SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6990

Diferencijalne deformacije u modelima za klasifikaciju slike

Pavo Matanović

Zagreb, srpanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6990

Pristupnik:	**Pavo Matanović (0036506316)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	prof. dr. sc. Siniša Šegvić

Zadatak: **Diferencijalne deformacije u modelima za klasifikaciju slike**

Opis zadatka:

Klasifikacija prirodnih slika važan je zadatak računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme vrlo zanimljive rezultate u okviru tog zadatka postižu duboki konvolucijski modeli. Ovaj rad razmatra mogućnost obogaćivanja dubokog modela deformabilnim afinim i poliharmoničnim transformacijama. U okviru rada, potrebno je odabrati okvir za automatsku diferencijaciju te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati postojeće pristupe za klasifikaciju slike. Predložiti arhitekturu dubokog modela koja bi uključivala diferencijabilne deformacije. Uhodati postupke učenja modela i validiranja hiperparametara. Primijeniti naučene modele te prikazati i ocijeniti ostvarene rezultate.. Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2020.

Zahvaljujem se mentoru prof. dr. sc. Siniši Šegviću, na pomoći pri izradi ovog rada i na trudu uloženom u prenošenje znanja. Zahvaljujem se roditeljima za pomoć i potporu tijekom školovanja.

SADRŽAJ

1. Uvod

Od izuma računala znanstvenici pokušavaju računalom imitirati ljudsku inteligenciju. Jedan od problema umjetne inteligencije je računalni vid — kako da računalo zaključi što se nalazi na slici? Područje računalnog vida koje se bavi prethodnim pitanjem je klasifikacija slika. Klasifikacija prirodnih slika važan je zadatak računalnog vida s mnogim zanimljivim primjenama. Iako su algoritmi umjetne inteligencije već duže vrijeme poznati, tek u zadnjih desetak godina događa se „procvat" UI zbog jeftinije računske snage. Jeftinija računska snaga omogućila je treniranje dubljih i kompleksnijih modela.

U posljednje vrijeme duboki konvolucijski modeli postižu vrlo dobre rezultate za problem klasifikacije prirodnih slika. Klasični modeli koji se koriste za klasifikaciju slika su konvolucijske neuronske mreže (engl. *Convolutional Neural Network*). Klasične CNN nisu invarijantne na translaciju i rotaciju ulaznih slika te neće uspješno klasificirati slike ako ih rotiramo za proizvoljan kut ili izvedemo neku drugu transformaciju (npr. translacija, skaliranje i rotacija).

U ovom radu razmatramo mogućnost proširenja modela deformabilnim poliharmoničnim transformacijama. Model proširujemo na način da mu dodamo modul za prostornu transformaciju (engl. *Spatial Transformer Network*, kratica STN) prije ulaza. STN se sastoji od lokalizacijske mreže, generatora rešetke te dijela za uzorkovanje originalne slike. Prednost ovakvog proširenja je mogućnost istovremenog treniranja STN i klasifikatora. Isti pristup rješavanju ovog problema koristi se u [5] i [17].

Još jedan način povećanja robusnosti klasfikatora pokazan je [5]. Osim na ulazu mreže, STN možemo dodatno ubaciti i nakon nekog sloja konvolucije.

2. Umjetna neuronska mreža

Umjetna neuronska mreža svoju inspiraciju crpi u ljudskom mozgu. Ljudski mozak sastoji se od velikog broja neurona koji rade paralelno. Poznato je da postoji više od 100 vrsta različitih neurona, a procijenjeno je da ukupan broj neurona u mozgu iznosi oko 10^{11}. Svaki neuron mozga u prosjeku dobiva informacije od 10^3 do 10^4 drugih neurona. Informacije u mozgu se obrađuju i serijski i paralelno.

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. (iz [2])

U poglavlju 2.1 pokazat ćemo model neurona od kojih se tvore umjetne neuronske mreže. Zatim ćemo u poglavljima 2.2, 2.3 i 2.4 pokazati osnovne gradivne jedinice od kojih se sastoje CNN.

2.1. Neuron

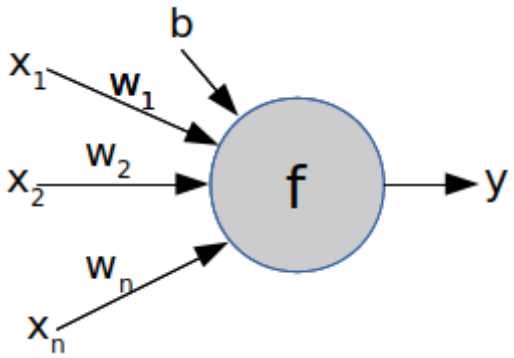
Model umjetnog neurona prikazan je na slici 2.1. Umjetni neuron se sastoji od n ulaza, jednog izlaza, prijenosne funkcije i praga (ne obavezno). Neuron množi vrijednost svakog ulaza x_i sa pripadajućom težinom w_i. Akumuliranoj vrijednosti zatim dodaje prag b (engl. *bias*) i propušta tu vrijednost kroz prijenosnu funkciju f.

Izlaz neurona matematički možemo prikazati kao:

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

Radi lakšeg baratanja, prilikom implementacije umjetnog neurona sa n ulaza, vrijednosti ulaza i težina držimo u matričnom obliku. Ulaze ćemo predstaviti vektorom $\boldsymbol{x} = [1\, x_1\, x_2\, \ldots\, x_n]^T$, a težine vektorom $\boldsymbol{w} = [b\, w_1\, w_2\, \ldots\, w_n]^T$. To nam omogućuje kompaktniji prikaz formule 2.1 i dan je izrazom 2.2:

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



Slika 2.1: Model umjetnog neurona. Izlaz neurona je težinska suma ulaza na koju primjenjujemo aktivacijsku funkciju.

Odabir aktivacijske funkcije f je bitan jer direktno utječe na rad neurona, a time i na rad cijele mreže. Za problem klasfikacije danas je najpopularnija zglobnica (engl. *Rectified Linear Unit*, kratica ReLU), a osim zglobnice koriste se sigmoidalna funkcija, tangens hiperbolni i propusna zglobnica (engl. *Leaky* ReLU).

2.1.1. Aktivacijske funkcije

Kao što je opisano u prethodnom poglavlju izbor aktivacijske funkcije važan je pri odabiru modela za rješavanje konkretnog problema. U nastavku ćemo dati opis trenutno najvažnijih aktivacijskih funkcija. Za svaku funkciju ćemo dati izraz, njenu derivaciju i ukratko komentirati prednosti i nedostatke.

Sigmoidalna funkcija (logistička) je dana izrazom 2.3, a njena derivacija izrazom 2.4.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x)) \tag{2.4}$$

Graf sigmoidalne funkcije prikazan je na slici 2.2a.
Prednosti sigmoidalne funkcije su derivabilnost, preslikavanje $\mathbb{R} \to [0,1]$ i njena nelinearnost. U starijim modelima je bila najvažnija aktivacijska funkcija za problem klasifikacije. *Nedostaci* sigmoide su gušenje gradijenta (nije pogodno za *backpropagation* algoritam) te što joj izlaz nije centriran oko nule.

Tangens hiperbolni je dan izrazom 2.5, a njegova derivacija izrazom 2.6.

$$\tanh(x) = \frac{2}{1+e^{-2x}} = 2\sigma(2x) - 1 \tag{2.5}$$

$$\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x) \tag{2.6}$$

Graf $\tanh$ prikazan je na slici 2.2b. Tangens hiperbolni sličan je sigmoidi, a njegova *prednost* u odnosu na sigmoidu je što mu je izlaz centriran oko nule.

Zglobnica je dana izrazom 2.7, a njena derivacija izrazom 2.8.

$$ReLU(x) = max(0, x) \tag{2.7}$$

$$\frac{dReLU(x)}{dx} = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{2.8}$$

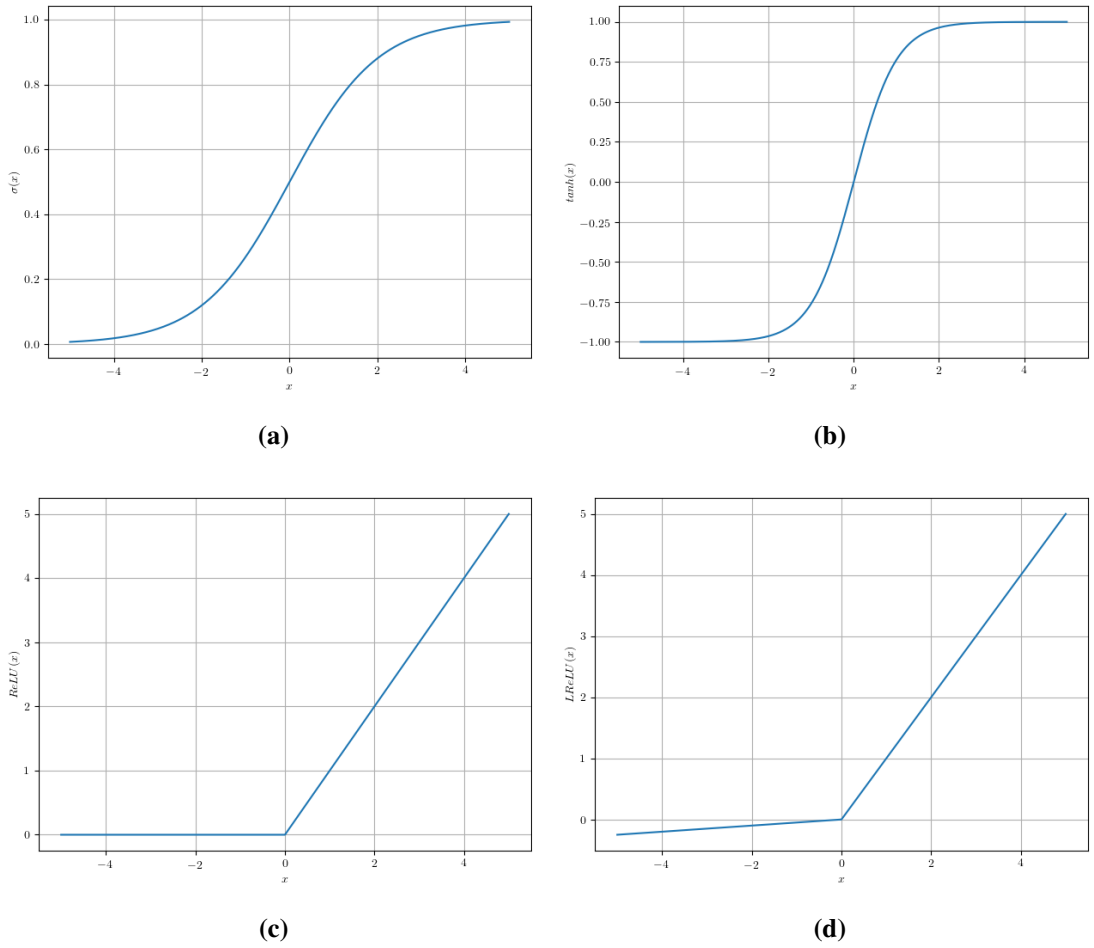
Graf zglobnice prikazan je na slici 2.2c. Zglobnica se danas najviše koristi, zbog mnogih *prednosti* u odnosu na prethodno navedene aktivacijske funkcije. Nema problem zasićenja s pozitivne strane (gradijent ima vrijednost 1 za $x > 1$ dok je u 2.4 i 2.6 približno 0) i jednostavna je za implementaciju. U [10] je pokazano da zglobnica ubrzava konvergenciju algoritma učenja do 6 puta u odnosu na $\tanh$. Glavni *nedostatak* zglobnice su *mrtvi neuroni* (engl. *dead neurons*). Ako se neuron inicijalno ne aktivira, algoritam učenja temeljen na gradijentima mu neće mijenjati težine (jer je gradijent u tom području 0) te će usporiti učenje. (vidi [12])

Propusna zglobnica je dana izrazom 2.9, a njena derivacija izrazom 2.10.

$$LReLU(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.9}$$

$$\frac{dLReLU(x)}{dx} = \begin{cases} \alpha & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{2.10}$$

Graf propusne zglobnice prikazan je na slici 2.2d. Glavna *prednost* propusne zglobnice je sprječavanje pojave *mrtvih neurona* jer gradijent za $x \leq 0$ nije nula nego neki mali broj α. Tipično $\alpha = 0.01$.



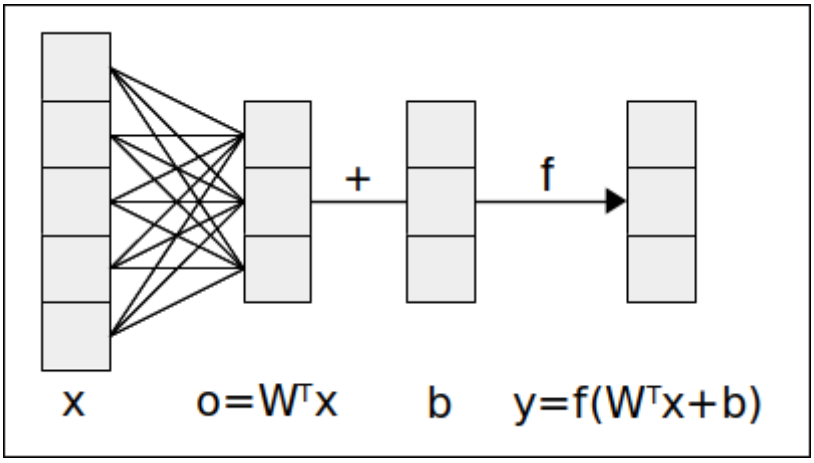
Slika 2.2: Grafovi aktivacijskih funkcija na intervalu $[-5, 5]$
(a) Sigmoida (b) Tangens hiperbolni (c) Zglobnica (d) Propusna zglobnica uz $\alpha = 0.05$

2.2. Potpuno povezani sloj (engl. *Fully Connected Layer*)

Potpuno povezani sloj najjednostavniji je gradivni element CNN-a. On modelira *jednu* nelinearnu transformaciju podataka: potpuno povezano afino preslikavanje i nelinearnu aktivaciju f. Ilustrativni prikaz rada potpuno povezanog sloja vidimo na slici 2.3. Matrica W definirana je kao

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} \tag{2.11}$$

gdje w_{ij} označava težinu kojom se ulaz x_i preslikava u izlaz o_j.



Slika 2.3: Potpuno povezani sloj. Izlaz svakog neurona računa se kao linearna kombinacija svakog od ulaza te se na nju djeluje aktivacijskom funkcijom.

Potpuno povezani sloj modelira funkciju danu s 2.12.

$$y(x; W, b) = f(W^T x + b) \tag{2.12}$$

Modeli koji se sastoji samo od potpuno povezanih slojeva nazivaju se potpuno povezani modeli (engl. *Fully Connected Network, FCN*).

2.3. Konvolucijski sloj

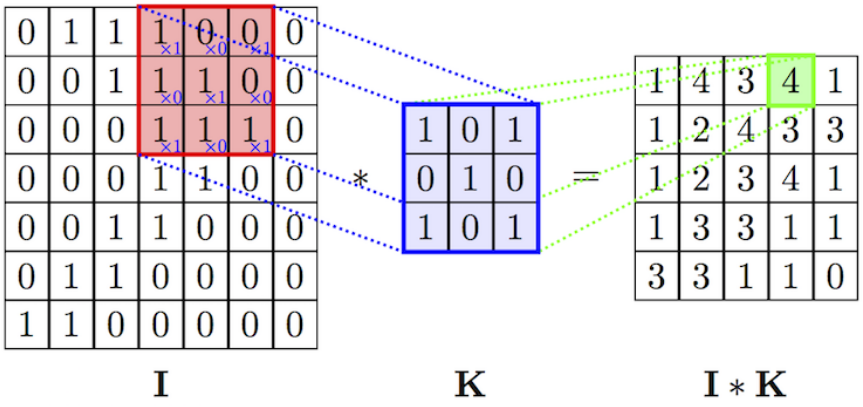
Konvolucijski sloj uvodimo u duboke modele kada trebamo klasificirati objekt koji je kompozicija značajki (npr. osoba ima glavu, glava ima lice, lice ima oči). U tom slučaju potpuno povezani modeli imaju sklonost da nauče šum jer su značajke ovisne o lokalnom susjedstvu, a kod FCN izlaz ovisi o svim ulazima te će FCN učiti *beskorisne* informacije o svim ulazima. Dodatno, FCN u tom slučaju svaku translacija treba posebno učiti jer je translatirana slika potpuno različita od originala za FCN.

Prednosti konvolucijskog sloja u odnosu na potpuno povezani su manji broj parametara (lokalne interakcije te dijeljenje težina) i brža evaluacija

Prinicip rada konvolucije ilustriran je na slici 2.4. Jezgra K (od engl. *kernel*) „klizi" po ulazu I te računa izlaz $I * K$. U strojnom učenju konvolucija je ekvivalentna unakrsnoj korelaciji te kada god kažemo konvolucija, zapravo mislimo unakrsna korelacija. [9]

Rezultat konvolucije naziva se *mapa značajki* i računa se po izrazu 2.13.

$$S(i,j) = (K \star I)(i,j) = \sum_{m=m_{min}}^{m_{max}} \sum_{n=n_{min}}^{n_{max}} K(m,n) \cdot I(i+m, j+n) \tag{2.13}$$



Slika 2.4: Princip rada konvolucije. [16] Jezgra K „klizi" po ulazu te se izlaz računa kao umnožak odgovarajućih elemenata ulaza i jezgre.

Za konvolucijski sloj trebamo definirati sljedeće parametre:

- veličinu jezgre
- dubinu (engl. *depth*) - broj mapi značajki (i jezgri) na izlazu sloja
- korak (engl. *stride*) - za koliko će se elemenata pomaknuti jezgra na ulazu za izračun idućeg elementa izlaza
- nadopunjavanje (engl. *padding*) - širina ruba koji se dodaje ulazu kako ne bi došlo do gubitka informacija

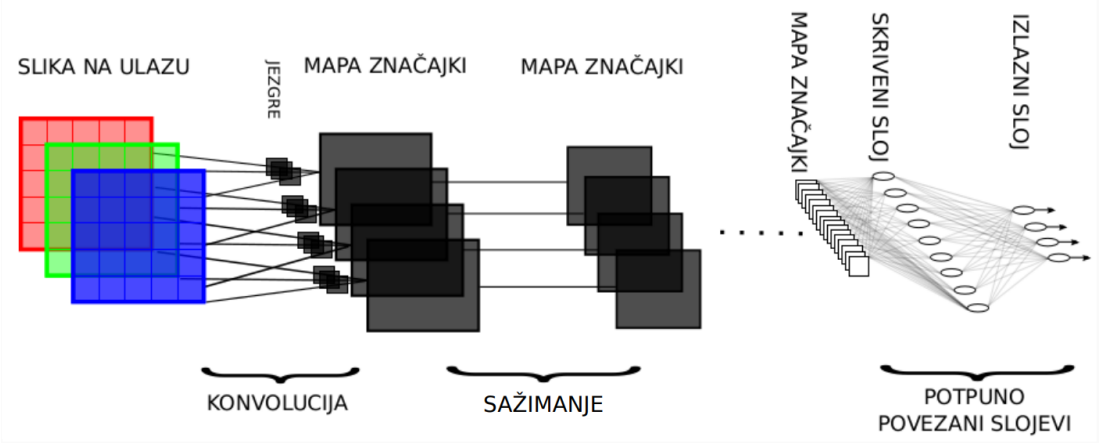
Označimo visinu i širinu ulaznog tenzora I sa H i W, a visinu i širinu izlaznog tenzora $I * K$ sa H' i W'. Visinu i širinu jezgre označimo sa K_h i K_w. Korak po visini i širini označit ćemo sa S_h i S_w. A nadopunjavanje, odnosno širinu ruba, označimo s P. Dimenzije izlaznog tenzora onda računamo po izrazima 2.14 i 2.15.

$$H' = \frac{H + 2 \cdot P - K_h}{S_h} + 1 \tag{2.14}$$

$$W' = \frac{W + 2 \cdot P - K_w}{S_w} + 1 \tag{2.15}$$

Klasična *struktura CNN* prikazana je na slici 2.5. Model se tipično sastoji od nekoliko naizmjeničnih slojeva konvolucije i sažimanja. Oni smanjuju dimenzionalnost značajki do veličine 1x1, a nakon njih dolaze potpuno povezani slojevi koji uče koliko je koja značajka važna za klasifikaciju.

Receptivno polje značajke je skup svih elemenata ulaznog sloja o kojim ta značajka ovisi. Za razliku od FCN, kod kojeg svaka značajka ima jednako veliko receptivno polje, receptivno polje kod CNN raste sa dubinom značajke.



Slika 2.5: Klasična struktura konvolucijskog modela. [9] Izmjenjuju se konvolucijski i slojevi za sažimanje kako bi izdvojili značajke ulaza. Te značajke na kraju prolaze kroz potpuno povezani sloj koji uči koje naučene značajke su bitne za koju klasu.

2.4. Sloj sažimanja (engl. *Pooling Layer*)

Sloj sažimanja obično uvodimo u CNN kada želimo ulazni tenzor preslikati u kategoričku varijablu. Njegova *uloga* je sažimanje prostorno bliskih značajki ulaza u jednu varijablu izlaza.

Ovisno o načinu sažimanja razlikujemo nekoliko tipova sloja sažimanja:

(a) sažimanje maksimalnom vrijednosti

(b) sažimanje srednjom vrijednosti

(c) sažimanje L^2 normom $\quad (\|x\|_2 := \sqrt{x_1^2 + \ldots + x_n^2})$

Sažimanje se obično provodi bez preklapanja ulaza tako da se podijeli u regije od kojih se svaka sažima u jednu značajku. Time se mapa značajki smanjuje k puta, pri čemu je k veličina regije.

2.5. Učenje neuronske mreže

Učenje neuronske mreže ugrubo dijelimo u 2 faze: *fazu učenja* i *fazu evaluacije*.
U fazi učenja model prilagođava svoje parametre na podacima za učenje, dok u fazi evaluacije provjeravamo uspješnost modela na podacima za evaluaciju. Podaci za evaluaciju nisu poznati modelu prilikom faze učenja, te služe za provjeru naučenog modela.

S obzirom na oblik podataka razlikujemo *nadzirano*, *nenadzirano* i *podržano učenje*. Nenadzirano učenje provodimo ako imamo samo podatke bez ciljnih vrijednosti, dok kod nadziranog učenja uz podatke imamo i ciljne vrijednosti. U podržanom učenju je dostupna povratna informacija o kvaliteti naučenog modela. U ovom radu bavit ćemo se nadziranim učenjem – klasifikacijom.

S obzirom na trenutak prilagođavanja parametara, razlikujemo 3 vrste učenja:

1. Pojedinačno (engl. *on-line*) učenje,
2. Učenje s mini-grupama (engl. *mini-batch*) i
3. Grupno (engl. *batch*) učenje.

Kod pojedinačnog učenja model prilagođava svoje parametre nakon svakog predočenog ulaza, kod grupnog učenja nakon svih predočenih ulaza, dok kod učenja s mini-grupama model prilagođava parametre nakon određenog broja ulaznih primjera. Broj primjera je određen unaprijed kao hiperparametar i veći je od 1, a manji od ukupnog broja primjera. U radu koristimo učenje s mini-grupama jer je najisplativije s obzirom na vremensku i memorijsku složenost.

2.5.1. Funkcija gubitka

U svrhu vrednovanja izlaza modela, potrebno je definirati funkciju gubitka.

Funkcija gubitka je mjera koliko je model pogriješio prilikom predviđanja ciljne klase.

Da bi mogli koristiti *Backpropgation* algoritam, potrebno je da funkcija gubitka bude derivabilna, tj. trebamo moći izračunati parcijalne derivacije s obzirom na svaki izlaz modela. Parcijalna derivacija nam govori koliko će se funkcija pogreške promijeniti ako promijenimo ulaz ili odgovarajuću težinu.

Za klasifikaciju se najčešće koristi **unakrsna entropija** (engl. *cross entropy*) ili negativna log izglednost. Za regresiju se koristi suma srednjih kvadratnih odstupanja.

U ovom radu koristimo unakrsnu entropiju, koja je dana izrazom 2.16

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N} log P(Y = y_i|\mathbf{x}_i) \tag{2.16}$$

gdje $P(Y = y_i|\mathbf{x}_i)$ označava vjerojatnost klasifikacije primjera $(\mathbf{x}_i, y_i)$ u točnu ciljnu klasu y_i.

2.5.2. *Backpropagation* algoritam

Algoritam propagacije pogreške unatrag ili *backpropagation* algoritam je postupak učenja (optimizacije parametara) na temelju gradijenta. Algoritam se provodi u 3 koraka:

1. **Prolaz unaprijed** — računa se izlaz modela $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$ i funkcija gubitka $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))$

2. **Prolaz unatrag** — računa se gradijent funkcije gubitka s obzirom na parametre modela $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))$

3. **Optimizacijski postupak** — ažuriraju se parametri modela varijantom gradijentnog spusta $\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))$

Proslijeđivanje greške unatrag provodi se iterativno od izlaznog prema ulaznom sloju. Korištenjem pravila ulančavanja za deriviranje kompozicije funkcija postupak računanja gradijenta postaje jednostavan i računski nezahtjevan. Dovoljno je, za svaki sloj izračunati parcijalnu derivaciju izlaza s obzirom na parametre (ako postoje) te s obzirom na ulaz (ako nismo došli do kraja).

U ovom radu koristi se optimizacijski postupak Adam (skraćeno od engl. *Adaptive Moment Estimation*). Algoritam je opisan u [6], a ovdje ga nećemo navoditi.

2.5.3. Normalizacija

Normalizaciju koristimo da bi poboljšali generalizaciju i ubrzali učenje modela. Cilj normalizacije je transformirati podatke tako da imaju sredinu nula (engl. *zero mean*) i varijancu jedan (engl. *unit variance*).

U ovom radu koristimo dva tipa normalizacije: **Normalizaciju nad cijelim skupom** i **Normalizaciju nad grupom** (engl. *batch-normalization*). Izrazi po kojima se podaci transformiraju su jednaki u oba tipa, samo se razlikuje skup nad kojim se računaju statistički parametri. Sredina μ i varijanca σ za normalizaciju nad cijelim skupom su konstantni, pa ih možemo izračunati unaprijed prije treniranja modela. Sredina cijelog skupa za treniranje za ispitni skup MNIST[11] iznosi $\mu_{\boldsymbol{x}} = 0.1307$, a varijanca $\sigma_{\boldsymbol{x}} = 0.3081$. Za razliku od normalizacije nad cijelim skupom, kod normalizacije nad grupom sredina i varijanca ovise o podacima koji čine grupu $B = \boldsymbol{x}_i,\ \forall i \in \{1..m\}$. U svakoj epohi učenja, skup za učenje se nasumično permutira te se parametri za norma-

lizaciju nad grupom mijenjaju dinamički po izrazima 2.17 i 2.18.

$$\mu_B := \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i \tag{2.17}$$

$$\sigma_B^2 := \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu_B)^2 \tag{2.18}$$

Tada normalizaciju svakog člana grupe računamo po izrazu 2.19, gdje je ϵ mali pozitivan broj i služi da izbjegnemo dijeljenje s nulom.

$$\widehat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.19}$$

***BatchNorm* sloj**

BatchNorm sloj sličan je normalizaciji po grupama, a razlika je u tome da *batchnorm* prati stare procjene sredine β i varijance γ. Izlaz *batchnorm* sloja dan je izrazom 2.20. Inicijalno su $\beta = 0$ i $\gamma = 1$.

$$BN_{\gamma,\beta}(\boldsymbol{x}_i) := \widehat{\boldsymbol{x}}_i \cdot \gamma + \beta \tag{2.20}$$

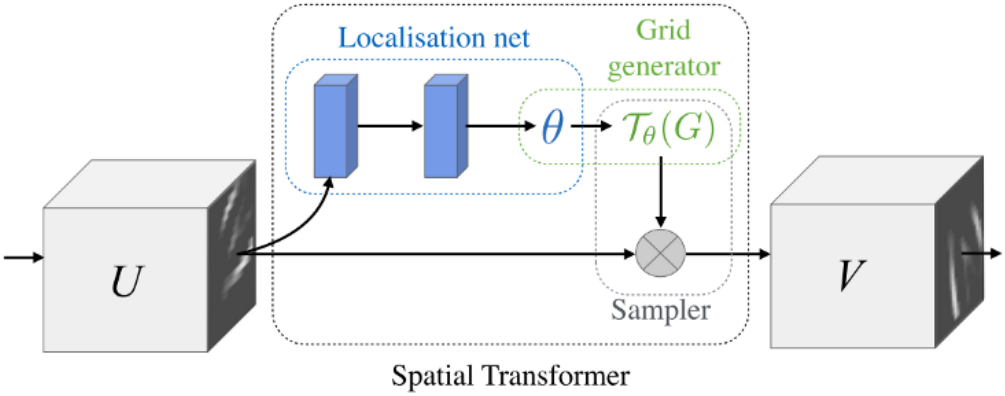
Prednosti batchnorm sloja su:

- izlazi imaju sredinu nula i varijancu jedan, što se dobro slaže sa slučajnom inicijalizacijom sljedećeg sloja
- sljedeća zglobnica u prosjeku propušta 50% ulaza, što raspršuje aktivacije neurona te model brže uči [4]

3. Modul za prostornu transformaciju

Modul za prostornu transformaciju (engl. *Spatial Transformer Network*, kratica STN) povećava prostornu invarijantnost CNN-a. Pogodan je za implementaciju u CNN jer se može dodati po principu „uključi i radi" – bez ikakvog dodatnog nadzora nad učenjem modela ni izmjene postupka učenja. [5]

STN se sastoji od 3 dijela, kako je prikazano na slici 3.1. Ulazna mapa značajki prosljeđuje se lokalizacijskoj mreži koja računa parametre θ. Zatim generator rešetke za uzorkovanje na osnovu tih parametara računa rešetku koju prosljeđuje sloju za uzorkovanje. Sloj za uzorkovanje određuje kamo i s kojim intenzitetom će se neki piksel sa ulaza preslikati u deformirani izlaz.



Slika 3.1: Arhitektura STN-a. [5] *Plavom* bojom je označena lokalizacijska mreža koja iz ulaza računa parametre θ. *Zelenom* bojom označen je generator rešetke koji računa deformaciju ulaza. *Sivom* bojom je označen sloj za uzorkovanje koji interpolira vrijednosti izlaznih elemenata na temelju rešetke $\mathcal{T}_\theta$ i vrijednosti elemenata ulaza U.

3.1. Lokalizacijska mreža

Lokalizacijska mreža prikazana je na slici 3.1 plavom bojom. Njezina uloga u STN je uz dani ulaz generirati parametre θ potrebne za generiranje rešetke za uzorkovanje. Ulaz je mapa značajki $U \in \mathbb{R}^{C \times H \times W}$ sa C kanala, visinom H i širinom W. Veličina tenzora θ varira o tipu transformacije, npr. za afinu transformaciju je veličina 6. Funkciju koju modelira lokalizacijska mreža dana je izrazom 3.1.

$$\theta = f_{loc}(U) \tag{3.1}$$

Funkcija $f_{loc}()$ može biti bilo kojeg oblika, npr. FCN ili CNN. U radu je korištena CNN kao lokalizacijska mreža, a detaljnija arhitektura mreže bit će opisana u poglavlju 5.

3.2. Generator rešetke za uzorkovanje

Generator rešetke za uzorkovanje je dio koji radi deformaciju ulaznog tenzora u modelu. On računa koordinate rešetke koja označava koji pikseli ulaza se preslikavaju na izlaz STN-a. Riječju *piksel* označavamo element generičke mape značajki, ne nužno slikovni element.

Izlaz generatora rešetke je skup $G = \{G_i\}$ piksela $G_i = (x_i^t, y_i^t)$ koji tvore izlaznu mapu značajki $V \in \mathbb{R}^{C \times H' \times W'}$. H' i V' su visina i širina rešetke, a C je broj kanala, koji je jednak za ulaznu i izlaznu mapu značajki.

U idućim potpoglavljima opisat ćemo dvije transformacije implementirane u ovom radu.

3.2.1. Afina transformacija

Afina transformacija u dvije dimenzije definirana je sa 6 parametara i dana je izrazom 3.2

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = A_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{3.2}$$

gdje su (x_i^t, y_i^t) ciljne (engl. *target*) koordinate u izlaznoj mapi značajki, a (x_i^s, y_i^s) izvorne (engl. *source*) koordinate u ulaznoj mapi značajki. Matricu A_θ inicijaliziramo tako da obavlja funkciju identiteta, tj. $\theta_{11} = \theta_{22} = 1$, ostali parametri $= 0$.

Afina transformacija definirana kao u izrazu 3.2 omogućuje da izrežemo, translatiramo, rotiramo, skaliramo i nakosimo ulaznu mapu značajki sa samo 6 parametara.

3.2.2. Thin-plate spline transformacija

Da bi objasnili thin-plate spline transformaciju, najprije moramo objasniti radij-funkcije i thin-plate spline funkciju.

Radij-funkcija

Za funkciju $\varphi : \mathbb{R}_+ \to \mathbb{R}$ kažemo da je radij-funkcija (engl. *radial basis function*, kratica RBF) ako je oblika

$$\varphi(\boldsymbol{x}, \boldsymbol{c}) = \varphi(\|\boldsymbol{x} - \boldsymbol{c}\|) \tag{3.3}$$

gdje kao normu $\|\cdot\|$ najčešće uzimamo 2-normu, tj. euklidsku udaljenost. Drugim riječima, funkcija je radij-funkcija ako joj vrijednost ovisi samo o udaljenosti točke $\boldsymbol{x} \in \mathbb{R}^d$ od točke $\boldsymbol{c} \in \mathbb{R}^d$, koja se naziva *centar*.

Neke od češće korištenih radij-funkcija možemo vidjeti u tablici 3.1. Označimo $r = \|\boldsymbol{x} - \boldsymbol{c}\|$.

Tablica 3.1: Neke od češće korištenih radij-funkcija [13]

Radij-funkcija	$\varphi(r)$	uvjeti za β
Gaussova	e^{-r^2}	–
Multikvadratna	$(-1)^{\lceil \beta/2 \rceil}(1 + r^2)^{\beta/2}$	$\beta > 0, \beta \notin 2\mathbb{N}$
Inverzna multikvadratna	$(1 + r^2)^{\beta/2}$	$\beta < 0$
Poliharmonična spline	$(-1)^{\lceil \beta/2 \rceil} r^\beta$	$\beta > 0, \beta \notin 2\mathbb{N}$
	$(-1)^{1+\beta/2} r^\beta \log(r)$	$\beta > 0, \beta \in 2\mathbb{N}$

U ovom radu koristit ćemo poliharmoničnu spline funkciju sa parametrom $\beta = 2$. Ona se još naziva i thin-plate spline funkcija. Kada uvrstimo $\beta = 2$ u izraz iz gornje tablice, dobivamo izraz za thin-plate spline funkciju

$$\varphi_{tps}(r) = r^2 \log(r) \tag{3.4}$$

Thin-plate spline transformacija

Thin-plate spline minimizira energiju prilikom savijanja tanke metalne ploče. Zamislimo beskonačnu tanku metalnu ploču i na njoj četiri točke, koje tvore kvadrat. Povučemo li dvije točke koje su na dijagonali u smjeru normale ravnine, a ostale dvije u suprotnom smjeru, ploča će se saviti tako da ima minimalnu energiju.

U [13] pokazano je da funkcija koja minimizira energiju savijanja ima oblik

$$s(x,y) = a_1 + a_2 x + a_3 y + \sum_{i=1}^{n_c} w_i \varphi_{tps}(\|(x,y) - (x_{c_i}, y_{c_i})\|) \tag{3.5}$$

te mora zadovoljavati sljedeće uvjete

$$\sum_{i=1}^{n_c} w_i = \sum_{i=1}^{n_c} w_i x_i = \sum_{i=1}^{n_c} w_i y_i = 0 \tag{3.6}$$

Parametri u izrazu 3.5 koje model prilagođava prilikom učenja su $a_{1..3}$ i $w_{1..n_c}$.

Thin-plate spline transformacija definirana je sa $n_c + 3$ parametra, a broj ovisi o broju izabranih kontrolnih točaka n_c. Zbog uvjeta u izrazu 3.6 jedna težina w_i se ne može slobodno mijenjati pa u implementaciji lokalizacijska mreža generira $n_c + 2$ parametra.

3.3. Sloj za uzorkovanje

Da bi napravio prostornu transformaciju značajki, sloju za uzorkovanje trebamo predati rešetku $\mathcal{T}_\theta$ zajedno sa ulaznom mapom značajki U, kako bi dobili izlaznu mapu značajki V.

Kako bi generirao mapu značajki, sloj za uzorkovanje uzima koordinate (x_i^s, y_i^s) iz rešetke $\mathcal{T}_\theta$ te na tim koordinatama provodi interpolaciju ulazne mape značajki kako bi odredio vrijednost (intenzitet) piksela na izlazu.

U radu koristimo *bilinearnu interpolaciju*. Izlaznu mapu značajki računamo pomoću formule 3.7 za svaku poziciju $i \in \{1 \dots H'W'\}$ i svaki kanal $c \in \{1 \dots C\}$.

$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \tag{3.7}$$

Važno je napomenuti da je bilinearna interpolacija derivabilna s obzirom na ulaz i parametre. Parcijalna derivacija s obzirom na ulaznu mapu značajki dana je izrazom 3.8, a parcijalna derivacija s obzirom na rešetku G izrazom 3.9 za $\dfrac{\partial V_i^c}{\partial x_i^s}$ te slično za $\dfrac{\partial V_i^c}{\partial y_i^s}$.

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_{n=1}^{H} \sum_{m=1}^{W} \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \tag{3.8}$$

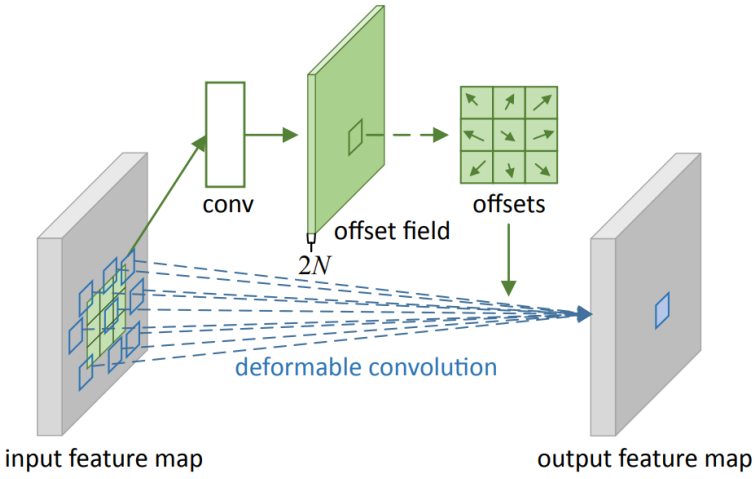
$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } w < x_i^s \end{cases} \tag{3.9}$$

3.4. Moderniji pristup modeliranju prostornih transformacija

Umjesto STN-a za prostornu transformaciju mogu se koristiti **deformabilne konvolucije**, kao u [18, 1]. Ključna razlika STN-a i deformabilnih konvolucija je doseg transformacije. STN obavlja globalnu prostornu transformaciju, dok deformabilne konvolucije obavljaju lokalnu.

Razliku u izvedbi između klasične i deformabilne konvolucije možemo vidjeti na slici 3.2. Klasična konvolucija za izračun koristi zeleni dio ulazne mape značajki. Deformabilna konvolucija dodaje još jednu paralelnu granu između ulaza i izlaza koja uči pomake. Izračun konvolucije se zatim provodi nad pomaknutim lokacijama ulazne mape značajki, označenim plavom bojom. Pomaknute koordinate najčešće nisu cjelobrojne, pa u tom slučaju provodimo bilinearnu interpolaciju.

Deformabilne konvolucije pokazale su se kao vrlo efikasan alat za modeliranje prostornih transformacija [14, 15].

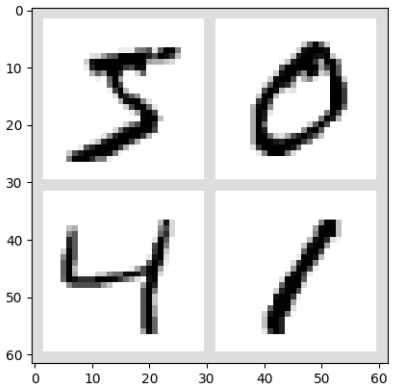


Slika 3.2: Deformabilna konvolucija [1]

4. Ispitni skup - MNIST

MNIST[11] je skup rukom pisanih znamenki. Sastoji se od 60 000 primjera u skupu za učenje i 10 000 primjera u skupu za testiranje. Sve slike su skalirane na veličinu 28x28 piksela i centrirane. Skup se sastoji od jednokanalnih (crno-bijelih) slika, a neke od njih možemo vidjeti na slici 4.1.

MNIST skup je često korišten u radovima iz područja računalnog vida te klasifikacija na ovom skupu služi kao „*hello world* program" za testiranje arhitektura neuronskih mreža.



Slika 4.1: Primjeri rukom pisanih brojeva iz MNIST skupa za treniranje

5. Programska izvedba

U ovom radu za implementaciju neuronskih mreža odabrali smo *Python*[1] verziju 3.8 i programski okvir *PyTorch*[2] verziju 1.4. Izbor se temeljio na lakoj i brzoj izradi modela koju PyTorch nudi, a opisana je kasnije u poglavlju.

5.1. Programski okvir PyTorch

PyTorch je programski okvir za automatsku diferencijaciju napisan za programski jezik Python. PyTorch je programski okvir otvorenog koda (engl. *open source*), a dizajnirao ga je i napravio Facebook.
Dvije najvažnije značajke Pytorch-a su:

- Računanje s višedimenzionalnim tenzorima (slično kao NumPy[3]) uz mogućnost izvršavanja operacija na GPU, što mu omogućava veće brzine izvođenja od paketa NumPy
- Mogućnost stvaranja i treniranja dubokih neuronskih mreža uz veliku fleksibilnost i brzinu, najviše zbog *autograd* paketa koji dinamički stvara graf izvođenja operacija te omogućava lako računanje gradijenata

Zbog *autograd* paketa, kada stvaramo model u PyTorch-u dovoljno je samo definirati slojeve (i po potrebi dodatne parametre) te kako će se izvršiti prolaz unaprijed. PyTorch na osnovu prolaza unaprijed računa gradijente parametara i stvara metodu za prolaz unatrag.

5.2. Struktura

Struktura programske podrške organizirana je u sljedeće direktorije:

[1] `https://www.python.org/`
[2] `https://pytorch.org/`
[3] `https://numpy.org/`

- **data** - podaci o skupu MNIST u IDX formatu.
- **runs** - dijagnostički podaci prilikom treniranja i evaluacije modela spremljeni pomoću torch.utils.tensorboard.SummaryWriter[4] klase
- **tests** - testovi za provjeru implementacije TPS transformacije i STN modula.
- **trained_nets** - parametri naučenih modela spremljeni u pkl formatu (spremanje se odvijalo nakon određenog broja epoha pa imamo spremljene i modele u međukoracima).

Ostale Python skripte nalaze se u korijenskom direktoriju, a to su:

- **mnist_classifier.py** - konvolucijski model bez STN-a
- **stn_mnist.py** - konvolucijski model sa STN-om
- **stn_module.py** - klasa modula STN
- **thinplate.py** - funkcije za thin-plate spline transformaciju
- **helpers.py** - pomoćne funkcije
- **experiments.py** - skripta za provođenje jednog eksperimenta
- **runner.py** - skripta za automatizirano pokretanje svih eksperimenata

5.3. Primjer izrade neuronske mreže

U ovom potpoglavlju dat ćemo primjer kako definirati arhitekturu neuronske mreže u PyTorch-u te kako trenirati definirani model.

Primjer učitavanja paketa možemo vidjeti u kodu 5.1. Za izradu modela trebaju nam sljedeći paketi:

- `torch` - korijenski paket PyTorch-a
- `torch.nn` - klase koje definiraju module i slojeve neuronskih mreža
- `torch.nn.functional` - funkcije koje koriste modeli, ali koje ne uče parametre
- `torch.optim` - klase za optimizaciju parametara modela
- `torch.utils.data` - za definiranje toka za učitavanje podataka
- `torchvision` - u njemu su definirane klase za skupove podataka (npr. MNIST, CIFAR10)

[4] `https://pytorch.org/docs/stable/tensorboard.html`

– `torchvision.transforms` - za definiranje transformacije skupa prilikom učitavanja podataka (npr. definiranje normalizacije na cijelom skupu)

Programski kod 5.1: Učitavanje PyTorch paketa

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import torch.optim as optim
5 import torch.utils.data
6 from torch.utils.tensorboard import SummaryWriter
7
8 import torchvision
9 import torchvision.transforms as transforms
```

Nakon učitavanja paketa moramo napraviti klasu koja definira arhitekturu i ponašanje našeg modela. Konstruktor klase dan je u kodu 5.2, a prolaz unaprijed definiran je u kodu 5.3. Klasa `nn.Module` je generička klasa i svaki model koji definiramo u PyTorch-u mora ju nasljeđivati. U konstruktoru stvaramo klase slojeva koje želimo koristiti u našem modelu i spremamo ih kao atribute instance razreda. Spremamo samo one slojeve koji trebaju učiti parametre (npr. konvolucijski i potpuno povezani sloj), a slojeve koji ne uče (npr. sloj sažimanja) ne spremamo.

Programski kod 5.2: Konstruktor modela bez STN-a

```
1  class CNN(nn.Module):
2    def __init__(self):
3      super(CNN, self).__init__()
4
5      self.conv1 = nn.Conv2d(1, 32, 5, padding=2, bias=False)
6      self.conv1_bn = nn.BatchNorm2d(32)
7      self.conv2 = nn.Conv2d(32, 64, 3, bias=False)
8      self.conv2_bn = nn.BatchNorm2d(64)
9
10     self.fc1 = nn.Linear(6 * 6 * 64, 120)
11     self.fc1_bn = nn.BatchNorm1d(120)
12     self.fc2 = nn.Linear(120, 80)
13     self.fc2_bn = nn.BatchNorm1d(80)
14     self.fc3 = nn.Linear(80, 10)
```

U klasi modela, osim konstruktora, moramo definirati i metodu `forward(self, x):` koja modelira prolaz unaprijed. Parametar `x` je ulazni 4D tenzor koji predajemo modelu. U linijama 4 i 6, koda 5.3 izvodimo slijedne operacije grupirane po klasičnom rasporedu slojeva u CNN. Prvo radimo konvoluciju, zatim normalizaciju po grupama, pa izlaz predajemo zglobnici i na kraju sažimamo s veličinom jezgre 2x2. U liniji 8 izravnamo međurezultat da ga možemo predati potpuno povezanom sloju. Zatim u linijama 10 i 11 vidimo kako se modelira potpuno povezani sloj. Prvo se radi težinska suma značajki, zatim normalizira te se na kraju djeluje sa zglobnicom. Izlaz zadnjeg sloja, ujedno i izlaz modela je samo linearna kombinacija prethodnog sloja te nema nikakvu aktivaciju.

Programski kod 5.3: Prolaz unaprijed za model bez STN-a

```
1   def forward(self, x):
2       # input 32x32x1 (2 padded 28x28 images 1-channel) ->
3       # 28x28x32 (Conv1) -> 14x14x32 (Max pool (2,2))
4       x = F.max_pool2d(F.relu(self.conv1_bn(self.conv1(x))), (2, 2))
5       # 14x14x32 -> 12x12x64 (Conv2) -> 6x6x64 (Max pool (2,2))
6       x = F.max_pool2d(F.relu(self.conv2_bn(self.conv2(x))), (2, 2))
7       # Flatten features
8       x = x.view(-1, 6 * 6 * 64)
9       # Fully connected layer
10      x = F.relu(self.fc1_bn(self.fc1(x)))
11      x = F.relu(self.fc2_bn(self.fc2(x)))
12      x = self.fc3(x)
13      return x
```

Jednostavan primjer treniranja modela prikazan je u kodu 5.4. Linije 1-5 redom definiraju: *device* označava na kojem će se uređaju pokretati model (GPU ili CPU), *net* instancira klasu `CNN` u kojoj smo definirali model, *cross_entropy_loss* je funkcija gubitka (koristimo unakrsnu entropiju), a *optimizer* je optimizator parametara modela (koristimo Adam). U liniji 7 definiramo transformacije koje radimo pri učitavanju primjera iz skupova za učenje i evaluaciju. Ovdje radimo normalizaciju nad cijelim skupom. Linije 9-12 služe za učitavanje train i test skupa, te definiraju tokove podataka u kojima je veličina mini-grupe 128. Linija 14 postavlja mrežu u način za učenje. Linije 18-22 pokazuju kako se izvodi jedna iteracija prilikom učenja. Prvo moramo optimizatoru postaviti gradijente na 0, zatim dovedemo ulaze na mrežu te spremimo izlaze koje usporedimo sa točnim oznakama klasa i izračunamo gubitak. Zatim od

gubitka pokrećemo prolaz unatrag da bi na kraju optimizator ažurirao parametre na osnovu izračunatih gradijenata u prolazu unatrag.

Programski kod 5.4: Treniranje modela

```
1  device = torch.device("cuda" if torch.cuda.is_available() else
       "cpu")
2  net = CNN()
3  net.to(device)
4  cross_entropy_loss = nn.CrossEntropyLoss()
5  optimizer = optim.Adam(net.parameters())
6
7  transform = transforms.Compose([ transforms.ToTensor(),
       transforms.Normalize((0.1307,), (0.3081,)) ])
8
9  trainset = torchvision.datasets.MNIST(root='./data', train=
       True, download=True, transform=transform)
10 trainloader = torch.utils.data.DataLoader(trainset, batch_size
       =128, shuffle=True)
11 testset = torchvision.datasets.MNIST(root='./data', train=
       False, download=True, transform=transform)
12 testloader = torch.utils.data.DataLoader(testset, batch_size
       =128, shuffle=False)
13
14 net.train()
15 for epoch in range(nepoch):
16   for i, data in enumerate(trainloader, 0):
17     inputs, labels = data[0].to(device), data[1].to(device)
18     optimizer.zero_grad()
19     outputs = net(inputs)
20     loss = cross_entropy_loss(outputs, labels)
21     loss.backward()
22     optimizer.step()
```

6. Eksperimentalni rezultati

6.1. Rezultati bez STN

6.1.1. Odabrana arhitektura

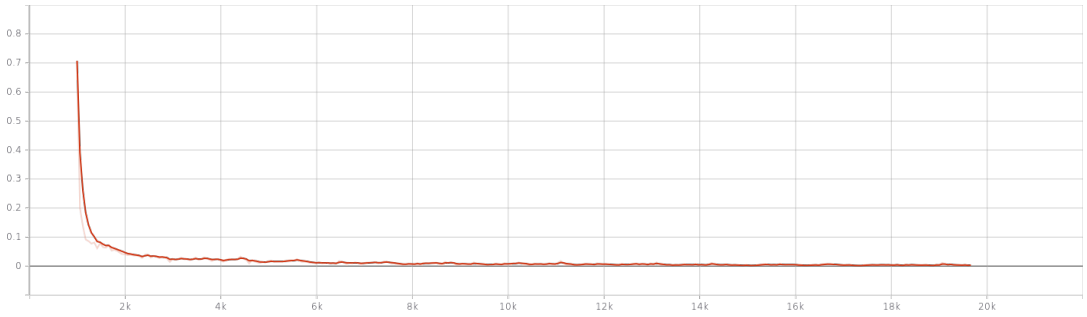
Kao osnovicu za usporedbu, konstruirali smo model bez modula za prostornu transformaciju. On ima sljedeću arhitekturu:

1. **Ulazni sloj** - slika dimenzija 28x28x1 - crno-bijele slike visine i širine 28 piksela
2. **Konvolucijski sloj** - 32 jezgre veličine 5x5, pomak je 1, a nadopunjavanje 2. Nakon slijedi normalizacija po grupama pa $ReLU$. Izlaz: 32 mape značajki veličine 28x28
3. **Sloj sažimanja** - veličina jezgre 2x2. Izlaz: 32 mape značajki veličine 14x14
4. **Konvolucijski sloj** - 64 jezgre veličine 3x3, pomak je 1, a nadopunjavanje 0. Nakon slijedi normalizacija po grupama pa $ReLU$. Izlaz: 64 mape značajki veličine 12x12
5. **Sloj sažimanja** - veličina jezgre 2x2. Izlaz: 64 mape značajki veličine 6x6
6. **Potpuno povezani sloj** - Ulaz: Mapa značajki veličine 64x6x6 koja se poreda u 2304 slijednih neurona. Izlaz: 120 neurona Aktivacijska funkcija je $ReLU$.
7. **Potpuno povezani sloj** - 80 neurona, aktivacijska funkcija je $ReLU$
8. **Potpuno povezani sloj** - 10 neurona

6.1.2. Rezultati

Na slici 6.1 vidimo gubitak unakrsne entropije prilikom učenja modela bez STN-a. Vidimo nagli pad u prve dvije epohe te kasnije lagani pad. Model je treniran u 20 epoha

sa stopom učenja 0.001. Svakih 5 epoha spremali smo trenutno stanje parametara modela. Najveću točnost koju postiže model na originalnom skupu za testiranje je 99.38%. Točnost modela računa se kao $\frac{\text{broj ispravno klasficiranih primjera}}{\text{ukupan broj primjera}}$.



Slika 6.1: Gubitak unakrsne entropije prilikom učenja.

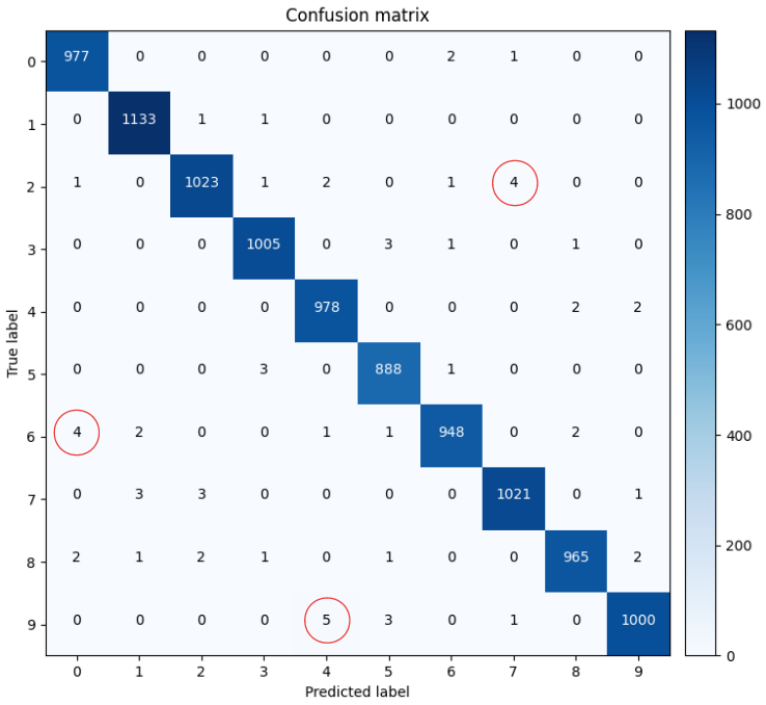
U tablici 6.1 možemo vidjeti matricu zabune za osnovni model. Redak matrice predstavlja točnu oznaku klase, a stupac predstavlja oznaku koju predviđa model. Ukupan broj ispitnih primjera je 10 000, a najčešće 3 greške modela su:

– broj 9 klasificiran kao 4 (5 krivih klasifikacija),

– broj 6 klasificiran kao 0 (4 krive klasifikacije) i

– broj 2 klasificiran kao 7 (4 krive klasifikacije).

Tablica 6.1: Matrica zabune za model bez STN-a.
Redak predstavlja točnu oznaku, a stupac oznaku koju predviđa model.



6.2. Rezultati sa STN

6.2.1. Odabrana arhitektura

Kao klasifikator za modele sa STN-om, korištena je ista arhitektura kao ona navedena u poglavlju 6.1.1. Arhitektura lokalizacijske mreže STN-a je sljedeća:

1. **Ulazni sloj** - slika dimenzija 28x28x1 - crno-bijele slike visine i širine 28 piksela
2. **Konvolucijski sloj** - 32 jezgre veličine 3x3, pomak i nadopunjavanje su 1. Nakon slijedi normalizacija po grupama pa $ReLU$. Izlaz: 32 mape značajki veličine 28x28
3. **Sloj sažimanja** - veličina jezgre 2x2. Izlaz: 32 mape značajki veličine 14x14
4. **Konvolucijski sloj** - 64 jezgre veličine 3x3, pomak i nadopunjavanje su 1. Nakon slijedi normalizacija po grupama pa $ReLU$. Izlaz: 64 mape značajki veličine 14x14
5. **Sloj sažimanja** - veličina jezgre 2x2. Izlaz: 64 mape značajki veličine 7x7
6. **Konvolucijski sloj** - 128 jezgri veličine 3x3, pomak i nadopunjavanje su 1. Nakon slijedi normalizacija po grupama pa $ReLU$. Izlaz: 128 mapi značajki veličine 7x7
7. **Adaptivni sloj sažimanja** - Izlaz: 64 mape značajki veličine 5x5
8. **Potpuno povezani sloj** - Ulaz: Mapa značajki veličine 128x5x5 koja se poreda u 3200 slijednih neurona. Izlaz: 100 neurona Aktivacijska funkcija je $ReLU$.
9. **Potpuno povezani sloj** - Aktivacijska funkcija je $\tanh$. Broj izlaznih neurona ovisi o vrsti transformacije:
   - **Afina transformacija** - 6 neurona
   - **Thin-plate spline transformacija** - $n_c + 2$ neurona gdje je n_c kvadrat veličine rešetke (npr. za veličinu rešetke 6, $n_c = 36$)

Sloj za uzorkovanje nakon provedene transformacije, iz ulazne slike i rešetke interpolira 28x28 piksela koji se dovode na ulaz klasifikatora definiranog u poglavlju 6.1.1.

6.2.2. Eksperimenti

Modele sa STN-om trenirali smo u 300 epoha te smo svakih 50 epoha spremali trenutne parametre modela. Stopa učenja modela sa STN-om je također 0.001 kao i bez STN-a. Modele smo trenirali na originalnom skupu za učenje, uz normalizaciju nad cijelim skupom. Prilikom evaluacije modela, slike iz skupa za evaluaciju smo slučajno pomaknuli i normalizirali. Slučajan pomak slike se sastoji od:

- Slučajne rotacije i
- Slučajne translacije.

Slučajna rotacija provodi se s parametrom α koji označava veličinu kuta (u stupnjevima) za koji se slika može maksimalno rotirati. Prilikom rotiranja slike, kut za koji se slika rotira odabire se nasumično iz intervala $\langle -\alpha, +\alpha \rangle$.

Slučajna translacija provodi se s parametrom t koji označava postotak visine i širine za koji se slika može translatirati. Prilikom određivanja translacije, pomaci dx i dy slike uzimaju se nasumično iz intervala $-W \cdot t < dx < W \cdot t$, odnosno $-H \cdot t < dy < H \cdot t$, gdje W označava širinu, a H visinu slike. Npr. za $t = 0.1$ i ulaznu sliku veličine 28x28, parametri dx i dy uzimaju se iz intervala $\langle -2.8, 2.8 \rangle$.

Prilikom slučajnog pomaka, dijelovi izlazne slike koji ispadaju iz granica ulazne slike, tj. za koje ne postoji odgovarajući ulazni piksel na temelju kojeg se može interpolirati vrijednost, popunjavali smo konstantom 0.

6.2.3. Rezultati

U tablici 6.2 prikazane su točnosti modela za provedene eksperimente iz prethodnog potpoglavlja. Modele smo ispitali za kombinacije parametara $\alpha \in \{0, 30, 45, 60, 90\}$ i $t \in \{0.0, 0.1, 0.2, 0.3\}$. Za svaki par (α, t), vrijednost prikazana u tablici je maksimalna točnost modela koji je evaluiran u svim kontrolnim točkama. U modelima sa STN-om kontrolna točka je svakih 5 epoha, a bez STN-a je svakih 50 epoha.

Rezultati s $\alpha = 90$ su generalno lošiji od ostalih. Također, vidimo da su uz $t = 0.3$ rezultati za model bez STN-a dosta lošiji od ostalih pomaka (~6%), dok su za modele sa STN-om nešto manje lošiji (~2%).

Eksperiment modela sa TPS transformacijom, veličine rešetke 10x10, nije uspio. Prilikom treniranja došlo je do velikog skoka (jednog reda veličine) na grafu gubitka oko 140-te epohe. Razlog skoka je, po našoj pretpostavci, velika stopa učenja. Zbog dugotrajnog procesa treniranja (oko 4 sata po modelu) i manjka vremena, pokus nismo ponavljali.

U tablici 6.2 također vidimo da modeli sa STN-om klasificiraju bolje od modela bez STN-a, što je tvrdnja koju smo htjeli pokazati u ovom radu. Između modela sa STN-om, najbolji se pokazao model sa afinom transformacijom, a zatim TPS modeli sa veličinom rešetke 6x6 pa 8x8. Očekivali smo da će TPS modeli sa većom rešetkom biti ekspresivniji od onih s manjom, međutim pokazalo se obratno. Razlog takvog ponašanja mogao bi biti da modeli s većom rešetkom bolje deformiraju ulaz pa si mogu „namjestiti" koji broj žele, npr. od broja 4 napraviti broj 9.

Najveću točnost od svih modela postiže model sa afinom transformacijom uz $(\alpha, t) = (0,\ 0.0)$ i iznosi 99.41%.

Tablica 6.2: Rezultati svih modela za različite slučajne rotacije i translacije ulaza.

		bez STN-a		STN tps			STN afina
Rot [°]	Trans [%]		$n_c =$	6x6	8x8	10x10	
0	0.0	99.38		99.31	99.37	97.89	**99.41**
	0.1	98.86		99.32	99.31	97.81	99.40
	0.2	98.74		99.22	99.19	97.68	99.31
	0.3	92.75		97.49	97.10	92.85	97.92
30	0.0	98.67		99.23	99.23	97.44	99.31
	0.1	98.60		99.29	99.24	97.35	99.29
	0.2	98.44		99.18	99.11	97.11	99.26
	0.3	92.25		97.51	96.79	92.31	97.88
45	0.0	98.68		99.15	99.20	97.54	99.28
	0.1	98.75		99.14	99.15	97.49	99.30
	0.2	98.46		99.18	99.13	97.32	99.27
	0.3	92.64		97.49	97.00	92.02	97.98
60	0.0	98.47		99.11	99.13	97.24	99.26
	0.1	98.32		99.16	99.05	97.29	99.26
	0.2	98.21		99.24	99.18	97.27	99.24
	0.3	91.64		97.61	96.78	91.83	97.82
90	0.0	94.51		96.46	96.49	92.46	96.52
	0.1	94.14		96.55	96.36	92.07	96.70
	0.2	93.56		96.47	96.21	91.53	96.54
	0.3	85.70		93.82	92.25	85.81	94.49

7. Zaključak

Klasifikacija prirodnih slika je važan problem računalnog vida. Na slikama stvarnog svijeta nikad nećemo dobiti da se objekt, koji trebamo klasificirati, nalazi savršeno u sredini slike i da je dobro orijentiran. Ovaj rad usredotočuje se na nadzirani pristup za strojno prepoznavanje znamenki.

Dali smo opis umjetnih neurona, slojeva od kojih se sastoje konvolucijski modeli i dodatnog modula za prostornu transformaciju. Rad modula za prostornu transformaciju direktno ovisi o odabiru transformacije za generator rešetke. U ovom radu koristili smo afinu i thin-plate spline transformaciju – sa veličinama rešetke 6x6, 8x8 i 10x10.

Pokazali smo da STN poboljšava uspješnost klasifikacije jer povećava prostornu invarijantnost modela. Velika prednost STN-a je laka i jeftina ugradnja u modele – po principu „uključi-i-radi".

Najveću točnost (99.41%) postiže model sa afinom transformacijom. Prema [7] trenutno najbolja točnost na skupu MNIST je 99.82%, a naši rezultati nisu daleko od toga.

Rezultati pokazuju da model s afinom transformacijom postiže bolju točnost od modela s TPS transformacijom. Očekivali smo povećanje točnosti s povećanjem veličine rešetke za modele s TPS-om, međutim rezultati pokazuju obratno.

Eksperiment modela s TPS-om veličine rešetke 10x10 nije uspio, pretpostavljamo zbog velike stope učenja.

U eventualnom daljnjem radu, trebalo bi ponoviti eksperimente s TPS-om veličine rešetke 10x10 te isprobati druge veličine rešetke. Preporučeno je ubrzati izračun TPS transformacije tako da se ne računa svaki put kvadrat razlike svih točaka rešetke, kao što je napravljeno u [17].

LITERATURA

[1] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, i Yichen Wei. Deformable convolutional networks. U *Proceedings of the IEEE international conference on computer vision*, stranice 764–773, 2017.

[2] Bojana Dalbelo Bašić, Marko Čupić, i Jan Šnajder. Slajdovi „Umjetne neuronske mreže" iz kolegija *Umjetna inteligencija*. URL `https://www.fer.unizg.hr/_download/repository/UI_12_UmjetneNeuronskeMreze[1].pdf`. [datum pristupa 7. Lipanj 2020.].

[3] Christoph Heindl. py-thin-plate-spline. GitHub repository, 2018. URL `https://github.com/cheind/py-thin-plate-spline`. [datum pristupa 18. Svibanj 2020.].

[4] Sergey Ioffe i Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] Max Jaderberg, Karen Simonyan, Andrew Zisserman, i koray kavukcuoglu. Spatial transformer networks. U C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, i R. Garnett, urednici, *Advances in Neural Information Processing Systems 28*, stranice 2017–2025. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf`.

[6] Diederik P Kingma i J Adam Ba. A method for stochastic optimization. arxiv 2014. *arXiv preprint arXiv:1412.6980*, 2019.

[7] Kamran Kowsari, Mojtaba Heidarysafa, Donald E Brown, Kiana Jafari Meimandi, i Laura E Barnes. Rmdl: Random multimodel deep learning for classification. U *Proceedings of the 2nd International Conference on Information System and Data Mining*, stranice 19–28, 2018.

[8] Josip Krapac i Siniša Šegvić. Slajdovi „Duboke unaprijedne mreže" iz kolegija *Duboko učenje*, . URL http://www.zemris.fer.hr/~ssegvic/du/du1feedforward.pdf. [datum pristupa 8. Lipanj 2020.].

[9] Josip Krapac i Siniša Šegvić. Slajdovi „Konvolucijski modeli" iz kolegija *Duboko učenje*, . URL http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf. [datum pristupa 9. Lipanj 2020.].

[10] Alex Krizhevsky, Ilya Sutskever, i Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. U F. Pereira, C. J. C. Burges, L. Bottou, i K. Q. Weinberger, urednici, *Advances in Neural Information Processing Systems 25*, stranice 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] Yann LeCun, Corinna Cortes, i Chris Burges. The MNIST Database. URL http://yann.lecun.com/exdb/mnist/. [datum pristupa 18. Svibanj 2020.].

[12] Andrew L Maas, Awni Y Hannun, i Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. U *Proc. icml*, svezak 30, stranica 3, 2013.

[13] Petar Palašek. Visual tracking of soft tissue targets in sequences of 3d ultrasound images. Diplomski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb/Rennes, 2012. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/palasek12ms.pdf.

[14] Josip Šarić, Marin Oršić, Tonći Antunović, Sacha Vražić, i Siniša Šegvić. Single level feature-to-feature forecasting with deformable convolutions. U *German Conference on Pattern Recognition*, stranice 189–202, 2019.

[15] Josip Saric, Marin Orsic, Tonci Antunovic, Sacha Vrazic, i Sinisa Segvic. Warp to the future: Joint forecasting of features and feature motion. U *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[16] Anh Vo. Deep learning – computer vision and convolutional neural networks, Veljača 2018. URL https://anhvnn.wordpress.com/2018/02/01/deep-learning-computer-vision-and-convolutional-neural-networks/. [datum pristupa 8. Lipanj 2020.].

[17] WarBean. tps_stn_pytorch. GitHub repository, 2017–2018. URL https://github.com/WarBean/tps_stn_pytorch. [datum pristupa 18. Svibanj 2020.].

[18] Xizhou Zhu, Han Hu, Stephen Lin, i Jifeng Dai. Deformable convnets V2: more deformable, better results. U *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, stranice 9308–9316. Computer Vision Foundation / IEEE, 2019.

[19] Ivan Šego. Klasifikacija slika kućnih brojeva dubokim konvolucijskim modelima. Završni rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2018. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/sego18bs.pdf.

SKRAĆENICE

CNN (engl. *Convolutional Neural Network*) – konvolucijska neuronska mreža

FCN (engl. *Fully conncted network*) – potpuno povezani model

LReLU (engl. *Leaky Rectified Linear Unit*) – propusna zglobnica

ReLU (engl. *Rectified Linear Unit*) – zglobnica

STN (engl. *Spatial Transformer Network*) – modul za prostornu transformaciju

TPS – Thin-plate spline

POPIS SLIKA

POPIS TABLICA

Diferencijalne deformacije u modelima za klasifikaciju slike

Sažetak

Rad daje uvod u umjetne neuronske mreže s naglaskom na konvolucijskim modelima. Ukratko je opisano učenje neuronske mreže. U radu uvodimo dodatni modul za prostornu transformaciju u konvolucijski model, kako bi poboljšali prostornu invarijantnost modela. Opisujemo arhitekturu dodatnog modula te dvije implementirane deformacije – afinu transformaciju i thin-plate spline transformaciju. Dana je programska implementacija modela te metode za učenje i evaluaciju istih. Modeli su trenirani na skupu MNIST. Prikazani su usporedni rezultati dvaju modela s modulom za prostornu transformaciju i modela bez dodatnog modula.

Ključne riječi: Strojno učenje, duboko učenje, duboke neuronske mreže, konvolucijske neuronske mreže, klasifikacija, modul za prostornu transformaciju, afina transformacija, thin-plate spline transformacija, MNIST

Differentiable deformations for image classification models

Abstract

This paper gives an introduction to artificial neural networks with emphasis on convolutional models. We described briefly learning process of neural network. In this work we introduce additional spatial transformer module in convolutional model to increase spatial invariance. We describe the architecture of additional module and two implemented deformations – affine transformation and thin-plate spline transformation. This paper gives code of model in PyTorch and methods for training and testing models. We trained models on MNIST dataset. We gave comparative results of two models with spatial transformations and third model without STN.

Keywords: Machine Learning, Deep Learning, Deep Neural Networks, Convolutional Neural Network, clasification, Spatial Transformer Network, Affine transformation, Thin-Plate Spline transformation, MNIST