Strojno učenje za boljšo povečavo slik

Blaž Rojc

11. november 2018

1 Uvod

Današnja fotografska oprema, od profesionalnih fotoaparatov do pametnih telefonov, je zmožna zajemati slike v visokih ločljivostih. Ker pogosto polna ločljivost slike ni potrebna, se večina slik pred shranjevanjem ali pošiljanjem zmanjša.

To samo po sebi ne bi bil problem, če bi se za spreminjanje velikosti uporabljali algoritmi, ki bi skušali čim bolj ohraniti kvaliteto slike. Ampak zaradi minimizacije stroškov procesiranja in shranjevanja veliko aplikacij uporablja bilinearno interpolacijo[1], ki zelo hitro spremeni velikost slike na račun kvalitete.

S ciljem boljše povečave slik sem ustvaril konvolucijsko nevronsko mrežo, ki vzame bilinearno interpolirano, v vsako dimenzijo štirikrat povečano sliko in vrne izboljšano verzijo z manj artifakti povečave. V tem besedilu bom predstavil delovanje nevronske mreže, njeno obliko, programsko okolje, ki sem ga uporabil, in množico slik, uporabljeno pri učenju in preizkušanju mreže.

Pri načrtovanju mreže sem se usmerjal po raziskavah s podobnim namenom.[1, 2, 3] V večini primerov je šlo za večslojne nevronske mreže, jaz sem se pa omejil na en sloj s konvolucijo.

Rezultati so sprejemljivi, mreža je zmožna očitno izboljšati kvaliteto slike v primerjavi z bilinearno interpolacijo samo.

2 Definicije

2.1 Konvolucija

Konvolucija [4] je računska operacija, podana s formulo

$$\text{izhod}(C_{izhod_j}) = \text{odmik}(C_{izhod_j}) + \sum_{\substack{C_{vhod_i} \\ \in C_{vhod}}} \text{utež}(C_{izhod_j}, C_{vhod_i}) \star \text{vhod}(C_{vhod_i}),$$

kjer sta C_{izhod_j} in C_{vhod_i} barvna kanala slike, $C_{vhod} = C_{izhod} = (\text{rdeč}, \text{zelen}, \text{moder})$, oznaka $\star$ pa označuje *diskretno korelacijo*, podano s formulo

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} \overline{f(m)} \cdot g(m+n).$$

[1] `https://en.wikipedia.org/wiki/Bilinear_interpolation`

Oznaka $\overline{f}$ nakazuje, da gre za kompleksno konjugiranko f, ampak ker so vsi vhodi, uteži in odmiki realna števila, uporabljamo pa le operaciji seštevanja in množenja, velja $\overline{f} = f$ in enačbo lahko zapišemo v obliki

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m) \cdot g(m+n).$$

Vemo, da ima funkcija uteži končno definicijsko območje, zato se omejimo le na k členov:

$$(f \star g)(n) = \sum_{m=0}^{k-1} f(m) \cdot g(m+n)$$

Število k imenujemo *jedro konvolucije*.

Slike so dvodimenzionalni objekti, v računalniku predstavljeni kot trojica dvodimenzionalnih matrik - vsaka predstavlja intenzivnosti enega izmed treh barvnih kanalov.[2] Torej moramo pojem konvolucije razširiti na dve dimenziji:

$$(f \star g)(x,y) = \sum_{m,n=0}^{k-1} f(m,n) \cdot g(x+m, y+n)$$

Vstavimo sedaj to formulo v definicijo konvolucije in računajmo za točko (x, y):

$$\text{izhod}(C_{izhod_j}, x, y) =$$
$$\text{odmik}(C_{izhod_j}, x, y) + \sum_{\substack{C_{vhod_i} \\ \in C_{vhod}}} \sum_{m,n=0}^{k-1} \text{utež}(C_{izhod_j}, C_{vhod_i}, m, n) \cdot \text{vhod}(C_{vhod_i}, x+m, y+n)$$

Opazimo: 'vhod', 'izhod' in 'odmik' so trodimenzionalne podatkovne strukture, 'utež' pa kar štiridimenzionalna. Take strukture imenujemo *tenzorji*.

2.2 Aktivacijska funkcija

Izraz *aktivacijska funkcija* označuje funkcijo, ki izhodno vrednost sloja mreže preslika v vrednost v želenem območju.[5] V globokih nevronskih mrežah se aktivacijske funkcije uporabljajo za vnos nelinearnosti, ki omogoča smiselno komponiranje operacij.[6]

Nekaj primerov aktivacijskih funkcij predstavljajo usmerjena linearna enota (ReLU):

$$f(x) = \begin{cases} 0 & \text{za } x < 0 \\ x & \text{za } x \geq 0, \end{cases}$$

sigmoid:

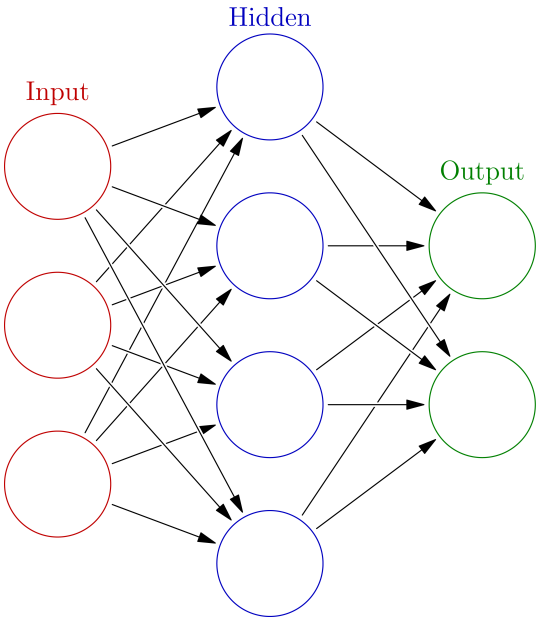
$$f(x) = \frac{1}{1+e^{-x}},$$

[2] `https://en.wikipedia.org/wiki/Raster_graphics`

in binarna stopnica:

$$f(x) = \begin{cases} 0 & \text{za } x < 0 \\ 1 & \text{za } x \geq 0. \end{cases}$$

2.3 Nevronska mreža



Slika 1: Primer nevronske mreže. vir: Wikipedia

Nevronska mreža je omrežje povezanih nevronov ali vozlišč.[7] Njena oblika se zgleduje po strukturi živalskih možganov.[8] Podobno se tudi "uči": Za vsak učni primer se prilagodi glede na razliko med njenim in ciljnim izhodom.

Prednost nevronske mreže v primerjavi s standardnimi algoritmi je, da za gradnjo mreže postopka iskanja rešitve ni potrebno točno definirati. Na razpolago moramo imeti le dovolj velik nabor parov vhodnih podatkov in pripadajočih izhodov - *učno množico* - ter dovolj računske moči. Posledično lahko z njimi rešujemo probleme, ki so preveč kompleksni za direktno algoritmično reševanje ali ko algoritma sploh ne moremo definirati, na primer klasifikacija slik, aproksimacija težko izračunljivih funkcij, regresijska analiza ...

Slabost nevronske mreže je njena nenatančnost. Večina nevronskih mrež ne doseže 100% natančnosti, kar pomeni, da niso primerne za življenjsko kritične namene. Poleg tega za doseganje dobrih rezultatov zahtevajo ogromne, raznolike učne množice in veliko računskega časa.[9]

2.4 Konvolucijska nevronska mreža

Konvolucijske nevronske mreže so eden izmed tipov usmerjenih nevronskih mrež. Navadno so sestavljene iz več slojev, kjer so posamezni sloji konvolucije, aktivacijske funkcije, združevalne funkcije ali *polno povezani sloji* - sloji, pri katerih je vsak nevron vhoda uteženo povezan z vsakim nevronom izhoda. Združevalne funkcije, kot so $\max(A)$, $\min(A)$ in $\text{avg}(A)$ zmanjšajo količino vmesnih podatkov, kar poenostavi proces učenja.[11]

Konvolucije emulirajo odziv nevronov na lastnosti vhodnih stimulusov, podobno kot ljudje bel krog na temnem ozadju zaznamo kot vir svetlobe.[10]

Pravilno načrtovane konvolucijske nevronske mreže potrebujejo relativno malo preprocesnega dela, kar omogoča hiter in enostaven razvoj programov, ki jih uporabljajo.

3 Oblika

3.1 Načrt mreže

Mreža je sestavljena iz dveh slojev, in sicer ene konvolucije z jedrom velikosti k ter aktivacijske funkcije pritrditve (ang. *clamp*), definirane kot:

$$clamp(x) = \begin{cases} 255 & \text{za } x > 255 \\ x & \text{za } x \in [0, 255] \\ 0 & \text{za } x < 0 \end{cases}.$$

Tu nelinearnost aktivacijske funkcije ne igra posebne vloge, pomaga pa omejiti izhodno vrednost v dopustno območje števil, ki jih lahko zaokrožimo v 8-bitno celo število.

Za vhod vzame povečano sliko dimenzij (širina, višina), vrne pa sliko dimenzij (širina$-k+1$, višina$-k+1$).

Izguba se računa s funkcijo *zglajene linearne izgube*, definirane kot:

$$\text{izguba}(\text{izhod}, \text{cilj}) = \frac{1}{n} \sum_{i=1}^{n} smooth(\text{cilj}_i - \text{izhod}_i),$$

kjer je funkcija *smooth* definirana kot:

$$smooth(x) = \begin{cases} 0.5x^2 & \text{za } |x| < 1 \\ |x| - 0.5 & \text{sicer.} \end{cases}$$

Ta funkcija ni toliko občutljiva na ekstremne primere kot MSE, ne potrebuje toliko računskega časa kot RMSE in da boljše rezultate kot vsota razdalj, ko se mreža približuje optimumu.[12]

Za optimizacijo se uporablja algoritem Adam. Ta je zelo podoben gradientnemu spustu, le da z uporabo dodatnih parametrov pohitri konvergenco k optimumu.[13]

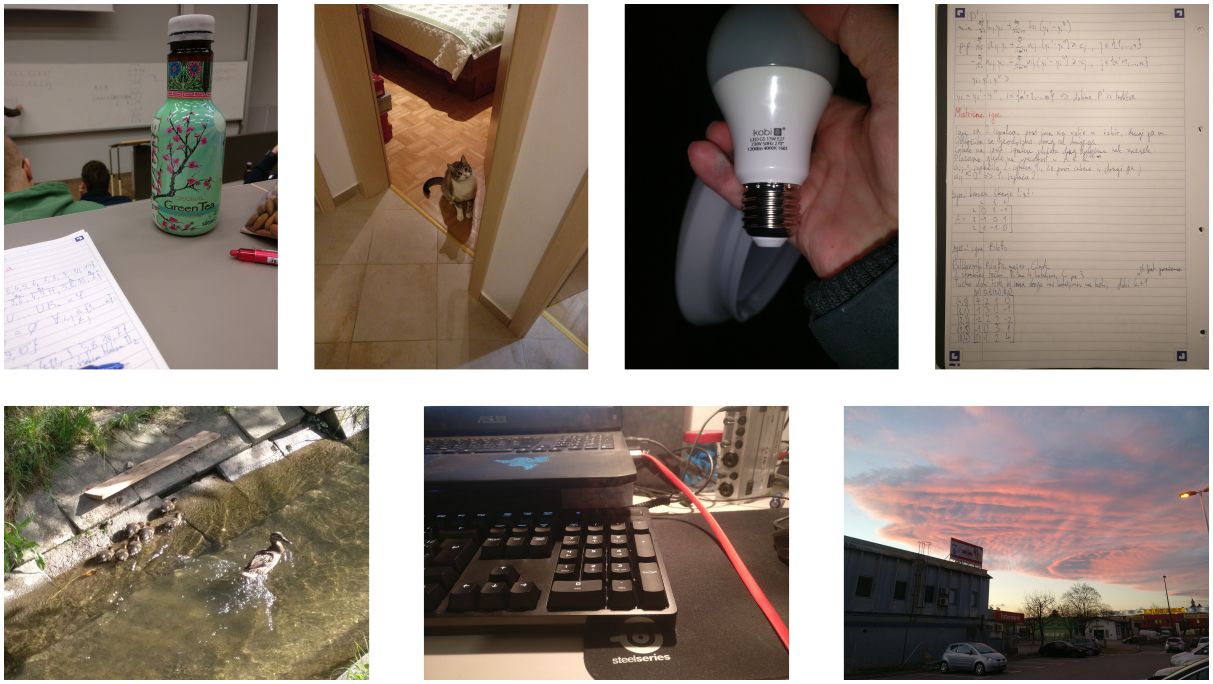
Vse računanje poteka v številih s plavajočo vejico v enojni natančnosti, `float32`.

3.2 Oblika podatkov

Za množico vhodnih podatkov sem izbral 644 slik, ki sem jih naredil z mojim pametnim telefonom v obdobju zadnjih dveh let in nekaj mesecev. Predstavljajo raznolik nabor, slikal sem ljudi, živali, naravo, zapiske, naključne predmete in še kaj.

Velikosti slik so različne, ampak večina jih je oblike (3480, 4640) ali (4640, 3480), kar predstavlja 16M pikslov. Stisnjene so z le rahlimi izgubami, vsebujejo pa zmerno količino statičnega šuma, kar sem pred učenjem poskušal omiliti z rahlim Gaussovim filtrom.

Slike so razdeljene na dva dela. Prvi del je učna množica 580 slik, drugi pa testna množica 64 slik. Slike so bile razdeljene naključno, s funkcijo `random.sample` v programskem jeziku



Slika 2: Primeri slik iz množice vhodnih podatkov.

Python 3.

4 Programsko okolje

Mrežo sem implementiral v programskem jeziku Python 3, uporabil sem knjižnico za strojno učenje PyTorch. Ker sem hotel izvajati učenje na grafični kartici, sem namestil tudi gonilnike za platformo CUDA in knjižnico cuDNN. Za delo s tenzorji sem uporabil knjižnico `numpy`, za delo s slikami pa knjižnico `scikit_image`.

Vsa izvorna koda je dosegljiva na javnem repozitoriju na naslovu `https://github.com/hackguy25/ImgResize`. Tam so zapisane tudi specifične verzije knjižnic in okolja.

5 Učenje

Mrežo sem učil s pomočjo skripte `train.py`. Vsako sliko sem naložil z diska, skrčil s faktorjem 4, povečal s faktorjem 4 z bilinearno interpolacijo in jo skupaj z originalom podal mreži. Ta je na grafični kartici izvedla učni korak z učno hitrostjo lr in vrnila napako pri trenutni sliki. To napako sem izpisal na standardnem izhodu za lažje sledenje napredku.

Učenje sem izvajal v iteracijah. V vsaki iteraciji sem na mreži izvedel en korak z vsako izmed 580 slik iz učne množice. Po vsaki iteraciji sem zmanjšal hitrost učenja, lr, tako da so posamezne slike le minimalno pripomogle k neželenemu odstopanju.

Vse učenje sem izvajal na mojem osebnem računalniku s procesorjem AMD Ryzen Thread-

ripper 1920X,[3] 32GB pomnilnika in grafično kartico Nvidia 1080 Ti.[4]

Ker je pretvorba slik potrošila veliko več časa kot samo učenje, sem postopek paraleliziral s pomočjo Pythonovega modula `multiprocessing`. V vsakem koraku sem naenkrat pretvoril 12 slik, vsako v svojem procesu, in izvedel učenje na 12 slikah iz prejšnjega koraka. Tako sem dosegel do 90% povprečno izkoriščenost procesorja in 60% povprečno izkoriščenost grafične kartice tekom postopka učenja.

Ustvaril sem dve različici mreže, ki se razlikujeta le v jedru konvolucije in času učenja.

Prva različica ima jedro velikosti $17 * 17$, učenje pa je potekalo v 15 iteracijah, kjer je bila hitrost učenja v i-ti iteraciji podana tako:

$$lr = 0.00001 \cdot 0,33^{i-1}$$

Druga različica ima jedro velikosti $25 * 25$, učenje pa je potekalo v 30 iteracijah, kjer je bila hitrost učenja v i-ti iteraciji podana tako:

$$lr = 0.00001 \cdot 0,85^{i-1}$$

Kljub večjemu jedru konvolucije druge različice učenje ni potekalo občutno počasneje.

Pri obeh različicah sem za izvedbo ene iteracije potreboval povprečno 12 minut. Za učenje prve različice sem porabil približno 3 ure, za učenje druge pa približno 6. Povprečna poraba energije računalnika med učenjem je nihala med 400 in 450W, torej sem za učenje obeh različic porabil nekje med 3,5 in 4 kWh električne energije.

6 Testiranje

Obe različici mreže sem testiral na 64 slikah iz testne množice. Slike sem predhodno skrčil na četrtino velikosti v vsako dimenzijo, nato pa jih obdelal z obema različicama mreže. Slika 3 prikazuje nekaj primerov povečanih slik, v primerjavi z bilinearno in bikubično interpolacijo.

7 Zaključek

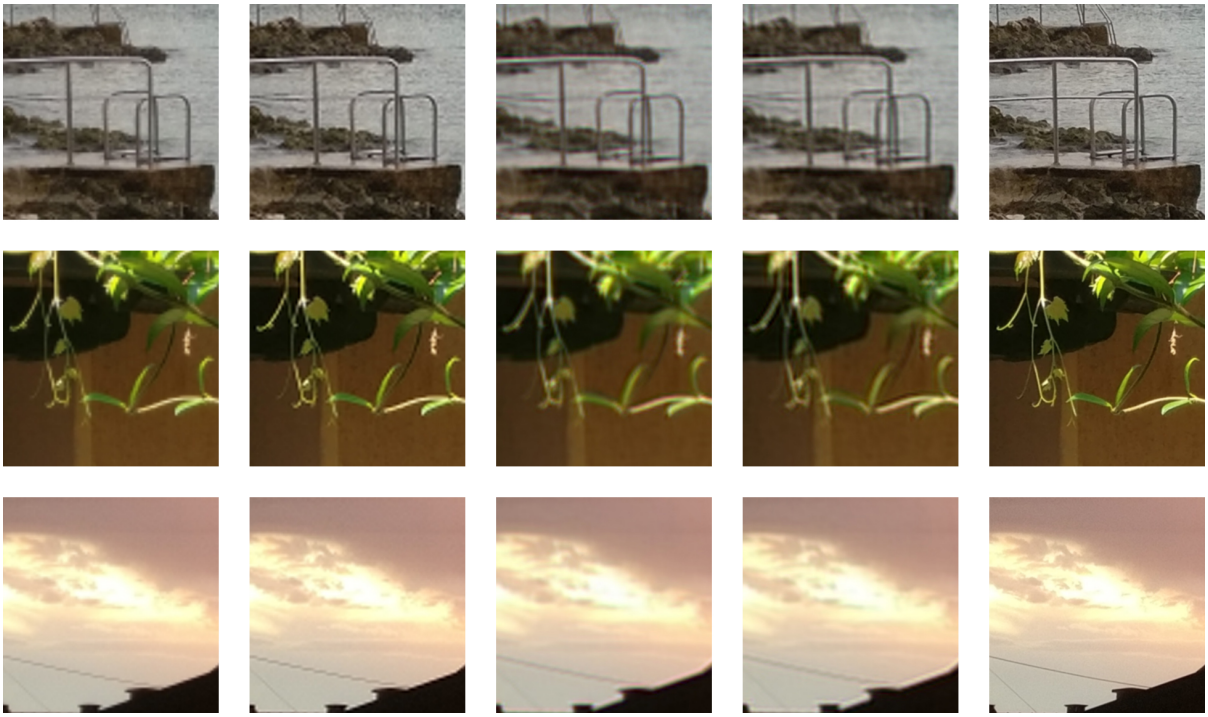
Povečava slik z mrežo se od povečave z eno izmed obstoječih interpolacij najbolj razlikuje večinoma v tem, da mreža ne ohranja toliko ostrine kot interpolacija, ampak veliko bolje ohranja obliko ostrih prehodov svetlosti.

Najbolj pozitivno bi na kvaliteto izhodnih slik vplivala večja učna množica, 580 slik ni veliko v primerjavi s prosto dostopnimi nabori slik.[14] Lahko bi tudi povečal število učnih iteracij, ampak že pri zadnjih iteracijah obstoječega učnega postopka so bile razlike minimalne. Uporabil bi lahko tudi večja jedra konvolucije, ampak glede na relativno majhno razliko med različicama menim, da pridobitek ne bi upravičil dodatne kompleksnosti.

Kljub tem pomanjkljivostim mislim, da je bila izdelava mreže uspeh.

[3] `https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-1920x`

[4] `https://www.nvidia.com/en-gb/geforce/products/10series/geforce-gtx-1080-ti/`



Slika 3: Primeri slik iz množice vhodnih podatkov, z leve: bilinearno, bikubično, konvolucija $17 * 17$, konvolucija $25 * 25$, izvorna slika

Literatura

[1] Alberto V. (2017). An Example of a Convolutional Neural Network for Image Super-Resolution. *Intel AI Academy* [Online]. Dosegljivo: `https://software.intel.com/en-us/articles/an-example-of-a-convolutional-neural-network-for-image-super-resolution`. [Dostopano 11. november 2018].

[2] Chao Dong *et al.*, "Learning a Deep Convolutional Network for Image Super-Resolution". The Chinese University of Hong Kong.

[3] Jiwon Kim *et al.*, "Accurate Image Super-Resolution Using Very Deep Convolutional Networks". Seoul National University, Korea.

[4] `torch.nn.Conv2d`. `https://pytorch.org/docs/stable/nn.html`. [Dostopano 11. november 2018].

[5] "What is an Activation Function?". `https://deepai.org/machine-learning-glossary-and-terms/activation-function`. [Dostopano 11. november 2018].

[6] Avinash Sharma V. (2017). Understanding Activation Functions in Neural Networks. *The Theory Of Everything*. [Online] Dosegljivo: `https://medium.com/the-theory-of-everything/`

understanding-activation-functions-in-neural-networks-9491262884e0. [Dostopano 11. november 2018].

[7] J. J. Hopfield. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*. [Online] Dosegljivo: http://www.pnas.org/content/pnas/79/8/2554.full.pdf. [Dostopano 11. november 2018].

[8] Marcel van Gerven, Sander Bohte. (2018). Artificial Neural Networks as Models of Neural Information Processing. *Frontiers in Computational Neuroscience*. [Online] Dosegljivo: https://www.frontiersin.org/research-topics/4817/artificial-neural-networks-as-models-of-neural-information-processing. [Dostopano 11. november 2018].

[9] Chris Edwards. (2015). *Growing pains for deep learning*. Communications of the ACM. 58 (7): 14–16.

[10] *Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation*. DeepLearning 0.1. LISA Lab. [Online] Dosegljivo: http://deeplearning.net/tutorial/lenet.html. [Dostopano 11. november 2018].

[11] Alex Krizhevsky *et al.* (2012). ImageNet Classification with Deep Convolutional Neural Networks. *NIPS 2012*. [Online] Dosegljivo: http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf. [Dostopano 11. november 2018].

[12] Ross Girshick. (2015). Fast R-CNN. *Microsoft Research*. [Online] Dosegljivo: https://arxiv.org/pdf/1504.08083.pdf. [Dostopano 11. november 2018].

[13] Diederik P. Kingma, Jimmy Lei Ba. (2015). Adam: A Method for Stochastic Optimization. *International Conference for Learning Representations*. [Online] Dosegljivo: https://arxiv.org/pdf/1412.6980.pdf. [Dostopano 11. november 2018].

[14] *Datasets*. Deep Learning. [Online] Dosegljivo: http://deeplearning.net/datasets/. [Dostopano 11. november 2018].