Recunoașterea vizuală a personajelor din Familia Simpsons

Stanciu Andrei Călin

Proiectul realizat se împarte în două categorii: rezolvarea primului task (detectarea tuturor fețelor), si rezolvarea celui de-al doilea task (detectarea unor 4 personaje diferite). Aproape toate tehnicile folosite și detaliile corespunzătoare vor fi descrise în raport cu prima cerință, urmând ca pentru a doua cerință să descriu pe scurt (micile) modificări aduse față de soluția implementată până în acel punct.

Observație: În acest document am introdus doar descrierea variantei finale, și unele modificări ce au condus spre aceasta. În total, am încercat mai mulți algoritmi (folosiți mai mult sau mai puțin corespunzător) - toate bucățile intermediate / nefolosite în versiunea finală au fost introduse în fișierele din folderul *aux_py* al proiectului.

Taskul 1

Culegerea de date pentru antrenare

Extragerea exemplelor pozitive a fost destul de directă, întrucât au fost puse la dispoziție fișiere ce conțin coordonatele a câteva mii de exemple pozitive. Singurele variațiuni apar la extragerea exemplelor negative:

- Varianta folosită pentru antrenarea clasificatorului final a constat (doar) în a culege bucăți random din fiecare imagine în care se aflau exemple pozitive, cu condiția de a nu avea nici o intersecție cu vreun chenar care indică exemple pozitive

- O altă variantă abordată (pe firul unei idei ce urmează a fi prezentată) a fost, pe lângă generarea de chenare random cu constrângerea de mai sus, un număr minim de exemple negative să prezinte un conținut ridicat de pixeli galbeni (pe parcursul acestei documentații, *galben* este definit ca fiind o submulțime a spațiului RGB, cu limitele ei determinate empiric – R > 150, G > 130, B < 80); un dezavantaj al acestei abordări este timpul relativ ridicat de colectare a datelor, motiv pentru care ele erau colectate separat si re-folosite la fiecare antrenare. În final, am renunțat de tot la această idee, întrucât antrenarea cu prima metodă de colectare a exemplelor negative dădea rezultate cel puțin la fel de bune.

Modelul folosit pentru extragerea de feature-uri și clasificare, și antrenarea lui

Modelul folosit este o implementare (modificată minimal) a rețelei neurale reziduale cu 18 straturi, ResNet18 (*fără preantrenare*) . Acesteia i s-a modificat dimensiunea inputului (72 x 72 x 3 canale de culoare) și ultimul strat (doar doi neuroni, cu activare softmax, pentru clasificare).

Implementarea acestei rețele a folosit primitivele puse la dispoziție de API ul din keras. Modelul a fost definit și antrenat în funcția train_resnet_task1 (respectiv train_resnet_task2 pentru a doua

cerință), iar blocurile reziduale (shortcut nemodificat, sau trecut printr-un layer convoluțional cu kernel 1x1) au fost implementate în două clase separate, care moștenesc clasa Layer din keras.

Optimizorul folosit este Stochastic Gradient Descent cu un learning rate de 10^-4 și momentum de 0.9. Am mai încercat tot SGD cu learning rate de 10^-3 și același momentum cu rezultate asemănătoare, și optimizorul Adam, dar care dădea rezultate inferioare.

Funcția de loss este binary crossentropy, batch size este 64, iar modelul a fost antrenat 5 epoci (batch size 32, sau un număr ceva mai mare sau mai mic de epoci a dat rezultate asemănătoare).

Realizarea de predictii

Realizarea predicțiilor respectă constrângerea enunțată în cerință, și anume aplicarea conceptului de *sliding window.* În cele ce urmează, voi ilustra în linii mari modul de execuție al funcției *predictions_task1,* precizând diferite aspecte cheie:

- Pentru fiecare imagine din setul de date, se construiește o matrice 2D de sume parțiale, definită astfel: y_mat[y][x] = numărul de pixeli *galbeni* în secțiunea [0: y, 0: x] a imaginii complete; această precalculare reduce timpul de aflare al numărului de pixeli galbeni pentru oricâte secțiuni arbitrare din imaginea corespunzătoare în O(1) (per query), în loc de O(n * m) (per query) asociat variantei naive. În practică, această optimizare reduce timpul de predictie de la ~100-300 secunde pe imagine, la ~1-5 secunde pe imagine

- Aplicarea sliding window-ului pentru diferite dimensiuni și xy-ratio: o abordare inițială a constat în redimensionarea imaginii inițiale și glisarea peste imaginea redimensionată cu un pătrat de dimensiuni constante (72 x 72), și reținerea coordonatelor corespunzătoare acelui window, în imaginea originală, ne-redimensionată; a doua variantă a constat în a selecta direct mai multe window-uri de dimensiuni diferite și glisarea acestora pe imaginea inițială, iar înainte de intrarea ferestrei în rețeaua neuronală, era redimensionată la 72 x 72 x 3. Deși echivalente dpdv. Matematic (și în mare măsură și computațional) prima variantă introducea mici inexactități în valoarea coordonatelor în urma unor operații de înmulțire / împărțire, și, ca o notă subiectivă, dimensiunile folosite păreau mai greu de controlat / implementarea era predispusă la mai multe bug-uri. Din punct de vedere al dimensiunilor efective, am folosit ferestre cu y/x ratio de 1 și 1.4, iar ca valori pentru x, toate valorile între 20 și 100 de pixeli inclusiv, cu pasul 10. O mulțime de alte valori au fost testate, (eg. y/x ratio 1.5 sau pasul pentru dimensiunea lui x 1 în loc de 10), acestea dădeau rezultate asemănătoare, dar creștea durata de predicție, întrucât erau procesate mai multe ferestre.

- Pentru fiecare fereastră, era verificată cantitatea de galben, comparată cu un prag superior și inferior de exact 0.3 (inferior) și 0.7 (superior): în cazul în care fereastra aleasă nu respectă aceste limite, nu este luată în calcul. În mod surprinzător, întregul mecanism de predicție este destul de senzitiv la modificările acestor limite (coborârea limitei inferioare la 0.2 și creșterea celei superioare la 0.75 a condus la un average precision de cca 0.4-0.5, spre deosebire de actualul 0.75 peste datele de validare). Această limită are mai multe roluri, și anume reducerea timpului de predicție, dar și eliminarea candidaților care (mai mult ca sigur) nu sunt fețe (atât cele care nu conțin deloc galben, dar și chenarele care conțin numai

galben, cum ar fi ferestrele foarte mici care conțin bucăți foarte mici de față - sau ferestrele peste suprafețe galbene, uniforme cum ar fi unele haine).

- Trecerea tuturor ferestrelor ce au depășit selecția anterioară prin resnet și obținerea predicțiilor corespunzătoare: cele care au o probabilitate de a nu fi față decât față sunt respinse. De precizat că, în urma selecției ferestrelor cu un minim de galben, cel puțin la prima vedere ar părea că rețeaua ar fi avantajată pe antrenarea cu exemple negative care de asemenea conțin mult galben, dar în urma unor teste, am observat că (cel puțin în implementarea curentă) nu se remarcă aproape nici o diferență.

- Aplicarea non-maximum suppression, cu un prag de 0.15 (este important ca pragul să fie destul de mic, dar nu 0, pentru a putea respinge inclusiv ferestrele foarte mici încapsulate în ferestrele mai mari, dar în același timp pentru a nu respinge ferestrele asociate fețelor apropiate).

- Salvarea predicțiilor, în formatul cerut

Taskul 2

Diferențe comparând cu taskul 1

Diferența semnificativă între taskul 1 și taskul 2 o prezintă rețeaua neuronală, și anume, în loc de 2 neuroni pe ultimul strat, am 6, corespunzător claselor: bart, homer, lisa, marge, unknown, negative (În plus, rețeaua pentru taskul 2 este antrenată pe 7 epoci, iar loss function-ul este categorical crossentropy).

Procedeul de predicție este aproape identic cu cel pentru taskul 1, cu excepția interpretării predicțiilor date de resnet: dacă clasa cu probabilitatea cea mai mare este unknown sau negative, fereastra este respinsă.

Legat de clasele folosite, implementarea modelului curent etichetează fiecare față unknown ca fiind negative, alături de exemplele care nu sunt fețe (lăsând clasa unknown nefolosită). Ambele variațiuni (cea descrisă anterior, cât și cea în care fețele unknown sunt păstrate în clasa lor separată) dau rezultate asemănătoare, cu un average precision puțin mai bun în cazul ultimei implementări (unknown inclus în negative).

Alte câteva detalii ale implementării

- Funcția folosită pentru calcularea IOU este implementată manual, prin metoda naivă de a număra direct numărul de pixeli din intersecție (dimensiunile de cel mult zeci/ 1 – 2 sute de pixeli permit această implementare) - oferă rezultate precise pentru cazuri arbitrare de intersecție.

- Aplicarea non-maximum suppression este de asemenea implementată manual, și anume prin sortarea (crescător după scor) a tuturor ferestrelor primite ca input, și recursiv, cât timp mai sunt ferestre rămase, extragerea ferestrei cu scor cel mai mare și eliminarea ferestrelor ce se suprapun cu aceasta (i.e. au IOU calculat ce depășește pragul setat).