Félig árnyékos karakterek megtalálása, felismerése

Kis Ádám

December 2017

1. Feladatleírás

A probléma félig árnyékos rendszámtáblák leolvasása képről. Ezt két lépésben kell megvalósítani:

- rendszámtábla megtalálása a képen
- rendszámtábla elolvasása

Ebben az első rész a feladat. Ehhez kaptam 500 db képet és hozzájuk tartozó címkéket, amelyek a rendszám pozícióját tartalmazzák az adott képen, 4 csúcsú poligon formályában.

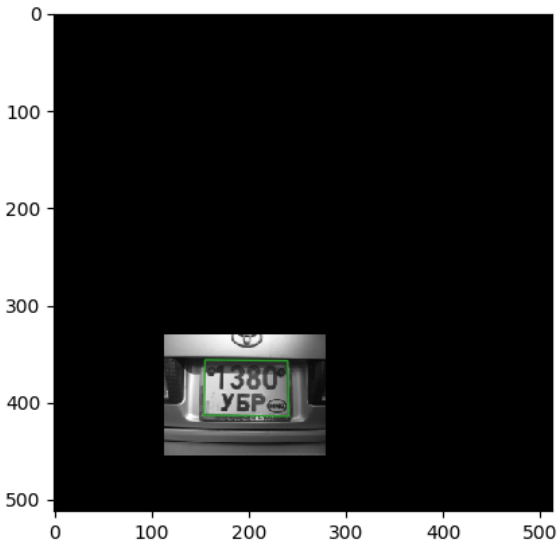
1. ábra. felcímkézett kép

Ez alapján kellett olyan algoritmust írnom, amely nem felcímkézett képen is megtalálja a rendszámtábla pozícióját. Ehhez a deep learning megközelítést választottam.

2. Előfeldolgozás

Az én esetemben az 500 példa között megtalálható fekete-fehér, színes kép, különböző felbontásokkal, ezért ezeket kell először azonos formára hozni. Mivel fekete-fehér képből nehéz (bár nem lehetetlen[3]) színes képeket csinálni, inkább a fordított utat választottam. A felbontást tekintve kísérleteztem 256x256 és 512x512-es felbontásokkal is. Ha a kép nagyobb, akkor egyszerűen átméreteztem. Egy képpont értéke az átméretezett képen szerűen a legközelebbi képpont értéke az eredeti képen.

Ha a kép kisebb, akkor két lehetőség van: átméretezni a képet a fenti módszerek valamelyikével, vagy kitölteni feketével a maradék részt (vagy is nullákkal). Itt mindkettővel próbálkoztam. A nullával kitöltést úgy csináltam, hogy véletlenszerűen válaszottam ki a kép hova kerüljön a fekete alapra, hogy elkerüljem, hogy a példahalmazban túl sok képen legyen a bal felső sarokban a rendszám.



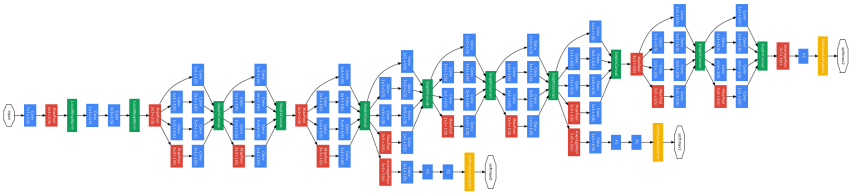
2. ábra. padded image

Ezek után még a képek standardizálása következett, azaz minden képpont értékéből kivontam az átlagot és elosztottam a szórással[4]:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{1}$$

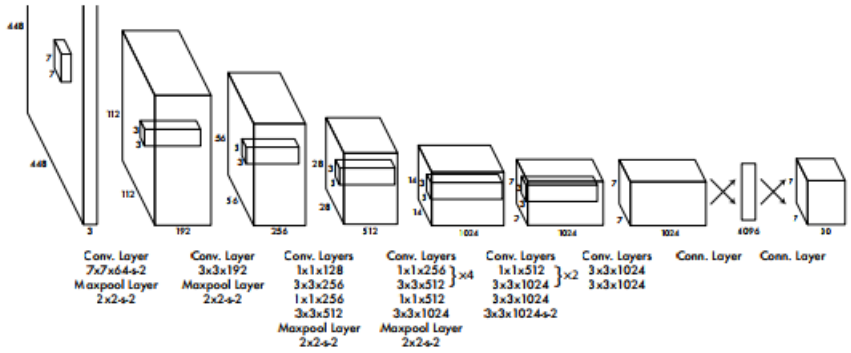
3. Háló architektúra

Az architektúra választásában legnagyobb szerepet a korlátozott erőforrásaim játszották. Mivel csak egy GeForce 1070 Ti videókártya áll a rendelkezésemre, az olyan hatalmas hálózatok használata mint az Inception[5] vagy a fast R-CNN[8] megoldások számomra nem ideálisak.



3. ábra. Inception architektúrája

Így eljutottam a viszonylag egyszerű YOLO architektúrához[7]. Ez tulajdonképpen nem különösebben forradalmi, a legegyszerűbb elemekhez tér vissza. A célja, hogy a költséges (bár hatékonyabb) megoldások helyett a kis hálóval is közel ugyanolyan eredményt érjen el.



4. ábra. YOLO architektúrája

Ahogy látható az 4. képen, az adatnak nincs több különböző útja mint az Inception esetében. (Megyjegyzés: ezeknél a több út szándékos: így el tudják könnyen osztani a feladatot több videókártyára – ennek számomra nincs akkora

előnye.) A háló kimenete 8 lebegőpontos szám, amelyek a rendszámtáblát leíró poligon 4 csúcsát jelentik.

4. Megvalósítás

A megvalósítást TensorFlow keretrendszer segítségével csináltam. Bár ilyen egyszerű architektúra implementációjára egy Keras-hoz hasonló jobban megfelelt volna a célnak, de ez rugalmasabb, hogy ha a jövőben egzotikusabb hálókat is ki szeretnék próbálni[6].

Az előfeldolgozás állítható számú párhuzamoson szálon fut (gyakorlatban 4 szálon futtattam – e fölött nem volt változás a feldolgozás sebességében), ezek adják be a példákat a hálónak. A háló tanulása batch learning[2] módszerrel történik, ahol a batch mérete állítható, ez egy hyperparameter.

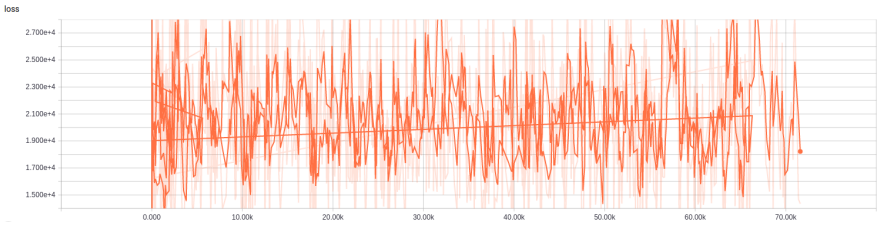
5. Hyperparameter tuning

További két hyperparameter: learning rate és momentum[10]. Ezeket próbáltam beálltani, de ezt nagyon nehéz egy videókártyával: ezt úgy szokták, hogy egy sok GPU-val rendelkező szervergépen elindítják a tanulást különböző paraméterbeállításokkal, majd ránéznek 1-2 nap múlva vagy akár 1 hét múlva és kiválasztják azt, amelyik a legjobban teljesített. Esetemben ez úgy nézett ki, hogy egy racionálisnak tűnő paraméter beállítással tanítottam egy éjszakát a hálót, majd ha nem adott jobb eredményt akkor másikkal próbálkoztam. Ez alapján a következő értékekre jutottam: learning rate 1e-4, momentum is 0.1 (a végén csak vettem az értékeket az eredeti publikációból[7], de próbálkoztam különböző értékekkel, nem sok sikerrel), batch size pedig 10. Mivel, mint említettem, nehéz megmondani az optimális értéket, ezek valószínűleg nem túl jók. A jövőben érdemes erre is koncentrálni, mert nagyban befolyásolhatják a végeredményt.

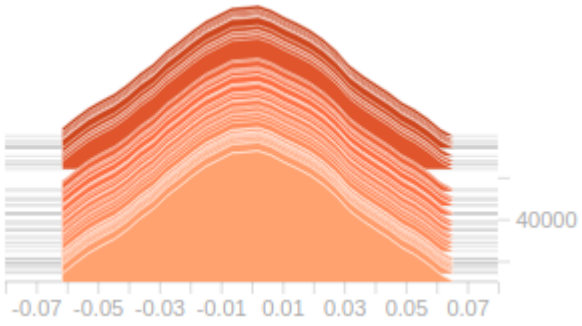
6. Eredmény

Eljutottam oda, hogy egy referencia implementációt elkészítettem, ez tanul ugyan, de nem jut sokra. Az eredmény mindössze annyi, hogy a háló bias-ait (eltolás paramétereit) beállítja úgy, hogy a kép közepére kerüljön a kimeneti poligon (a rendszámtábla pozíciója a háló szerint), de innen nem konvergál a model tovább. A háló alsobb rétegei elkezdenek specializálódni, de a többi réteg nem.

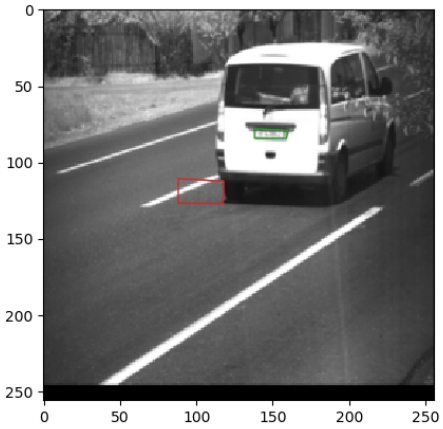
Ahogy a 6. ábrán is látszik, a második réteg súlyainak eloszlása nem változik az iterációkkal. Ez azt jelenti, hogy ez a réteg nem tanul. A neurális hálóknál jellemző, hogy az alsó rétegek tanulnak be először, mert addig a felette lévő rétegeknek nincs mi alapján dönteni, de itt ez igen sok iteráció után is így marad.



5. ábra. Veszteség értékei az iterációk függvényében - 70ezer ciklus után sincs változás



6. ábra. A második konvolúciós réteg súlyainak hisztogramja. A mélység az iterációk számát jelenti.



7. ábra. Egy tipikus eredmény - pirossal a háló kimenete

7. További irány

Ez a megoldás egyelőre nem működik. Ennek számos oka lehet:

- kis méretű tanítóhalmaz
- hyperparaméterek rosszak
- architektúra nem ideális

A jövőben a következő megoldásokat tervezem kipróbálni:

- példa generálás: a képeket lehet torzítani, zajjal ellátni, kivágni belőle darabokat
- hyperparameter beállítása: erre is léteznek módszerek amelyekbe nem mentem bele, de első sorban időigényes. Mivel a programom működik (ha nem is ad jó kimenetet), a lehetőség adott, hogy különböző paraméterekkel próbálkozzak
- súlyok inícializálása: jelenleg a súlyok véletlenszerűen vannak inícializálva. Erre léteznek okosabb módszerek[4]
- képek előfeldolgozása: pl. szokványos éldetekció futtatása a képre először. Ennek alternatívája, hogy a háló legalsó rétegét úgy inícializálom, hogy eleve éldetektorok legyenek
- vonal pásztázó: függőleges pásztázó algoritmusként használni a hálót: egyszerre csak egy sort kap bemenetnek, és az alapján kell eldöntenie, hogy abban a sorban van-e rendszám. Ez gyorsabb lenne és ezáltal jobban tanítható korlátozott számítási kapacitás esetén
- előtanítás hasonló példahalmazon: a házszámok elolvasása pl. elég hasonló probléma. Ez egy bevett módszer (Karpathy említi a Stanford-on tartott CS231n kurzusán.) kevés példa esetén.
- Komponens alapú tanulás: be lehet tanítani a háló egy darabját pl járművek felismerésére, majd ezt be lehet kötni mint input a többi réteg számára
- befoglaló téglalap megtalálása poligon helyett
- capsule network[9]
- neural turing machine: különösen jól teljesít kevés példából tanulás esetén[1]

Szeretném hangsúlyozni, hogy ezen javaslatok nem zárják ki egymást, sőt, legtöbbször inkább kiegészítik.

Hivatkozások

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

[2] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, 2012.

[3] Jeff Hwang and You Zhou. Image colorization with deep convolutional neural networks.

[4] 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.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Aakash Nain. Tensorflow or keras? which one should i learn? https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0. Accessed: 2017-12-10.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[8] 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.

[9] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3857–3867, 2017.

[10] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.