Vaja 4: Razpoznavanje slik z globokim učenjem

Povzetek

Z razvojem tehnologije in vse lažjim dostopom do velike količine podatkov, je globoko učenje (*ang. deeplearning*) trenutno najbolj razširjen pristop strojnega učenja in predstavlja osnovo vseh sodobnih algoritmov. Za razliko od tradicionalnih pristopov globoki algoritmi ne potrebujejo prednačrtovanja filtrov za analizo podatkov, saj njihove lastnosti spoznajo samodejno ob upoštevanju kriterijske funkcije, ki jih vodi skozi proces učenja. Pri tej vaji boste spoznali postopek načrtovanja in uporabe globokega algoritma za razvrščanje slik, ki temelji na konvolucijski nevronski mreži (*ang. Convolutional Neural Network - CNN*). Programsko kodo boste razvijali z uporabo jezika Python in deeplearning knjižnice PyTorch (https://pytorch.org/docs/1.5.1).

1 Izločanje robov z uporabo knjižnjice PyTorch

Za začetek boste z uporabo konvolucije izračunali robove slike hotela, ki ste jo uporabljalji na vajah pri predmetu Računalniški vid:

http://vision.fe.uni-lj.si/classes/RV/vaje/vaja4/hotel.jpg

Iz spletne učilnice predmeta si prenesite predpripravljeno Python skripto `edge_detection.py`. V omenjeni skripti boste našli kodo, ki s knjižnjico OpenCV prebere RGB sliko. Vaša naloga je, da s pomočjo konvolucijskega sloja `torch.nn.Conv2d` najprej izvedete pretvorbo RGB slike v sivinsko po formuli:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B.$$

Potrebujete torej konvolucijski sloj s tremi filtri velikost 1×1, kjer je utež prvega filtra $w_1 = 0.299$, utež drugega $w_2 = 0.587$ in utež tretjega $w_3 = 0.114$.

Po inicializaciji konvolucijskega sloja `Conv2d` bodo vrednosti njegovih uteži naključne. Nastavitev na prave vrednosti izvedite z `torch.nn.Parameters`. Bodite pozorni na vrstni red filtrov. Preverite kakšen je vrstni red barvnih kanalov matrike slike!

Na podoben način ustvarite še en konvolucijski sloj, tokrat za izračun gradientov sivinske slike s Sobelovim operatorjem. Definirajte torej dva filtra velikosti 3×3, edega za vertikalne robove (S_x) in enega za horizontalne robove (S_y):

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \; S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

.

Rezultata konvolucije (matrike vertikalnih in horizontalnih robov) združite v eno matriko, z izračunom magnitud gradientov:

$$G = \sqrt{G_x^2 + G_y^2}$$

Rezultat naj bi bila sivinska slika, kjer imajo robovi svetlejšo barvo. Za izolacijo robov od ostanka slike eksperimentalno določite prag upragovljanja, s katerim boste sivinsko sliko pretvorili v binarno.
Na koncu z OpenCV funkcijo `imshow()` prikažite originalno, sivinsko, ter binarno sliko robov hotela. Vse tri slike shranite s funkcijo `imwrite()`. Ne pozabite, da morate sivinsko in upragovljeno sliko normalizirati, ter pretvoriti v ustrezno obliko (ki jo OpenCV zahteva) preden ju prikažete oz. shranite. Konvolucijski sloji namreč delajo s tako imenovanimi tenzorji (ang. *tensor*), ki niso nič drugega kot matrike, ki jih lahko preselite na grafično kartico (ang. *GPU*) za pohitritev matematičnih operacij, ki se izvajajo. Normaliziranim tenzorjem torej najprej spremenite vrstni red kanalov iz $[N, H, W]$ v $[H, W, N]$ (s `.permute()`), nato pa jih predvorite v *numpy* matrike (s `.numpy()`) tipa `uint8` (s `.astype(np.uint8)`).

2 Večrazredno razvrščevalnje slik

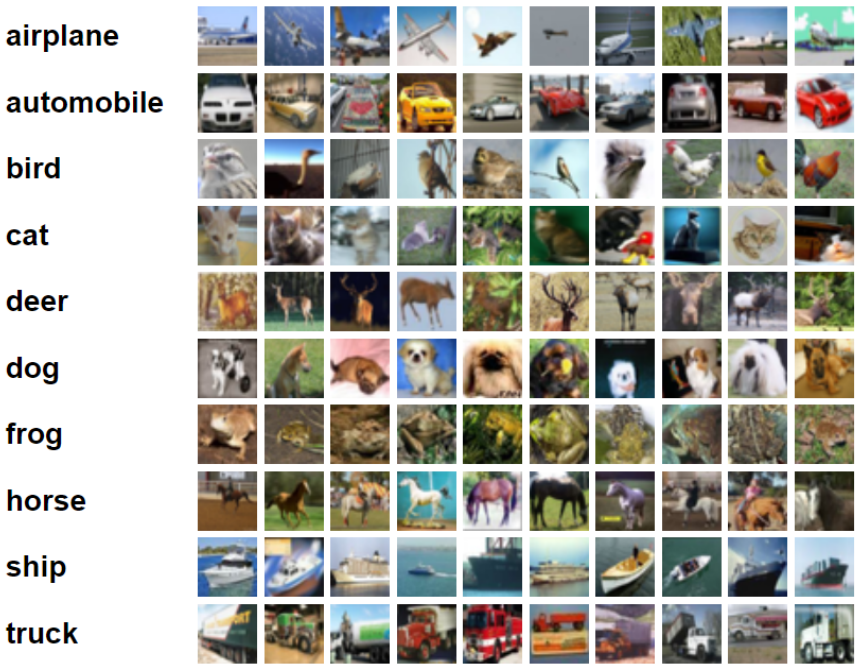
V nadaljevanju boste s knjižnjico PyTorch zgradili večplastno konvolucijsko nevronsko mrežo, podobno mreži VGG [1], ki ste jo obravnavali na predavanjih. V ta namen uporabite Python skripte iz .zip datoteke `classifier.zip` (objavljena v spletni učilnici eFE), ki jih ustrezno dopolnite z manjkajočimi funkcijami.

2.1 Podatkovne baze

Za učenje in evaluacijo mreže boste najprej uporabili podatkovno bazo MNIST [2, 3] (Slika 2.1). Slikovna baza vsebuje 70.000 slik števk bele barve na črnem ozadju. Velikost vsake od slik je 28×28 slikovnih elementov.

Slika 1: Primeri slik iz podatkovne baze MNIST [2, 3].

Kodo, ki jo boste razvili, dodatno testirajte še na slikovni bazi CIFAR10 [4]. Za razliko od MNIST, CIFAR10 sestavlja 60.000 barvnih slik velikosti 32×32 slikovnih elementov, enakomerno razdeljenih v deset razredov: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* in *truck* (Slika 2.1).



Slika 2: Primeri slik iz podatkovne baze CIFAR10.

Obe podatkovni bazi sta na voljo na laboratorijskih računalnikih v mapi `/storage`.

Branje podatkov iz lokalnega diska se izvaja v skripti `dataloader.py`, kjer funkcija `make_dataset()` sestavi seznam učnih in testnih slik, ter seznam pripadajočih oznak (od 0 do 9) tako, da je vsak od MNIST oz. CIFAR10 razredov označen z eno od teh oznak. Funkcija `load_data()` potem poskrbi za ustrezno preoblikovanje slik iz tvorjenih seznamov in sicer tako, da jim najprej po potrebi spremeni velikost (`Resize()`), nato pa jih normalizira (`Normalize()`). V nadaljevanju s pomočjo modula `torch.utils.data` (z uporabo klase `Dataset` in `Dataloader`) definiramo še postopek dinamičnega (sprotnega) branja slik. Dejansko branje podatkov izvajamo s klaso `Dataloader(Dataset)`, z uporabo metode `def __getitem__`.

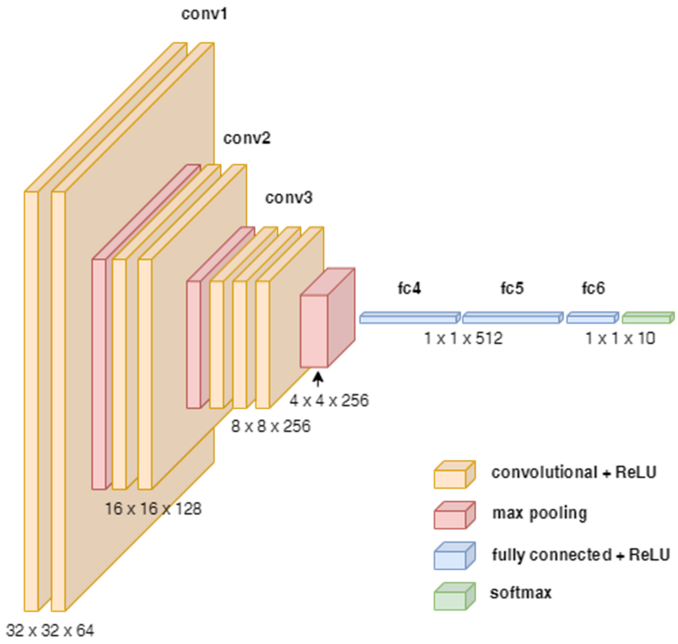
2.2 Načrtovanje konvolucijske nevronske mreže

V nadaljevanju, v klasi `CNN(nn.Module)` skripte `network.py` ustvarite arhitekturo konvolucijske nevronske mreže. Plasti definirajte znotraj inicializacijske metode `def __init__` in sicer tako, da praznemu modelu `net = nn.Sequential()` s klicem `net.add_module(name, module)` dodate vsako plast posebej. Arhitektura naj bo skladna s tabelo na Sliki 2.2.

Mreža naj torej vsebuje:

- 7 konvolucijskih slojev (`torch.nn.Conv2d()`) s filtri velikosti 3×3. Pred izračunom konvolucije, naj se na vsak rob vhodnih matrik dodaja ena vrstica ničel (`padding` $= 1$). Pri izračunu

RGB image 32x32x3
conv3-64/p1 conv3-64/p1
maxpool2/s2
conv3-128/p1 conv3-128/p1
maxpool2/s2
conv3-256/p1 conv3-256/p1 conv3-256/p1
maxpool2/s2
FC-512
FC-512
FC-10
softmax



konvolucije pa naj se filtri premikajo s korakom 1 (`stride` = 1).

- 3 združevalne sloje z maksimizacijo (`torch.nn.MaxPool2d()`) s filtri velikosti 2×2 in korakom 2.
- 3 polnopovezane sloje `torch.nn.Linear`.
- 9 aktivacijskih slojev ReLU (`torch.nn.ReLU`), ki se nahajajo za vsakim konvolucijskim in polnopovezanim slojem, razen za zadnjim polnopovezanim slojem.

Funkcijo `softmax` bomo implemetirali posebej, v okviru kriterijske funkcije učenja mreže.

Sedaj, ko ste sestavili arhitekturo mreže, bo funkcija `def forward` klase `CNN(nn.Module)` posredovala (ang. *feedforward*) podatke med plastmi nevronske mreže z upoštevanjem vrstnega reda, ki ste ga definirali v `def __init__`.

2.3 Učenje nevronske mreže

Učenje nevronske mreže se začne, ko zaženete skripto z imenom `train.py`, ki ob njenem klicu omogoča nastavitev različnih parametrov učenja, kot so: ime podatkovne baze (`--dataset`), pot do podatkov (`--dataroot`), število iteracij (*ang. epoch*) učenja (`epochs`) itd. V tej skripti najprej z `load_data()` preberemo seznam učnih in testnih podatkov, nato še incializiramo mrežo `model = CNN()`.

Ker je globoko učenje računsko zelo zahtevna naloga, bomo za učenje uporabljali grafično kartico (`device="cuda:0"`), ki znatno pohitri izvajanje matematičnih operacij. Ob inicializaciji spremenljivk, ki so neposredno povezane z nevronsko mrežo (na primer vhodne matrike mreže) moramo tudi te preseliti na grafično kartico, kjer se nahajajo uteži modela:

```
images = torch.empty(size=(B, N, H, W), dtype=float32, device=device)
```

Uteži modela optimiziramo na podlagi napake t.i. kriterijske funkcije, ki je odvisna od tipa problema, ki ga rešujemo. V našem primeru izvajamo večrazredno razvščanje vzorcev in zato bomo za kriterijsko funkcijo uporabili križno entropijo (`torch.nn.CrossEntropyLoss()`). Za posodobitev vrednosti modela pa bomo uporabljali optimizacijski model `Adam` [5].

Preden začnemo z procesiranjme slik, moramo nastaviti še režim delovanja nevronske mreže (`model.train()` za učenje mreže, ter `model.eval()` za evaluacijo naučenih uteži). Učenje mreže, je iterativni postopek, kjer se na začetku izračuna napaka (`loss`) razvrščanja slik iz trenutne podmnožice učne baze. Na podlagi te napake v nadaljevanju izračunamo še gradiente (`loss.backward()`), ter ustrezno posodobimo vrednosti parametrov modela tako, da je napaka manjša (`optimizer.step()`).

2.4 Testiranje nevronske mreže

Ko obdelamo vse učne vzorce, mrežo evaluiramo z uporabo vzorcev iz testne množice. Sripto `train.py` dopolnite s kodo za izračun napake razvrščanja testnih vzorcev. Za vsako serijo testnih vzorcev izračunajte število pravilno razvrščenih slik (`predicted_labels=test_labels`). Klasifikacijsko točnost izračunajte tako, da število pravilno razvrščenih vzorcev delite s skupnim številom vzorcev. Poiščite vsaj 10 vzorcev, ki niso bili pravilno razvščeni.

Literatura

[1] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In 3rd International Conference on Learning Representations, ICLR, 2015.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient–Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[3] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The MNIST Database of Handwritten Digits. `http://yann.lecun.com/exdb/mnist/`. Accessed: 2021-03-14.

[4] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toronto, 2009.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In 3rd International Conference on Learning Representations, ICLR, 2015.