Fakulta elektrotechniky a informatiky

Fakulta Elektrotechniky a Informatiky

Katedra kybernetiky a umelej inteligencie

Predmet : **Základy hlbokého učenia**
kurz 2018 / 2019

Zadanie číslo # 1 (esej) :

Využitie deep learningu v biometrickom zabezpečení

kľúčové slova : Autentifikácia, Biometrické zabezpečenie, Deep learning, Overovanie identity, Neurónová sieť, FaceNet

---

Spracovali :
Michal Bavlšík


KKUI
Katedra kybernetiky
a umelej inteligencie

Obsah

1 Úvod

S každodenným technologickým pokrokom a šírením konektivity do všetkých častí našich životov je čoraz viac jasné, že staré metódy zabezpečenia sú nedostačujúce na ochranu toho, čo nám je najvzácnejšie. Naštastie biometrické zabezpečenie je v dnešnej dobe prístupnejšie ako hocikedy predtým, čím nám dáva nový pohľad na možnosti chrániť ako aj fyzický, tak aj digitálny svet. Jednou z prelomových techník na poli biometrického zabezpečenia je využitie špeciálneho algoritmu z oblasti umelej inteligencie a to tzv. hĺbkových neurónových sietí nazývaných aj "deep learning". V tejto práci sa objasní termín biometrika, biometrický systém, stručne sa popíšu niektoré biometrické modality a priblíži sa implementácia určitých algoritmov deep learningu do uvedenej problematiky.

2 Špecifikácie oblasti aplikácie a popis problému

Termín Biometrika sa zkladá z dvoch slov Bio (Grécke slovo pre život) a Metrika (Meranie). Biometrika je vetva informačných technológií, ktorá sa zaoberá identitou jednotlivcov identifikovaním osobných črtov.

Biometrika je momentálne veľmi spomínanou témou v oblasti zabezpečenia informačných technológií keďže poskytuje vysokú presnosť v identifikácii v porovnaní s konvenčnými metódami napríklad znázornenými v tabuľke nižše.

Autentifikačná metóda	**Výhody**	**Nevýhody**
Fyzické Zariadenia (karty, ID, pas, atď.)	■ Môže sa vystaviť nový. ■ Štandartný prenos do inej krajiny alebo zariadenia	■ Môže byť ukradnuté. ■ Môže byť ľahko falsifikované. ■ Môže sa zdielať. ■ Jedna osoba môže byť registrovaná pod viacerými identitami.
Založené na znalostiach (heslo,PIN, atď.)	■ jednoduché a ekonomické. ■ Pri problémoch vie byť vystavené nové bez problémov.	■ Môže byť uhádnuté alebo prelomené. ■ Dobré heslá sú ťažko zapamätateľné. ■ Môže byť zdieľané. ■ Jedna osoba môže byť registrovaná pod viacerými identitami .
Biometrika	■ Nemôže byť stratené ukadnuté, uhádnuté ani zdieľané. ■ Pomerne ľahká identifikácia duplicitných identít. ■ Najväčšia miera zabezpečenia.	■ V niektorých prípadoch sa môže dať obísť. ■ Nemôže sa nahradiť ani utajiť. ■ Pri strate biometrických údajov sa nemôžu údaje zmeniť.

2.1 Čo je Biometrika?

Biometrika je technológia, ktorá sa používa na identifikáciu, analýzu a meranie fyzických charakteristík a správania sa jednotlivca.

Každá ľudská bytosť je jedinečná v medziach jeho charakteristík, ktoré ho odlišujú od ostatných. Fyzikálne atribúty ako napríklad odtlačky prstov, farba zreničiek, tvar ruky, prejavy správania, tón a farba hlasu využívajú biometrické systémy ku :

- Identifikácii a overeniu osoby.
- Povolenie prístupu ku určitej časti systému.
- Udržovaniu systému pred neetickým zaobchádzaním.

2.2 Čo je Biometrický systém?

Biometrický systém je technológia, ktorá má na vstupe fyzické črty alebo črty správania a po analýze vie identifikovať či je jednotlivec pre systém známy alebo nie.

Základné pojmy systému sú:
Biometrický vzor - Digitálna referencia jednoznačných charakteristík, ktoré sú extrahované z biometrickej vzorky.

Kandidát/Subjekt – osoba, ktorá vkladá biometrickú vzorku.

Identifikácia uzavretého setu – O osobe sa vie, že sa nachádza v databáze.

Zápis – Nastáva ak subject používa biometrický system po prvý krát, kedy sa nahrajú jeho základné informácie a jeho biometrické črty.

Stupeň falošnej prípustnosti (FAR) – Miera možnosti, že biometrický system nesprávne identifikuje neautorizovaného uživateľa ako autorizovaného. (čím menšia hodnota tým lepšie)

Stupeň falošného odmietnutia (FRR) Miera možnosti, že biometrický system nesprávne identifikuje autorizovaného uživateľa ako neautorizovaného.

Identifikácia otvoreného setu – Nemáme garantované, že osoba sa v database už nachádza.

Úloha – Prehľadávanie databázy biometrickým systémom na zhodu.

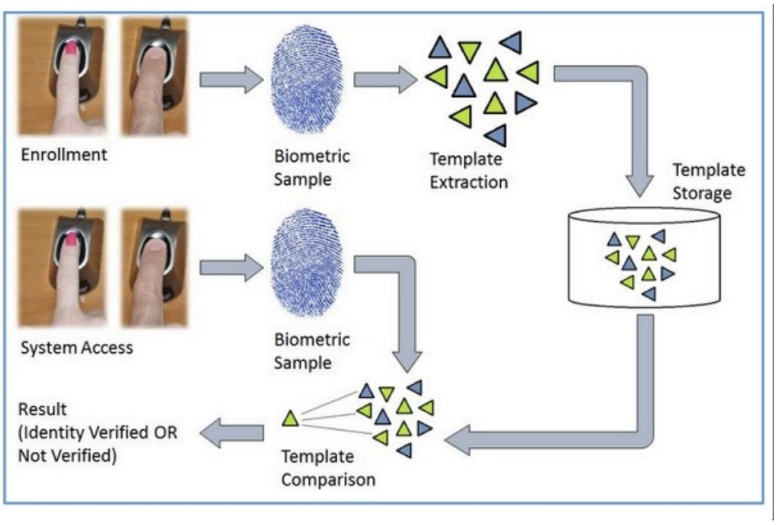
Autentifikácia (Identifikácia) - metóda vyhľadávania a porovnávania vstupu s údajmi v databáze typu jeden-ku-všetkým

Verifikácia - metóda vyhľadávania a porovnávania vstupu s údajmi v databáze v reálnom čase typu jeden-ku-jednému

Autorizácia - proces priraďovania prístupových práv ku autentifikovaným alebo overeným uživateľom

2.3 Princíp fungovania biometrického systému

Štyri základné kroky pre autentifikáciu a verifikáciu :
Získanie vzorky v reálnom čase od kandidáta (pomocou senzorov)
Extrakcia prominentných dát zo vzorky (použitím výpočtového systému)
Porovnanie novej vzorky so vzorkami v databáze (pomocou algoritmu)
Prezentácia výstupu. (Potvrď alebo zamietni kandidáta)



2.4 Biometrická modalita

Kategória biometrickej črty na vstupe, s ktorými pracuje biometrický systém. Čím viac dát vieme zo vzorky vyčítať, tým väčšia šanca že systém bude spoľahlivejší.

• Typy biometrických modalít
Človek je charakterizovaný niekoľkými črtami, ktoré môžu byť využité ako biometrické modality adelíme ich do troch skupín:

- Fyziologické
- Charakterové

- Ich kombinácia

Nasledujúca tabuľka ukazuje body, ktoré odlišujú jednotlivé modality:

Fyziologická modalita	**Behaviorálna modalita**	**Kombinácia oboch modalít**
Táto modalita sa týka tvaru a veľkosti tela.	Táto modalita súvisí so zmenou ľudského správania v čase.	Táto modalita zahŕňa obidve znaky, v ktorých vlastnosti závisia od fyzických aj behaviorálnych zmien.
Napríklad - •Rozpoznávanie odtlačkov prstov •Systém rozpoznávania geometrie ruky •Systém rozpoznávania tváre •Systém rozpoznávania dúhovky •Sietnicový skenovací systém •Systém rozpoznávania DNA	Napríklad - • Chôdza • Rytmus písania • Podpis	Napríklad - Rozpoznanie hlasu Závisí od zdravia, veľkosti a tvaru hlasiviek, nosných dutín, ústnej dutiny, tvaru pery atď. Ale aj emocionálneho stavu, veku, choroby (správania) človeka.

2.4.1 Rozpoznanie odtlačku prsta

Najviac známe a používané riešenie v oblasti autentifikácie ľudí v biometrických systémoch. Hlavným dôvodom je dostupnosť desiatich zdrojov merania a ľahkosť získavania. Každá osoba ma jedinečný odtlačok prsta, ktorý pozostáva z reliéfov, rýh a smeru čiar.

Poznáme tri základné typy reliéfov a to klenba, slučka a závit. Jedinečnosť odtlačku prsta je daná týmito vlastnosťami ako aj maličkosťami ako vetveniu a bodmi kde reliéfy končia.

2.4.2 Rozpoznanie tváre

Rozpoznanie tváre je založené na určení tvaru a veľkosti čeľuste, brady, tvaru a umiestnenia očí, obočia, nosa, pier a lícnych kostí. Jeden prístup je použiť 2D skenery, ktoré začínajú čítať geometriu tváre a zaznamenávať ju na mriežku. Geometria tváre sa prenesie do databázy ako sústava bodov resp. vektorov. Porovnávacie algoritmy vykonávajú porovnávanie tváre a na výstupe sú výsledky napr. v podobe verifikácie. Tejto časti sa budeme bližšie venovať v popise algoritmov deep learningu.

2.4.3 Rozpoznanie dúhovky

Rozpoznávanie dúhovky funguje na základe modelu dúhovky v ľudskom oku. Dúhovka je pigmentovaná elastická tkanina, ktorá má v strede nastaviteľný kruhový otvor. Ovláda priemer zreničky. U dospelých ľudí je štruktúra dúhovky počas celého života stabilná. Vzory ľavého a pravého oka sú odlišné. Vzory a farby dúhovky sa menia z osoby na osobu.

2.4.4 Rozpoznanie geometrie ruky

Zahŕňa meranie dĺžky a šírky dlaní, povrchovej plochy, dĺžky a polohy prstov a celkovej kostnej štruktúry ruky. Ruka človeka je jedinečná a môže byť použitá na identifikáciu osoby od ostatných.

2.4.5 Rozpoznanie sietnice

Sietnica je vrstva na zadnej strane očnice, ktorá pokrýva 65% vnútorného povrchu oka. Obsahuje fotosenzitívne bunky. Sietnica každého človeka je jedinečná kvôli komplexnej sieti krvných ciev, ktoré dodávajú krv.

Je to spoľahlivá biometria, pretože sietnicový vzor zostáva nezmenený po celý život človeka, s výnimkou osôb s diabetom, glaukómom alebo niektorými degeneratívnymi poruchami.

V procese snímania sietnice je osoba požiadaná o odstránenie šošoviek alebo okuliarov. Infračervený svetelný lúč s nízkou intenzitou svieti do očí človeka po dobu 10 až 15 sekúnd. Toto infračervené svetlo je absorbované krvnými cievami, ktoré vytvárajú štruktúru krvných ciev počas snímania. Tento vzor je potom digitalizovaný a uložený v databáze.

2.4.6 Rozpoznanie pomocou DNA

Deoxyribonuklerová kyselina (DNA) je genetický materiál, ktorý sa nachádza u ľudí. Každý človek, okrem jednovaječných dvojčiat, je jednoznačne identifikovateľný vlastnosťami nachádzajúcimi sa v ich DNA, ktorá sa nachádza v jadre bunky. Existuje množstvo zdrojov, z ktorých sa môžu zhromažďovať vzorky DNA, ako sú krv, sliny, nechty, vlasy atď.

2.4.7 Rozpoznanie kroku

Krok je spôsob chôdze človeka. Ľudia vykazujú pri chôdzi odlišné črty, ako je držanie tela, vzdialenosť medzi dvoma nohami pri chôdzi, kymácanie sa atď., ktoré pomáhajú jedinečne ich rozpoznať.

2.4.8 Rozpoznanie podpisov

V tomto prípade sa kladie väčší dôraz na vzorce správania, v medziach ktorých je písaný podpis, ako spôsob zobrazenia podpisu v grafickej podobe.

Vzory správania zahŕňajú zmeny v časovaní písania, pauzy, tlaku, smeru ťahov a rýchlosti počas podpisovania. Mohlo by byť jednoduché duplikovať grafický vzhľad podpisu, ale nie je ľahké napodobňovať podpis tým istým správaním, ktoré osoba vykazuje pri podpisovaní.

Táto technológia sa skladá z pera a špecializovaného písacieho tabletu, ktoré sú pripojené k počítaču na porovnanie a overovanie šablón. Vysokokvalitný tablet dokáže zachytiť charakteristiky správania, ako je rýchlosť, tlak a časovanie pri podpisovaní.

2.4.9 Rozpoznanie písania na klávesnici

Takýto systém analyzuje model písania subjektu, rytmus a rýchlosť písania na klávesnici. Meranie času a času letu (prstov) sa takisto používajú pri rozpoznaní tohoto typu.

2.4.10 Rozpoznanie hlasu

Rozpoznávanie hlasu sa tiež nazýva rozpoznávanie hovoriaceho človeka. V čase zápisu musí používateľ hovoriť slovo alebo frázu do mikrofónu. Je to nevyhnutné na získanie vzorovej reči kandidáta. Elektrický signál z mikrofónu sa konvertuje na digitálny signál pomocou analógového na digitálny (ADC) konvertor. Zaznamenáva sa do pamäte počítača ako digitalizovaná vzorka. Počítač potom porovnáva a pokúša sa spárovať vstupný hlas kandidáta s uloženou digitalizovanou hlasovou vzorkou a identifikuje kandidáta. V tejto sfére sa využíva tak isto deep learning v podobe spracovanie časových radov alebo spektrálna analýza.

2.4.11 Multimodálna biometria

Všetky biometrické systémy, o ktoré su vyššie spomenuté, boli unimodálne, teda využívajú jediný zdroj informácií na autentifikáciu. Ako sa uvádza v názve, multimodálne biometrické systémy pracujú na prijímaní informácií z dvoch alebo viacerých biometrických vstupov. Multimodálny biometrický systém zvyšuje rozsah a rôznorodosť vstupných informácií, ktoré systém zbiera od používateľov na účely autentifikácie.

2.5 Systém rozpoznania uživateľa na základe tváre

Kôli zachovaniu rozsahu práce vyberieme len jednu modalitu a to rozpoznávanie tváre pre jej súčasne nízky výkon a možné zvýšenie výkonu pomocou deep learningu. Referencie na rozpoznanie hlasu [1] a rozpoznanie odtlačku prsta [2] doplníme kôli ich popularite.

V roku 2015 výskumníci spoločnosti Google vydali článok FaceNet[3], ktorý používa hlbokú konvolučnú sieť čo sa považuje za súčasne najmodernejšiu architektúru pre extrakciu atribútov z 2D obrazu a jeho klasifikáciu.

Na datasete LFW (Labeled Faces in the Wild) dosiahla novú rekordnú presnosť 99,63%. Vďaka tomu sa stala populárnou voľbou v zabezpečovacích systémoch ale aj open source hobby pro-

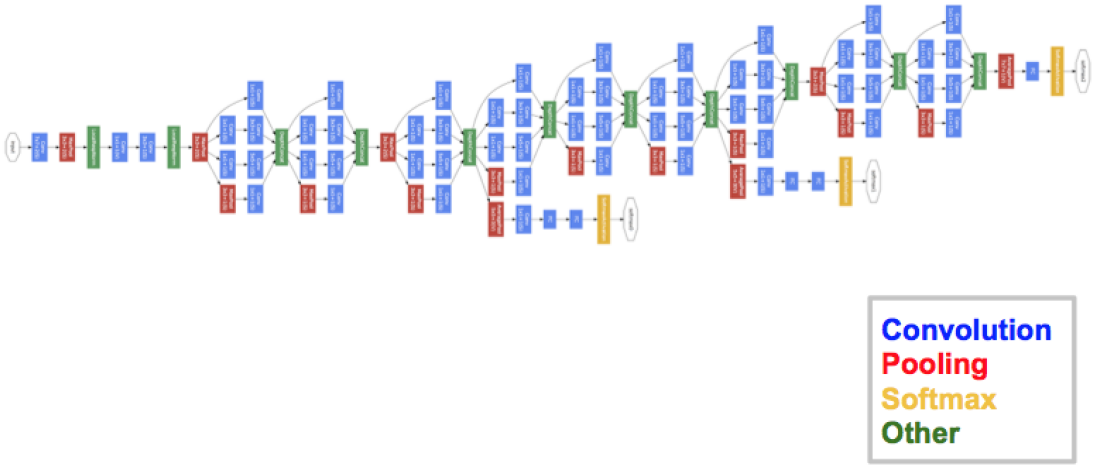
jektoch. Bližšie si ju popíšeme v ďalších kapitolách.

3 Výber typu neurónových sietí a jej topológia

FaceNet využíva hlbokú konvolučnú sieť. V článku skúšali 2 architektúry: ZeilerFergus [4] štýl a Inception [5] štýl siete.
Všeobecná topológia riešenia vyzerá takto:

Dávky (batches) vstupujú do Inception V2 (prípadne V3 siete) po ktorej nasleduje L2 normalizácia výstupu, ktorý sa zakódoval do 128 dimenzionálneho vektora (embedding) slúžiaceho na výpočet triplet loss (chyby).

Samotná topológia Inception hlbokej neurónovej siete (deep architecture):



Keďže tieto vektory embeddingov sú reprezentované v zdieľanom vektorovom priestore, na výpočet rozdielnosti (podobnosti tvárí) sa využíva jednoduchá vektorová vzdialenosť.

4 Hyperparametre a parametre siete

Sieť má 7.5 milióna parametrov a 1.6 Bilióna FLOPS.

Hyperparametre:
Optimizer: SGD(lr=1e-5, momentum=0.9, nesterov=True, decay=1e-6)
Loss: triplet loss
BatchNorm: BatchNormalization(momentum=0.995,epsilon=0.001,scale=False)

5 Topológia a popis neurónovej siete

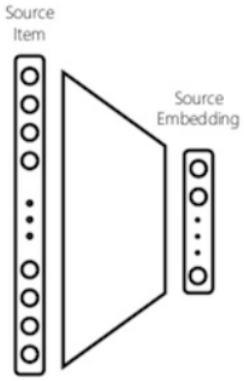
type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

Table 2. **NN2.** Details of the NN2 Inception incarnation. This model is almost identical to the one described in [16]. The two major differences are the use of L_2 pooling instead of max pooling (m), where specified. The pooling is always 3×3 (aside from the final average pooling) and in parallel to the convolutional modules inside each Inception module. If there is a dimensionality reduction after the pooling it is denoted with p. 1×1, 3×3, and 5×5 pooling are then concatenated to get the final output.

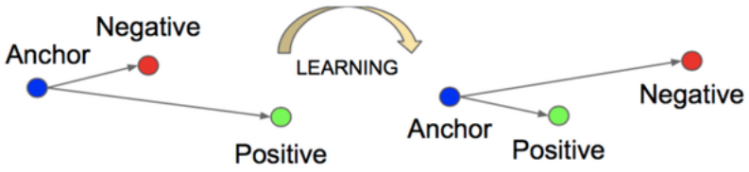
Sieť má 22 vrstiev ak rátame iba vrstvy s parametrami (alebo 27 vrstiev ak rátame pooling). Celkový počet nezávislých stavebných blokov siete je okolo 100.

6 Výber L funkcie a jej matematický popis

Ako L funkciu sieť využíva tzv. Triplet Loss. Triplet loss závisí od minimalizácie vzdialenosti od pozitívnych príkladov, pričom maximalizuje vzdialenosť od negatívnych príkladov.
Embedding je reprezentovaný funkciou f(x) Rd . Táto funkcia kóduje obrázok x(ai) do d-dimenzionálneho Euklidovského priestoru. Navyše v článku obmedzili toto kódovanie aby existovalo v d-dimenzionálnom hyperpriestore, čiže —f(x)—2 = 1 ako na obrázku nižšie:



Takáto chybová funkcioa vznikla na základe článku [19] v kontexte klasifikátora na báze najbližšieho suseda. V tomto prípade sa chceli uistiť, že obrázok x(ai) - alebo kotva (anchor) špecifickej osoby je bližšie ku všetkým ostatným obrázkom x(pi) - (positive) rovnakej osoby ako ku x(ni) - (negative) obrázkom iných osôb. Toto je zvizualizované v obrázku nižšie.



Matematický popis chybovej funkcie je:

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Tým pádom chceme:

$$\|x_i^a - x_i^p\|_2^2 + \alpha < \|x_i^a - x_i^n\|_2^2, \forall (x_i^a, x_i^p, x_i^n) \in \mathcal{T} \ . \quad (1)$$

Správne zvolený triplet loss je kritický pre rýchlu konvergenciu učenia. Na jednej strane chceli použiť mini-batche, ktoré vedú na lepšiu konvergenciu keď sa využíva Stochastic Gradient Descent (SGD) [20]. Vo väčšine experimentov využívali v článku batch size = 1800.

7 Vyhodnotenie popísaného experimentu

V článku navrhli metódu ktorá sa priamo učila ako zakódovať výstup do 128 dimenzionálneho priestoru, ktoré ju odlišujú od ostatných článkov (ktoré využívajú bottlenck vrstvy alebo concatenation).

Tento kompletný systém zjednodušuje architektúru trénovania. Vstup tak isto vyžaduje iba minimálne predspracovanie (orezanie, otočenie atď.)

8 Čas spracovania, skúsenosti, presnosť

Rozloženie trénovacej vzorky:

#training images	VAL
2,600,000	76.3%
26,000,000	85.1%
52,000,000	85.1%
260,000,000	86.2%

8.1 Výkonnosť na LFW datasete

Na vyhodnotenie využili 9 trénovacích rozdelení na výber hranice L2-vzdialenosti.
Klasifikácia bola prevedená na desiatom rozdelení s hranicou 1.242 pre všetky testovacie rozdelenie.
Dosiahnutá presnosť bola 98.870.15%

8.2 Výkonnosť na Youtube Faces DB

Využívali priemernú podobnosť všetkźch párov prvých 100snímkov ktoré detekovat detektor tváre v každom videu. To zabezpečilo presnosť 95.120.39%.

9 Zhodnotenie a záver

Biometrické rozpoznávanie ponúka sľubný prístup k bezpečnostným aplikáciám, s jasnými výhodami nad klasickými metódami. Pozitívna vlastnosť biometrických črtov spočíva v tom, že sú založené na niečom, čím človek je alebo niečom čo robí, takže si nemusí pamätať zložité heslá, ani sa starať o karty alebo kľúče ale hlavne, že úroveň zabezpečenia je takmer rovnaká pre všetkých používateľov v systéme.

Hoci biometria ponúka množstvo výhod, ešte nebola masívne prijatá a implementovaná. Zmeniť to môže klesajúca cena hardwaru a využitie algoritmov umelej inteligencie, napr. ako deep learning. Momentálne popredné technológie a výskumy vsádzajú práve na spomínané algoritmy

a to čo sa dnes považuje za špičkovú technológiu sa môže čoskoro stať archaické v porovnaní s výhodami ako rýchlosť a presnosť, ktoré so sebou prináša umelá inteligencia resp. deep learning ako taký.

V súčastnej dobe existujú už mnohé ďalšie a sofistikovanejšie alternatívy ku spomínanej architektúre FaceNet, ktorá bola ale dlho považovaná za state-of-the-art vrámci Face recognition systémov pracujúcich s 2D obrazom. Využitie triplet loss značne vylepšilo presnosť Inception hlbokej neurónovej siete čo ju robilo vhodnou pre použitie v jednoduchých biometrických systémoch.

10 referencie

[1] Variani, Ehsan Lei, Xin McDermott, Erik Lopez Moreno, Ignacio Gonzalez-Dominguez, Javier. (2014). Deep neural networks for small footprint text-dependent speaker verification. Proc. ICASSP. 4052-4056. 10.1109/ICASSP.2014.6854363.

[2] Pavol Marák, Alexander Hambalík. Fingerprint recognition system using artificial neural network as feature extractor:ndesign and performance evaluation. 10.1515/tmmp-2016-0035 Tatra Mt. Math. Publ. 67 (2016), 117–134

[3] arXiv:1503.03832[cs.CV]: FaceNet: A Unified Embedding for Face Recognition and Clustering

[4] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. CoRR, abs/1311.2901, 2013.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014

11 Bonusove časti - voliteľne

11.1 Implementácia systému v Kerase

```
"""Inception-ResNet V2 model for Keras.
Model naming and structure follows TF-slim implementation
(which has some additional layers and different number of
filters from the original arXiv paper):
https://github.com/tensorflow/models/blob/master/research/slim/nets/inception
Pre-trained ImageNet weights are also converted from TF-slim,
which can be found in:
https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-m
```

```
# Reference
- [Inception-v4, Inception-ResNet and the Impact of
   Residual Connections on Learning](https://arxiv.org/abs/1602.07261) (AAAI
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os

from . import get_submodules_from_kwargs
from . import imagenet_utils
from .imagenet_utils import decode_predictions
from .imagenet_utils import _obtain_input_shape


BASE_WEIGHT_URL = ('https://github.com/fchollet/deep-learning-models/'
                   'releases/download/v0.7/')

backend = None
layers = None
models = None
keras_utils = None


def preprocess_input(x, **kwargs):
    """Preprocesses a numpy array encoding a batch of images.
    # Arguments
        x: a 4D numpy array consists of RGB values within [0, 255].
    # Returns
        Preprocessed array.
    """
    return imagenet_utils.preprocess_input(x, mode='tf', **kwargs)


def conv2d_bn(x,
              filters,
              kernel_size,
              strides=1,
              padding='same',
              activation='relu',
              use_bias=False,
              name=None):
```

```
    """Utility function to apply conv + BN.
    # Arguments
        x: input tensor.
        filters: filters in `Conv2D`.
        kernel_size: kernel size as in `Conv2D`.
        strides: strides in `Conv2D`.
        padding: padding mode in `Conv2D`.
        activation: activation in `Conv2D`.
        use_bias: whether to use a bias in `Conv2D`.
        name: name of the ops; will become `name + '_ac'` for the activation
            and `name + '_bn'` for the batch norm layer.
    # Returns
        Output tensor after applying `Conv2D` and `BatchNormalization`.
    """
    x = layers.Conv2D(filters,
                      kernel_size,
                      strides=strides,
                      padding=padding,
                      use_bias=use_bias,
                      name=name)(x)
    if not use_bias:
        bn_axis = 1 if backend.image_data_format() == 'channels_first' else 3
        bn_name = None if name is None else name + '_bn'
        x = layers.BatchNormalization(axis=bn_axis,
                                      scale=False,
                                      name=bn_name)(x)
    if activation is not None:
        ac_name = None if name is None else name + '_ac'
        x = layers.Activation(activation, name=ac_name)(x)
    return x


def inception_resnet_block(x, scale, block_type, block_idx, activation='relu'
    """Adds a Inception-ResNet block.
    This function builds 3 types of Inception-ResNet blocks mentioned
    in the paper, controlled by the `block_type` argument (which is the
    block name used in the official TF-slim implementation):
        - Inception-ResNet-A: `block_type='block35'`
        - Inception-ResNet-B: `block_type='block17'`
        - Inception-ResNet-C: `block_type='block8'`
    # Arguments
        x: input tensor.
        scale: scaling factor to scale the residuals (i.e., the output of
```

```
        passing `x` through an inception module) before adding them
        to the shortcut branch.
        Let `r` be the output from the residual branch,
        the output of this block will be `x + scale * r`.
    block_type: ``block35``, ``block17`` or ``block8``, determines
        the network structure in the residual branch.
    block_idx: an `int` used for generating layer names.
        The Inception-ResNet blocks
        are repeated many times in this network.
        We use `block_idx` to identify
        each of the repetitions. For example,
        the first Inception-ResNet-A block
        will have `block_type='block35', block_idx=0`,
        and the layer names will have
        a common prefix ``block35_0``.
    activation: activation function to use at the end of the block
        (see [activations](../activations.md)).
        When `activation=None`, no activation is applied
        (i.e., "linear" activation: `a(x) = x`).
# Returns
    Output tensor for the block.
# Raises
    ValueError: if `block_type` is not one of ``block35``,
        ``block17`` or ``block8``.
"""
if block_type == 'block35':
    branch_0 = conv2d_bn(x, 32, 1)
    branch_1 = conv2d_bn(x, 32, 1)
    branch_1 = conv2d_bn(branch_1, 32, 3)
    branch_2 = conv2d_bn(x, 32, 1)
    branch_2 = conv2d_bn(branch_2, 48, 3)
    branch_2 = conv2d_bn(branch_2, 64, 3)
    branches = [branch_0, branch_1, branch_2]
elif block_type == 'block17':
    branch_0 = conv2d_bn(x, 192, 1)
    branch_1 = conv2d_bn(x, 128, 1)
    branch_1 = conv2d_bn(branch_1, 160, [1, 7])
    branch_1 = conv2d_bn(branch_1, 192, [7, 1])
    branches = [branch_0, branch_1]
elif block_type == 'block8':
    branch_0 = conv2d_bn(x, 192, 1)
    branch_1 = conv2d_bn(x, 192, 1)
    branch_1 = conv2d_bn(branch_1, 224, [1, 3])
```

```
        branch_1 = conv2d_bn(branch_1, 256, [3, 1])
        branches = [branch_0, branch_1]
    else:
        raise ValueError('Unknown Inception-ResNet block type. '
                         'Expects "block35", "block17" or "block8", '
                         'but got: ' + str(block_type))

    block_name = block_type + '_' + str(block_idx)
    channel_axis = 1 if backend.image_data_format() == 'channels_first' else
    mixed = layers.Concatenate(
        axis=channel_axis, name=block_name + '_mixed')(branches)
    up = conv2d_bn(mixed,
                   backend.int_shape(x)[channel_axis],
                   1,
                   activation=None,
                   use_bias=True,
                   name=block_name + '_conv')

    x = layers.Lambda(lambda inputs, scale: inputs[0] + inputs[1] * scale,
                      output_shape=backend.int_shape(x)[1:],
                      arguments={'scale': scale},
                      name=block_name)([x, up])
    if activation is not None:
        x = layers.Activation(activation, name=block_name + '_ac')(x)
    return x


def InceptionResNetV2(include_top=True,
                      weights='imagenet',
                      input_tensor=None,
                      input_shape=None,
                      pooling=None,
                      classes=1000,
                      **kwargs):
    """Instantiates the Inception-ResNet v2 architecture.
    Optionally loads weights pre-trained on ImageNet.
    Note that the data format convention used by the model is
    the one specified in your Keras config at `~/.keras/keras.json`.
    # Arguments
        include_top: whether to include the fully-connected
            layer at the top of the network.
        weights: one of `None` (random initialization),
              'imagenet' (pre-training on ImageNet),
```

```
            or the path to the weights file to be loaded.
    input_tensor: optional Keras tensor (i.e. output of `layers.Input()`)
        to use as image input for the model.
    input_shape: optional shape tuple, only to be specified
        if `include_top` is `False` (otherwise the input shape
        has to be `(299, 299, 3)` (with `''channels_last''` data format)
        or `(3, 299, 299)` (with `''channels_first''` data format).
        It should have exactly 3 inputs channels,
        and width and height should be no smaller than 75.
        E.g. `(150, 150, 3)` would be one valid value.
    pooling: Optional pooling mode for feature extraction
        when `include_top` is `False`.
        - `None` means that the output of the model will be
            the 4D tensor output of the last convolutional block.
        - `''avg''` means that global average pooling
            will be applied to the output of the
            last convolutional block, and thus
            the output of the model will be a 2D tensor.
        - `''max''` means that global max pooling will be applied.
    classes: optional number of classes to classify images
        into, only to be specified if `include_top` is `True`, and
        if no `weights` argument is specified.
# Returns
    A Keras `Model` instance.
# Raises
    ValueError: in case of invalid argument for `weights`,
        or invalid input shape.
"""
global backend, layers, models, keras_utils
backend, layers, models, keras_utils = get_submodules_from_kwargs(kwargs)

if not (weights in {'imagenet', None} or os.path.exists(weights)):
    raise ValueError('The `weights` argument should be either '
                     '`None` (random initialization), `imagenet` '
                     '(pre-training on ImageNet), '
                     'or the path to the weights file to be loaded.')

if weights == 'imagenet' and include_top and classes != 1000:
    raise ValueError('If using `weights` as `"imagenet"` with `include_top`'
                     ' as true, `classes` should be 1000')

# Determine proper input shape
input_shape = _obtain_input_shape(
```

```
    input_shape,
    default_size=299,
    min_size=75,
    data_format=backend.image_data_format(),
    require_flatten=include_top,
    weights=weights)

if input_tensor is None:
    img_input = layers.Input(shape=input_shape)
else:
    if not backend.is_keras_tensor(input_tensor):
        img_input = layers.Input(tensor=input_tensor, shape=input_shape)
    else:
        img_input = input_tensor

# Stem block: 35 x 35 x 192
x = conv2d_bn(img_input, 32, 3, strides=2, padding='valid')
x = conv2d_bn(x, 32, 3, padding='valid')
x = conv2d_bn(x, 64, 3)
x = layers.MaxPooling2D(3, strides=2)(x)
x = conv2d_bn(x, 80, 1, padding='valid')
x = conv2d_bn(x, 192, 3, padding='valid')
x = layers.MaxPooling2D(3, strides=2)(x)

# Mixed 5b (Inception-A block): 35 x 35 x 320
branch_0 = conv2d_bn(x, 96, 1)
branch_1 = conv2d_bn(x, 48, 1)
branch_1 = conv2d_bn(branch_1, 64, 5)
branch_2 = conv2d_bn(x, 64, 1)
branch_2 = conv2d_bn(branch_2, 96, 3)
branch_2 = conv2d_bn(branch_2, 96, 3)
branch_pool = layers.AveragePooling2D(3, strides=1, padding='same')(x)
branch_pool = conv2d_bn(branch_pool, 64, 1)
branches = [branch_0, branch_1, branch_2, branch_pool]
channel_axis = 1 if backend.image_data_format() == 'channels_first' else
x = layers.Concatenate(axis=channel_axis, name='mixed_5b')(branches)

# 10x block35 (Inception-ResNet-A block): 35 x 35 x 320
for block_idx in range(1, 11):
    x = inception_resnet_block(x,
                               scale=0.17,
                               block_type='block35',
                               block_idx=block_idx)
```

```
# Mixed 6a (Reduction-A block): 17 x 17 x 1088
branch_0 = conv2d_bn(x, 384, 3, strides=2, padding='valid')
branch_1 = conv2d_bn(x, 256, 1)
branch_1 = conv2d_bn(branch_1, 256, 3)
branch_1 = conv2d_bn(branch_1, 384, 3, strides=2, padding='valid')
branch_pool = layers.MaxPooling2D(3, strides=2, padding='valid')(x)
branches = [branch_0, branch_1, branch_pool]
x = layers.Concatenate(axis=channel_axis, name='mixed_6a')(branches)

# 20x block17 (Inception-ResNet-B block): 17 x 17 x 1088
for block_idx in range(1, 21):
    x = inception_resnet_block(x,
                               scale=0.1,
                               block_type='block17',
                               block_idx=block_idx)

# Mixed 7a (Reduction-B block): 8 x 8 x 2080
branch_0 = conv2d_bn(x, 256, 1)
branch_0 = conv2d_bn(branch_0, 384, 3, strides=2, padding='valid')
branch_1 = conv2d_bn(x, 256, 1)
branch_1 = conv2d_bn(branch_1, 288, 3, strides=2, padding='valid')
branch_2 = conv2d_bn(x, 256, 1)
branch_2 = conv2d_bn(branch_2, 288, 3)
branch_2 = conv2d_bn(branch_2, 320, 3, strides=2, padding='valid')
branch_pool = layers.MaxPooling2D(3, strides=2, padding='valid')(x)
branches = [branch_0, branch_1, branch_2, branch_pool]
x = layers.Concatenate(axis=channel_axis, name='mixed_7a')(branches)

# 10x block8 (Inception-ResNet-C block): 8 x 8 x 2080
for block_idx in range(1, 10):
    x = inception_resnet_block(x,
                               scale=0.2,
                               block_type='block8',
                               block_idx=block_idx)
x = inception_resnet_block(x,
                           scale=1.,
                           activation=None,
                           block_type='block8',
                           block_idx=10)

# Final convolution block: 8 x 8 x 1536
x = conv2d_bn(x, 1536, 1, name='conv_7b')
```

```
if include_top:
    # Classification block
    x = layers.GlobalAveragePooling2D(name='avg_pool')(x)
    x = layers.Dense(classes, activation='softmax', name='predictions')(x
else:
    if pooling == 'avg':
        x = layers.GlobalAveragePooling2D()(x)
    elif pooling == 'max':
        x = layers.GlobalMaxPooling2D()(x)

# Ensure that the model takes into account
# any potential predecessors of `input_tensor`.
if input_tensor is not None:
    inputs = keras_utils.get_source_inputs(input_tensor)
else:
    inputs = img_input

# Create model.
model = models.Model(inputs, x, name='inception_resnet_v2')

# Load weights.
if weights == 'imagenet':
    if include_top:
        fname = 'inception_resnet_v2_weights_tf_dim_ordering_tf_kernels.h
        weights_path = keras_utils.get_file(
            fname,
            BASE_WEIGHT_URL + fname,
            cache_subdir='models',
            file_hash='e693bd0210a403b3192acc6073ad2e96')
    else:
        fname = ('inception_resnet_v2_weights_'
                 'tf_dim_ordering_tf_kernels_notop.h5')
        weights_path = keras_utils.get_file(
            fname,
            BASE_WEIGHT_URL + fname,
            cache_subdir='models',
            file_hash='d19885ff4a710c122648d3b5c3b684e4')
    model.load_weights(weights_path)
elif weights is not None:
    model.load_weights(weights)

return model
```

```
import keras.backend as K
import tensorflow as tf
from keras.applications.inception_resnet_v2 import InceptionResNetV2
from keras.layers import Input, Dense, concatenate, Lambda
from keras.models import Model
from keras.utils import plot_model

from config import img_size, channel, embedding_size


def build_model():
    base_model = InceptionResNetV2(include_top=False, weights='imagenet', inp
                                   pooling='avg')
    image_input = base_model.input
    x = base_model.layers[-1].output
    out = Dense(embedding_size)(x)
    image_embedder = Model(image_input, out)

    input_a = Input((img_size, img_size, channel), name='anchor')
    input_p = Input((img_size, img_size, channel), name='positive')
    input_n = Input((img_size, img_size, channel), name='negative')

    normalize = Lambda(lambda x: K.l2_normalize(x, axis=-1), name='normalize'

    x = image_embedder(input_a)
    output_a = normalize(x)
    x = image_embedder(input_p)
    output_p = normalize(x)
    x = image_embedder(input_n)
    output_n = normalize(x)

    merged_vector = concatenate([output_a, output_p, output_n], axis=-1)

    model = Model(inputs=[input_a, input_p, input_n],
                  outputs=merged_vector)
    return model


if __name__ == '__main__':
    with tf.device("/cpu:0"):
        model = build_model()
```

```
print(model.summary())
plot_model(model, to_file='model.svg', show_layer_names=True, show_shapes

K.clear_session()
```