrika	Érték
Accuracy	0.9022
False negative ratio	0.0986
False positive ratio	0.0216
Average midpoint distance	0.3161
Average midpoint distance after non-max suppression	0.1532

Magyarázat:

- Accuracy: egy adott ablakon belül ha a háló rendszámtáblát érzékelt, annyak mekkora részében tartalmazott ténylegesen
- False negative ratio: a pozitív példák mekkora hányadát érzékelte negatívnak
- False positive ratio: a negatív példák mekkora hányadát érzékelte pozitívnak
- Average midpoint distance: Euklídészi távolsága az eredeti és a háló által kiadott téglalapnak. Itt a kimenet 0 és 1 közé van normálva, azaz a bal felső sarok a (0, 0) pont, a jobb alsó az (1, 1). A távolság a legközelebbi címkéhez van viszonyítva.

Jól látható, hogy annak kitalálása, hogy az adott ablakban van-e rendszámtábla jelentősen könnyebb feladat, mint annak pontos pozíciójának megtalálása, holott azt gondolnánk, utóbbihoz kell az előbbi.

12.1. Megjegyzés az eredményekhez

A kis adathalmaz miatt nem használtam keresztvalidációt és az alfa értékét (azaz hogy mekkora confidence felett tenkintsük úgy, hogy rendszámtáblát érzékelt a háló) is utólag löttem be 0.5-re, ami azért komoly publikációban elfogadhatatlan lenne (remélem), de gyakorlatban nem biztos: például az alfát be lehetne állítani egy tesztidőszak alatt.

A batch mérete a kiértékelés alatt is 10 maradt, de ezt a csúszóablak miatt nem tartom problémának, mivel a kép mindig legalább 10 kisebbre van bontva. Ennek megfelelően a batch normalization a tanításnak megfelelően volt beállítva, nem pedig csúszóátlaggal. Nem tudom ennek van-e jelentősége, de eltér a szokásostól.

13. További irány

Rengeteg lehetőség van még a fejlesztésre:

1. mindenképpen érdemes megpróbálkozni mélyebb hálóval - általában segít
2. nagyobb adathalmazon való betanítás mindenképpen javasolt, főleg amiatt, hogy minden próbálkozást ezen végeztem, és kiegyensúlyozott adathalmazt készíteni nehéz. Pont pár napja volt egy hír, hogy a legjobb klasszifikátorokon is ki tudnak fogni új képek[1]
3. region proposal használata a kiértékelés sebességének növelése érdekében külön kihívásokat tartalmaz
4. továbbra is vágyam egy neural turing machine kipróbálása[3], bár ezek sajnos nem futnak túl hatékonyan a videókártyákon, valószínűleg ez is az oka annak, hogy bíztató eredmények dacára nem nagyon hallani gyakorlati hasznukról
5. azt is jó lenne látni, hogy meddig javítható az eredmény pusztán azzal, hogy a hálót tovább tanítjuk - nem tudom, hogy elérte-e a reprezentációs képessége határát

14. Megjegyzés

Nem tartozik az e félévi munkámhoz (azért valamennyire igen, hiszen meglévő kódot bővíteni legtöbbször plusz munkával jár), de a teljesség igénye miatt megemlítem az alábbi, korábbi félévekben leimplementált módszereket:

- az első réteg Gábor-wavelet-re és élkeresőkre volt inícializálva
- sample augmentation módszerek közül mindent használok, amit korábban írtam
- minden ReLU előtt batch normalization réteg van
- az input pipeline az egyetlen, fent említett véletlenszerű átméretezést leszámítva teljesen megegyezik a korábbiakkal
- a belső rétegeket ortogonális kernelekre inícializáltam

Hivatkozások

[1] This object-recognition dataset stumped the world's best computer vision models. http://news.mit.edu/2019/object-recognition-dataset-stumped-worlds-best-computer-vision-models-1210. Accessed: 2018-05-10.

[2] NI Glumov, EI Kolomiyetz, and VV Sergeyev. Detection of objects on the image using a sliding window mode. *Optics & Laser Technology*, 27(4):241–249, 1995.

[3] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.