UNIVERZITET SINGIDUNUM

Tehnički Fakultet

Optimizacija metoda za prepoznavanje lica za rad na mobilnim uređajima

- master rad -

Mentor: **Kandidat:**

prof. dr ***Milan Milosavljević*** ***Davor Jordačević***

Beograd, ***2022.***

Sadržaj

Uvod

Analiza lica predstavlja jedan od bitnih procesa u našim životima. Ljudi analizom lica prikupljaju bitne podatke o drugim osobama. Ovo uključuje podatke o broju godina, polu, rasnoj pripadnosti, ali ih i na osnovu lica možemo i prepoznati. Takođe možemo prepoznati da li je osoba srećna ili tužna, ili pak neku drugu emociju. Pokreti usana su važni u oblasti prepoznavanja govora, kao i sve popularnijoj oblasti kao što je generisanje lažnih snimaka. Metode analize lica nam mogu reći gde je usmeren pogled neke osobe, odnosno šta privlači njenu pažnju, i ovo može biti posebno interesantno u marketingu i šopovima, kao i kazinima. U medicini ove metode mogu biti od koristi za prepoznavanje nekih bolesti, poput autizma koji se odlikuje time što osobe imaju poteškoće da iskažu svoje emocije. Sve navedene metode koriste kako ljudi, tako i računari.

Metode analize lica nalazimo na svakom koraku, od otključavanja telefona licem, do plaćanja licem putem određenih aplikacija, automatskog tagovanja na društvenim mrežama, video nadrzoru, pametnim kućama. Ova lista se može nastaviti, i to je samo primer koliko je ova oblast korisna, i koliko će tek biti.

Tokom vremena su se smenjivale razne metode za prepoznavanje lica, počev od metoda koje nisu koristile neuronske mreže, pa sve do danas popularnih dubokih neuronskih mreža. Jedan od problema sa ovakvim mrežama je to što su kompleksne, i u većini slučaja je potrebno posedovati skup hardver kako bi se ovakve mreže pokrenule i radile u realnom vremenu. Glavni problem koji je razmatran u ovom radu jeste optimizacija dubokih neuronskih mreža za prepoznavanje lica, za rad na uređajima sa malom potrošnjom električne energije, kao što su mobilni telefoni, mini računari (IoT računari) poput Raspberry PI računara, i ostali edge uređaji poput Nvidia Jetson serije. Većina ovakvih uređaja poseduje ARM procesore, ili određene Intel i AMD čipove sa malom potrošnjom, što ih stavlja u istu kategoriju. Bitna razlika je u tome da ovakvi uređaji nemaju jake grafičke procesore (ili kartice), te je sama paralelizacija modela smanjena, i optimizacija je neophodna.

Metode optimizacije su neophodne, jer je potreba za manjim, bržim i manje zahtevnim modelima sve veća, i sa napretkom IoT uređaja ova oblast dostiže vrhunac. Pored IoT uređaja, u današnje vreme se većina mobilnih telefona otključava uz pomoć lica, ali i veliki broj mobilnih aplikacija koristi lice korisnika za određene verifikacije.

Jedna od mreža koja je postigla izuzetne rezultate na zadacima prepoznavanja lica jeste duboka neuronska mreža pod nazivom ArcFace [1]. Cilj ovog rada je da koristeći jednu već istreniranu i napravljenu mrežu, bez značajnog gubitka preciznosti, ubrzamo mrežu, i testiramo njenu brzinu na mobilnom uređaju. U ovom radu korišćene su analitičke metode kako bi se svaka celina razložila na delove i bolje objasnila.

Postoji veliki broj metoda koje se mogu koristiti za ubrzanje dubokih neuronskih mreža, i veliki broj alata koji se mogu koristiti, poput specijalizovanih razvojnih okvira za kvantizaciju i

pruning, kvantizaciju sa kalibracijom, prenos znanja raznim metodama, spajanje slojeva radi ubrzanja.

Kao metode za ubrzanje u ovom radu korišćene su:

1. Kvantizacija
2. Prunning
3. Zamena backbone arhitekture
4. Prenošenje znanja metodom učitelj-student.

Neuronske mreže su veoma zahtevne. Pored vremena koje je potrebno za inferencu (pokretanje) mreže, potrebno je imati određeni prostor na disku za čuvanje same mreže kao i određenu količinu radne memorije (RAM). Iako se dostupni resursi svakodnevno povećavaju, novijim procesorima, brzim memorijama, većim prostorom, često se dešava da postoje mesta gde je količina resursa ograničena. Ovo je slučaj sa edge uređajima, mobilnim telefonima i ostalim hardverom koji se napaja baterijama. U slučajevima kada brzina nije presudna, sama optimizacija je neophodna kako bi se smanjima potrosnja električne energije i sačuvala baterija.

Ukoliko sa druge strane naše modele želimo držati na oblaku (eng. cloud), optimizacija modela bi dovela do smanjenih troškova hardvera, ali i do manje emisije ugljen dioksida i ostalih komponenti. Oblak nije uvek idealno rešenje. U slučajevima gde je jako često potrebno pokretati inference mreža, servisi poput Amazona mogu biti jako skupi.

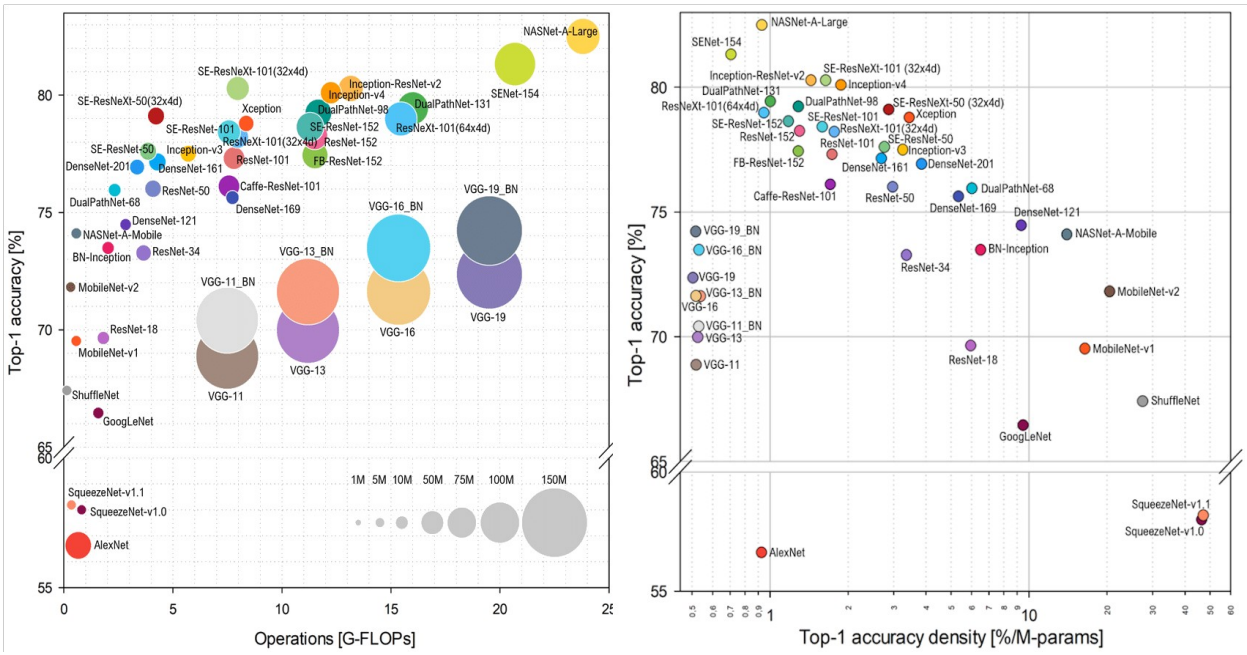
Ukoliko bi naše modele pokretali direktno na mobilnim uređajima i ostalim uređajima koji se napajaju baterijama, samo se nameće da je optimizacija neophodan korak, iako sama brzina nije uvek kritična.

Uzmimo za primer Resnet50 mrežu [6]. Ova mreža sadrži 26 miliona parametra. Tokom jednog prolaza, potrebno je izračunati 16 miliona aktivacija. Ukoliko koristimo 32-bitni brojeve sa pokretnim zarezom za čuvanje težina i aktivacija, potrebno je 168MB radne memorije. Optimizacijom ovih parametra možemo doći do mreže koja je manje memorijski zahtevna.

1. Arhitektura modela

Pre kretanja razvoja modela za bilo koju situaciju, prva potrebna stvar je osmisliti i izabrati željenu arhitekturu. Ovaj korak je ključan zbog toga što iz njega proizilaze ostali koraci koji će biti potrebni kako bi se model mogao pokrenuti na mobilnom uređaju. Na slici 1 moguće je videti vizuelno reprezentovano kako veličina i kompleksnost modela utiču na broj operacija po sekundi (GFLOPS), ali i na preciznost samog modela na ImageNet setu podataka [40]. Na istoj slici se mogu videti i oba modela koja će biti korišćena u nastavku ovog rada, a oni su ResNet34 i ResNet18.

Veliki broj parametara u modelu se može interpretirati i kao veće zauzeće memorije na grafičkoj kartici. Premda to za slučaj rada na mobilnim uređajima ne deluje bitno, to nije tako. Ovo se može reflektovati i na naš slučaj zbog toga što su resursi na mobilnim uređajima ograničeni, i paralelizacije koje je moguće postići na desktop uređajima, nije moguće ostvariti na mobilnim telefonima. Moderne desktop grafičke kartice poseduju preko 8GB radne memorije u koju se mogu smestiti parametri modela, sa značajno više radne memorije za ostatak softvera, dok je kod mobilnih telefona ova memorija deljena, i često manja od pomenutih 8GB. Upravo zbog ovoga, manje parametra znači bolji rad na mobilnim uređajima, ali manje parametra ne mora da se preslika kao manja preciznost, i upravo je to razlog postojanja specijalizovanih arhitektura za mobilne uređaje.



Slika 1. (Levo) Poređenje preciznosti modela na ImageNet setu podataka u odnosu na veličinu modela i broja parametara [41]. (Desno) Prikaz efikasnosti modela u odnosu na broj parametara.

Sa slike 1 vidimo kako veliki broj modela uspeva da postigne balans između dobre preciznosti i manjeg broja parametara. Ovakvi modeli se služe raznim tehnikama, ne samo kreiranjem manjeg modela, već i raznim ručno rađenim funkcijama (slojevima) koji su sami po sebi manje zahtevni od već postojećih i implementiranih u razvojne okvire za duboko učenje. Arhitekture specijalizovanih modela nisu dovoljne u svakom slučaju, već i način na koji se koriste, odnosno razvojni okviri preko koga se pokreću. Više o specifičnim funkcijama mobilnih platforma će biti reči na kraju ovog rada.

Osim ručnog kreiranja modela, sve popularnije su tehnike automatskog pronalaženja modela uz pomoć razvojnih okvira poput AutoML-a, AutoPyTorch-a i NASLib-a. U ovim metodama se

najbolji model pronalazi automatski, tako da zadovoljava određene kriterijume, što bi u ovom slučaju bila efikasnost i preciznost.

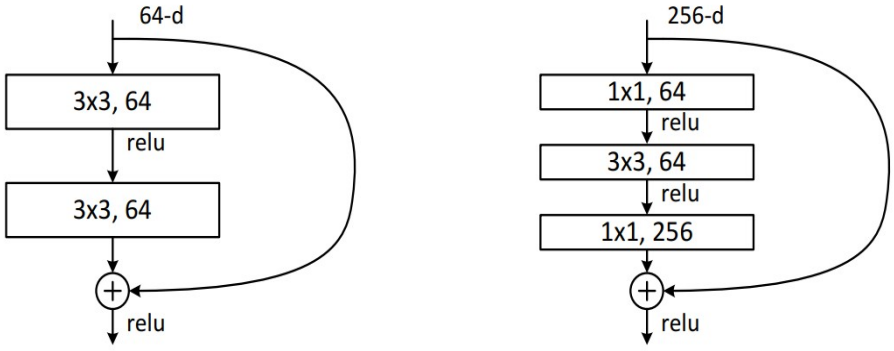
U ovom radu ćemo se zadražati na jednoj kompleksnijoj mreži. Počnimo ovu temu pričom o ResNet arhitekturi, nakon čega nam slede koraci njene optimizacije.

1.1 ResNet arhitektura

ResNet mreža predstavlja konvolucionu neuronsku mrežu koja sadrži određena poboljšanja. Ova arhitektura je nastala 2015 godine, i od trenutka nastanka do sada, predstavlja najkorišćeniju arhitekturu. Naime, ResNet, ili kako se često naziva rezidualna mreža sadrži određene skokve (prečice) preko jednog ili više narednih slojeva.

Motivacija za korišćenjem ResNet blokova je bila rešiti problem degradacije gradijenta u dubljim slojevima mreže [30], i dokazati da je moguće krerati duboku neuronsku mrežu, bez gubitka preciznosti sa dodavanjem novih slojeva. Pored ovog, ovo je pomoglo i u rešavanju problema prokletstva dimenzionalnosti [29]. U klasičnim neuronskim mrežama, svaki sloj ima izlaz ka narednom sloju, dok kod rezidualnih blokova, svaki sloj ima izlaz ka narednom sloju, ali i ka slojevima 2-3 skoka unapred.

Klasičan rezidualni blok se sastoji od 2 konvoluciona sloja veličine 3x3, sa ReLU aktivacionom funkcijom. Ovakvi slojevi su korišćeni u mrežama poput ResNet18 i ResNet34, dok su za dublje mreže poput ResNet50, ResNet101, ResNet150 korišćeni blokovi sa uskim grlom (Slika 2). Ovakvi blokovi se sastoje od dva konvoluciona sloja veličine 1x1, između kojih se nalazi jedan konvolucioni sloj veličine 3x3. Osim toga, sadrže i već pomenuti skok. Osim osnovne namene ovih blokova, dokazano je da oni uspešno sprečavaju overfiting mreže (problem velikom podudaranja).



Slika 2. Primer ResNet bloka [6].

Overfiting predstavlja situaciju u kojoj model savršeno nauči da vrši predikciju za instance iz trening seta, ali ima veoma slabu sposobnost predikcije za instance koje se i malo razlikuju od naučenih, odnosno model nije sposoban da generalizuje, što u osnovi predstavlja kriterijum koji je ključan kod razvoja ne samo modela koji se koriste za prepoznavanje lica, već i svih ostalih modela.

Kod ResNet modela, broj nakon imena predstavlja broj slojeva u modelu. Na većini zadataka za koje se ove mreže koriste se primećuje da bolju preciznost postižu mreže sa više slojeva, odnosno ResNet100 je bolji od ResNet50, ResNet50 od ResNet34, ResNet34 od ResNet18. Ovu činjenicu ćemo iskorititi kasnije u radu za treniranje mreže sa prenosom znanja, kada ćemo uz pomoc ResNet34 mreže istrenirati ResNet18 mrežu.

Konkretno u ovom radu, koristićemo jednu modifikaciju ResNet mreže, pod nazivom IResNet [20]. Ako za primer uzmemo ImageNet [21] set podataka i mrežu sa 50 slojeva, IResNet dostiže preciznost od 77.31%, dok ResNet 76.12%. Mada ovo ne deluje kao značajan pomak, ImageNet set je veliki, sadrži mnogo klasa i milione slika. Ono što je i bitnije od skoka u preciznosti je to što se do ovoga došlo bez povećanja kompleksnosti modela. Zamena same arhitekture mreže sa ResNet na IResNet je prva stvar koja može dovesti do boljih performansi.

Treba uzeti u obzir da se i dalje radi o kompleksnoj arhitekturi mreže, te da se uvek može odabrati neka od opcija specijalizovanih za rad na uređajima sa ograničenim resursima.

1.2 IResNet arhitektura

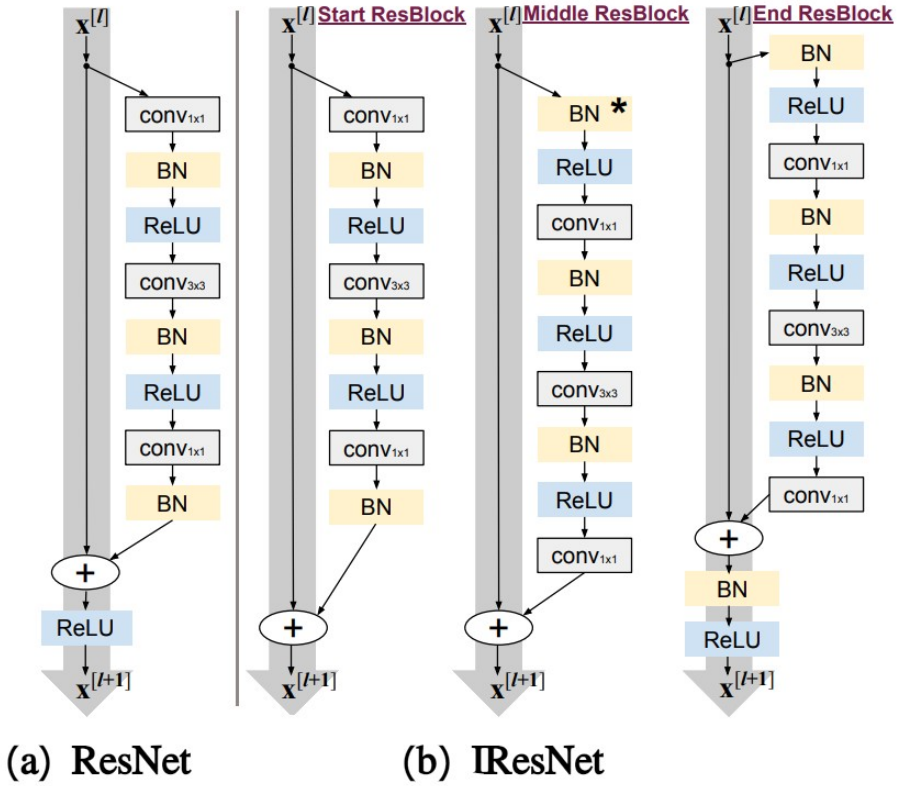
Kako bi se ResNet arhitektura konvertovala u IResNet, za početak je takvu mrežu podeliti na više faza, u zavisnosti od dubine mreže. Svaka faza se može dodatno podeliti na više delova, u ovom slučaju tri dela (početak, sredina i kraj). Kod originalne ResNet arhitekture svi rezidualni blokovi su isti, bez obzira o kojoj se fazi i delu faze govori.

Ono što se primećuje da u glavnoj grani i toku signala kroz mrežu ne postoji ni jedan sloj batch normalizacije. Ovo je prva stvar izmenjena u novoj mreži, dodavanje batch normalizacije van rezidulanog dela, u poslednjoj fazi.

Originalna mreža sadrži ReLU aktivacione funkcije u glavnom toku signala, što može negativno uticati na propagaciju signala na početku treninga (ne dozvoljavajući propagaciju negativnih vrednosti), i ovo se može videti na slici N (a).

U IResNet arhitekturi, postoje tri tipa rezidualnih blokova, svaki malo drugačiji od originalne implementacije. Ono što je najbitnije je da se ReLU aktivacija ne pojavljuje u svakom od ovih

blokova (u glavnom toku signala), već samo u blokovima koji se koriste u krajenjem delu svake od faza u mreži. IResNet blokovi su predstavljeni na slici 3 (b).



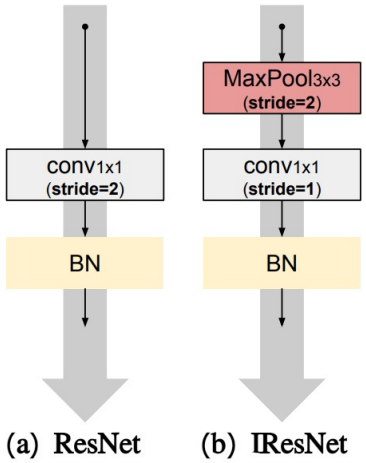
Slika 3. (a) ResNet blok u originalnoj mreži. (b) IResNet blokovi (početni, srednji i krajnji).

Pre svake ReLU aktivacije je dodat sloj batch normalizacije kako bi se stabilisao signal (osim u prvom rezidualnom bloku u srednjem sloju svake faze).

IResNet arhitektura omogućava lakšu propagaciju signala kroz mrežu, omogućavajući brži trening mreže, što može dovesti do smanjenja resursa potrebnih za trening i smanjiti troškove. Druga prednost ove arhitekture je mogućnost treniranja mnogo dubljih mreža nego korišćenjem klasične rezidualne arhitekture (do čak 3000 slojeva).

ResNet mreža ima dve putanje kojima je moguća propagacija signala. Ukoliko ulazne dimenzije nisu zadovoljavajuće, primenjuje se takozvana prečica [6], čija je uloga da dimenzije dovede na željenu veličinu. Ova prečica u originalnom modelu sadrži dva sloja, gde prvi sloj predstavlja 2D konvoluciju korišćenjem pomeraja sa vrednošću 2, nakon čega sledi sloj batch normalizacije.

Kod IResNet modela, pre ova dva sloja uveden je MaxPool sloj, koji koristi kernel veličine 3x3 sa pomerajem vrednošću 2. 2D konvolucija u ovom slučaju koristi pomeraj sa vrednošću 1. Izmene u prečici koja rešava probleme nepoklapanja veličina se mogu videti na slici 4.



Slika 4. (a) Prečica u ResNet mreži (b) Prečica u IResNet mreži.

2. Kvantizacija

Prva metoda za optimizaciju modela je prebacivanje težina modela i ulaza u model sa korišćenja decimalnih brojeva na celobrojne. Ovo deluje jednostavno teorijski, ali u praksi to nije slučaj.

Kao što znamo, brojeve ne možemo u stvarnom obliku čuvati u memoriji, već ih moramo enkodovati nulama i jedinicama. Postoje dve osnovne reprezentacije brojeva, decimalna i celobrojna.

Celi brojevi se predstavljaju u numeričkom sistemu sa bazom 2. U zavisnosti od opsega koji želimo, u računarstvu se koriste sledeći oblici celih brojeva:

1. int8 ili short (-128 do 127)
2. uint8 (0-255)
3. int16 ili long (-32768 do 32767
4. uint16 (0 do 65535)

Ukoliko želimo da prikažemo realan broj, moramo se odreći putpune preciznosti. Na primer, broj ⅓ u decimalnom zapisu se može napisati kao 0.333.. sa beskonačno mnogo brojeva koje ne možemo prikazati u memoriji. Da bi ovo popravili, koristimo brojeve sa pokretnim zarezom (slika 5).

Decimalni zapis broja je u formi n bitne reči:

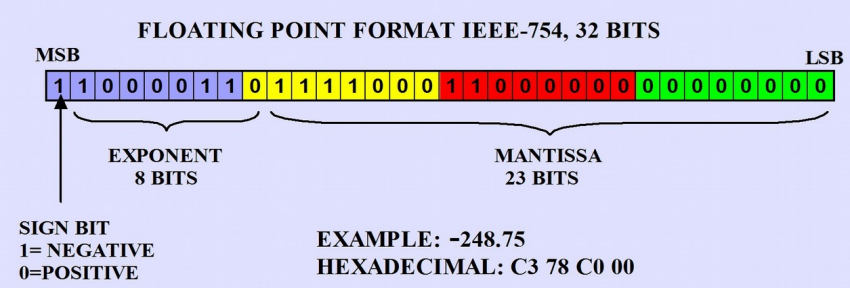
$$\text{ZNAČAJNA\_VREDNOST} \times \text{OSNOVA}^{\text{EKSPONENT}}$$

gde je osnova obično 2, ali može biti i 10. Za naš slučaj ćemo uzeti da je 2.

Slično celim brojevima, postoje različiti tipovi decimalnih zapisa:

5. half ili float16 (1 bit znak, 5 bit-a eksponent, 10 bit-a značajna vrednost)
6. single ili float32 (1 bit znak, 8 bit-a eksponent, 23 bit-a značajna vrednost)
7. double ili float64 (1 bit znak, 11 bit-a eksponent, 52 bit-a značajna vrednost)

Ukoliko bi probali da pomnožimo dva broja, videćemo da su operacije sa brojevima sa pokretnim zarezom mnogo više korišćene nego celobrojna aritmetika. U praksi, brzina svake operacije veoma zavisi od hardvera. Na primer, moderni procesori na desktop računarima mogu izvršavati aritmetičke operacije sa brojevima sa pokretnim zarezom skoro pa jednakom brzinom kao i sa celim brojevima. Sa druge strane, grafičke kartice su mnogo više optimizovane za korišćenje float32 tipa podataka.



Slika 5. Primer brojeva sa pokretnim zarezom.

Ideja sa kvantizacijom je sledeća. Uzmimo za primer sloj neuronske mreže sa izlazom u opsegu od *[-a, a)*, gde je *a* bilo koji realni broj.

Prvo, potrebno je skalirati izlaz tako da odgovara opsegu *[-128, 128)*.

$$x \mapsto \left\lfloor 128 \frac{x}{a} \right\rfloor \tag{1}$$

Uzmimo konkretan primer:

$$\begin{pmatrix} -0.18120981 & -0.29043840 \\ 0.49722983 & 0.22141714 \end{pmatrix} \begin{pmatrix} 0.77412377 \\ 0.49299395 \end{pmatrix} = \begin{pmatrix} -0.28346319 \\ 0.49407474 \end{pmatrix} \tag{2}$$

Opseg brojeva je u opsegu od *(-1, 1)*. Nakon kvantizacije, dobili bi sledeće:

$$\begin{pmatrix} -24 & -38 \\ 63 & 28 \end{pmatrix} \begin{pmatrix} 99 \\ 63 \end{pmatrix} = \begin{pmatrix} 4770 \\ 8001 \end{pmatrix} \tag{3}$$

Već u ovom slučaju vidimo da rezultat zapravo i nije int8 tipa. Do ovoga dolazi jer se množenjem dva 8-bitna int broja, dobija 16-bitni int broj. Ako uradimo de-kvantizaciju rezultata upotrebom sledeće transformacije, dobićemo rezultat koji nije identičan kao početne vrednosti.

$$x \mapsto \frac{ax}{16384} \tag{4}$$

$$\begin{pmatrix} -0.2911377 \\ 0.48834229 \end{pmatrix} \tag{5}$$

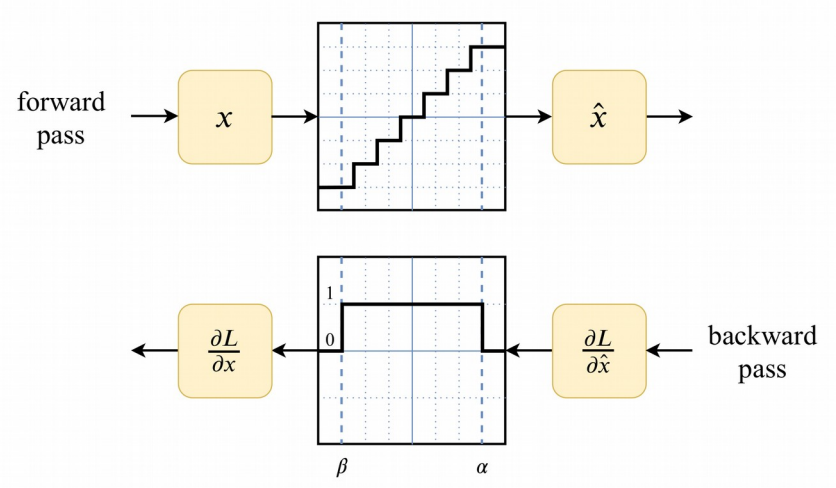
Ovo je očekivano, s obzirom na to da se prilikom procesa kvantizacije gubi preciznost.

U ovom primeru smo pokazali primer kvantizacije u int8 format, ali je moguće koristiti i ostale formate, kao i kombinacije. Na primer, procesi množenja matrica u mreži se mogu kvantizovati u int8, ali aktivacione funkcije u float16.

Postoje dva tipa kvantzacije kod neuronskih mreža:

1. Trening sa kvantizacijom
2. Kvantizacija nakon treninga

Kod treninga sa kvantizacijom, kvantizaciju težina vršimo tokom treninga. U ovom slučaju se čak i gradijent računa sa kvantizovanim težinama. Prednost ove metode je veća preciznost prilikom korišćenja int8 tipa nego u narednoj metodi. Kod ovakve vrste kvantizacije, aktivacione funkcije takođe bivaju kvantizovane (slika 6).



Slika 6. Kvantizacija aktivacione funkcije i propagacije unazad.

Kod kvantizacije posle treninga, model se najćešće trenira sa float32 ili float16 težinama i ulazom, nakon ćega se vrši kvantizacija. Prednost ove metode je jednostavnost, ali je mana veći gubitak preciznosti.

U praksi, performanse strogo zavise od hardvera. Mreža kvantizovana u int8 format će najbolje rezultate postići na specijalizovanom hardveru za takav tip kalkulacija. Pored hardvera, neophodna je i podrška razvojnog okvira koji će se koristiti za pokretanje modela.

Iako ova tehnika deluje obećavajuće, njeno korišćenje mora biti pažljivo. Neuronske mreže su izuzetno kompleksne funkcije, ali to što su kontinualne ih ne čini otporne na nagle promene.

Kvantizacija nam u nekim slučajevima može pomoći, ali ukoliko je cilj korisiti float16 preciznost kako bi se smanjila potrošnja resursa tokom treninga, treba biti oprezan. Grafičke karte rade sa takozvanim dense vektorima kako bi popunili vektor koji je 1024 bit-a. Ukoliko koristimo potpunu preciznost, odnosno 32 bit-a, kako bi popunili jedan vektor, potrebno je 32 primerka tokom treninga. Ovaj način treninga je poznat i kao mini-batch. Ukoliko koristimo 16-bitnu preciznost, a ne promenimo veličinu mini-batch-a, potrošićemo polovinu raspoloživih resursa. Zbog toga je potrebno i povećati veličinu mini-batch-a prilikom korišćenja float16 preciznosti.

Do ovoga dolazi zbog toga što grafičke kartice koriste takozvanu SIMD arhitekturu. Ovo je vrsta paralelne arhitekture, koja se zasniva na jednom toku instrukcija nad više tokova podataka, odnosno omogućava paralelizam podataka.

Mini-batch metoda može poboljšati performanse prilikom treninga, ali stvara još jedan dodatni problem. U radu [7] je dokazano da mini-batch metoda može smanjiti sposobnost mreže da generalizuje, što će automatski dovesti do dužeg treninga ukoliko želimo da postignemo podjednako dobre rezultate kao kod korišćenja potpune preciznosti. Metode poput augmentacije podataka mogu poboljsati generalizaciju, ali dovode do relativno oštrog lokalnog minimuma.

Jos jedno potencijalno rešenje za popravljanje rezulata generalizacije kod velikog batcha u mini-batch metodi je dinamicko povećanje batch-a tokom treninga [8].

Još jedan od potencijalnih problema prilikom kvantizacije je odsecanje vrednosti. U praksi, proces kvantziacije može dovesti do toga da vrednosti iz opsega [α,β] budu prebačene u opseg [αq,βq]. Programski jezici će automatski odseći vrednosti veće od ovog opsega. Kada bi se primenom dekvantizacije vrednosti iz opsega [αq,βq] vratile u opseg [α,β], doći ce do gubitka preciznosti.

3. Pruning

Neuronske mreže su u današnje vreme postale veoma popularne zbog toga što mogu rešiti veliki broj kompleksnih zadataka. Kako bi rešile ove zadatke, veliki broj novih modela ima

veliki broj parametra i samim tim je inferenca spora, ali je i veličina modela jako velika. Jedna od metoda koja nam može pomoći je pruning.

Pruning predstavlja proces odbacivanja viška parametra kod potpuno obučenih struktura. Pruningom se nakon procesa treninga mogu iterativno odbacivati težine, ali je ovaj proces moguće uraditi i tokom treninga. Pruningom je moguće i iterativno odbacivanje neurona.

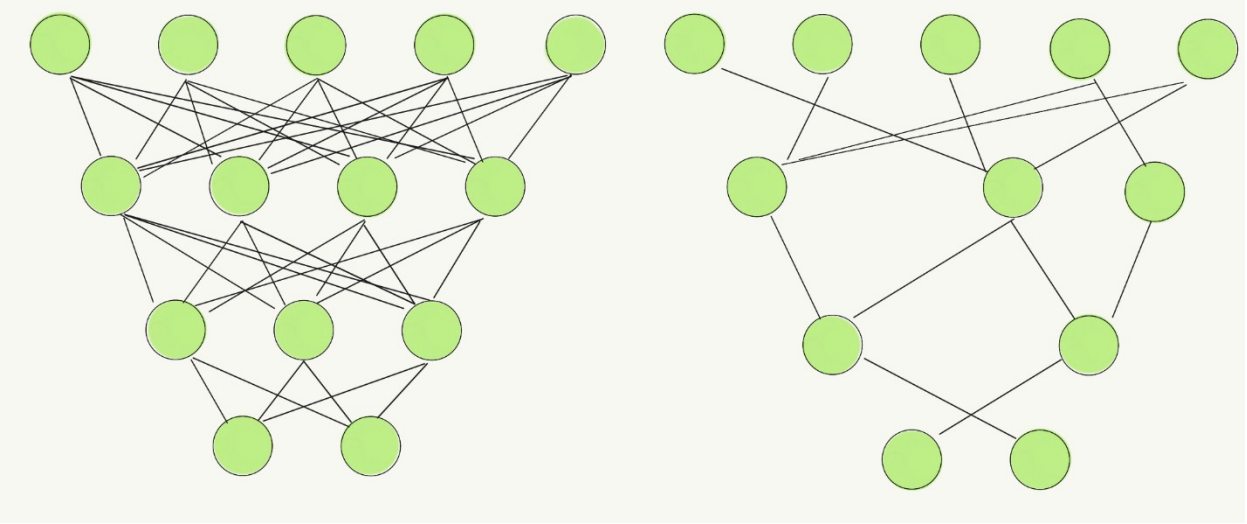
Odbacivanjem parametra se smanjuje i vreme potrebno za trening (broj parametra u prostoru za pretragu je manji), kao i za inferencu iz istog razloga, ali je i sam model manje memorijski zahtevan.

Kao i kod procesa kvantizacije, odbacivanje parametra može dovesti i do gubitka preciznosti. Fokus u ovom radu je post-trening pruning težina. Ovo neće skratiti vreme potrebno za trening, ali će skratiti vreme inference.

Kod odbacivanja težina modela, postavljamo individualne paremetre na nulu, što će imati jednak efekat odbacivanju.

Odbacivanje neurona je proces koji je bolje optimizovan. Treba obratiti pažnju da je odbacivanje kompletnih neurona rizično i može imati veliki impakt na preciznost modela.

Postavlja se pitanje koju metodu odabrati. Ukoliko želimo da odbacimo težine modela (ovde je reč o delimičnom odbacivanju, ne potpunom), želimo da odbacimo one koje su manje bitne modelu. Za ovo postoje različite heuristike koje nam mogu pomoći da odlučimo koje su težine manje bitne, tako da imamo najmanji impakt na preciznost njihovim odbacivanjem.



Slika 7. Primer pruninga težina i neurona.

Jedna od metoda je odbacivanje težina zasnovana na njihovoj vrednosti. Pošto je odbacivanje jednako nuliranju istih, u ovoj metodi možemo odbaciti sve koje su blizu nuli. Ovo se može postići korišćenjem L2 norme.

Ako proces pruninga želimo da radimo iterativno tokom obučavanja mreže, moguće je koristiti aktivacije neurona na trening skupu i odbacivati neurone čija je aktivacija blizu nuli. Slično ovoj ideji, mogu se odbacivati neuroni koji imaju identične izlaze, što u nekim slučajevima može značiti da rade istu stvar.

Za potrebe ovog rada, pruning ćemo izvršiti nad težinama Conv2D modula našeg modela u količini od 50% koristeći L1 normu, poznatiju kao Manhatten distancu.

Postoje dve vrste pruninga, struktuirano i nestruktuirano. Kod struktuiranog pruninga, lokacije težina u tensoru su struktuirane, odnosno tensor se i nakon pruninga može prikazati u obliku dense tensora, dok se za razliku od ovoga, kod nestruktuiranog pruninga tensori predstavljaju sparse tensorima.

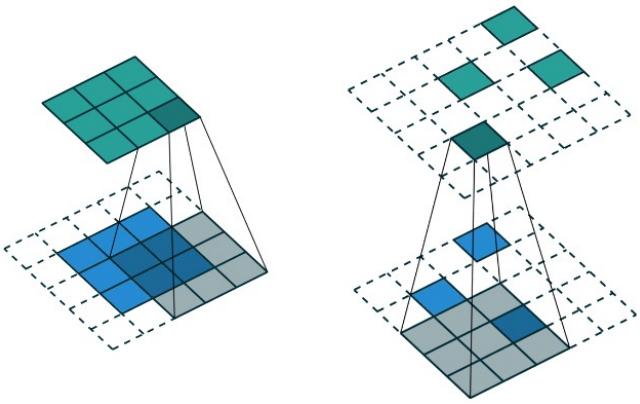
Dense tensori se mogu prikazati na sledeći način

$$W = \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{m,1} & W_{m,2} & \dots & W_{m,n} \end{bmatrix} \tag{6}$$

dok su sparse tensori reprezentovani kao

$$W = \begin{bmatrix} m \\ n \\ (i_1, j_1), W_{i_1,j_1} \\ (i_2, j_2), W_{i_2,j_2} \\ \vdots \\ (i_k, j_k), W_{i_k,j_k} \\ \vdots \end{bmatrix} \tag{7}$$

Pošto se pruning u našem slučaju izvršava nad težinama, na slici 8 je vizuelno demonstriran proces konvolucije pre korišćenja sparse tensora i nakon.



Slika 8. Dense konvolucija (desno). Sparse konvolucija (levo).

Kako bi se proces pruninga izvršio, potrebno je definisati masku (matrica), čijim će se množenjem dobiti novi vektori. Ova matrica se definiše na osovu određenih pravila, u našem slučaju je to L1 norma.

U [9] urađeno je detaljnije istraživanje kako kompresija modela utiče na model za prepoznavanje ekspresija lica. Autori rada su pokušali da odgovore na dva bitna pitanja koja se u vezi sa ovim temom. Prvo je koliko je efikasna kompresija modela u kontekstu prepoznavanja ekspresija. Kvantizacijom je postignuta kopresija od čak 4-4.5 puta, tok je pruning takodje smanjio veličinu modela na disku. Preciznost modela je bila ista kao i kod originalnog modela, sa čak 80 odbacivanja parametra, a iznosi 82%.

Drugo bitno pitanje je da li kopresija modela pojačava bajas. Pokazano je da nakon kompresije dolazi do povećanog ispoljavanja bajasa u smislu polova. Autori ovog rada nisu testirali preciznost modela nakon kompresije na setu podataka podeljenom po ljudskim rasama, što je čest problem u sistemima za prepoznavanje lica, i treba biti pažljiv.

4. Prenošenje znanja metodom učitelja i studenta

U poslednjih par godina, modeli dubokih neuronskih mreža postaju sve bolji i bolji u rešavanju određenih kompleksnih zadataka. Kako bi se postigli ovakvi rezulati, primenjuju se arhitekture koje su kompleksne i zahtevne, kako za implementaciju, tako i za trening i inferencu. Ovakvi modeli sa milionima i milijardama parametra se mogu pokretati na oblaku, ali će troškovi ovakvih servisa biti veliki. Sa druge strane, nije ih moguće ni pokrenuti na većini edge uređaja, ili ako je moguća, vrlo često nije isplativa.

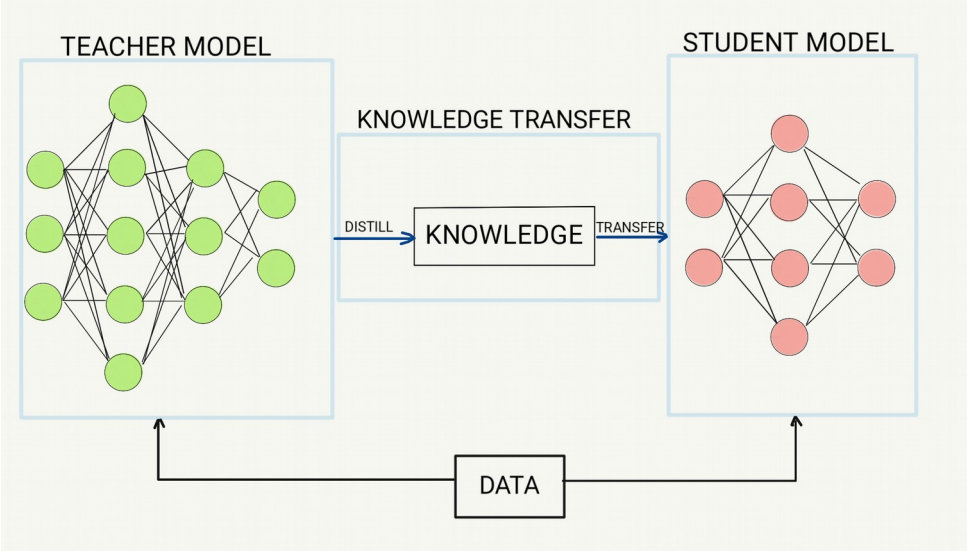
Proces prenošenja znanja metodom učitelja i studenta (slika 9) se može posmatrati kao proces kompresije modela u manju mrežu.

Kompleksniji modeli teoretski imaju veći prostor za pretragu parametra nego manji modeli. Ako pretpostavimo da se isti, ili slični prostor može pokriti manjom mrežom, tada bi prostor konvergencije mreže učitelja trebalo da se preklapa sa prostorom konvergencije mreže studenta. Ovo treba uzeti sa dozom rezerve, zbog toga što je ovo samo teorijska pretpostavka, i nije uvek tačna.

Koraci potrebni za prenošenje znanja (korišćeni u ovom radu) ovom metodom su:

1. Trening veće učitelj mreže
2. Evaluacija učitelj mreže
3. Trening manje student mreže na istom skupu podataka
4. Evaluacija student mreže
5. Ostvarivanje veze između učitelj i student mreže (kreiranje funkcije gubitka)
6. Trening student mreže uz pomoć učitelja na novom skupu podataka
7. Evaluacije nove student mreže
    a. Prolaz napred kroz učitelj mrežu
    b. Prolaz napred kroz student mrežu
    c. Povratna propagacija kroz student mrežu koristeći grešku učitelj mreže

Prenošenje znanja sa veće na manju mrežu može pomoći boljoj generalizaciji manje mreže. Do ovoga dolazi zbog mogućnosti većih mreža da bolje generalizuju. Takođe, veće mreže pokazuju bolje rezultate u radu sa slikama manjeg kvaliteta i manje rezolucije [2].



Slika 9. Proces obučavanja mreže metodom učitelj-student.

Kao i kod svih neuronskih mreža, i učitelj i student mreže imaju svoje funkcije gubitka (eng. Loss function). Konkretno se radi o ArcFace funkciji [1]. ArcFace funkcija je nastala iz prethodno najčešće korišćene funkcije, a to je Softmax funkcija. Softmax funkcija je postizala

dobre rezulate pre pojave ArcFace funkcije, ali je njen nedostatak predstavljao to što trening ovakvom funkcijom ne optimizuje izlazna obeležja dovoljno dobro kako bi se u velikim skupovima podataka lice moglo prepoznati. U nastavku sledi proces prepravljanja ArcFace funkcije, počeći od Softmax funkcije:

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{W_{yi}^T x_i + b_{y_i}}}{\sum_{i=1}^{n} e^{W_j^T x_i + b_j}} \tag{8}$$

gde $x_i \in R^d$ označava da obeležje i-tog primerka pripada y_i-toj klasi. U našem slučaju, dimenzije vektora obeležja d su 512 na osnovu [1, 33, 34, 35, 36]. $W_j \in R^d$ označava y-tu kolonu težine $W \in R^{d \times n}$ i $b_j \in R^n$ predstavlja bajes.

Ako se bajes fiksira $b_j = 0$ kao u [34], a težine po [37] trasnformišu kao $W_j^T x_i = \|W_j\| \|x_i\| \cos\theta_j$, gde θ_j predstavlja ugao između težina W_j i obeježja x_i.

Prateći [34 , 36, 33] individualne težine se normalizuju l2 normalizacijom kao $\|W_j\| = 1$. Takođe, po [38, 36, 33, 39] obeležja se normalizuju l2 normalizacijom i reskaliranjem uz pomoć parametra s. Normalizacije težina i obeležja, kao i reskaliranje obeležja dovode do toga da predikcije ArcFace modela zavise samo od ugla koji preklapaju obeležja i težine. Naučena obeležja će biti distribuirana po hipersferi sa radijusom s.

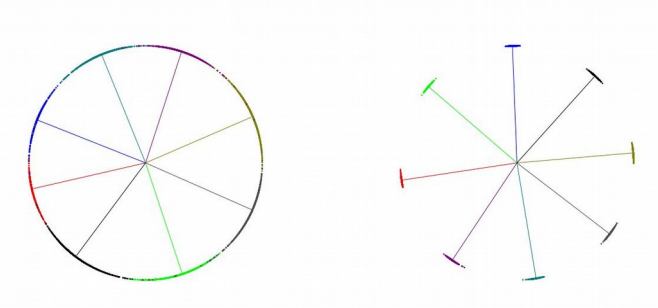
$$L_2 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{scos(\theta_{j\,i})}}{e^{scos(\theta_{j\,i})} + \sum_{j=1, i \neq y_i}^{n} e^{scos(\theta_j)}} \tag{9}$$

Na prethodnu formulu dodajemo marginu m koja će pojačati kompaktnost klasa, ali povećati raštrkanost klasa.

$$L_{Arc} = -\frac{1}{N}\sum log\frac{e^{s(cos(\theta_{y_i} + m))}}{e^{s(cos(\theta_{y_i} + m))} + e^{scos(\theta_y)}} \tag{10}$$

Kao izlaz iz modela zasnovanog na ArcFace funkciji, dobićemo 512-dimezionalni vektor. Ovaj vektor predstavlja obeležje lica koje želimo da prepoznamo. Sam proces prepoznavanja nakon ovoga se neće odvijati u modelu, već zasebno korišćenjem neke od metoda pretraga. Izbor najbolje metode u ovom koraku može dovesti do velikom usporenja, ali i greške ukoliko se odabere pogrešno.

Na slici 10 se može videti primer razdvanja klasa na MNIST datasetu [3] korišćenjem ArcFace funkcije.



(a) Softmax (b) ArcFace

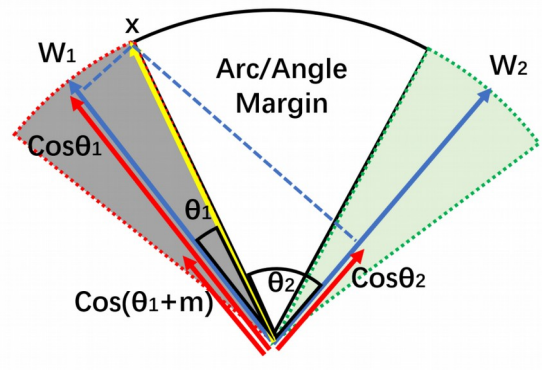
Slika 10. Primer separacije 8 klasa korišćenjem Sotfmax i ArcFace funckija.

Sama ArcFace funkcija je dizajnirana tako da u 512-dimenzionalnom prostoru raspoređuje klase tako da je vrlo jednostavno moguće naći slične klase. Pošto sama funkcija ima za cilj da minimizira distance između instanci iste klase, a maksimizira distancu između instanci različitih klasa, i pošto je bilo reči o tome da predikcije zavise samo od ugla, upravo je to ono kako se same instance, odnosno klase mogu razlikovati. Kosinusna distanca (formula 11), ili obrnuta metrika kosinusna sličnost (formula 12) su idealne metrike u ovom slučaju, jer nam mogu pomoći pronalazak željene klase izračunavanjem ugla između dve instance.

$$similarity = cos(\theta) = \frac{AB}{\|A\|\,\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{11}$$

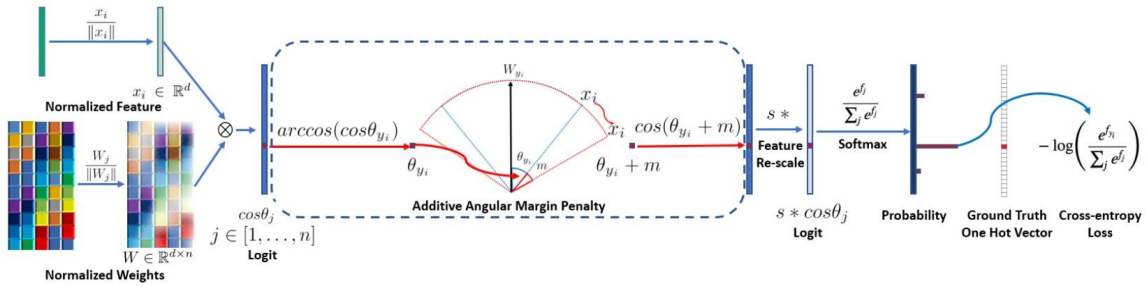
$$distance = 1 - similarity = 1 - cos(\theta) = \frac{AB}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{12}$$

Na slici 11 je vizuelno prikazano kako izgleda Arc margina između dve klase.



Slika 11. Geometriska interpretacija Arcface funkcije. Siva i zelena boja predstavljaju prostore dveju klasa. Arcface ne samo da kompresuje obeležja klasa, već i odgovara geometrijskoj distanci na hipersfericnoj površini.

Proces treninga mreže korišćenjem Arcface funkcije se može videti na slici 12.



Slika 12. Proces treninga mreže korišćenjem ArcFace funkcije.

Osim funkcije gubitka za dve osnovne mreže, neophodno je definisati i funkciju gubitka između dve mreže.

Ovu funkciju smo obeležili sa L_{total} i ona predstavlja kombinaciju klasičnog ArcFace loss-a mreže (L_{Arc}) koju treniramo, odnosno mreže studenta, i L_{KD} funkcije.

L_{KD} predstavlja kombinovanu funkciju gubitka koja se sastoji iz dva dela. Prvi deo predstavlja CrossEntropyLoss, dok je drugi deo MeanSquaredErrorLoss. Pošto se svi vektori obeležja normalizuju, L_{KD} će biti jako mali, tako da se po predlogu iz [4] množi parametrom $\lambda = 100$ tokom prvog koraka treninga.

$$L_{KD} = \frac{1}{N}\sum_{i \in N}(1 - \frac{1}{n}\sum_{D}^{j=1}(\phi_i^S(x)_j - \phi_i^T(x)_j)^2)$$

(13)

Ukupna funkcija gubitka je jednaka

$$L_{total} = L_{Arc} + \lambda L_{KD}$$

(14)

Ovo je u kodu urađeno na sledeći način (kod 1):

```
loss_v1 = criterion(thetas, label)
loss_v2 = configKD.cfg.w*criterion2(features_student, features_teacher)
loss_v = loss_v1 + loss_v2
```

Kod 1. Definisanje ukupne funkcije gubitka.

gde je *header* definisan kao (kod 2):

```
header = losses.ArcFace(in_features=configKD.cfg.embedding_size,
out_features=configKD.cfg.num_classes,                    s=configKD.cfg.s,
m=configKD.cfg.m).to(local_rank)
```

Kod 2. Definisanje header ArcFace funkcije.

U kodu 2 *criterion* predstavlja *CrossEntropyLoss* funkciju, dok *criterion2* predstavlja *MSELoss* funkciju.

Početni learning rate iznosi 0.1, ali je tokom treninga varijabilan, i definisan je na sledeći način (kod 3):

```
def lr_step_func(epoch):
    return ((epoch + 1) / (4 + 1)) ** 2 if epoch < -1 else 0.1 ** len([m for m in [8, 14, 20, 25]
if m -
    1 <= epoch])
```

Kod 3. Funkcija za menjanje koraka učenja.

Za optimizaciju ciljne funkcije kod obe mreže se koristi stohastički gradijentalni spust (kod 4).

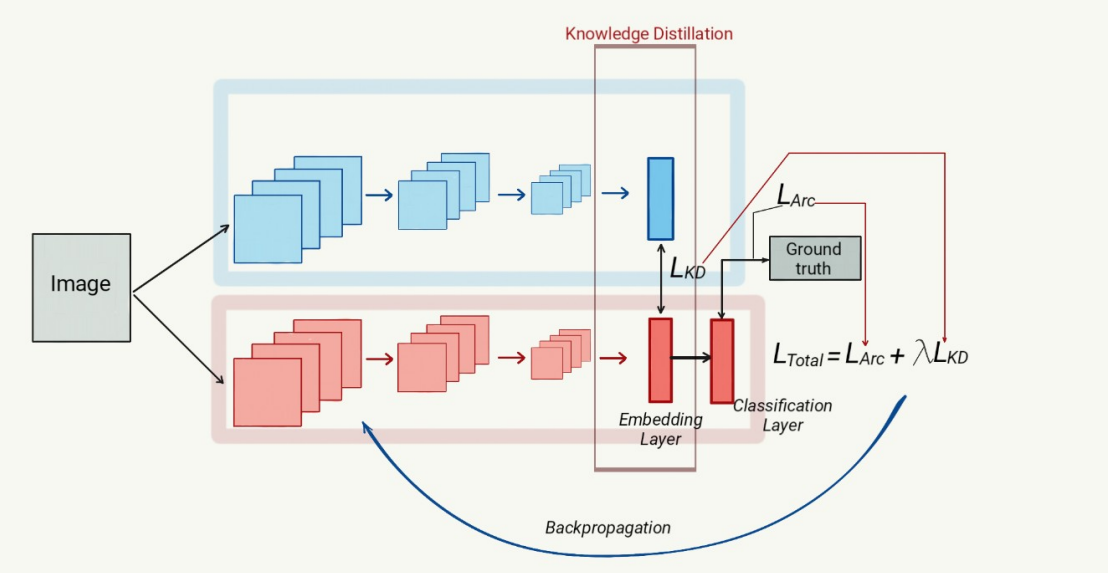
```
opt_backbone_student = torch.optim.SGD(
    params=[{'params': backbone_student.parameters()}],
    lr=configKD.cfg.lr / 512 * configKD.cfg.batch_size * world_size,
    momentum=0.9, weight_decay=configKD.cfg.weight_decay
)
opt_header = torch.optim.SGD(
    params=[{'params': header.parameters()}],
    lr=configKD.cfg.lr / 512 * configKD.cfg.batch_size * world_size,
    momentum=0.9, weight_decay=configKD.cfg.weight_decay
)
```

Kod 4. Stohastički gradijentni spust se koristi za obučavanje modela.

Neke implementacije ArcFace mreža koriste Adam algoritam. Ovo predstavlja mesto za eksperimente. Činjenica je da Adam algoritam brže konvergira, ali je u [25] cilj bio dokazati da sporiji proces treninga korišćenjem SGD algoritma dovodi do boljeg procesa generalizacije.

Generalizacija mreža za prepoznavanje lica je voema bitna. Ovakve mreže u toku treninga mogu videti hiljade, pa čak i milione lica, sa po 5-20 lica za svaku osobu, ali to nije dovoljno. U realnim uslovima, osobe koje se se pojavila u setovima podataka se neće toliko često pojavljivati u realnim uslovima korišćenja mreže. Postoje slučajevi kada se ova činjenica može iskoristiti, a to su slučajevi pravljenja mreže za specifične uslove, poput mesta gde znamo da će se pojavljivati samo određeni tip ljudi, i želimo ih sa što većom preciznošću prepoznati, dok nam generalizacija ne igra toliko ulogu. Ipak, modeli koji će se pokretati na mobilnim telefonima se u većini slučajeva neće koristiti u kakvim uslovima, te je u ovom radu kao algoritam optimizacije korišćen stohastički gradijentalni spust.

Proces treninga student mreže uz pomoć učitelja je prikazan na slici 13.



Slika 13. Proces treninga korišćenjem KD metode.

Metode prenosa znanja mogu pomoći u procesu prepoznavanja lica na više načina. Jedna od mogućnosti je treniranje kvalitetnije mreže za prepoznavanje osoba sa maskama [17]. U ovom slučaju je moguće i za učitelja i za studenta koristiti istu mrežu. Kako bi se obučila mreža koja će bolje generalizovati na slikama osoba sa maskama, potrebno je kroz učitelj mrežu provući slike osoba bez maski, dok kroz student mrežu slike osoba sa maskom. Ovime se postiže to da mreža student daje slične rezultate na slikama osoba sa maskom, kao i učitelj mreža na slikama bez maske. Ova metoda može biti korisna i u slučajevima kada je neophodno da se sistem ponaša dobro i sa osobama koje nose naočare za sunce. U [19] pokazano je kako mreža za prepoznavanje lica sa ResNet arhitekturom ima gubitak preciznosti kada se radi sa osobama koje nose naočare.

Osim modela za enkodovanje lica, ključni korak u procesu prepoznavanja je detekcija. Proces detekcije se sastoji iz dve faze. Prva faza je detekcija koordinata na kojima se lice nalazi, a druga faza je detekcija ključnih tačaka na licu, na osnovu kojih će se uraditi poravnanje lica. Ukoliko je neophodno ovakve modele pokretati na uređajima sa ograničenim resursima, prenos znanja je metoda koja se može koristiti. U ovom slučaju je moguće sa boljeg i sporijeg modela preneti znanje na brži i manje precizniji model. Takođe, ukoliko je neophodno detektovati osobe sa maskom, ili osobe sa određenim odevnim predmetima na glavi, ovo je polje gde će veći modeli bolje generalizovati, te je prenos znanja jedno od rešenja [18]. U slučaju detekcije ključnih tačaka na licu (regresija tačaka), modeli sa više slojeva i parametra obično daju kvalitetnije predikcije.

Sav programski kod u ovom radu je napisan u PyTorch razvojnom okviru, i skalabilan je. Proces treninga je moguće pokrenuti na procesoru, grafičkoj kartici na jednom računaru, više grafičkih kartica na jednom računaru, ili na više računara. Za distribuirani trening je korišćen

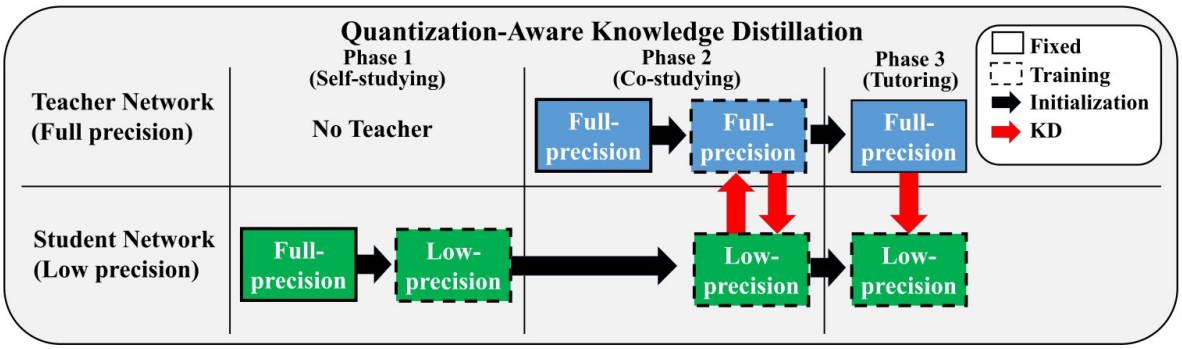
PyTorch DistributedDataParallel modul sa NCCL backendom (kod 5). Kako bi se postiglo ubrzanje procesa treninga korišćeni su CUDA i CUDNN paketi razvijeni od strane NVIDIA-e.

```
dist.init_process_group(rank=args.local_rank, backend='nccl', init_method='env://',
world_size=1)
local_rank = args.local_rank
torch.cuda.set_device(local_rank)
rank = dist.get_rank()
world_size = dist.get_world_size()
...

for ps in backbone_student.parameters():
   dist.broadcast(ps, 0)
...

dist.destroy_process_group()
```

Kod 5. Inicijalizacije distribuiranog modula za trening modela na više računara i više grafičkih kartica.

Za pomoćne funkcije i preprocesiranje seta podataka korišćen je MXNet razvojni okvir.

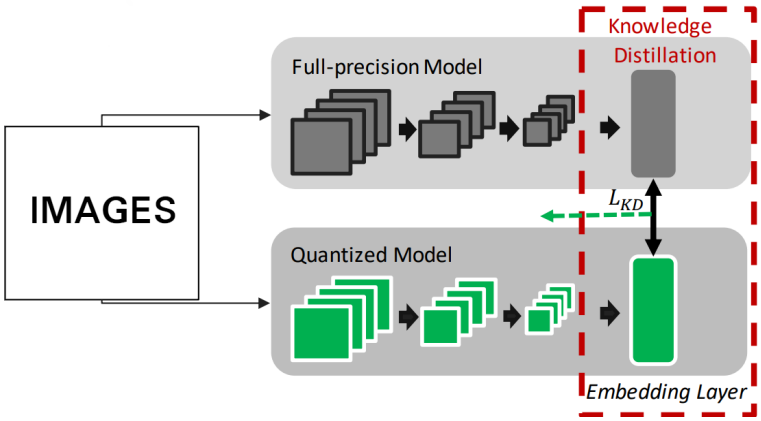
Kod metoda prenosa znanja od koristi može biti i korišćenje više učitelj mreža, s tim što treba uzeti u obzir da je potrebno prilagoditi loss funkciju, kao i prenos znanja sa kvantizacijom (eng. QKD) [26], ili bez seta podataka [28].



Slika 14. Proces obučavanja korišćenjem QKD metodom. Prva faza je obučavanje bez učitelja. Druga faza predstavlja dodatno obučavanje sa učiteljem, sa razlikom u tome što se i učitelj mreža dodatno obučava i prilagođava student mreži, dok se u krajnjoj trećoj fazi obučava samo student mreža.

Obučavanja modela sa kvantizacijom se može izvršiti na više načina. Jedan od načina je prikazan na slici 14. U ovoj metodi proces je sačinjem od 3 faze. Počinje se od toga da imamo učitelj mrežu koju možemo i dodatno obučavati. Tokom prve faze, već obučenu student mrežu bez kvantizacije kvantizujemo, nakon čega se obučavanje nastavlja sa kvantizacijom. U drugoj fazi se nastavlja obučavanje sa prenosom znanja za koje se koristi učitelj mreža koja se takođe obučava, ali bez kvantizacije. Na kraju se tokom treće faze obučava samo student mreža metodom prenosa znanja, takođe sa kvantizacijom.

Proces prikazan na slici 14 nije neophodan, već se dobri rezultati mogu postići i samo treningom sa kvantizacijom [27]. Kod kvantiazcije je neophodno obratiti pažnju na sve mane koje proizilaze iz korišćenja ove metode. Ovaj proces je prikazan na slici 15.



Slika 15. Proces obučavanja korišćenjem kvantizacije tokom procesa obučavanja.

5. Pokretanje mreža na mobilnim uređajima

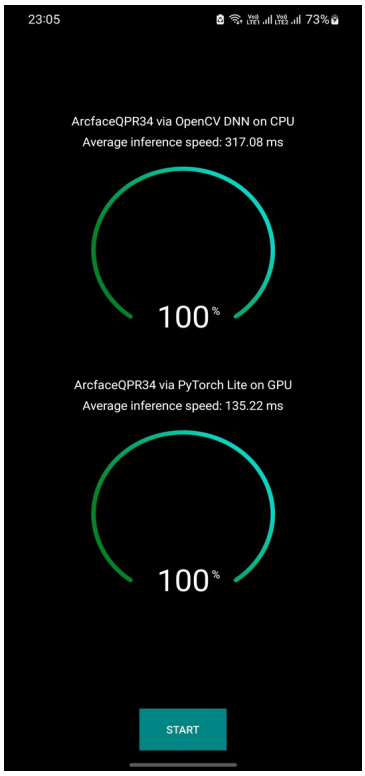
Za potrebe ovog rada kreirana je aplikacija za Android platformu (slika 16), preko koje je izmerena brzina svake mreže. U aplikaciji se svaka mreža pokreće preko procesora (CPU) i preko mobilnog grafičkog čipa (GPU). Sve mreže su testirane na Xiaomi Poco X3 telefonu sa Qualcomm SM7150-AC Snapdragon 732G ARM procesorom (2x2.3 GHz Kryo 470 Gold & 6x1.8 GHz Kryo 470 Silver), i Adreno 618 mobilnim grafičkim čipom, kao i 6GB ram memorije.

Za razvoj je korišćen Kotlin programski jezik i Gradle build sistem.

OpenCV [5] je biblioteka razvijena u Intelovom istraživačkom centru. Prvi put se pojavila 2000 godine. Prva mobilna verzija biblioteke se pojavila 2010 godine. Korišćenjem OpenCV biblioteke modeli su testirani na ARM procesoru korišćenjem ugrađenog DNN modula. Alternativa korišćenju mreža direktno na procesoru je korišćenje Vulkan arhitekture.

Drugi način pokretanja mreža u ovom radu je korščenjem PyTorch razvojnog okvira. Prednost ove metode je što je moguće koristiti mobilni grafički čip preko Vulkan arhitekture.

PyTorch razvojni okvir podržava čuvanje modela u formatu koji se može učitati na mobilnim uređajima, ali je ovakve modele prethodno potrebno optimizovati.



Slika 16. Izgled mobilne aplikacije. Moguće je ubaciti dva modela, i u samoj aplikaciji pronaći koji od dva modela je brži na mobilnom uređaju.

Za to je potrebno pozvati odeđenu PyTorch funkciju koja će izvršiti sledeće optimizacije nad modelom:

1. Spajanje 2D konvolucije i batch normalizacije. Težine i bajes Conv2d slojeva se shodno tome ažuriraju.
2. Prepisivanje grafa tako da se zamene Conv2d i Linear funkcije njihovim predefinisanim parnjacima koje se mogu koristiti uz pomoć Guglove XNNPACK biblioteke na ARM procesorima.
3. Ograničavanje ReLU / Hardtanh funkcija.
4. Brisanje neurona koji bi bili odbačeni Dropout slojem, kako se to ne bi radilo tokom pokretanja.
5. Brisanje parametra 2D konvolucije i pomeranje istih u koren modula. Ovo će smanjiti veličinu modela, bez gubljenja preciznosti.

Mobilna aplikacija kreirana u ovom radu predstavlja okruženje za eksperimente, te je lako moguće nadograditi je i promeniti modele dubokih neuronskih mreža koji se pokreću. Ovo je čini korisnom za širu upotrebu i pruža mesto za napredak.

Kako bi rezultati bili konzistentni, potrebno je primeniti isti preprocesing u oba razvojna okvira.

Ukoliko je reč o OpenCV-u, preprocesiranje možemo primeniti kreiranjem OpenCV blob objekta iz slike (kod 6).

```
val TARGET_IMG_WIDTH = 112.0
val TARGET_IMG_HEIGHT = 112.0
MEAN = Scalar(0.485, 0.456, 0.406)
val blob = Dnn.blobFromImage(image, 1.0, Size(TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT), MEAN, true, false)
```

Kod 6. Preprocesiranje slika u OpenCV razvojnom okviru.

U PyTorch-u je takođe moguće uraditi preprocesiranje koristeći već ugrađene funkcije (kod 7).

```
inputTensor = TensorImageUtils.bitmapToFloat32Tensor( bitmap,
   TensorImageUtils.TORCHVISION_NORM_MEAN_RGB,
   TensorImageUtils.TORCHVISION_NORM_STD_RGB
)
```

Kod 7. Preprocesiranje slika u PyTorch razvojnom okviru.

Sama infrenca, odnosno pokretanje modela je jako jednostavno na ovaj način (kod 8). Počnimo od OpenCV-ja.

```
dnnNet.setInput(blob)
val result = mutableListOf<Mat>()
dnnNet.forward(result, layerNames)
val transposed = Mat()
transpose(result[0], transposed)
```

Kod 8. Inferenca modela u OpenCV razvojnom okviru.

Dok je kod PyTorch-a ovo još jednostavnije (kod 9).

```
val outputTensor = module.forward(IValue.from(inputTensor)).toTensor()
```

Kod 9. Inferenca modela u PyTorch razvojnom okviru.

6. Rezultati primenjenih metoda

Primena metoda optimizacije može biti proces koji developeri redovno primenjuju, te se sam proces u nekim slučajevima može završiti za kratak vremenski period. Sa druge strane je preciznost, koja se može pogoršati u nekim slučajevima. U nastavku sledi analiza rezultata primenjenih metoda.

6.1 Rezultati kvantizacije

U tabeli 1 prikazane su veličine modela pre i posle kvantizacije težina iz float32 u int8 format (slika 17).

	ArcFace R18	ArcFace R34
Baseline	91.8 MB	130.4 MB
Kvantizacija	23.1 MB	32.9 MB

Tabela 1. Prikaz veličine modela u MB pre i posle kvantizacije.

PyTorch razvojni okvir i dalje ne podržava eksportovanje kvantizovanih modela za rad na mobilnim uređajima, tako da se zbog ovoga nismo zadržali na merenju performansi i preciznosti istih. Postoje druge metode za kvantizaciju, poput kvantizacije samog ONNX modela, ali to nije tema ovog rada. Ono što je bitno za slučajeve kada je potrebna manja veličina softvera na uređajima, kvantizacija može smanjiti veličinu modela i do par puta.

Slika 17. Primer kvantizacije težina iz float32 tipa podataka u int8 tip podataka

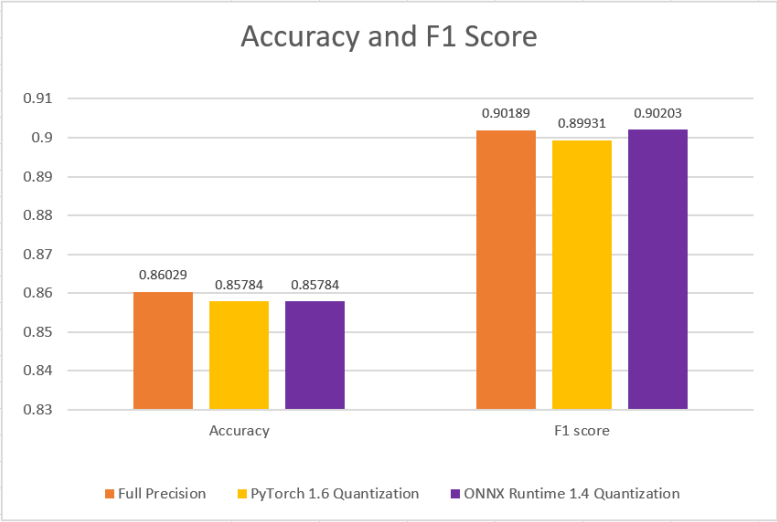
S obzirom da je ONNX standard široko korišćen, i većina modela se na mobilnim uređajima pokreće preko ONNXRuntime razvojnog okvira, ukoliko kvantizacija tokom treninga nije moguća, ovo predstavlja jednostavnije rešenje nego korišćenje PyTorch-a koji je i dalje u beta verziji. Prednost ONNXRuntime biblioteke je i bolja podrška nego PyTorch razvojni okvir koji za Android i dalje nema stabilnu verziju koja poseduje sve operacije kao desktop varijanta.

ONNXRuntime podržava dve vrste kvantizacije:

1. Dinamičku kvantizaciju

2. Statičku kvantizaciju

Prednost ONNXRuntime paketa je što generalno pruža bolje rezultate kvantizacije, uključujući i manju veličinu modela, a jednaku, a nekad i bolju preciznost (slika 18).



Slika 18. Poređenje PyTorch i ONNXRuntime kvantizacije na primeru BERT modela [32].

6.2. Rezultati pruninga

Veličina modela nakon pruninga se neće promeniti. Do ovoga dolazi i zbog toga što i ako težine modela ili neurone odbacimo, ovo je kao što smo već spomenuli ekvavilentno zamenom nulama. Sa strane memorije je ovo isto kao i čuvanje neke vrednosti, te se veličina modela neće promeniti. Ukoliko želimo da zaista smanjimo veličinu modela, potrebno je sačuvati takozvani sparse model.

Ukoliko pričamo o preciznosti, pruningom ne možemo poboljšati naš model, već samo možemo ubrzati inferencu (tabela 2).

Ukoliko uzmemo u obzir sve prethodno navedeno, kao najbolje rešenje se nameće korišćenje manje mreže sa punom preciznošću. Pošto manje mreže same po sebi teže dostizu preciznost kao kompleksniji modeli, ukoliko primenimo prenos znanja metodom učitelj-student, poboljšaćemo preciznost manjeg modela.

	ArcFace R18	ArcFace R34
Baseline OpenCV	195ms	349ms
Baseline Pytorch	142ms	249ms
Pruning OpenCV	193ms	340ms

Pruning Pytorch	137ms	215ms

Tabela 2. Rezultati pruninga izraženi u milisekundama.

Kao što se može videti iz tabele 2, pruningom dobijamo brže izvršavanje inference modela. Efekat pruninga je izraženiji kod većih mreža. Ovo se može videti i ukoliko se uporede MobileNet [13] i Resnet arhitekture. MobileNet arhitektura je brža, te metode poput kvantizacije i pruninga mogu povećati brzinu modela, ali je efekat znatno manji nego kod Resnet mreže.

Osim toga, mreže poput MobileNet-a su veoma male, te se kvantizacijom može izgubiti mnogo više korisnih konekcija u mreži.

Treba uzeti u obzir da tema ovog rada nije detaljno istraživanje naprednih metoda za kvantizaciju i pruning, koje bi znatno više smanjile vreme potrebno za izvršavanje modela. Neke od naprednijih metoda pruninga su:

1. Pruning sa objašnjenjem [10]

2. Pruning korišćenjem GAN mreža [11] – tehnologije koja stoji iza popularnih DeepFake projekata.

3. Pruning filtera [12]

Pored navedenih metoda, izvršavanje kvantizacije i pruninga tokom treninga će dovesti do boljih rezultata (preciznosti) nego izvršavanje ovih funkcija nakon treninga. Do ovoga će doći iz razloga što se gubitak preciznosti kompresijom tokom treninga može kompenzovati dodatnim treningom.

6.3. Rezultati prenosa znanja

Ono što dobijamo prenosom znanja je manje kompleksna mreža sa boljom preciznošću. Pored toga, dobijamo mrežu koja je manje zahtevna za pokretanje, što je ono gde ova metoda pokazuje pun potencijal.

Vreme inference za IResnet R18 mrežu sa prenosom znanja je identično kao i vreme inference originalne IResnet R18 mreže.

Nakon dodatnog treninga ArcFace mreže, sa IResnet R18 arhitekturom kao student mreže, i ArcFace IResnet R34 mreže kao učitelj mreže, poboljšana je preciznost student mreže, i ona sada iznosi 0.89%.

Originalna preciznost student mreže je bila 0.86%, dok je preciznost učitelj mreže 0.94%.

Premda ovo ne deluje kao značajno poboljšanje, ovi rezultati su ostvareni treningom nad samo 40000 slika, što je manje od jedne epohe treninga.

Za trening je korišćen faces_emore set podataka, dok je za validaciju korišćen LFW set podataka. Set podataka faces_emore je pravljen specifično za trening originalne ArcFace mreže sa ResNet arhitekturom.

LFW set podataka [22] predstavlja bazu podataka sačinjenu od 13233 slika i 5749 osoba. Ovaj set podataka je napravljen sa ciljem testiranja algoritama za prepoznavanje lica bez uslova pod kojima su podaci skupljani.

U ovom setu podataka osobe su podeljene u parove, konkretno pozitivne i negativne. Pozitivni parovi se sastoje od dve slike iste osobe, dok se negativni parovi sastoje od dve slike različitih osoba. Ovo je ilustrovano na slici 19.

Slika 19. (Levo) Pozitivni parovi u LFW setu podataka. (Desno) Negativni parovi u LFW setu podataka [23].

Slike lica u LFW datasetu nisu poravnate. Autori originalnog dataseta su priložili i dodatni dataset lica poravnatih deepfunneled metodom [24]. Ipak, slike poravnate ovom metodom nisu odgovarajuće za naš metod enkodovanja lica zbog toga što lice ne zauzima dovoljan procenat slike. Za potrebe ovog rada je odsečen centralni deo sa svake slike, a zatim skalirao kako bi se zadovoljile dimenzije mreže (112x112), ali i uslov da lice zauzima većinu prostora na slici. Na kraju procesa poravnanja, želimo dobiti set slika na kojima su oči uvek na istoj pravoj (slika 20).

Slika 20. (Levo) Lice poravnato Deepfuneled metodom. (Desno) Lice kropovano naivnom metodom.

Skaliranje slika ovom metodom nije idealno, ali dovodi do interesantne pojave. A to je da je preciznost modela smanjena zbog skaliranja slika na veću rezoluciju, kako manjeg, tako i većeg modela. Ovo je idealna prilika da se isprobaju metode prenosa znanja.

Rezultati metode prenosa znanja, kao i test brzine se mogu videti na slici 22. Preciznost svakog modela ja pronađena na osnovu najboljeg tresholda, odnosno praga ispod kog se osoba smatra prepoznatom. Vrednosti koje su upoređuju ovim pragom predstavljaju kosinusnu distancu između dva vektora obeležja lica. Preciznost modela se računa po formuli 15:

$$Accuracy = 1 - EER \tag{15}$$

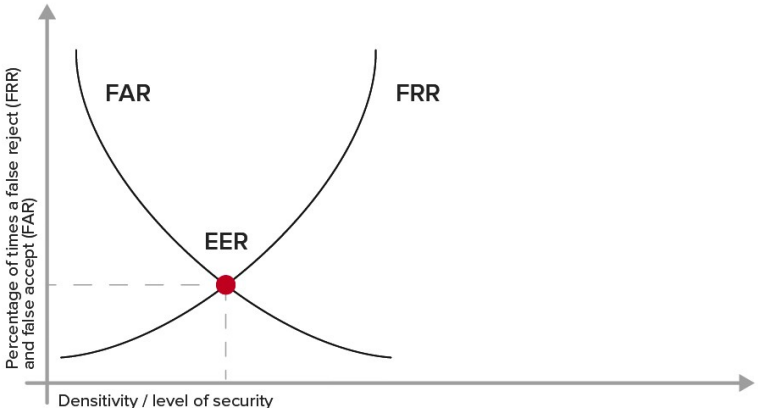
gde EER predstavlja equal error rate. EER predstavlja tačku u kojoj se stopa lažnog odbacivanja i lažnog prihvatanja poklapaju (slika 21).

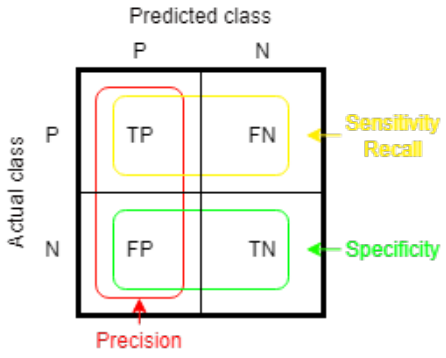
$$EER = \frac{FAR+FRR}{2} \tag{16}$$

$$FAR = \frac{FN}{TP+FN} \tag{17}$$

$$FRR = \frac{FP}{FP+FN} \tag{18}$$

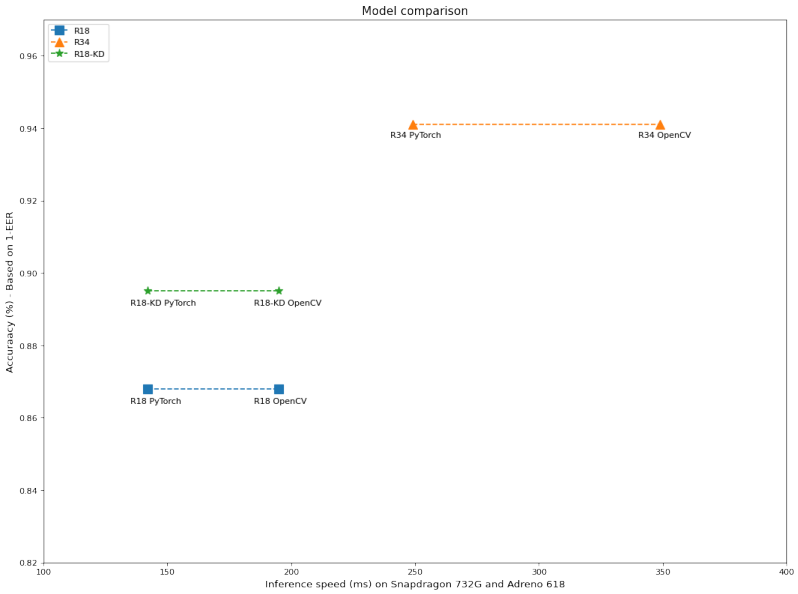
Vrednosti TP, TN, FP, FN se mogu dobiti iz matrice konfuzije (slika 21).





Slika 21. (Levo) Equal Error Rate. (Desno) Matrica konfuzije.

Brzina modela je merena u milisekundama na Xiaomi Poco X3 mobilnom telefonu, koji poseduje Qualcomm Snapdragon 732G mobilni procesor (Octa core 2x2.3 GHz Kryo 470 Gold & 6x1.8 GHz Kryo 470 Silver), i Adreno 618 mobilni grafički čip.



Slika 22: Prikaz rezultata metode prenosa znanja tehnikom učitelj student.

Bitno je spomenuti prednosti i mane metode prenosa znanja, kao i kada je moguće iskoristiti je. Jedna od glavnih prednosti ove metode je upravo razlog zbog kojeg je odabrana kao glavna tema ovog rada, a to je trening manjih modela uz pomoć veće i preciznije mreže. Ovo se može proširiti na slučajeve kreiranja mreža za prepoznavanje zaklonjenih lica, kao i za trening manjih mreža za detekciju uz pomoć većih i preciznijih modela.

Ipak, bitno je spomenuti i njene mane. Sama metoda zahteva dva modela (za slučaj treninga manje mreže uz pomoć veće), što će automatski proizvesti zahtev za znatno više resursa tokom treninga, ali i sporiji trening. Ovo dodatno zahteva i drugi model, koji je potrebno kreirati i trenirati kako bi se postigli željeni rezultati. Sve navedeno nije idealno ukoliko su vremenski okviri kratki i zahteva znatno više programiranja i modelovanja. Kao i dodatne podatke koji će se koristiti za prenos znanja.

7. Specifične opcije određenih platforma

Metode poput kvantizacije, pruninga i prenosa znanja su metode koju se mogu koristiti u procesu razvoja, ili nakon procesa razvoja. Ukoliko upotrebe ovih metoda nakon treninga ne dovedu do željenih rezultata, određeno ubrzanje se može postići korišćenjem specifinčih opcija platforme na kojoj se mreže pokreću.

U našem slučaju, na Android mobilnih telefonima je moguće koristiti NNAPI. NNAPI predstavlja C API koji služi za pokretanje računarski zahtevne operacije. Ovaj API je moguće koristiti na najnižem nivou preko C programskog jezika, ali i uz pomoć razvojnih okvira kao što su TensorFlow Lite, Caffe2, ONNXRuntime i PyTorch.

Ovo je alternativa korišćenju Vulkan arhitekture, i na ovaj način je moguće pokretati neuronske mreže kako na procesoru, grafičkom čipu, tako i na specijalizovan čipovima na određenim modelima mobilnih telefona.

Druga opcija na Android uređajima jesu specifične biblioteke za određene vrste procesora. Na uređajima koje pokreće Qualcomm Snapdragon platforma, moguće je koristiti Qualcomm Neural Processing Engine na višem nivou, ili Qualcomm Hexagon NN na nižen nivou. Prednost ove opcije je što je moguće koristiti ONNXRuntime o kome je već bilo reči.

Ukoliko je na raspolaganju Huawei platforma, moguće je koristiti HiAI platformu, za koju je moguće konvertovati modele iz TensorFlow, Caffe, ONNX razvojnih okvira. Takođe, ukoliko se radi o Samsung telefonima, moguće je koristiti Samsung Exysos NPU i Samsung Neural SDK.

Telefoni sa MediaTek procesorima mogu koristiti mogućnosti poput MediaTek APU procesora (eng. AI Processing Units).

XNNPACK koji je gore spomenut pruža i opciju pokretanja takozvanih sparsifikovanih mreža (proređenih). Ovo je još jedan od pokušaja dobijanja bržih inferenci, ali je i tokom ovog postupka moguće izgubiti na preciznosti modela. Metode poput ove daju najbolje rezultate na modela zasnovanim na invertovanim rezidualnim blokovima [31], poput blokova u MobileNetV2 arhitekturi. Sam proces sparsifikacije će dovesti do manjeg modela i brže inference, te se gubitak preciznosti može nadoknaditi korišćenjem većeg i kvalitetnijeg modela. Ova operacija je vrlo jednostavna za implementaciju uz pomoć TensorFlow Lite razvojnog okvira u kombinaciji sa XNNPACK bibliotekom, ali je neophodno model od samog početka razviti tako da podržava ovu operaciju. Sa druge strane, sparsifikacije se može videti kao alternativa prenosu znanja, gde se umesto treninga uz pomoć većeg modela, direktno koristi taj model, uz određena ubrzanja i mali gubitak preciznosti.

Google, kao firma koja stoji iza Android platforme pruža veoma dobru podršku, te je još jedan od razvojnih okvira koji pruža mogućnost pokretanja modela na mobilnim uređajima MLKit, koji osim što pruža mogućnost pokretanja modela kreiranih korišćenjem TensorFlow-a, podržava i veoma laku inferencu hostovanih modela, te se sam model ne mora fizički nalaziti na uređaju. Pored toga što je razvijen od strane Google-a, MLKit je moguće instalirati i koristiti i na Apple mobilnim uređajima. Još jedna prednost ovakvog razvojnog okvira je što pruža samim

programerima mogućnost izbora programskog jezika, za razliku od većine drugih specijalizovanih rešenja koja je moguće iskoristiti samo uz pomoć C++ jezika, MLKit je vrlo lako moguće koristiti i uz pomoć Java i Kotlin jezika.

Osim Android telefona, Apple uređaji takođe pružaju mogućnost pokretanja modela na specijalizovan hardveru, koji se u ovom slučaju naziva Neural Engine, i potrebno je koristiti Apple Core ML razvojni okvir.

U gore navedenom tekstu se spominje veliki brok razvojnih okvira, kako novih, tako i nekih dobro poznatih i široko korišćenih. Ipak, treba imati u vidu da ukoliko se radi o novim razvojnim okvirima, podrška nije uvek dobra, i za većinu problema ne postoji odgovor. Takodje, ukoliko je za mrežu predviđeno korišćenje i unapređenje u budućnosti, bolje je odabrati razvojni okvir koji ima veću podršku i broj ljudi koji ga koriste, kako se ne bi obustavio njegov razvoj.

8. Zaključak

U ovom radu smo predstavili nekoliko korisnih rešenja za smanjenje veličine mreže, kao i njeno ubrzanje. Optimizacija mreža je oblast koja se brzo razvija, i u poslednje dve godine se pojavljuje sve više mreža koje su same po sebi brze i nije ih potrebno dodatno ubrzavati. Neke od popularnih arhitektura su MobileNet [13], EfficientNet [14], ShuffleNet [15], SqueezeNet [16]. Zamena arhitektura nije uvek jednostavno kao zamena ResNet-a IResNet-om. Uvođenjem novih arhitektura sa manje parametra dovodi do novih problema prilikom treninga istih, te ovakvi problemi nisu uvek jednostavni. Sofver kao usluga (eng. Saas) je sve popularnija opcija, gde se korišćenje modela lincencira, te plaća po broju poziva. Na ovaj način je moguće osigurati sam model od nedozvoljenog korišćenja, ali sa druge strane pojednostaviti implementaciju sa strane klijenta. Ipak, i kod ovakvih sistema je bitna brzina modela, te se često koriste sve metode pomenute u ovom radu kako bi se smanjili troškovi električne energije, hardvera, oblaka. Prenos znanja je metoda koja je počela da se razvija relativno kasno u odnosu na sve ostale pomenute metode.

Uzimajući u obzir sve navedeno, dolazimo do zaključka da dizajnirane specijalizivanih modela za mobilne uređaje nije jednostavan proces, već često podrazumeva kombinaciju više koraka ukoliko je cilj rad u realnom vremenu. Iz svega navedenog proizilazi i potreba za kreiranje lakših, manjih modela sa manje parametra, i korišćenje novijih principa koji pokazuju značajna ubrzanja.

Kako se noviji mobilni uređaji pojavljuju sa sve više memorije, lako je osloniti se na to i iskoristiti bolji, ali kompleksniji model. Ne treba zaboraviti da na tržištu mobilnih uređaja i dalje većinu tržišta zauzimaju telefoni koji ne samo da ne mogu da po snazi pariraju novim uređajima iz visokog segmenta, nego je i sama softverska i hardverska podrška ne toliko dobra.

Još jedan od faktora je i vreme razvoja. Sam trening mreže poput MobileNet-a od početka do pokretanja na mobilnom uređaju može biti gotov i za jedan dan, ali operacije poput pruninga, kvantizacije, prenosa znanja, sparsifikacije mogu oduzeti mnogo više vremena, a čak i ne uvek dati željene rezultate. Osim toga, sam trening većih mreža za prenos znanja je proces koji je iscrpniji, kako sa stanovišta neophodnih resursa, tako i za vreme potrebno za trening.

Kombinacija svih prethodno navedenih metoda je još zahtevniji proces i može dovesti do naglog gubutka preciznosti modela ukoliko se ne radi pažljivo. Ukoliko je primena metoda kvantizacije i pruninga većeg modela dovoljna da se on ubrza na nivo nekog manjeg modela, sa zadovoljavajućom preciznošću, same metode prenosa znanja ne bi bilo potrebe koristiti.

Metode prenosa znanja su i dalje nova oblast u mašinskom učenju i njihov razvoj je tek počeo. Vremenom će ove metode tek doživeti svoj maksimum, odnosno postizanje SOTA rezultatata, što se već može videti sa nekima od njih. Njihova primena u oblasti analize lica je sve češća, a s obzirom da se proces prepoznavanja ne sastoji samo od samog enkodinga lica i pretrage, već i od njegove detekcije, kao i detekcije ključnih tačaka, metode prenosa znanja se ovde nameću kao opcija za kreiranje manjih, ali brzih i preciznih modela.

Literatura

[1] Deng, Jiankang, et al. "Arcface: Additive angular margin loss for deep face recognition." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

[2] Wang, Mengjiao, et al. "Improved knowledge distillation for training fast low resolution face recognition model." *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019.

[3] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

[4] Boutros, Fadi, et al. "PocketNet: extreme lightweight face recognition network using neural architecture search and multistep knowledge distillation." *IEEE Access* 10 (2022): 46823-46833.

[5] Bradski, Gary. "The openCV library." *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25.11 (2000): 120-123.

[6] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[7] Keskar, Nitish Shirish, et al. "On large-batch training for deep learning: Generalization gap and sharp minima." *arXiv preprint arXiv:1609.04836* (2016).

[8] Byrd, Richard H., et al. "Sample size selection in optimization methods for machine learning." *Mathematical programming* 134.1 (2012): 127-155.

[9] Stoychev, Samuil, and Hatice Gunes. "The effect of model compression on fairness in facial expression recognition." *arXiv preprint arXiv:2201.01709* (2022).

[10] Yeom, Seul-Ki, et al. "Pruning by explaining: A novel criterion for deep neural network pruning." *Pattern Recognition* 115 (2021): 107899.

[11] Lin, Shaohui, et al. "Towards optimal structured cnn pruning via generative adversarial learning." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

[12] Li, Hao, et al. "Pruning filters for efficient convnets." *arXiv preprint arXiv:1608.08710* (2016).

[13] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." *arXiv preprint arXiv:1704.04861* (2017).

[14] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International conference on machine learning*. PMLR, 2019.

[15] Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

[16] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

[17] Huber, Marco, et al. "Mask-invariant face recognition through template-level knowledge distillation." *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)*. IEEE, 2021.

[18] Jin, Haibo, et al. "Learning lightweight face detector with knowledge distillation." *2019 International Conference on Biometrics (ICB)*. IEEE, 2019.

[19] Guo, Jianzhu, et al. "Face synthesis for eyeglass-robust face recognition." *Chinese Conference on biometric recognition*. Springer, Cham, 2018.

[20] Duta, Ionut Cosmin, et al. "Improved residual networks for image and video recognition." *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021.

[21] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." *International journal of computer vision* 115.3 (2015): 211-252.

[22] Huang, Gary B., et al. "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments." *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*. 2008.

[23] Zheng, Tianyue, Weihong Deng, and Jiani Hu. "Cross-age lfw: A database for studying cross-age face recognition in unconstrained environments." *arXiv preprint arXiv:1708.08197* (2017).

[24] Huang, Gary, et al. "Learning to align from scratch." *Advances in neural information processing systems* 25 (2012).

[25 Hardt, Moritz, Ben Recht, and Yoram Singer. "Train faster, generalize better: Stability of stochastic gradient descent." *International conference on machine learning*. PMLR, 2016.

[26] Kim, Jangho, et al. "Qkd: Quantization-aware knowledge distillation." arXiv preprint arXiv:1911.12491 (2019).

[27] Boutros, Fadi, Naser Damer, and Arjan Kuijper. "QuantFace: Towards Lightweight Face Recognition by Synthetic Data Low-bit Quantization." arXiv preprint arXiv:2206.10526 (2022).

[28] Lopes, Raphael Gontijo, Stefano Fenu, and Thad Starner. "Data-free knowledge distillation for deep neural networks." arXiv preprint arXiv:1710.07535 (2017).

[29] Bellman, Richard. "Dynamic programming." *Science* 153.3731 (1966): 34-37.

[30] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998): 107-116.

[31] Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[32] Wolf, Thomas, et al. "Huggingface's transformers: State-of-the-art natural language processing." arXiv preprint arXiv:1910.03771 (2019).

[33] Wang, Feng, et al. "Normface: L2 hypersphere embedding for face verification." Proceedings of the 25th ACM international conference on Multimedia. 2017.

[34] Liu, Weiyang, et al. "Sphereface: Deep hypersphere embedding for face recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[35] Zhang, Xiao, et al. "Range loss for deep face recognition with long-tailed training data." Proceedings of the IEEE International Conference on Computer Vision. 2017.

[36] Wang, Hao, et al. "Cosface: Large margin cosine loss for deep face recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[37] Pereyra, Gabriel, et al. "Regularizing neural networks by penalizing confident output distributions." arXiv preprint arXiv:1701.06548 (2017).

[38] Ranjan, Rajeev, Carlos D. Castillo, and Rama Chellappa. "L2-constrained softmax loss for discriminative face verification." arXiv preprint arXiv:1703.09507 (2017).

[39] Wang, Feng, et al. "Additive margin softmax for face verification." IEEE Signal Processing Letters 25.7 (2018): 926-930.

[40] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009.

[41] Bianco, Simone, et al. "Benchmark analysis of representative deep neural network architectures." IEEE access 6 (2018): 64270-64277.