SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 480

Duboka konvolucijska arhitektura EfficientNetV2

Karlo Frankola

Zagreb, srpanj 2022.

Zahvaljujem se obitelji na iskazanoj podršci kroz cjelokupno studiranje te mentoru Siniši Šegviću na svoj pruženoj pomoći pri izradi rada.

SADRŽAJ

1. Uvod

Računalni vid je područje umjetne inteligencije kojim se nastoji riješiti određene probleme ljudskog vida računalima. Jedan od najčešćih problema koji pokušavamo riješiti je klasifikacija, a to znači da želimo da računalo prepozna što se nalazi na nekoj slici. Na primjer, ako se na nekoj slici nalazi pas, želimo da računalo to može prepoznati.

Područje računalnog vida doživjelo je pravi procvat pojavom dubokog učenja te se primijetilo da je za mnoge probleme upravo to jedno od najboljih načina rješavanja. Tu dolazimo do problema zbog ogromne količine računalnih resursa koji su potrebni za trenirati takve duboke neuronske mreže. Za neke velike mreže potrebno je čak višetjedno treniranje na velikom broju TPU i GPU jezgri. Zbog toga se mnogi trude pronaći mreže koje će postići dovoljnu dobru točnost, a da ipak ne iziskuju toliko računalnih resursa. Neke od takvih mreža su EfficientNet (s dvije verzije) i MobileNet (s tri verzije).

Ovaj rad će proučavati klasifikaciju slika na skupu podataka CIFAR10[4] i sljedeće mreže: ResNet[3], EfficientNet[7] i EfficientNetv2[8]. Pokušat će se replicirati rezultati iz znanstvenih radova i pojasniti temeljni koncepti dubokog učenja i modela koje koristimo za raspoznavanje slika.

2. Strojno učenje

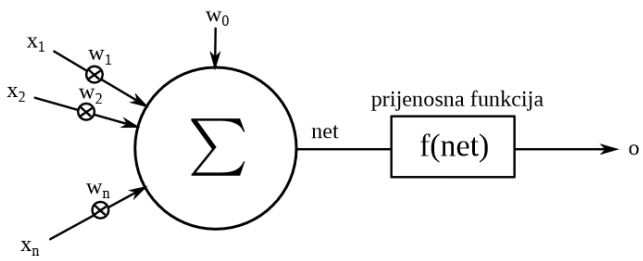
2.1. Teorijska podloga

Algoritmi strojnog učenja su algoritmi koji mogu učiti iz podataka. Kažemo da računalni program uči iz iskustva uzimajući u obzir neki zadatak (npr. klasifikacija) i mjeru performansi ako se njegova performansa na zadatku poboljšava s iskustvom[2].

U zadataku klasifikacije najčešće mjerimo preciznost modela, tj. proporciju točnih primjera za koje model vraća točni izlaz. Želimo da naši algoritmi rade dobro za podatke koje još nisu vidjeli pa te podatke dijelimo na skup podataka za treniranje i skup podataka za testiranje. Mjerimo performansu na skupu podataka za testiranje te tako provjeravamo generalizira li model ili je samo *napamet* naučio podatke. Kada model ima dobre performanse na skupu za treniranje, a ne na skupu za testiranje kažemo da je *pretreniran*.

2.2. Umjetne neuronske mreže

Umjetna neuronska mreža je skup međusobno povezanih jednostavnih procesnih elemenata (neurona) čija se funkcionalnost temelji na biološkom neuronu i koji služe distribuiranoj paralelnoj obradi podataka[1].



Slika 2.1: Osnovni model umjetnog neurona

Na slici 2.1 vidimo osnovni model umjetnog neurona. Vrijednosti sa svakog ulaza

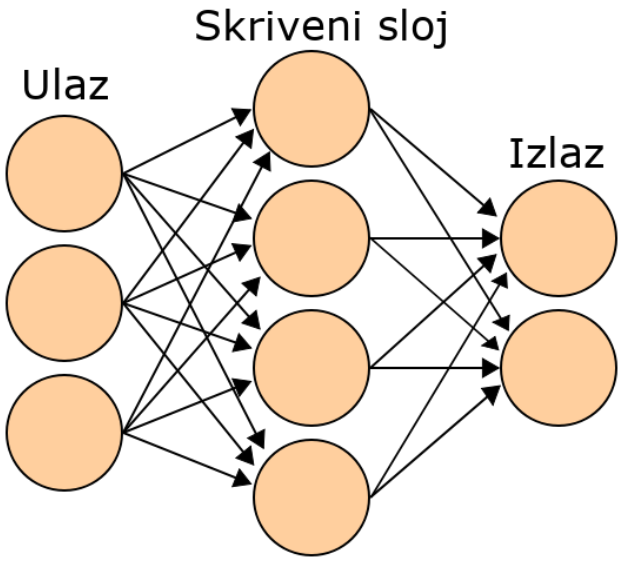
x_i množe se s osjetljivošću tog ulaza i akumuliraju. Ukupnoj sumi dodaje se pomak w_0 te se takva akumulirana vrijednost propušta kroz neku nelinearnu diferencijabilnu prijenosnu funkciju čime nastaje izlazna vrijednost. Ovakav umjetni neuron još se naziva i *perceptron* po izvornom radu u kojem je definiran. Element w_0 također nazivamo prag i označavamo s b (engl. *bias*). Funkciju neurona tada možemo zapisati kao:

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{2.1}$$

Zbog jednostavnosti često postavljamo da je $b = 1 \times w_0$ te uz vektor značajki ulaza $\boldsymbol{x}$ gdje je $x_0 = 1$ i vektor težina $\boldsymbol{w}$ jednadžbu neurona možemo zapisati kao:

$$y = f(\boldsymbol{w}^T \boldsymbol{x}) \tag{2.2}$$

Jedna od arhitektura umjetnih neuronskih mreža je unaprijedna neuronska mreža ili višeslojni perceptron. Cilj unaprijedne neuronske mreže je aproksimacija neke funkcije f^*. Na primjer za klasifikator, $y = f^*(\boldsymbol{x})$ preslikava ulaz $\boldsymbol{x}$ u kategoriju y. Unaprijedna mreža definira preslikavanje $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ i uči vrijednosti parametara $\boldsymbol{\theta}$ koji rezultiraju najboljom aproksimacijom funkcije[2].



Slika 2.2: Prikazana neuronska mreža sastoji se od 3 sloja. Ulaznog sloja od 3 neurona, skrivenog sloja od 4 neurona te izlaznog sloja od 2 neurona. Arhitekturu možemo kraće zapisati na način: 3x4x2.

Unaprijedne neuronske mreže zovu se mrežama jer su najčešće prikazane kompozicijom više funkcija. Na primjer ako neke 3 aproksimirane funkcije najjednostavnije

povežemo u lanac možemo imati na primjer: $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$. U ovom slučaju je $f^{(1)}$ prvi sloj mreže, $f^{(2)}$ drugi sloj i tako dalje. Ukupna duljina takvog lanca daje *dubinu* modela te termin "duboko učenje" proizlazi iz ove terminologije. Ilustracija takve proizvoljne mreže nalazi se na slici 2.2.

Po uzoru na jednadžbu 2.2 možemo prikazati i jednadžbu čitavog sloja. Tada ćemo umjesto vektorskog zapisa težina neurona imati matrični zapis. Svaki stupac u matrici težina $\boldsymbol{W}$ bit će vektor težina pojedinog neurona $\boldsymbol{w_j}$:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_0} & \dots & \boldsymbol{w_j} & \dots & \boldsymbol{w_n} \end{bmatrix} \tag{2.3}$$

Uz korištenje proširenog ulaza sa $x_0 = 1$ za prag jednadžbu sloja možemo zapisati kao:

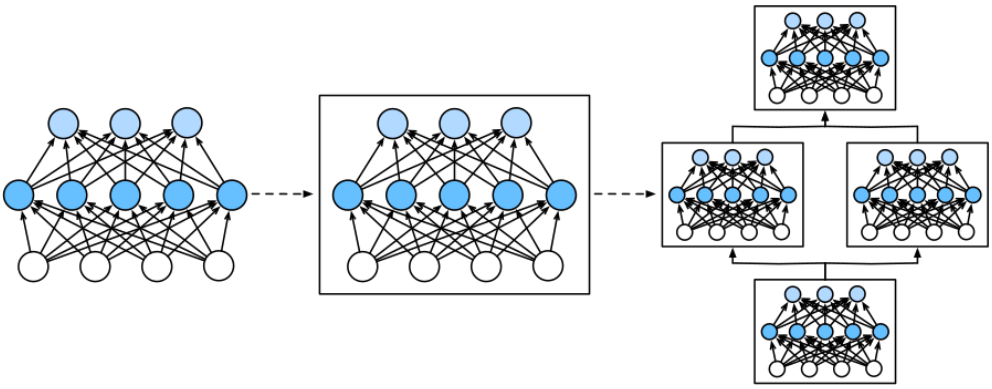
$$\boldsymbol{y} = f(\boldsymbol{W} \cdot \boldsymbol{x}) \tag{2.4}$$

Tijekom treniranja mi želimo da naša finalna funkcija aproksimira neku funkciju pa znamo samo kakva ona treba biti na zadnjem, izlaznom sloju. Slojeve između ulaznog i izlaznog sloja nazivamo *skrivenim slojevima* jer naši podaci za treniranje ne prikazuju željeni izlaz za svaki od tih slojeva.

2.3. Duboko učenje

Duboko učenje je specifična podvrsta strojnog učenja koja iskorištava višeslojne modele. U početku smo pričali o linearnom modelu neurona s jednim izlazom te smo nakon tog uveli umjetnu neuronsku mrežu s više slojeva, ali nažalost ni sami slojevi nisu dovoljna apstrakcija.

Moderne duboke neuronske mreže mogu koristiti stotine slojeva i pokazalo se da je često jako prigodno pričati o komponentama koje su veće od individualnog sloja, ali manje od cijelog modela. Na primjer ResNet arhitektura se sastoji od ponavljajućih *grupa slojeva* te bi implementacija takve mreže sloj po sloj bila izrazito naporna. Kako bi lakše implementirali takve kompleskne mreže uvodimo pojam **bloka** neuronske mreže [11].



Slika 2.3: Prikaz ponavljajućeg uzorka više slojeva pretvorenog u blok kao komponentu većih neuronskih mreža. Preuzeto iz [11].

2.4. Konvolucijske neuronske mreže

Ako koristimo unaprijedne neuronske mreže, kod rada sa slikama dolazimo do određenih problema. Pokazat ćemo zašto unaprijedni modeli nisu pogodni za rad sa slikama kao motivaciju za konvolucijske modele.

Slike mogu biti varijabilne veličine te ih možemo prikazati kao trodimenzionalni tenzor $x \in \mathbb{R}^{WHC}$ gdje W prikazuje širinu slike, H prikazuje visinu slike, a C je broj ulaznih kanala (npr. C=3 za RGB sliku ili C=1 za monokromatsku sliku). Problem do kojeg dolazimo ovdje je da bi morali naučiti različito veliku matricu težina $\boldsymbol{W}$ za svaku veličinu ulazne slike. Izuzev toga, čak i ako je veličina ulazne slike fiksna, broj potrebnih parametara bio bi prevelik za slike razumnih rezolucija, tada bi tek prva matrica težina morala imati veličinu $W \times H \times C \times D$ gdje je D broj izlaznih neurona u sljedećem sloju. I zadnji problem je taj što ako se neki uzorak pojavljuje u određenoj lokaciji na slici, možda ga nećemo prepoznati ako se pojavi na drugoj lokaciji na sliki. Na primjer kod klasifikacije ako se u skupu podataka za treniranje psi pojavljuju samo u gornjem lijevom kutu slike, svejedno bismo željeli da možemo prepoznati psa u donjem desnom kutu slike. To jest, unaprijedni modeli možda neće pokazivati svojstvo translacijske invarijantnosti jer ne postoje dijeljene težine na različitim lokacijama.[5]

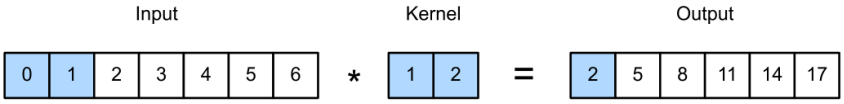
Kako bismo riješili te probleme, koristit ćemo konvolucijske neuronske mreže u kojoj ćemo operaciju množenja matrica zamijeniti s operacijom konvolucije. Osnovna ideja konvolucije je ta da ćemo ulaz podijeliti u 2-dimenzionalne komade slike koji se preklapaju te ćemo svaki komad uspoređivati s manjim skupom matrica težina ili *jezgri*. Komadi su često maleni (najčešće 3x3 ili 5x5) čime je broj parametara značajno smanjen. Sama operacija konvolucije nam omogućava svojstvo translacijske invarijantnosti.

2.4.1. Konvolucijski slojevi

Prvo ćemo opisati konvoluciju u jednoj dimenziji pa onda konvoluciju u dvije dimenzije i tada ćemo opisati kako se koristi kao ključna komponenta u konvolucijskim neuronskim mrežama.

Konvolucija između dvije funkcije, npr. $f, g : \mathbb{R}^D \to \mathbb{R}$, definirana je kao:

$$[f * g](\boldsymbol{z}) = \int_{\mathbb{R}^D} f(\boldsymbol{u}) g(\boldsymbol{z} - \boldsymbol{u}) d\boldsymbol{u} \tag{2.5}$$



Slika 2.4: Prikaz 1d konvolucije. Preuzeto iz [5]

Probajmo vidjeti što se dogodi kad u jednadžbi 2.5 zamijenimo funkcije s konačnim vektorima (o kojima možemo razmišljati kao funkcijama definiranima na konačnom skupu točaka). Recimo da je funkcija f evaluirana u točkama $\{-L, -L+1, \ldots, 0, 1, \ldots, L\}$ da dobijemo vektor težina (također zvan i jezgra) od w_{-L} do w_L. Neka je sada funkcija g evaluirana u točkama $\{-N, \ldots, N\}$ da dobijem vektor značajki od x_{-N} do x_N. Gornja jednadžba sada postaje:

$$[\boldsymbol{w} * \boldsymbol{x}](i) = w_{-L}x_{i+L} + \cdots + w_{-1}x_{i+1} + w_0 x_i + w_1 x_{i-1} + \cdots + w_L x_{i-L} \tag{2.6}$$

Vidimo da rezultat dobivamo tako da "preokrenemo" vektor težina i posumiramo lokalne prozore u svakoj točki kao što to možemo vidjeti na slici 2.4.

Vrlo slična operacija ovoj je ona u kojoj ne preokrenemo vektor $\boldsymbol{w}$:

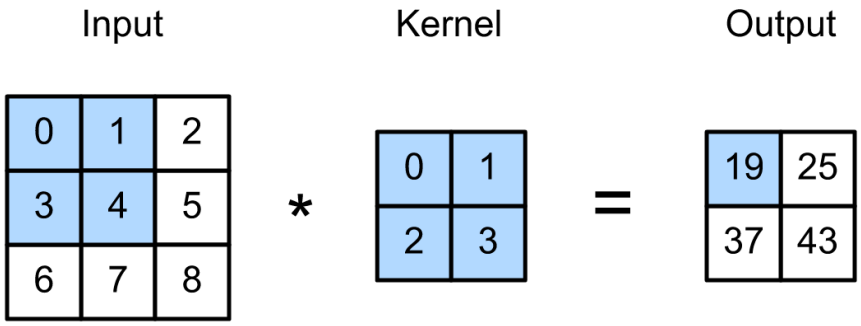
$$[\boldsymbol{w} * \boldsymbol{x}](i) = w_{-L}x_{i-L} + \cdots + w_{-1}x_{i-1} + w_0 x_i + w_1 x_{i+1} + \cdots + w_L x_{i+L} \tag{2.7}$$

Ova jednadžba se zove unakrsnom korelacijom. Ako je težinski vektor simetričan, što je često i slučaj, tada su unakrsna korelacija i konvolucija iste. U literaturi dubokog učenja termin "konvolucija" se često koristi umjesto termina unakrsne korelacije.

Također možemo evaluirati težine na nenegativnoj domeni kako bi mogli eliminirati negativne indekse. Tada gornja jednadžba postaje:

$$[\boldsymbol{w} * \boldsymbol{x}](i) = \sum_{u=0}^{L-1} w_u x_{i+u} \tag{2.8}$$

Često radimo sa slikama pa će nam biti izrazito zanimljiva konvolucija u 2 dimenzije.

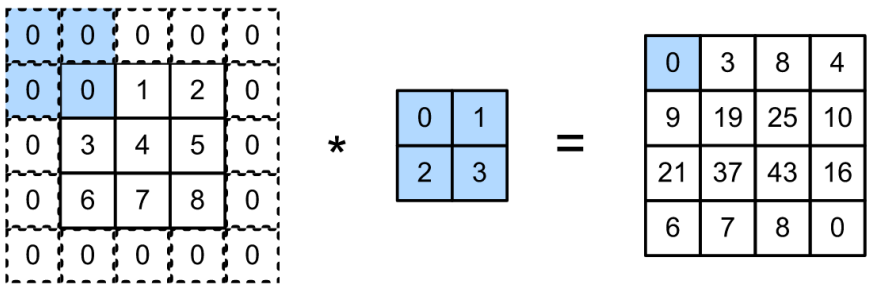


Slika 2.5: Prikaz 2d konvolucije. Preuzeto iz [11]

U 2 dimenzije jednadžba 2.8 postaje:

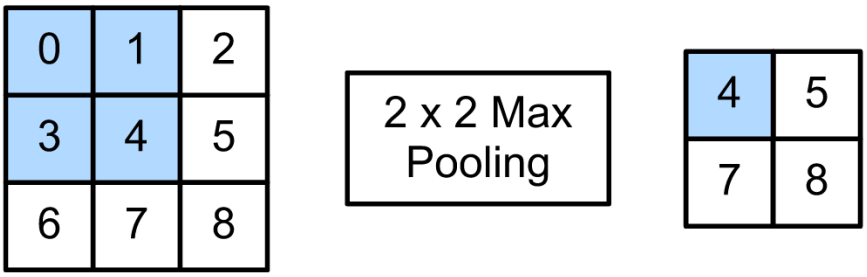
$$[\boldsymbol{W} * \boldsymbol{X}](i,j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v} x_{i+u,j+v} \tag{2.9}$$

gdje je 2-dimenzionalna jezgra $\boldsymbol{W}$ velika $H \times W$. Operaciju konvolucije možemo zamisliti kao način *detekcije značajki*. Tada se 2-dimenzionalni izlaz operacije konvolucije $\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X}$ zove mapa značajki. Za razliku od običnog neurona gdje nam je izlaz bio samo jedan broj, sada imamo i višedimenzionalni izlaz.



Slika 2.6: Prikaz 2d konvolucije s nadopunjavanjem nulama. Preuzeto iz [11].

U ovoj operaciji konvolucije imamo još dvije bitne značajke, a to su nadopunjavanje nulama (engl. *zero-padding*) i pomak (engl. *stride*) [11]. Za operaciju konvolucije koju smo prikazali na slici 2.5 vidimo da za ulaz veličine 3x3 i jezgru veličine 2x2 dobivamo izlaz veličine 2x2. Te dimenzije možemo promijeniti nadopunjavanjem nulama kako je prikazano na slici 2.6. Pomak predstavlja vrijednosti za koje pomičemo jezgru udesno i prema dolje. Npr. pomak $(2,3)$ označava da ćemo jezgru pomicati za dva polja udesno te za tri polja prema dolje.



Slika 2.7: Prikaz sažimanja maksimalnom vrijednošću regije. Preuzeto iz [11].

2.4.2. Slojevi sažimanja

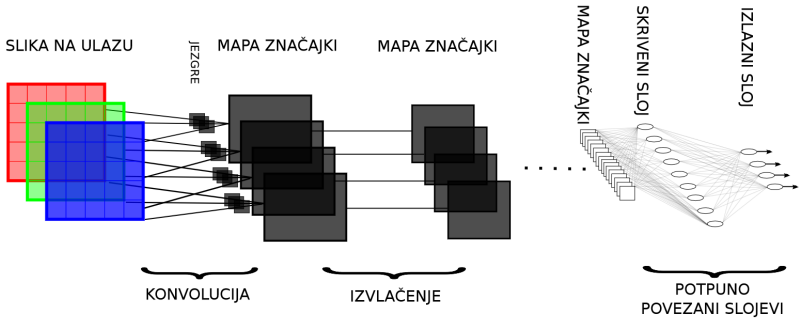
Drugi bitan dio konvolucijskih mreža su slojevi sažimanja. Oni nam također povećavaju invarijantnost na pomak, a ta invarijantnost nam je vrlo korisna ako nam je bitnije prepoznati prisutnost neke značajke, nego njezinu točnu lokaciju. Na slici 2.7 prikazano je sažimanje maksimalnom vrijednošću regije, a još koristimo i sažimanje usrednjavanjem, sažimanje Gaussovim usrednjavanjem i sažimanje Lp metrikama.

2.5. Efikasnost neuronskih mreža

Postoje različite mjere efikasnosti neuronskih mreža. Jedna od mjera efikasnosti su "FLOPs" (Floating Point Operations), a oni nam govore koliko je takvih operacija potrebno kako bi se pokrenuo jedan prolaz unaprijed na modelu. Dakle što je broj FLOP-ova veći to je model sporiji. Druga stvar koju možemo mjeriti je broj parametara i memorijsku efikasnost, naravno da nije isto ako model ima 100 milijuna parametara ili 10 milijuna. Često se to odražava i na FLOP-ove, ali nisu ista stvar. O mjerama efikasnosti bit će riječ u daljnjem tekstu kad budemo diskutirali o efikasnosti izabranih modela.

3. Konvolucijske arhitekture

Konvolucijske arhitekture su arhitekture neuronskih mreža koje u svojim slojevima koriste gore navedene slojeve uz različite dodatke. Rezidualne mreže, na primjer, iskorištavaju mogućnosti rezidualnih blokova kako bi lakše naučili prepoznavanje te su od svog začetka jedan od najpopularnijih modela u računalnom vidu[3].

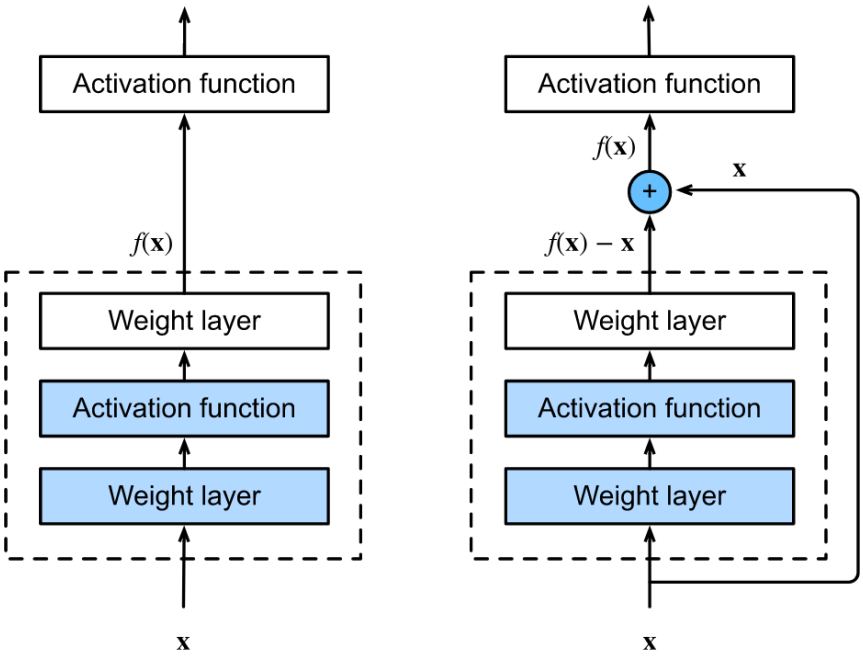


Slika 3.1: Prikaz opće arhitekture konvolucijske mreže. Preuzeto iz [9].

Opća arhitektura konvolucijskih mreža dana je na slici 3.1. Na ulazu dobivamo sliku s 3 kanala u RGB modu te kroz više slojeva konvolucije i sažimanja dobivamo mape značajki preko kojih možemo prepoznati nešto na slici. U zadnjem dijelu mreže prikazani su potpuno povezani slojevi koji koriste finalni vektor značajki na ulazu za neku predikciju. Treba napomenuti i da potpuno povezani slojevi na kraju konvolucijskih mreža nisu nužni te se danas često koristi i globalno sažimanje usrednjavanjem (engl. *Global Average Pooling*) gdje se izlaz toga daje direktno u softmax sloj.

3.1. Rezidualna mreža - ResNet

Rezidualne mreže iskorištavaju mogućnosti rezidualnih blokova. Glavno obilježje rezidualnih blokova je postojanje preskočnih veza. Njihova uloga je da u sljedeći blok prenose ulaz trenutnog bloka koji nije bio transformiran prolaskom kroz slojeve prošlog bloka. Rezidualni blok se može opisati sljedećim izrazom:

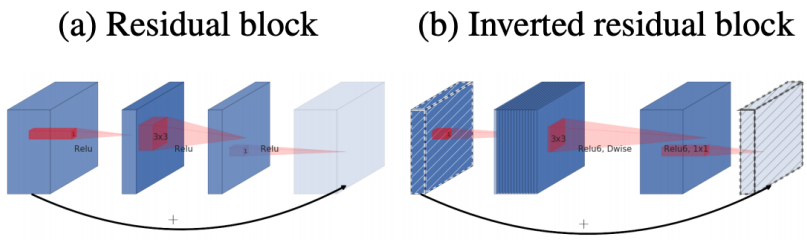


Slika 3.2: Razlika između običnog bloka (lijevo) i rezidualnog bloka (desno). Preuzeto iz [11]

$$x_{l+1} = f(h(x_l) + F(x_l, W_l)) \tag{3.1}$$

gdje su x_l i x_{l+1} ulaz i izlaz l-tog bloka, a F rezidualna funkcija. $h(x_l)$ je funkcija identiteta koja prenosi netransformirani ulaz u rezidualni blok. Rezidualna funkcija F je implementirana kao niz transformacija nad ulaznim podacima. Ulazni podaci se transformiraju prolaskom kroz konvolucijske slojeve bloka. Na slici 3.2 grafički je prikazan izgled rezidualnog bloka.

3.2. EfficientNet



Slika 3.3: Razlika između rezidualnog bloka (lijevo) i invertiranog rezidualnog bloka (desno). Preuzeto iz [6].

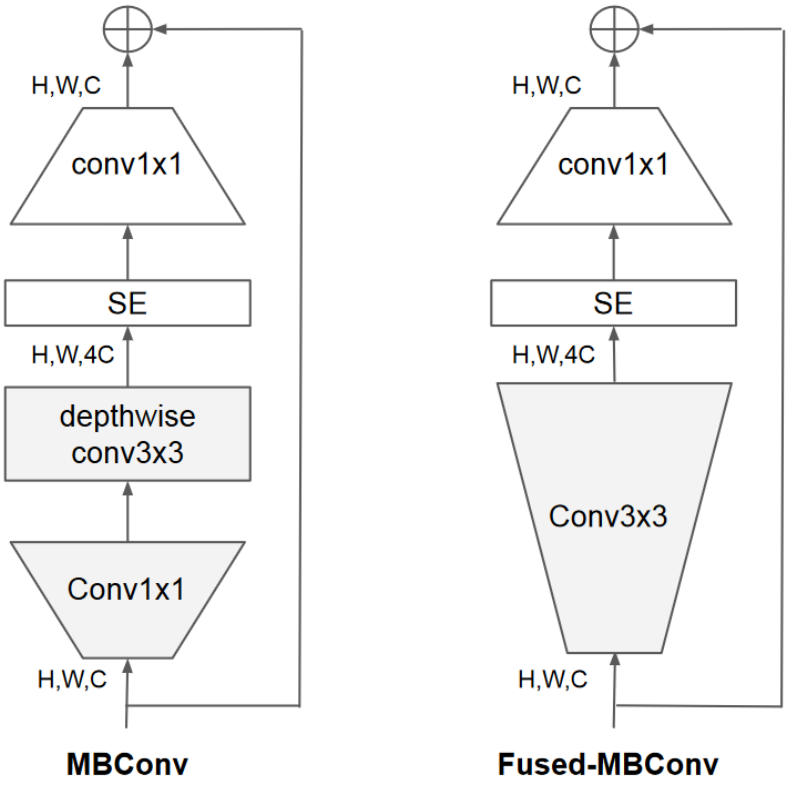
EfficientNet je vrsta mreže koja intenzivno koristi invertirane rezidualne blokove,

često zvane i MBConv blokovima jer su prvi put zamišljene u MobileNetV2 konvolucijskoj arhitekturi [6]. Invertirani rezidualni blokovi za razliku od rezidualnih blokova preskočnom vezom povezuju "uska grla" s manjim brojem kanala što je i ilustrirano na slici 3.3.

Rad EfficientNet također uvodi novu verziju skaliranja preko pretrage neuralne arhitekture kako bi maksimizirao točnost uz što manji broj parametara.

3.3. EfficientNetV2

EfficientNetV2 zamišljen je kao nadogradnja na prethodnu verziju te nije čudno da koristi sličnu arhitekturu kombinacije invertiranih rezidualnih blokova, ali također uvodi jednu novinu: *Fused-MBConv* blokove u ranijim fazama mreže za bolju parametarsku efikasnost. Na slici 3.4 vidimo da se umjesto 1x1 konvolucije i 3x3 konvolucije u dubinu koristi 3x3 konvolucija. To se promijenilo jer su konvolucije u dubinu jako spore u ranijim slojevima, ali su izrazito efektivne u kasnijim slojevima pa se novi blok koristi samo u početku mreže.



Slika 3.4: Razlika između invertiranog rezidualnog bloka MBConv (lijevo) i Fused-MBConv bloka (desno). Preuzeto iz [8]

4. Eksperimentalni rezultati

Gore opisane mreže evaluirane su na slobodno dostupnom skupu podataka CIFAR10 te su validirani hiperparametri. U ovom poglavlju opisat ću rezultate te cijeli proces treniranja i izrade.

4.1. Programska potpora

Programska izvedba cijelog rada ostvarena je u programskom jeziku Python uz korištenje *PyTorch* biblioteke za baratanje skupovima podataka te za učenje modela. Biblioteka "timm" korištena je kao izvor modela za treniranje.

4.1.1. PyTorch

PyTorch je programski okvir za duboko učenje koji omogućuje jednostavno kreiranje modela, njihovo treniranje te sve potrebne operacije prilagodbe skupa podataka i slično. PyTorch tenzori imaju ugrađenu GPU podršku te se operacije nad tenzorima mogu izvoditi na grafičkim procesorskim jedinicama. Jako je jednostavno prebacivati modele i tenzore s procesora na grafičku jedinicu i obrnuto. Korištenje PyTorch biblioteke znatno olakšava cjelokupni rad oko dubokog učenja te nam omogućava koncentraciju na bitne dijelove: izgradnju i evaluaciju modela.

4.1.2. timm

"timm" ili "Torch Image Models"[10] je biblioteka otvorenog koda koja sadrži različite implementacije modela za slike u PyTorch verziji. Naime, i EfficientNet i EfficientNetV2 su izvorno napisani u TensorFlow-u pa je korištenje ove biblioteke za gotov model puno olakšalo proces rada.

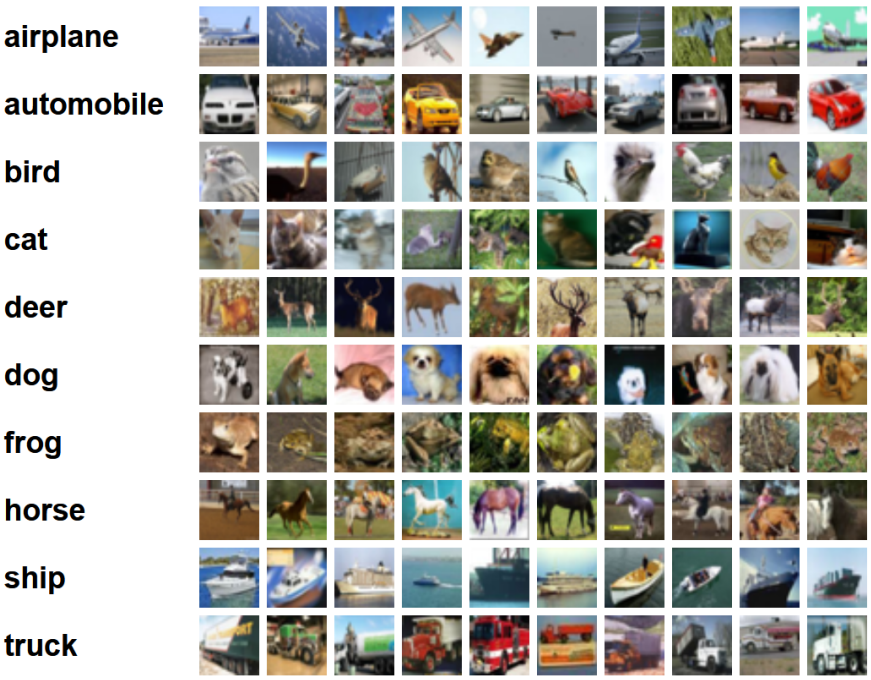
Korištene su najmanje verzije gore navedenih modela kako bi se ubrzao proces učenja. To su ResNet-18, EfficientNet-B0 i EfficientNetV2-S.

4.1.3. Oblikovanje programske potpore

Sama programska potpora sastoji se od dvije datoteke: train.py i main.py. Glavni program prima određene hiperparametre i postavke preko standardnog ulaza, pokreće modele i trenira ih na skupu podataka. Kad završi s treniranjem sprema vizualizaciju gubitka i točnosti te uvijek spremi najbolji model. Datoteka train.py osmišljena je da olakša postupak treniranja te se u njoj mogu namjestiti vrijednosti hiperparametara s kojima ona pokreće zadani model, sprema trening log i pamti koji modeli su istrenirani kako se ne bi radio dupli posao.

4.2. CIFAR10 skup podataka

CIFAR10 je skup podataka koji se sastoji od 60000 32x32 RGB slika u 10 klasa. U današnje vrijeme često se koristi za demonstraciju prijenosnog učenja (engl. *Transfer Learning*), ali ja sam ga koristio kao skup za treniranje i evaluaciju zbog praktičnih razloga - ostali skupovi podataka su jednostavno preveliki kako bih ih mogao evaluirati na slobodno dostupnim računalnim resursima. Na primjer ImageNet se sastoji od oko 1.3 milijuna slika različitih veličina te je veličine od oko 166GB.

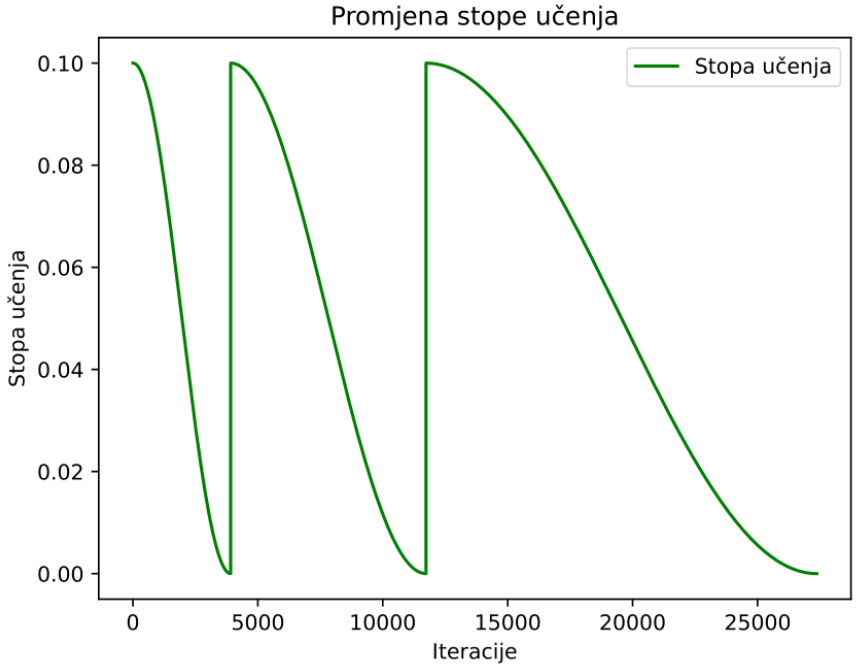


Slika 4.1: Vizualizacija skupa podataka CIFAR10 s 10 nasumičnih slika po klasi [4].

4.3. Treniranje mreža

Mreže su trenirane na osobnom računalu s Nvidia GTX 1650 Ti grafičkom karticom. Korišteno je kosinusno kaljenje s toplim ponovnim pokušajima (engl. *Cosine Annealing with Warm Restarts*) kao što je prikazano na slici 4.2 kroz broj iteracija optimizacije. Modeli su trenirani kroz 70 epoha. Korištene su standardne augmentacije slike za CIFAR10 - nasumični izrez sa nadopunom od 4 piksela i nasumično zrcaljenje. To bi trebalo poboljšati generalizaciju na skupu podataka za testiranje i smanjiti prenaučenost.

Treniranje ResNet-18 modela je u prosjeku trajalo 30 minuta, EfficientNet-B0 modela 42 minute, EfficientNetV2-S modela oko 75 minuta. Čak i s tim duljim vremenom treniranja nisu primjećene velike razlike u performansama i točnosti.



Slika 4.2: Prikaz promjene stope učenja kroz različite iteracije treniranja modela. Ovdje je prikazana promjena stope učenja za maksimalnu stopu učenja od 0.03, ali graf bi slično izgledao i za stope učenja 0.05 i 0.1.

4.4. Mjerenja

U tablici 4.1 prikazani su rezultati za svaku kombinaciju hiperparametara. Modeli su trenirani na cijelom skupu podataka i konačno evaluirani na testnom skupu podataka.

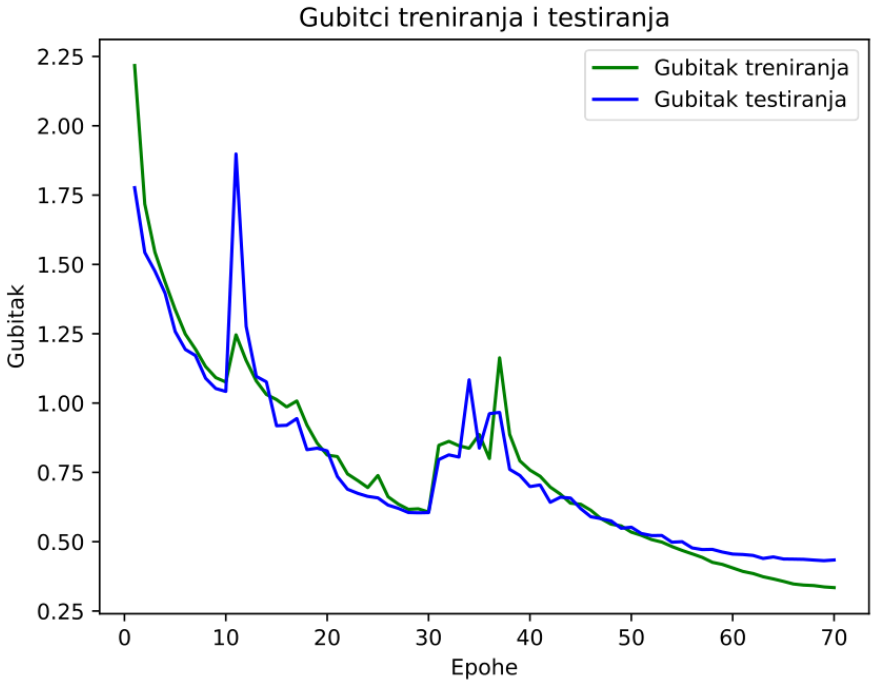
Ono što je interesantno je da u ResNet-18 modelu ne vidimo velike varijacije u točnosti rezultati - najmanja točnost je 83.94%, a najveća 86.79%. Dok u EfficientNet mrežama to nije slučaj i ogromne su varijacije između postavljenih hiperparametara.

Stopa učenja	Zalet	Weight Decay	ResNet18	EfficientNet-B0	EfficientNetV2-S
0.03	0.85	0	0.8400	0.8037	0.8008
0.03	0.85	0.0001	0.8482	0.8172	0.8200
0.03	0.85	1.00E-05	0.8394	0.8047	0.8001
0.03	0.9	0	0.8439	0.8067	0.7951
0.03	0.9	0.0001	0.8530	0.8324	0.8174
0.03	0.9	1.00E-05	0.8485	0.8120	0.7915
0.03	0.95	0	0.8478	0.8237	0.7573
0.03	0.95	0.0001	0.8626	0.7545	0.6727
0.03	0.95	1.00E-05	0.8485	0.7665	0.7842
0.05	0.85	0	0.8449	0.8153	0.8081
0.05	0.85	0.0001	0.8517	0.8361	0.7979
0.05	0.85	1.00E-05	0.8439	0.8234	**0.8255**
0.05	0.9	0	0.8490	0.8012	0.7659
0.05	0.9	0.0001	0.8604	0.8498	0.7907
0.05	0.9	1.00E-05	0.8492	0.8189	0.7948
0.05	0.95	0	0.8508	0.6521	0.7437
0.05	0.95	0.0001	0.8610	0.8242	0.7061
0.05	0.95	1.00E-05	0.8473	0.7690	0.7633
0.1	0.85	0	0.8494	0.7902	0.7694
0.1	0.85	0.0001	0.8600	0.8449	0.7919
0.1	0.85	1.00E-05	0.8495	0.8275	0.7839
0.1	0.9	0	0.8519	0.7175	0.6679
0.1	0.9	0.0001	0.8654	**0.8545**	0.7070
0.1	0.9	1.00E-05	0.8563	0.8039	0.6912
0.1	0.95	0	0.8529	0.6800	0.7187
0.1	0.95	0.0001	**0.8679**	0.8380	0.7796
0.1	0.95	1.00E-05	0.8530	0.7738	0.7430

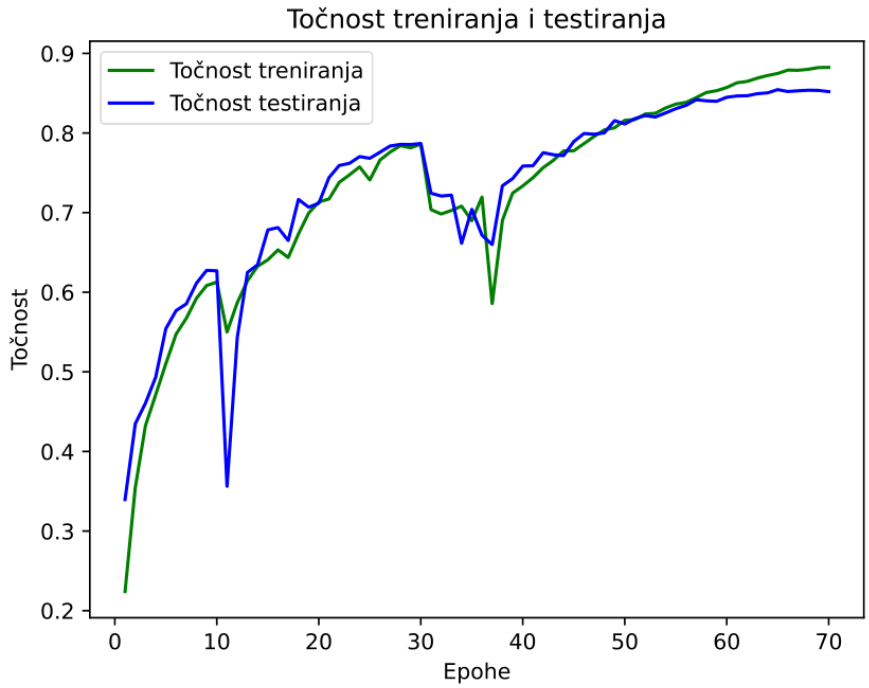
Tablica 4.1: Prikaz rezultata ovisno o odabranoj stopi učenja, weight decayju i zaletu za izabrane modele. Najbolji rezultati po mreži su istaknuti masno.

Nadalje su prikazani grafički rezultati promjene točnosti i gubitka kroz epohe tre-

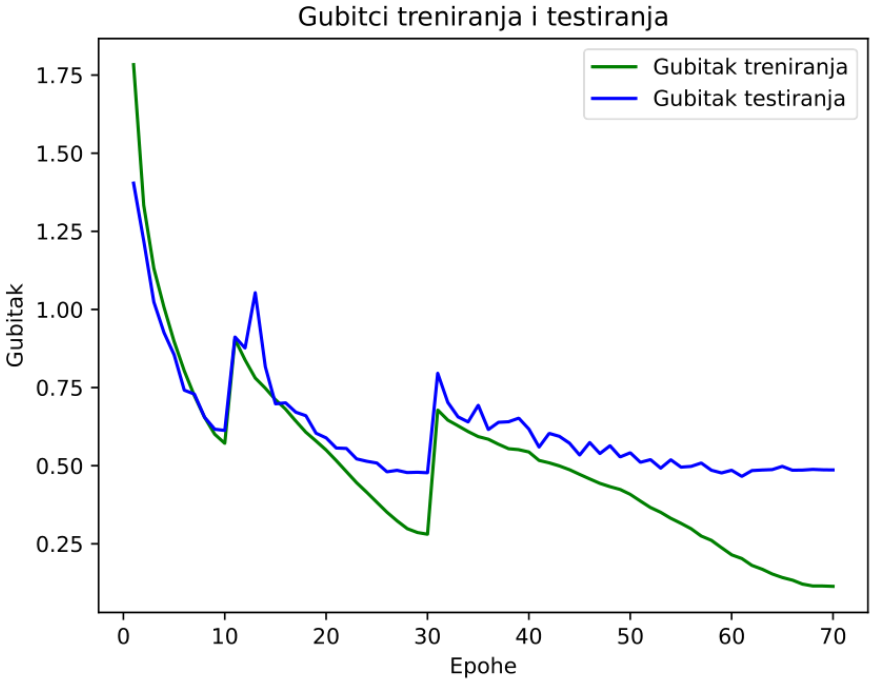
niranja na skupu podataka za treniranje i testiranje samo za najbolje rezultate.



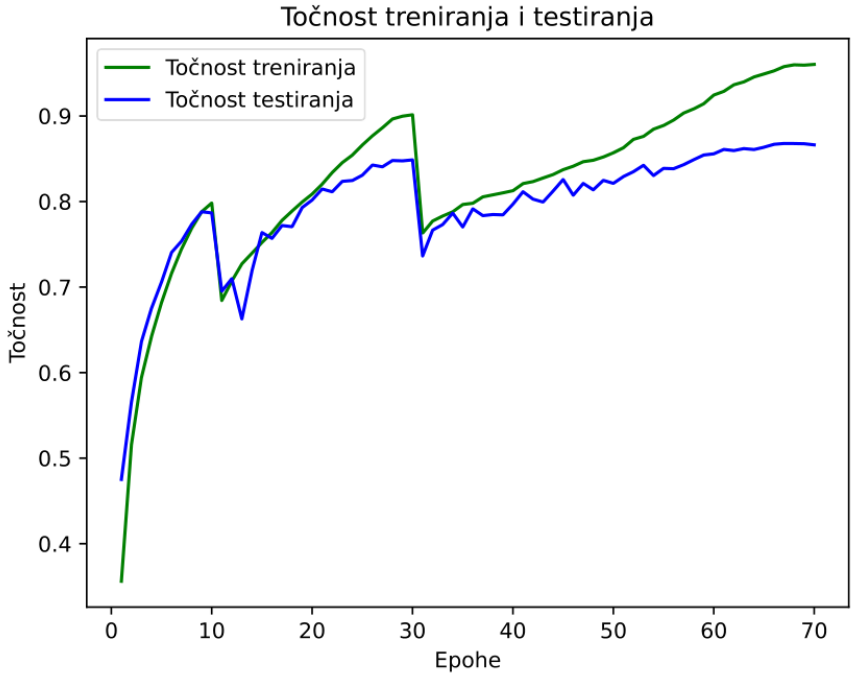
Slika 4.3: Prikaz promjene gubitka kroz epohe za ResNet-18.



Slika 4.4: Prikaz promjene točnosti kroz epohe za ResNet-18.

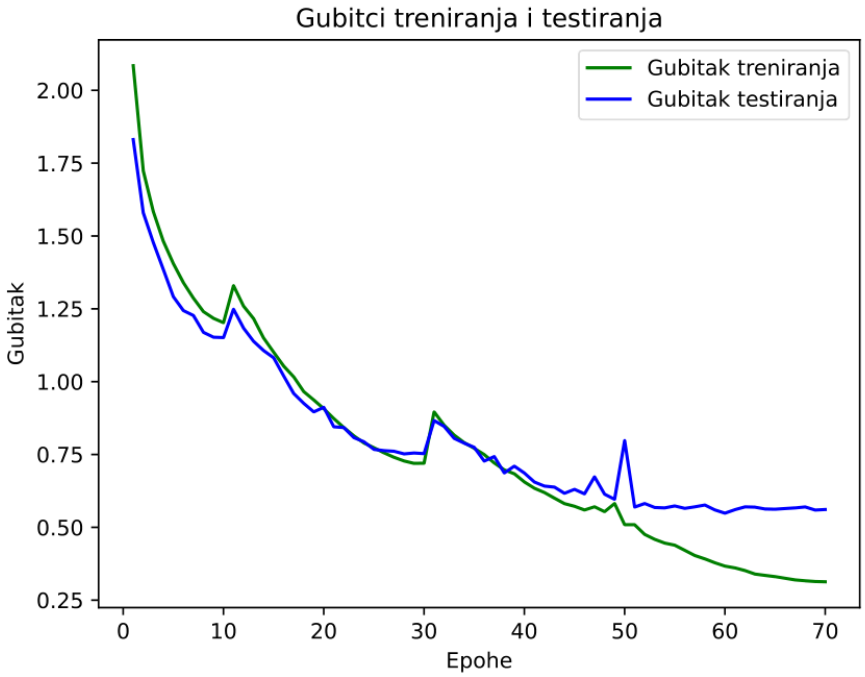


Slika 4.5: Prikaz promjene gubitka kroz epohe za EfficientNet.

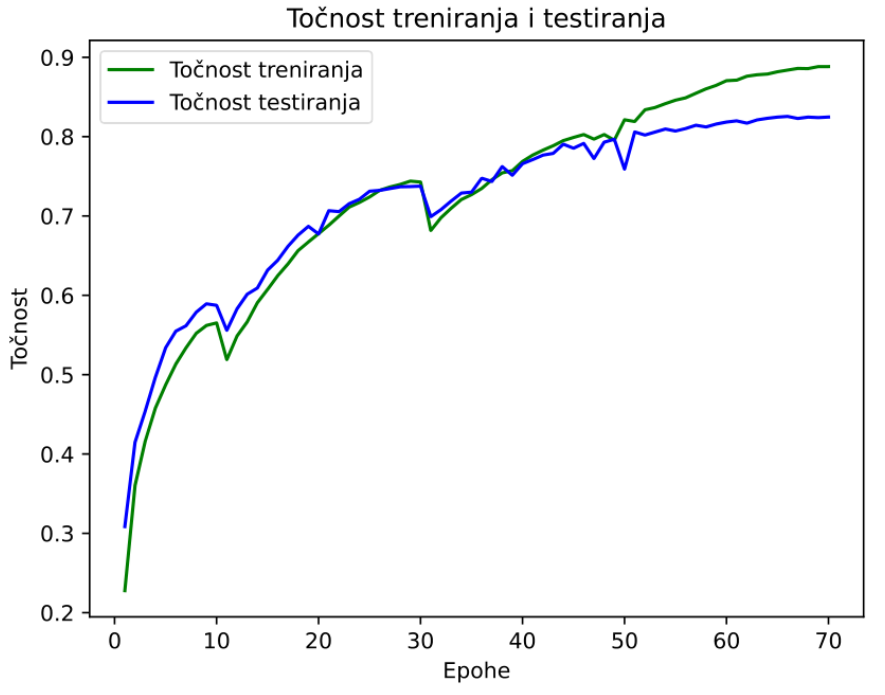


Slika 4.6: Prikaz promjene točnosti kroz epohe za EfficientNet.

Ovakvi relativno loši rezultati točnosti te velika varijabilnost u rezultatima EfficientNet-ova upućuju da nešto nije u redu s modelima. Naime, sve ove mreže u korijenskom konvolucijskom sloju imaju pomak jednak 2. To uglavnom nije problem ako se radi



Slika 4.7: Prikaz promjene gubitka kroz epohe za EfficientNetV2.



Slika 4.8: Prikaz promjene točnosti kroz epohe za EfficientNetV2.

na slikama većih rezolucija, ali na CIFAR10 skupu podataka sa svojim rezolucijama 32x32 modeli jednostavno ne mogu dovoljno dobro naučiti s toliko velikim pomakom.

Teoriziramo da je i EfficientNetV2-S sa svojih 20,19 milijuna parametara teško naučiti na relativno malenom skupu podataka. Nadalje su promijenjeni pomaci u korijenskim konvolucijama te se koristi EfficientNetV2-B0, noviji model sa skaliranjem prve verzije.

Stopa učenja	Zalet	Weight Decay	ResNet18	EfficientNet-B0	EfficientNetV2-B0
0.03	0.85	0	0.8822	0.8624	0.8747
0.03	0.85	0.0001	0.8865	0.8624	0.8774
0.03	0.85	1.00E-05	0.8829	0.8677	0.8750
0.03	0.9	0	0.8895	0.8725	0.8769
0.03	0.9	0.0001	0.8916	0.8788	0.8809
0.03	0.9	1.00E-05	0.8840	0.8717	0.8835
0.03	0.95	0	0.8897	0.8879	0.8749
0.03	0.95	0.0001	0.9005	0.9000	0.8866
0.03	0.95	1.00E-05	0.8906	0.8870	0.8759
0.05	0.85	0	0.8850	0.8789	0.8682
0.05	0.85	0.0001	0.8923	0.8862	0.8908
0.05	0.85	1.00E-05	0.8861	0.8753	0.8734
0.05	0.9	0	0.8933	0.8799	0.8848
0.05	0.9	0.0001	0.8975	0.9010	0.8925
0.05	0.9	1.00E-05	0.8954	0.8837	0.8844
0.05	0.95	0	0.8951	0.8879	0.8636
0.05	0.95	0.0001	0.9070	**0.9119**	0.8861
0.05	0.95	1.00E-05	0.8966	0.8939	0.8747
0.1	0.85	0	0.8956	0.8844	0.8791
0.1	0.85	0.0001	0.9023	0.9074	0.8927
0.1	0.85	1.00E-05	0.8926	0.8917	0.8763
0.1	0.9	0	0.8920	0.8890	0.8799
0.1	0.9	0.0001	0.9072	0.8965	0.8813
0.1	0.9	1.00E-05	0.8957	0.8928	0.8811
0.1	0.95	0	0.8956	0.8887	0.8374
0.1	0.95	0.0001	**0.9131**	0.9012	**0.8987**
0.1	0.95	1.00E-05	0.8969	0.8970	0.8420

Tablica 4.2: Ponovljeni eksperimenti s promijenjenim modelima. Najbolji rezultati u pretrazi su istaknuti masno.

Pri ponovljenim eksperimentima vidimo puno bolje rezultate s kojima smo relativno zadovoljni. Pretpostavljamo da smo pronašli najbolje hiperparametre te provodimo konačna treniranja s duplo više epoha (140). Također skaliramo EfficientNet modele na broj parametara sumjerljiv s ResNet18 kako bi dobili bolju usporedbu modela.

U konačnom treniranju dobivamo sljedeće rezultate:

	ResNet18	EfficientNet-B3	EfficientNetV2-B3
Broj parametara	11.18M	10.71M	12.84M
Točnost	0.9218	0.9341	0.9164

Tablica 4.3: Tablični prikaz najboljih dobivenih modela.

Na najboljim dobivenim modelima provodili smo mjerenja efikasnosti, zauzeća memorije i vremena potrebnog za obradu jedne slike na GPU i CPU. Ti rezultati su prikazani dolje. Ova testiranja provedena su u Google Colab-u zbog memorijskih ograničenja vlastite grafičke kartice.

	ResNet18	EfficientNet-B3	EfficientNetV2-B3
Broj parametara	11.18M	10.71M	12.84M
Broj operacija (GMAC)	81.62	43.20	72.28
Zauzeće memorije GPU (MiB)	1420	8387	5172
Vrijeme obrade GPU (ms)	40.79 ± 0.35	117.57 ± 0.68	88.81 ± 0.63
Vrijeme obrade CPU (ms)	2673.25 ± 29.39	3893.17 ± 85.59	3300.36 ± 84.17

Tablica 4.4: Rezultati mjerenja evaluacije na jednoj nasumično generiranoj trokanalnoj slici rezolucije 750x750. Evaluacija se ponovila 100 puta nakon čega se izračuna prosjek i standardna devijacija.

5. Zaključak

5.1. Pregled

Klasifikacija slika važan je problem računalnog vida. Konvolucijske mreže postižu vrhunske rezultate u klasifikaciji slika, ali pronalaženje parametarski efikasnih arhitektura veliki je problem.

Iako EfficientNet-B3 (10.7 milijuna parametara) ima manje parametara i manji ukupni broj operacija pri evaluaciji od ResNet-18 (11.18 milijuna parametara) modela, razlika u točnosti bilo je u samo 1.23 postotna boda. Nadalje vidimo da taj manji broj operacija nije značio i manje potrebno vrijeme pri obradi slike, EfficientNet-B3 je imao daleko najlošije rezultate od tog te je u prosjeku skoro 3 puta sporiji od ResNet-18 modela. Ogromnu razliku u memorijskim zauzećima (EfficientNet skoro 6 puta više memorije, EfficientNetV2 skoro 4 puta više memorije od ResNet modela) objašnjavamo upotrebom dubinski separabilnih konvolucija u tim modelima zbog kojih oni moraju pamtiti puno više među rezultate pri unaprijednim i unatražnim prolazima. Iako to nije bio glavni cilj rada, također možemo vidjeti koliko paralelizacija obrade tenzora na GPU instancama povećava brzinu i preko 40 puta.

Količina razlike u vremenu obrade unatoč manjem ukupnom broju operacija nije uspjela biti potpuno objašnjenja. Klasični TensorFlow model postiže puno bolje inferencijske rezultate od ovih te je jedan moguć razlog neoptimizirana operacija grupnih konvolucija u PyTorch biblioteci. Naime, TensorFlow ima vlastitu, puno bržu implementaciju te operacije te time postiže bolje rezultate u svom izvornom modelu.

5.2. Budući rad

Budući rad bi mogao uključiti fino ugađanje na predtreniranim modelima i usporedbu tako dobivenih rezultata. Čini se da EfficientNet-ovi u takvim slučajevima znatno premašuju točnost, a i parametarsku efikasnost rezidualnih mreža.

U ovom radu se namještanju hiperparametara pristupilo ručno: odabrale su se neke moguće vrijednosti te provela iscrpna pretraga nad njima koji daju najbolji rezultat na testnom skupu. U daljnjem radu moglo bi se razmotriti korištenje rješenja koja automatski podešavaju hiperparametre, a mogli bi se provesti i eksperimenti s različitim optimizatorima, primjerice Adam-om.

Također bi bilo dobro istražiti koliko operacija dubinski separabilne konvolucije u stvari odmaže modelima u PyTorch-u. Tu bi se ti blokovi mogli zamijeniti klasičnim konvolucijskim blokovima ili se također može provesti testiranje smanjenjem preciznosti parametara na FP16 umjesto FP32 tipa podataka. Takav tip podataka u PyTorch-u pri obradi grupnih konvolucija trebao bi omogućiti korištenje optimiziranije paralelizacije za takav tip operacije.

LITERATURA

[1] Bojana Dalbelo Bašić, Marko Čupić, i Jan Šnajder. Uvod u umjetnu inteligenciju, slajdovi na prezentacijama, 2020.

[2] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[4] Alex Krizhevsky. The cifar-10 dataset. URL https://www.cs.toronto.edu/~kriz/cifar.html.

[5] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL probml.ai.

[6] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, i Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

[7] Mingxing Tan i Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL http://arxiv.org/abs/1905.11946.

[8] Mingxing Tan i Quoc V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, abs/2104.00298, 2021. URL https://arxiv.org/abs/2104.00298.

[9] Vedran Vukotić. Raspoznavanje objekata dubokim neuronskim mrežama. 2014.

[10] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[11] Aston Zhang, Zachary C. Lipton, Mu Li, i Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Duboka konvolucijska arhitektura EfficientNetV2

Sažetak

Klasifikacija slika važan je problem u računalnom vidu s mnogim zanimljivim primjenama. Trenutno stanje tehnike koristi duboke konvolucijske mreže. Velika računalna složenost tih modela dovodi do problema u treniranju i kasnijoj evaluaciji. To je dovelo da se posebno traže konvolucijske arhitekture koje kroz puno manji broj parametara svejedno mogu ostvariti komparabilnu točnost. U okviru rada odabran je okvir za automatsku diferencijaciju te biblioteke za rad s tenzorima i slikama. Ukratko je opisana teorijska podloga strojnog i dubokog učenja te neke klasične konvolucijske arhitekture. Istražene su efikasne arhitekture EfficientNet i EfficientNetV2 te uspoređeni njihovi rezultati s rezidualnim mrežama.

Ključne riječi: Duboko učenje, konvolucijske arhitekture, računalni vid, ResNet, EfficientNet, EfficientNetv2

Deep Convolutional Architecture EfficientNetV2

Abstract

Image classification is an important problem in computer vision with many interesting applications. Current state of the art techniques use deep convolutional networks. The large computational complexity of such models leads to problems in training and later evaluation of them. This lead to a special search for convolutional architectures that even with significantly less parameters are able to achieve comparable accuracy. In this paper, an automatic differentiation framework was chosen, and libraries for working with tensors and images. The theoretical basis for machine and deep learning was introduced and some classic convolutional architectures were explained. Efficient architectures of EfficientNet and EfficientNetV2 were researched and their results were compared to residual networks.

Keywords: Deep learning, convolutional architectures, computer vision, ResNet, EfficientNet, EfficientNetv2