Vysoké Učení Technické v Brně
Fakulta informačních technologií

Strojové učení a rozpoznávaní 2020

Dokumentácia k projektu

Detektor jedné osoby z obrázku obličeje a hlasové nahrávky

Marek Žiška **(xziska03)**
Martin Osvald **(xosval03)**

Brno, 30.04.2020

Obsah

1 Audio systém

1.1 Popis systému

1.1.1 Spracovanie nahrávok

Na vytvorenie systému som použil deep learning knižnicu Keras. Na vyhodnocovanie som sa rozhodol použiť extrakciu príznakov z audio nahrávok. Na extrakciu príznakov som najprv potreboval vyfiltrovať audio nahrávky zo zložiek. Tieto audio nahrávky som načítaval pomocou funkcie *parse_audio_files* z jednej zložky *train* v ktorej boli zložky *non_target* a *target*. Jednotlivé cesty k nahrávkam som si uložil do zoznamu, z ktorého som následne mohol spracovávať jednotlivé nahrávky.

1.1.2 Extrakcia príznakov

Extrakcia prebieha vo funkcií *get_features*. V tejto časti som dosť experimentoval. Na jednoduchú extrakciu príznakov som použil knižnicu *librosa*. Pri návrhu môjho prvého modelu som z audio nahrávok extrahoval len MFCC príznaky, ktoré mapujú charakteristiky ľudského hlasu. Pri prvom modely som z týchto príznakov vytváral spektogramy, na ktorých som som trénoval konvolučnú neurónovú sieť. Túto metódu som nakoniec ale nepoužil z dôvodu nepresných výsledkov a použil som metódu, v ktorej som si vyextrahované príznaky, názov osoby a nahrávacieho sedenia uložil do csv súbora. Vo funkcií *get_set_of_data* som z csv súbora vytvoril dátové sety:
Train_data - numpy pole reprezentujúcé príznaky nahrávok pre dané osoby. Jednotlivé príznaky som s *StandardScaler* vyštandardizoval aby boli dáta viac menej normálovo distribuované a teda znížime možnosť neočakávane zlého natrénovania neúrónovej siete.
Train_labels - pole veľkosťou korešpondujúce *Train_data*, obsahujúce hodnoty 0 pre neznáme osoby a 1 pre hľadanú osobu.
Pri extrakcií a klasifikácií MFCC príznakov som sa ale nedostal k dobrým výsledkom, klasifikátor klasifikoval všetky nahrávky ako neznáme poprípade rozpoznal len zopár cieľových nahrávok. Rozhodol som sa skúsiť rozšíriť trénovací dataset. Rozmýšľal som nad transformáciami nahrávok alebo extrahovaním ďalších príznakov z audio nahrávok. Ako prvé som skúsil extrahovať ďalšie príznaky. Po ktorých pridaní som už dostal rozumnejšie výsledky a k transformáciam nahrávok som sa nedostal. Okrem MFCC som teda skúšal príznaky ako:
Spektrálny centroid - ukazuje na frekvenciu, okolo ktorej je sústredené najvačšie množstvo energie signálu.
Spektrálny rollof - Určuje, kde je uložených 90% energie spektra.

Spektrálny bandwidth - Ukazuje na rozloženie frekvencie.
Zero crossing rate - Počet prechodov signálu nulou.
Chroma príznak - Indikuje koľko energie je v každej triede tónu.
Postupne som tieto príznaky pridával/kombinoval a najväčšiu presnosť som dosiahol pri zahrnutí všetkých príznakov.

1.1.3 Návrh modelov neuronovej siete

Prvý návrh modelu pozostával z dvoch Dense vrstiev, vstupný tvar dát bol počet príznakov extrahovaných z náhravky, aktivačnú funkciu prvej vrstvy som zvolil *relu.* s tým že výstupnú dimenzionalitu som zvolil na 32, druhá vrstva používala aktivačnú funkciu *softmax*, ktorý sa typicky používa pre koncové vrstvy. Výstupná dimenzionalita bola 2, keďže klasifikátor rozdeloval dve triedy(target/non_target). Model som kompiloval s optimalizačnou funkciou *adam*, ktorý funguje na princípe stochastického poklesu gradientu a je nenáročný na výkon. Loss funkciu som zvolil *sparse_categorical_crossentropy*, ktorá je vhodná pre systémy s dvoma a viac výslednými triedami. Pri trénovaní modelu som sa viedol tým že počet epôch budem optimalizovať tak aby sedeli na práve taký počet, kedy sa presnosť modelu na trénovacích dátach blížil k jednej. Bolo to hlavne z dôvodu aby sa model nepretrénoval. Tento model avšak nepriniesol žiadne dobré výsledky, všetky nahrávky označoval za neznáme. Následne som skúšal zvyšovať počet skrytých vrstiev v návrhu modelu a to v takom zmysle aby počty výstupných dimenzionalít regresívne klesali. Taktiež som pri trénovaní zaviedol validačný set dát a všimol som si že, model už dávať presnejšie výsledky. Pri týchto obmenách som sa dopracoval až k výslednému modelu, z ktorého presnosťou som bol uspokojený.

1.1.4 Generovanie výsledkov

Pri generovaní výsledkov som použil funkcie:
model.predict_classes - ktorá vrátila tvrdé rozhodnutie 0 alebo 1 podľa veľkosti apriórnych pravdepodobností tried
mode.predict_proba - vráti tuple apriórnych pravdepodobností pre každú triedu. Z týchto pravdepodobností som generoval skóre. Skóre teda predstavovalo s akou pravdepodobnosťou si je systém istý tvrdým rozhodnutím.
A z týchto dát som vytvoril požadovaný finálny formát.

1.1.5 Čo by som zmenil

Pri trénovaní by som skúsil nahrávky transformovať a vygenerovať tak viacero nahrávok najmä pre hľadanú osobu, aby sa počty nahrávok pre jednotlivé triedy

mierne vyrovnali. Pri trénovaní by sa model natrénoval presnejšie, plus validačný set by dával väčší zmysel, keďže zastúpenie oboch tried by v ňom bolo väčšie nie ako to je v prípade odovzdaného systému, kde pri trénovaní prevažovala trieda *non_target*.

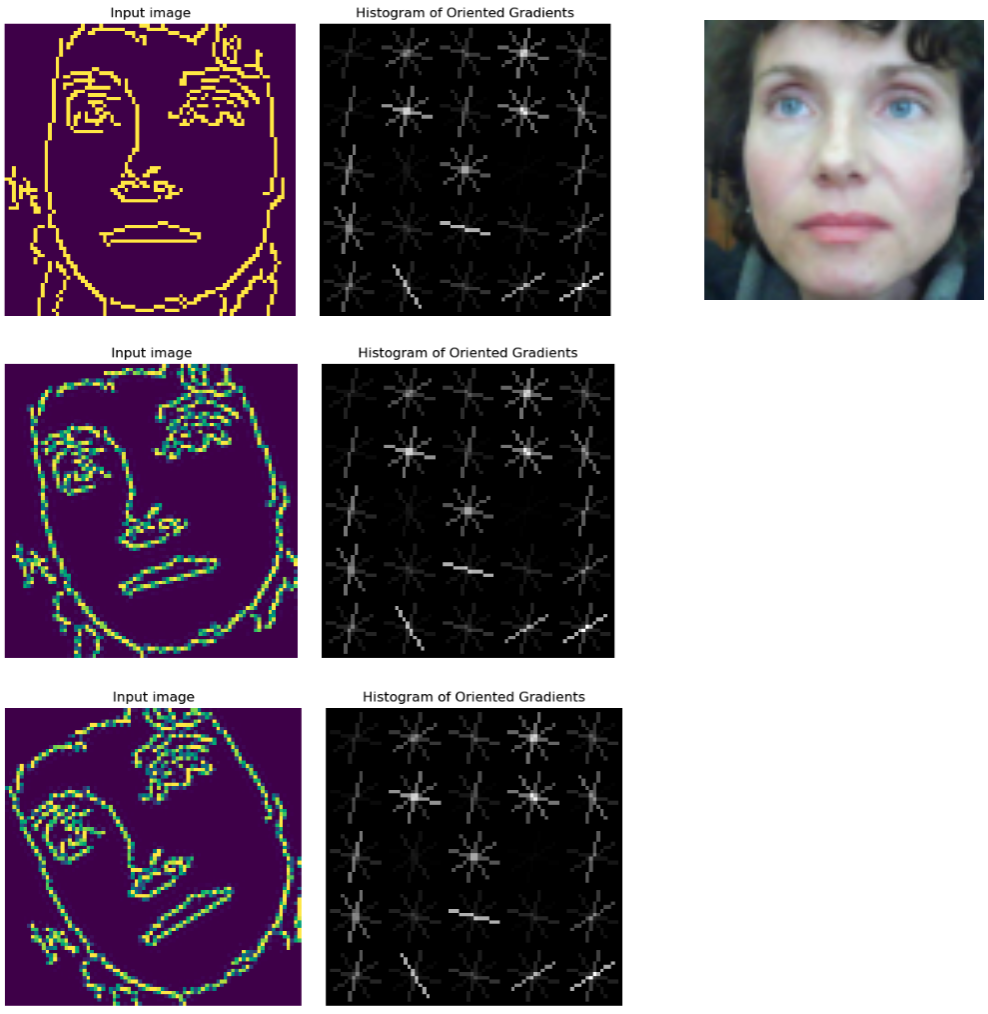
1.2 Návod na zreprodukovanie výsledkov.

Ako prvé je samozrejme potrebné nainštalovanie všetkých nástrojov, uvedených v README. Pre trénovanie bude potrebné vytvoriť zložku *train*, v ktorej budú dve zložky *non_target* a *target*. V zložkách sú obrázky a nahrávky nehľadaných a hľadanej osoby, na týchto dátach bude systém natrénovaný. Zložku s vyhodnocovacími/nevidenými dátami. *eval*. Trénovanie neurónovej siete som vykonával v jupyter notebooku *audio_classifier_MLP.ipynb*. Keďže pri trénovaní neurónovej siete vznikajú rôzne váhy, preto som model na ktorom sa vyhodnocovali evaluačné dáta uložil separátne, aby bolo možné totožné zreprodukovanie výsledkov(súbor *audio_classifier_MLP.py*).

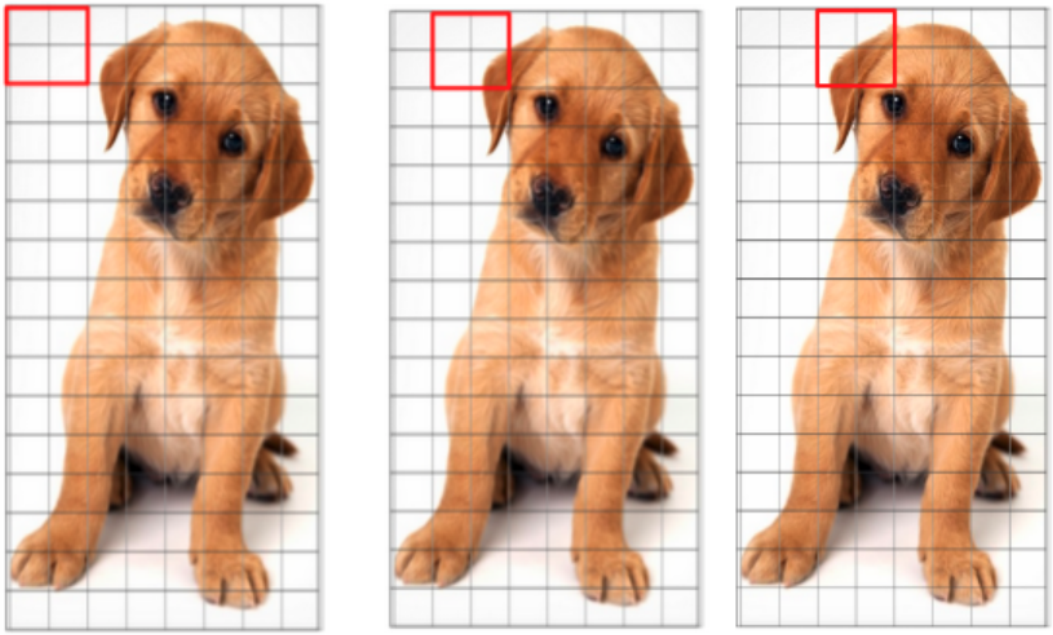
2 Image systém

2.1 Popis systému - image_classificator_training.py

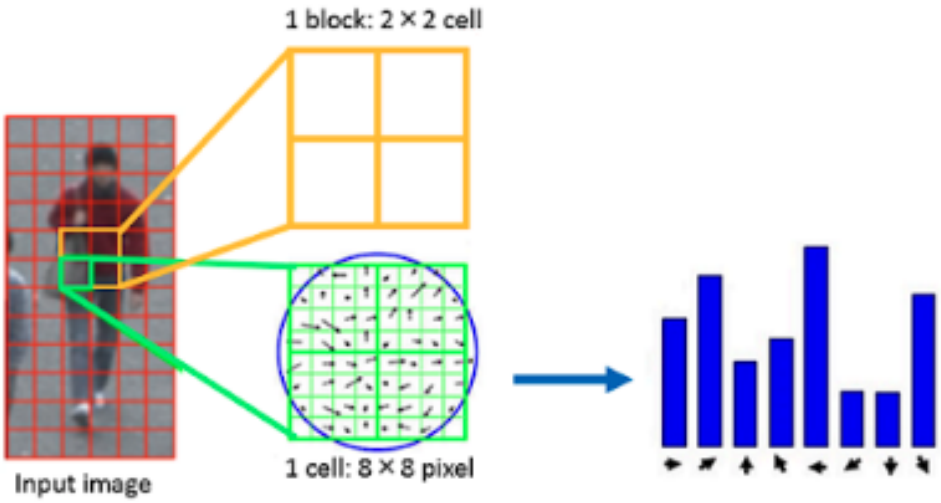
Program načíta obrázky pomocou png2fea ktorá je upravená z IKRLIB.
Png2fea() otvorí obrázok, zmení mu farbu z knižnice cv2 na COLOR_BGR2GRAY a resizne na dimenziu (80,80). Ked sa resizne tak pomocou imutils zvýrazni hrany. Takýto obrázok potom pridá do slovníka ktorý vráti všetky takto upravené obrázky.
Evaluate() prechádza obrázky v slovníku extrahuje príznaky Histogramu Orientovaných Gradientov (HOG) z obrázku. Kvôli veľmi malému množstvu obrázkov potom obrázok točí do + 90 stupňov a do – 90 stupňov a znova extrahuje príznaky HOG. V súbore config.json sa dá upraviť cesta z ktorej zložky sa budú načítavať obrázky ako aj options HOG gradientu a ako aj flag na print. Flag print ak je nastavený na 1 tak nám ukazuje ako narába s obrázkami v Evaluate .

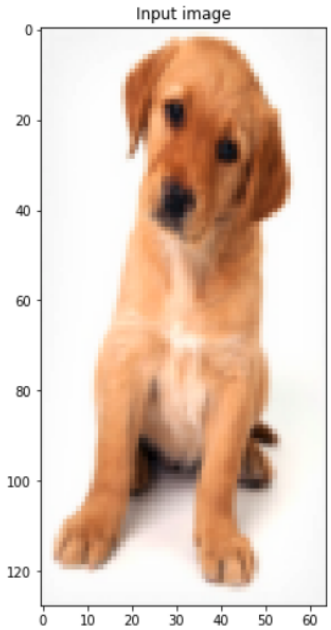


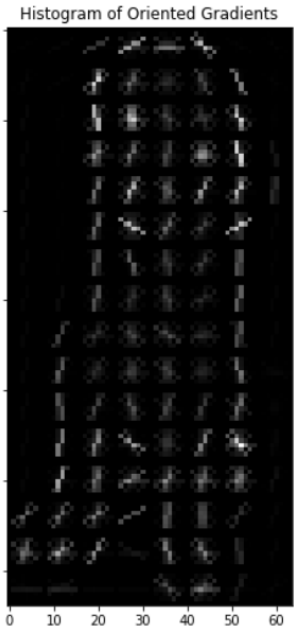
HOG funguje na princípe že obrázok rozdelí do jednotlivých takzvaných cells (mriežok) Toto číslo sa dá tiež upraviť v config.json file pod premenou $\text{pixels}_p er_c ell, cell\_per\_blockznamenakolkotvorekomprejdemvrmciblocku.$



V rámci týchto cellov sa zistí rozdiel medzi jednotlivými pixelmi a podla toho sa určí smer vektoru. Za nás to robí funckia HOG ktorá nam vráti array príznakov .



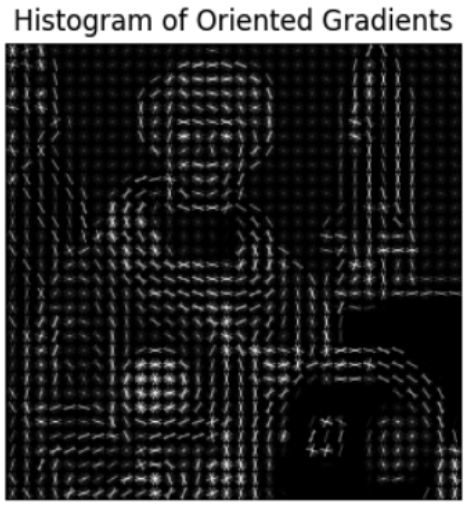




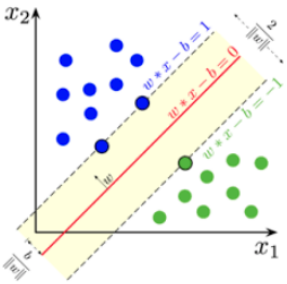
Ku príznakom pridávame ďalší vektor a to buď 1 alebo 0 poďla toho či sme dali non_target_train obrázok alebo target_train. Tieto príznaky s označením potom trénujeme v Supporteddvector machine svc ktorá je lineárna.

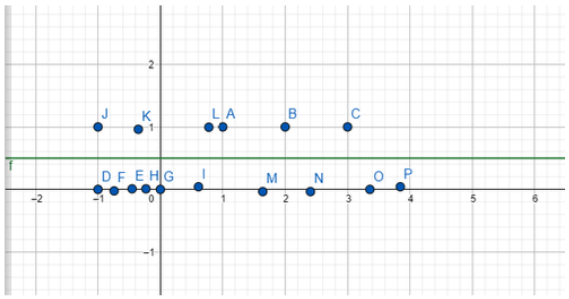
```
print("Training a Linear SVM Classifier")
svc =svm.SVC(kernel='linear', C=5.0,probability=True)
svc.fit(datas,labels)
print("Finished training of Linear SVM Classifier")
joblib.dump(svc, image_classifier)
print("Saving Linear SVM Classifier")
```

Trénovanie je veľmi rýchle kvôli tomu že do hog features dávame len dvojfarebné obrázky a rozdelili sme ho na málo blockov.
Testovaním sa presnosť zlepšila ak cells_per_block dáme menšie čislo kvôli tomu že ak hog gradient vytiahne viac príznakov ako je na obrázku dole tak klasifikátor potom skoro vždy nevedel určiť či sa daná osoba je na obrázku. Je to kvôoli tomu ,že by tam bolo o dosť viac hrán a tým pádom aj väčšia nepresnosť.



SVM nám dokáže oddeliť dve skupiny bodov pomocou nejakého vectoru . SVM využívame na rozdelnie či na obrázku je daný target alebo není .





2.2 image_classifier_evaluation.py

Znova prechádza obrázky a otvára ich spôsobom ako v pri trenovaní . (Otvorí , zmení farbu , resizne na (80,80) a zvýrazni hrany) Potom extrahuje hog features rovnako ako pri trénovaní a tým že obrázok má rovnakú veľkosť tak nerieši scaling.
Score sa zistí pomocou tohto kódu , kde nastavíme hranu na 0.29 precent. Predict_proba nám dáva dva výsledky a to percentuálne ako si je istý či to je a či to není daný target. Pravdepodobne toto zohralo rolu na Cdet. Snažil som sa vyhovieť požiadavke čim som si viac istejši skóre , takže keď je rozhodnutie 0 dávam prvé percentá ak je 1 dávam druhé precentá .

2.3 Čo by sa dalo zlepšiť

V image klasifikátore sa dali upraviť parametre hog na optimálnu veľkosť príznakov z HOG , teraz sa asi menej príznakov hľadá ako by sa malo. Taktiež by sa dal zväčšiť dataset pomocou scalingom obrázkov. Vyhodnocovanie klasifikátoru pri obrázkov by taktiež vyhodnocovalo pomocou roznych velkosťí obrázku tak ako sa to robí klasicky podľa hog klasifikátoru. Poprípadne by sa dalo ešte z obrázku najprv vyrezať tvár pomocou univerzálneho predtrénovaného klasifikátoru na tváre , resiznuť pootáčať a natrénovať neuronovú sieť. Taktiež na nontarget by sa dal zväčšiť dataset pomocou obrázkov z internetu.

```
datas.append(fd)
#label = model.predict(fd.reshape(1, -1))[0]
label = int((model.predict_proba(fd.reshape(1, -1))[:, 1] >= 0.29).astype(bool))
score = model.predict_proba(fd.reshape(1, -1))
#score = str('{0:.0%}'.format (score[0][1]))

if label == 0 :
    score = score[0][0]
    pass
else:
    count+=1
    score = score[0][1]


table[image.split('/')[-1].split('.')[0]] = (label, score )
names.append( image.split('/')[-1].split('.')[0])
labels.append(label)
scores.append(score)
```

3 Image a Audio systém

3.1 Popis systému

V súbore both.py sa prechádzajú súbory postupne tak ako v audio klasifikátore tak aj v image klasifikátore. Zistí sa percentuálne score a ako aj tvrdé rozhodnutie obidvoch klasifikátorov. Ak tvrdý výsledok je rovnaký u obidvoch klasifikátorov tak sa robí aritmetický priemer obidvoch klasifikátorov. Audio klasifikátor bol podľa testovanie presnejší oproti obrázkovému klasifikátoru tak sme mu dali väčšiu váhu. Ak sa score líšili viac ako o 25 percent tak sme sa rozhodovali podľa toho ktorý klasifikátor bol istejší vo výsledku.

```
if sound_deci != image_deci:
    treshold = abs(sound_perc-image_perc)
    if treshold > 0.25:
        if sound_perc > image_perc:
            decision = sound_deci
        else:
            decision = image_deci
    else:
        decision = sound_deci

else:
    decision = sound_deci
```