SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3944

Duboke neuronske arhitekture za lokalizaciju objekata

Matija Folnović

Zagreb, lipanj 2015.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću i asistentima na pomoći pri izradi ovog rada. Zahvaljujem i obitelji i prijateljima na podršci kroz cijeli studij.

Sadržaj

1. Uvod

Područje računalnog vida (engl. *Computer Vision*) kao dio područja umjetne inteligencije (engl. *Artificial Intelligence*) bavi se dohvaćanjem, obrađivanjem, analiziranjem i razumijevanjem slika. Izazov računalnog vida je približiti se ljudskom vidu, što je i danas daleko od riješenog.

Dugo kroz povijest umjetne inteligencije problemi su se pokušavali riješiti ručnim modeliranjem algoritama specifičnim za problem. Danas, tehnikama strojnog učenja (engl. *Machine learning*) pristupamo rješavanju problema na način da *algoritam* temeljem skupa podataka sam nauči riješiti problem, bez da mu *eksplicitno* modeliramo problem.

Jedan od ključnih problema kojim se bavi područje strojnog učenja je temeljem ulaznih podataka odabrati kategoriju kojoj predani ulaz pripada. Takav problem se naziva klasifikacijski problem. Primjer toga je *spam* filter, tj. klasifikacija mailova u kategorije: *spam* ili *nije spam*. Postoji više vrsta učenja takvih *modela*: jedan od njih je nadzirano učenje (engl. *supervised learning*), gdje je ideja da *model* tijekom učenja koristi parove ulaznih i izlaznih podataka, te nauči temeljem ulaznih podataka *pogađati* izlazni podatak s ciljem da njegovo *pogađanje* bude što bliže danom izlaznom podatku. U suprotnosti s time, nenadzirano učenje (engl. *unsupervised learning*) tijekom učenja koristi samo ulazne podatke - i pokušava naučiti ključne uzorke u podacima.

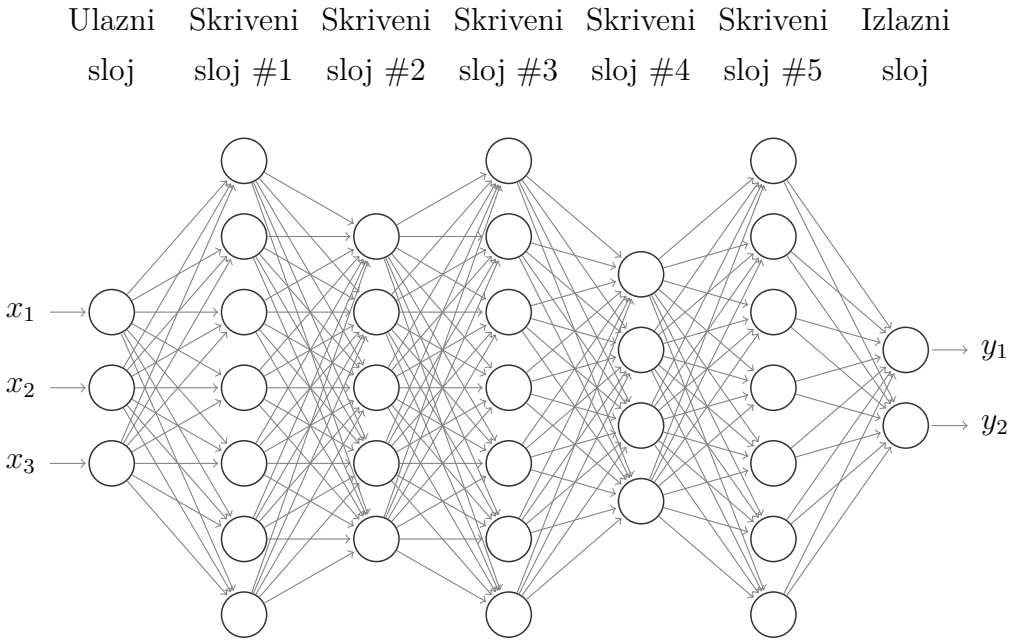
Glavna tema ovog rada je lokalizacija objekata u slici korištenjem dubokih neuronskih mreža (engl. *Deep Neural Networks - DNN*). Lokalizacija objekata na slici predstavlja problem traženja manjeg objekta (npr. prometnog znaka) na većoj slici (npr. slika ceste iz auta). Neuronske mreže ćemo koristiti za klasifikaciju malih slikovnih okana (engl. *window, patch*) te ćemo koristeći tu informaciju oblikovati jednostavan algoritam za lokalizaciju objekata na slici. Razlog odabira

dubokih neuronskih mreža je njihov golem uspjeh u nekim problemima računalnog vida, ali i drugim područjima. Pogledat ćemo i drugu vrstu neuronskih mreža koje se zovu translacijski autoenkoder, koju ćemo iskoristiti za oblikovanje malo drugačijeg algoritma za lokalizaciju, baziranom na ovom koji koristi klasifikaciju slikovnih okana.

Kako bismo stekli potrebnu podlogu za glavnu temu ovog rada, u poglavlju 2 će biti objašnjene najobičnije umjetne neuronske mreže (engl. *Artificial Neural Networks - ANN*), kako rade i kako se tipično uče. U poglavlju 3 ćemo se pozabaviti drugom vrstom neuronskih mreža koje se nazivaju konvolucijske neuronske mreže (engl. *Convolutional Neural Networks - CNN*). One se najčešće koriste u situacijama kada radimo sa zvukom ili slikom, a u tom poglavlju ćemo vidjeti i zašto. U 4. poglavlju ćemo vidjeti drugačiju vrstu neuronskih mreža koje se nazivaju autoenkoderi (engl. *Autoencoders*). Jednom kada smo postavili temelje za ovaj rad, u poglavlju 5 ćemo vidjeti varijantu autoenkodera koja se naziva translacijski autoenkoder, te ćemo u poglavlju 6 pojasniti dvije inačice lokalizacije objekata, jedna koristeći konvolucijske neuronske mreže, a druga koristeći translacijski autoenkoder. U poglavlju 7 ćemo vidjeti detalje implementacije. I na kraju, u poglavlju 8 ćemo evaluirati dobivene rezultate i usporediti lokalizaciju objekata koristeći konvolucijske neuronske mreže i lokalizaciju koristeći translacijski autoenkoder.

2. Umjetna neuronska mreža

Umjetna neuronska mreža (engl. *Artificial Neural Network - ANN*) je skup međusobno povezanih neurona, po uzoru na naš živčani sustav. Jedan primjer veće neuronske mreže dan je na slici 2.1.



Slika 2.1: Primjer neuronska mreža

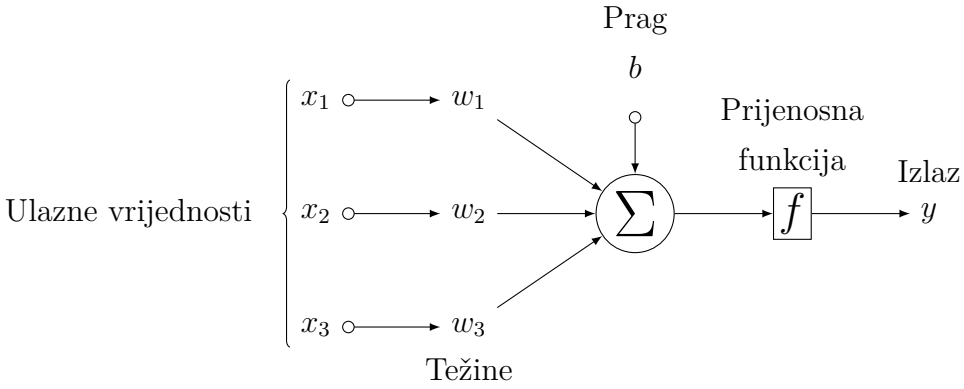
Mreža na svom ulazu dobiva skup ulaznih vrijednosti $[x_1, x_2, x_3]$. Te tri vrijednosti ulaze u sve neurone u 1. skrivenom sloju, koji izvedu neki jednostavni matematički izračun, te svoje rezultate proslijede neuronima u 2. skrivenom sloju. Zatim ti neuroni ponove istu stvar i proslijede rezultate 3. skrivenom sloju. Cijeli postupak se ponavlja do izlaznog sloja, čije vrijednosti $[y_1, y_2]$ predstavljaju izlaze neuronske mreže.

2.1. Neuron

Na slici 2.2 prikazan je neuron koji kao ulaz prima skup vrijednosti $[x_1, x_2, \ldots, x_n]$, i na izlazu daje jednu vrijednost y. Izlaz neurona definiran je izrazom u nastavku.

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

U izrazu 2.1 imamo dvije nepoznate vrijednosti i jednu nepoznatu funkciju koje zajedno predstavljaju parametre neurona. Vrijednosti $[w_1, w_2, \ldots, w_n]$ nazivamo težinama (engl. *weights*), vrijednost b nazivamo pragom (engl. *bias*), a funkcija f se naziva prijenosna funkcija (engl. *activation function*).



Slika 2.2: Neuron s tri ulazne vrijednosti $[x_1, x_2, x_3]$

2.1.1. Prijenosna funkcija

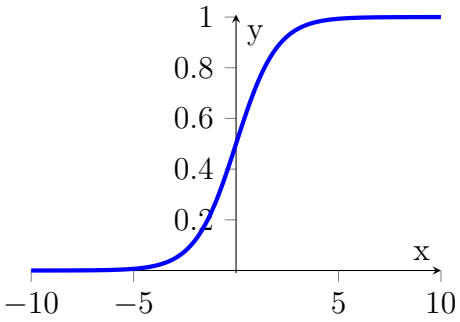
Neuron bez prijenosne funkcije (ili s prijenosnom funkcijom $f(x) = x$) predstavlja najobičniju linearnu kombinaciju ulaznih parametara, kao što je prikazano izrazom 2.1. Tada kod višeslojnih mreža skriveni slojevi gube smisao, jer će i tada izlazna vrijednost biti linearna kombinacija ulaznih vrijednosti. Zaključujemo da neuronske mreže bez nelinearnih aktivacijskih funkcija ne mogu naučiti bilo kakvu nelinearnost koje u stvarnim podacima uvijek ima.

Tipično se koriste sigmoidalne prijenosne funkcije: logistička funkcija i tangens hiperbolični, čiji su izrazi prikazani u nastavku.

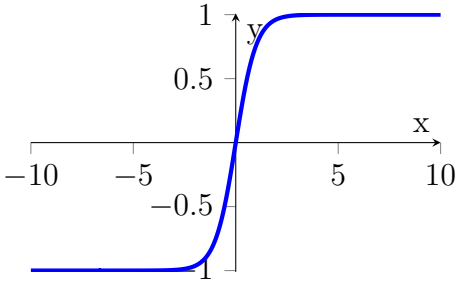
$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

Također, njihovi grafovi su prikazani na slici 2.3.



(a)



(a)

Slika 2.3: Prijenosne funkcije: (a) logistička funkcija, (b) tangens hiperbolični

Zbog načina učenja neuronskih mreža bitne su derivacije prijenosnih funkcija, zbog čega slijedi njihov izvod: prvo za logističku funkciju, a zatim za tangens hiberbolični.

$$
\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} \\
\frac{d\sigma}{dx} &= -\left(\frac{1}{1+e^{-x}}\right)^2 \cdot e^{-x} \cdot (-1) \\
&= \frac{1}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \cdot e^{-x} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}
$$

$$
\begin{aligned}
f(x) = tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\
\frac{df}{dx} &= \frac{\left(\frac{d}{dx}(e^x - e^{-x})\right) \cdot (e^x + e^{-x}) - (e^x - e^{-x}) \cdot \left(\frac{d}{dx}(e^x + e^{-x})\right)}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x}) \cdot (e^x + e^{-x}) - (e^x - e^{-x}) \cdot (e^x - e^{-x})}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
&= 1 - f(x)^2 = 1 - tanh(x)^2
\end{aligned}
$$

Ako se radi o klasifikacijskom problemu, u izlaznom sloju nalazi se po jedan izlaz za svaku kategoriju, čije su vrijednosti ograničene na interval $[0, 1]$ koje predstavljaju vjerojatnost kategorije. Zbog zahtjeva da suma vjerojatnosti mora biti jednaka 1, umjesto logističke sigmoidalne funkcije ili *tanh* koristimo *Softmax* funkciju, prikazanu u nastavku.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

2.2. Nadzirano učenje

Prije učenja neuronske mreže potrebno je dizajnirati njezinu arhitekturu. To uključuje broj skrivenih slojeva, broj neurona u svakom sloju te korištene prijenosne funkcije. Primjer takve arhitekture prikazan je na slici 2.1. Ono što preostaje je odrediti težine w_{ij} svakog neurona.

Jedan od načina pristupa učenju mreže (tj. težina neurona) je nadzirano učenje (engl. *supervised learning*). Kao što je u uvodu rečeno, ideja je da *algoritmu* za učenje damo skup ulaznih podataka X za koje su poznate izlazne vrijednosti Y. *Algoritam* temeljem ulaznih podataka X treba naučiti vrijednosti težina neurona za koje je procjena vrijednosti neuronske mreže Y' najbliža vrijednostima Y.

Nadzirano učenje koristi N parova ulaznih podataka dimenzionalnosti M i N parova izlaznih podataka dimenzionalnosti K, takvi da je svaki ulazni podatak $X_i = [x_1, x_2, \ldots, x_M] \in R^M$, a svaki izlazni podatak $Y_i = [y_1, y_2, \ldots, y_K] \in R^K$. Definirajmo $X \in R^N \times R^M$ kao skup ulaznih podataka, $Y \in R^N \times R^K$ kao skup pripadajućih izlaznih podataka, a $Y' \in R^N \times R^K$ izlazi neuronske mreže. Preostaje definirati funkciju mjere preciznosti vrijednosti izlaza neuronske mreže $E(Y, Y')$. Cilj učenja neuronske mreže je učenje vrijednosti težina neurona za koje je vrijednost $L = \sum_{i=1}^{N} E(Y_i, Y_i')$ minimalna. Smijemo zbrajati pogreške zbog činjenice da su podaci međusobno nezavisni.

Postoje dva razreda problema:

1. Klasifikacija (engl. *Classification*): ulazni podaci se smještaju u kategorije ili klase - primjer toga dan je u uvodu, klasifikacija mailova u kategorije

spam ili nije-spam

2. Regresija (engl. *Regression*): iz ulaznih podataka se izračunava kontinuirana vrijednost - primjer toga je izračun cijene stana temeljem podataka o broju soba, kvadraturi itd.

Tipično se kod regresije za funkciju mjere pogreške koristi L^2 norma (euklidska norma):

$$E(Y, Y') = \|Y - Y'\|^2 = \sum_{i=1}^{K} (y_i - y_i')^2$$

Kod klasifikacije se mogu koristiti (y_i je točna klasa, a y_i' je izračunata):

1. cross-entropy: $L = \sum_{i=1}^{N} y_i \cdot log(y_i')$
2. negative log-likelihood: $L = -\frac{1}{N} \cdot \sum_{i=1}^{N} log(p(y_i' = y_i))$

2.3. Algoritam unazadne propagacije pogreške

Izlaz neuronske mreže je funkcija ulaza i svih težina neurona $y(x, w_{11}, w_{12}, \ldots, w_{ij}, \ldots, w_{NM})$ (gdje je N broj slojeva, a M broj neurona u sloju). Za potrebe algoritma za učenje parametara neuronske mreže, potrebno je znati odrediti gradijent ∇y - što je uloga algoritma unazadne propagacije pogreške (engl. *Backpropagation*).

2.3.1. Intuicija iza algoritma

U nastavku je prikazana relativno jednostavna funkcija i cilj je izvesti njezin gradijent.

$$f(x, y) = \frac{x + y^2}{(x + y)^3 + y} \tag{2.4}$$

Gradijent ove funkcije moguće je izvesti na "školski" način:

$$\frac{\partial f}{\partial x} = \frac{1 \cdot ((x + y)^3 + y) - (x + y^2) \cdot (3 \cdot (x + y)^2)}{((x + y)^3 + y)^2}$$

$$\frac{\partial f}{\partial y} = \frac{2 \cdot y \cdot ((x + y)^3 + y) - (x + y^2) \cdot (3 \cdot (x + y)^2 + 1)}{((x + y)^3 + y)^2}$$

$$\nabla(f) = \left\{ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\}$$

Za potrebe provjere da je gradijent točno izveden algoritmom unazadnom propagacijom pogreške, prikazane su vrijednosti funkcije i gradijent u točki $x = 2, y = 3$:

$$f = \frac{11}{128} \qquad \frac{\partial f}{\partial x} = \frac{-697}{16384} \qquad \frac{\partial f}{\partial y} = \frac{-17}{4096}$$

Za gradijent jedne relativno jednostavne funkcije dobiveni su komplicirani i veliki izrazi u koje je teško uvrštavati brojeve. Algoritam unazadne propagacije pogreške temelji se na derivaciji kompozicije funkcije:

$$z = z(y(x))$$
$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

i lančanom pravilu derivacije funkcija s više varijabli:

$$z = f(x(t), y(t))$$
$$\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{\partial y} \cdot \frac{dy}{dt}$$

Izvod gradijenta funkcije (2.4) koristeći algoritam unazadne propagacije pogreške slijedi u nastavku. U prvom koraku funkcija f se rastavlja na manje funkcije čijom se kompozicijom dobiva funkcija f:

$$y_2 = y^2$$
$$brojnik = x + y_2$$
$$xpy = x + y$$
$$xpy_3 = xpy^3$$
$$nazivnik = xpy_3 + y$$
$$nazivnikinv = \frac{1}{nazivnik}$$
$$f = brojnik \cdot nazivnikinv$$

Algoritam se svodi na to da unazad (od f do x i y) provodimo izraz za derivaciju kompozicije funkcije.

$$\frac{\partial f}{\partial brojnik} = nazivnikinv \tag{2.5}$$

$$\frac{\partial f}{\partial nazivnikinv} = brojnik \tag{2.6}$$

$$\begin{aligned}\frac{\partial f}{\partial nazivnik} &= \frac{\partial f}{\partial nazivnikinv} \cdot \frac{\partial nazivnikinv}{\partial nazivnik} \\ &= \frac{\partial f}{\partial nazivnikinv} \cdot \frac{-1}{nazivnik^2}\end{aligned} \tag{2.7}$$

$$\frac{\partial f}{\partial xpy_3} = \frac{\partial f}{\partial nazivnik} \cdot \frac{\partial nazivnik}{\partial xpy_3} = \frac{\partial f}{\partial nazivnik} \cdot 1 \tag{2.8}$$

$$\frac{\partial f}{\partial xpy} = \frac{\partial f}{\partial xpy_3} \cdot \frac{\partial xpy_3}{\partial xpy} = \frac{\partial f}{\partial xpy_3} \cdot 3 \cdot xpy^2 \tag{2.9}$$

$$\frac{\partial f}{\partial y_2} = \frac{\partial f}{\partial brojnik} \cdot \frac{\partial brojnik}{\partial y_2} = \frac{\partial f}{\partial brojnik} \cdot 1 \tag{2.10}$$

$$\begin{aligned}\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial brojnik} \cdot \frac{\partial brojnik}{\partial x} + \frac{\partial f}{\partial xpy} \cdot \frac{\partial xpy}{\partial x} \\ &= \frac{\partial f}{\partial brojnik} \cdot 1 + \frac{\partial f}{\partial xpy} \cdot 1\end{aligned} \tag{2.11}$$

$$\begin{aligned}\frac{\partial f}{\partial y} &= \frac{\partial f}{\partial nazivnik} \cdot \frac{\partial nazivnik}{\partial y} + \frac{\partial f}{\partial xpy} \cdot \frac{\partial xpy}{\partial x} \\ &= \frac{\partial f}{\partial nazivnik} \cdot 1 + \frac{\partial f}{\partial xpy} \cdot 1\end{aligned} \tag{2.12}$$

Izraz (2.7) ovisi samo o izrazu (2.6) (koji je definiran iznad izraza (2.7)), izraz (2.8) ovisi samo o izrazu (2.7) (koji je definiran iznad izraza (2.8)), sve do izraza (2.12) koji ovisi samo o izrazima (2.7) i (2.9) (koji su definirani iznad izraza (2.12)). Odnosno, svi gornji izrazi ovise samo o izrazima koji su definirani iznad njih, što znači da su nam kod izračuna gornjih derivacija sve vrijednosti nepoznatih derivacije zapravo poznate. Dodatno, funkcije $\frac{\partial f}{\partial x}$ i $\frac{\partial f}{\partial y}$ sadrže sumu drugih derivacija (zbog lančanog pravila funkcija s više varijabli).

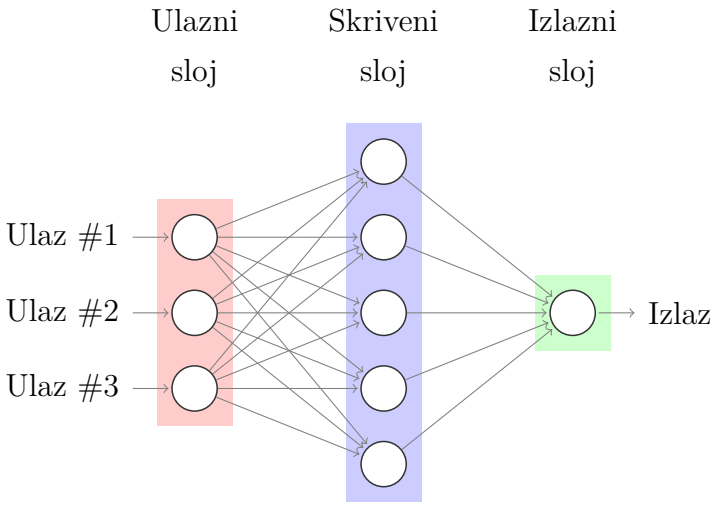
Ovakvi izrazi su poželjni jer implementacija u većini programskih jezika postaje vrlo elegantna, jer možemo računati derivacije funkcija od zadnje prema prvoj, proširivati i mijenjati funkciju bez značajnih promjena u izračunu gradijenta. U izvornom kodu u nastavku izrazi dx i dy nisu izračunati kao suma derivacija, nego su vrijednosti pribrajane kroz postupak.

```
x = 2
y = 3

# prolaz prema naprijed (forward pass)
y2 = y ^ 2                                  # y2 = 9
brojnik = x + y2                            # brojnik = 11
xpy = x + y                                 # xpy = 5
xpy3 = xpy \^ 3                              # xpy3 = 125
nazivnik = xpy3 + y                         # nazivnik = 128
nazivnikinv = 1.0 / nazivnik                # nazivnikinv = 1/128
f = brojnik * nazivnikinv                   # f = 11/128

# prolaz prema natrag (backward pass)
# unazadna propagacija pogreške za f = brojnik * nazivnikinv
dbrojnik = nazivnikinv                      # dbrojnik = 1/128
dnazivnikinv = brojnik                      # dnazivnikinv = 11
# unazadna propagacija pogreške za nazivnikinv = 1/nazivnik
dnazivnik = -1.0 / nazivnik^2 * dnazivnikinv # dnazivnik = -11/16384
# unazadna propagacija pogreške za nazivnik = xpy3 + y
dxpy3 = dnazivnik                           # dxpy3 = -11/16384
dy = dnazivnik                              # dy = -11/16384
# unazadna propagacija pogreške za xpy3 = xpy ^3
dxpy = dxpy3 * 3 * xpy ^ 2                  # dxpy = -825/16384
# unazadna propagacija pogreške za xpy = x + y
dx = dxpy                                   # dx = -825/16384
dy += dxpy                                  # dy = -209/4096
# unazadna propagacija pogreške za brojnik = x + y2
dx += dbrojnik                              # dx = -697/16384
dy2 = dbrojnik                              # dy2 = 1/128
# unazadna propagacija pogreške za y2 = y ^2
dy += dy2 * 2 * y                           # dy = -17/4096
```

Izvorni kod 2.1: Primjer izračuna gradijenta za funkciju u izrazu 2.4

2.3.2. Primjena na neuronske mreže

Na slici 2.4 prikazan je primjer dvoslojne neuronske mreže. Na ulazu se nalaze tri vrijednosti: x_1, x_2 i x_3, a na izlazu jedna vrijednost y. Mreža sadrži jedan skriveni sloj s 5 neurona. Težine između neurona na ulaznom sloju i neurona u skrivenom sloju označavamo s W^1_{ij}, a težine između neurona u skrivenom sloju i izlaznom sloju s W^2_{ij}.



Slika 2.4: Neuronska mreža - 2 sloja

U nastavku su prikazani izrazi neuronske mreže u matričnom obliku.

$$X = [x_1 x_2 x_3]^T$$

$$W^1 = \begin{bmatrix} W^1_{11} & W^1_{12} & W^1_{13} \\ W^1_{21} & W^1_{22} & W^1_{23} \\ W^1_{31} & W^1_{32} & W^1_{33} \\ W^1_{41} & W^1_{42} & W^1_{43} \\ W^1_{51} & W^1_{52} & W^1_{53} \end{bmatrix}$$

$$H = W^1 X$$

$$W^2 = \begin{bmatrix} W^2_{11} & W^2_{12} & W^2_{13} & W^2_{14} & W^2_{15} \end{bmatrix}$$

$$Y = W^2 H = [y]$$

Prolaz prema naprijed (engl. *forward pass*) prikazan je u nastavku.

```
1 # dot(A, B) mnozi matrice A i B
2 # transpose(A) transponira matricu A
3
4 H = dot(W1, transpose(X))
5 Y_ = dot(W2, H)
```

Izvorni kod 2.2: Prolaz prema naprijed za neuronsku mrežu na slici 2.4

U ovom primjeru za mjeru pogreške koristimo funkciju prikazanu u nastavku.

$$E = \frac{1}{2}\|Y' - Y\|^2$$
$$\frac{\partial E}{\partial Y'} = Y' - Y$$

Prolaz prema natrag (engl. *backward pass*) prikazan je u nastavku.

```
1  # Y je točni izlaz, a Y_ je izračunati izlaz.
2
3  # unazadna propagacija pogreške za E
4  diff = Y - Y_
5
6  # unazadna propagacija pogreške za Y_
7  dW2 = dot(diff, H)
8  dX = dot(diff, X)
9
10 # unazadna propagacija pogreške za H
11 dW1 = dot(dX, X)
```

Izvorni kod 2.3: Prolaz prema natrag za neuronsku mrežu na slici 2.4

2.4. Gradijentni spust

Neuronske mreže najčesće se uče algoritmom gradijentni spust (engl. *Gradient descent*). Algoritam se sastoji od 2 koraka, koji se izvršavaju fiksni broj iteracija:

1. Izračunaj gradijente pogreške za težine: $\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}$.
   Ovo se najčesće radi algoritmom unazadne propagacije pogreške.

2. Ažuriraj težine u smjeru suprotnome od gradijenta:

$$w_1 = w_1 - \alpha \cdot \frac{\partial E}{\partial w_1}$$
$$w_2 = w_2 - \alpha \cdot \frac{\partial E}{\partial w_2}$$
$$\ldots$$
$$w_n = w_n - \alpha \cdot \frac{\partial E}{\partial w_n}$$

Koeficijent α naziva se faktorom učenja (engl. *learning rate*). Što je on manji - to je preciznije učenje, a što je veći - to je treniranje brže.

Najjednostavnija inačica gradijentnog spusta računa gradijent koristeći cijeli podatkovni skup. Kod velike količine podataka, u svakoj iteraciji algoritam prolazi kroz cijeli skup da bi izračunao gradijent i samo jednom ažurirao težine. Ovo se može ubrzati particioniranjem cijelog podatkovnog skupa na male dijelove (engl. *batch*) te izračunom gradijenata i ažuriranjem težina za svaki dio pojedinačno. Ovakva inačica algoritma naziva se minibatch stohastični gradijentni spust (engl. *Minibatch Stochastic Gradient Descent, Minibatch SGD*). U praksi, ova inačica algoritma brže konvergira minimumu.

Implementacija cijelog algoritma je prikazana u nastavku.

```
1  class NeuralNetwork:
2    # ažurira težine mreže
3    # svaka tezina ažurira se izrazom: w = w - learning_rate * dw
4    # (gdje je w težina, a dw derivacija funkcije greške po w)
5    def update_weights(grad, learning_rate)
6
7    # implementira algoritam unazadne propagacije pogreške
8    # vraća gradijent pogrečke za svaku težinu mreže
9    def backpropagation(X, Y)
10
11   # izračunava izlaz mreže za zadani ulaz
12   def output(X)
13
14 # racuna pogrešku učenja
15 # npr. (broj točnih klasifikacija) / (ukupan broj primjera)
16 def calculate_error(net, X, Y)
17
18 # num_iters = broj iteracija, num_batches = broj dijelova
19 # learning_rate = faktor učenja
20 def gradient_descent(net, X, Y, num_iters, num_batches,
       learning_rate):
21   best_error = inf
22   best_net = None
23
24   for i in range(num_iters):
25     for i in range(num_batches):
26       X_batch = X[i * num_batch : (i + 1) * num_batch]
27       Y_batch = Y[i * num_batch : (i + 1) * num_batch]
28       grad = net.backpropagation(X_batch, Y_batch)
29       net.update_weights(grad, learning_rate)
30
31     error = calculate_error(net, X, Y)
32     if error < best_error:
33       best_error = error
34       best_net = net
35
36   return best_net
```

Izvorni kod 2.4: Implementacija algoritma minibatch stohastični gradijentni spust

3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže su neuronske mreže koje rade s dvodimenzionalnim podacima (npr. slika), tj. $X \in \mathbb{R}^N \times \mathbb{R}^N$, u odnosu na umjetne neuronske mreže koje rade sa skalarnim vrijednostima. U odnosu na obične neuronske mreže, značajke ovdje nazivamo mapom značajka (engl. *feature map*), a težine neurona sada nazivamo jezgra (engl. *kernel*). Također, u ovom radu je pretpostavljeno da se radi o monokromatskim slikama (nijanse sive).

Postoje dvije vrste slojeva kod konvolucijskih neuronskih mreža: konvolucijski sloj (engl. *convolutional layer*) i sloj sažimanja (engl. *pooling layer*). Ideja konvolucijskog sloja je da djeluje kao filter za bitne značajke slike. Konvolucijski sloj je definiran svojom jezgrom $K \in \mathbb{R}^M \times \mathbb{R}^M$. Ako je $X \in \mathbb{R}^N \times \mathbb{R}^N$ ulaz u neuron konvolucijskog sloja, izlaz je definiran kao:

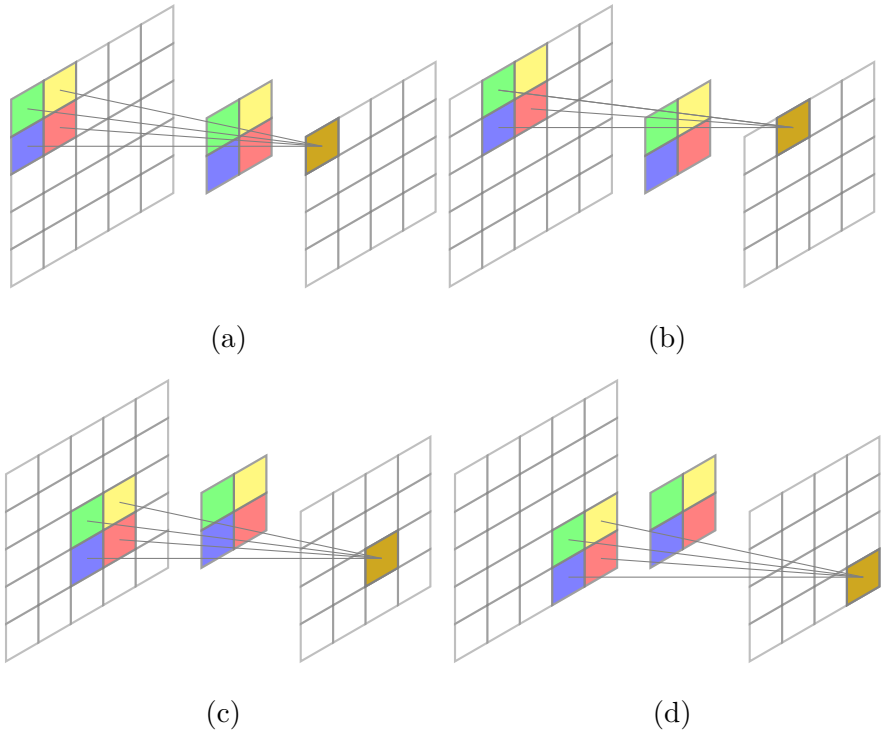
$$Y = X * K$$

Gdje operator $*$ predstavlja 2D konvoluciju. Izlaz je matrica dimenzija $Y \in \mathbb{R}^{N-M+1} \times \mathbb{R}^{N-M+1}$.

U nastavku je prikazan primjer konvolucije nad matricama $X \in \mathbb{R}^5 \times \mathbb{R}^5$ i $K \in \mathbb{R}^2 \times \mathbb{R}^2$. Rezultat operacije konvolucije je matrica $Y \in \mathbb{R}^4 \times \mathbb{R}^4$.

$$\begin{aligned}
Y_{00} &= X_{00} \cdot K_{00} + X_{01} \cdot K_{01} + X_{10} \cdot K_{10} + X_{11} \cdot K_{11} \\
Y_{01} &= X_{01} \cdot K_{00} + X_{02} \cdot K_{01} + X_{11} \cdot K_{10} + X_{12} \cdot K_{11} \\
&\dots \\
Y_{03} &= X_{03} \cdot K_{00} + X_{04} \cdot K_{01} + X_{13} \cdot K_{10} + X_{14} \cdot K_{11} \\
Y_{10} &= X_{10} \cdot K_{00} + X_{11} \cdot K_{01} + X_{20} \cdot K_{10} + X_{21} \cdot K_{11} \\
&\dots \\
Y_{33} &= X_{33} \cdot K_{00} + X_{34} \cdot K_{01} + X_{43} \cdot K_{10} + X_{44} \cdot K_{11}
\end{aligned}$$

Ilustracija konvolucije nad matricama X i K je prikazana na slici 3.1.



Slika 3.1: Ilustracija konvolucije: (a) za Y_{00}, (b) za Y_{10}, (c) za Y_{22} i (d) Y_{33}

Kao što je rečeno, osim konvolucijskog sloja postoji i sloj sažimanja. Postoji više vrsta slojeva sažimanja, među kojima su: sažimanje usrednjavanjem (engl. *mean pooling*), sažimanje maksimalnom vrijednovsću (engl. *max pooling*) itd. U ovom radu korišteno je sažimanje maksimalnom vrijednovsću, stoga će samo ono biti objašnjeno u nastavku.

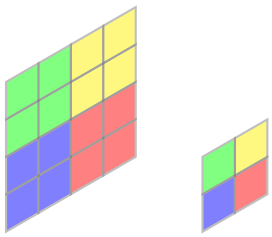
Ideja je da sliku $X \in \mathbb{R}^N \times \mathbb{R}^N$ particioniramo na dijelove dimenzija $K \times K$, gdje je K konstanta sažimanja. Tada za svaku particiju uzimamo maksimalnu vrijednost unutar nje, zbog čega na izlazu dobivamo $Y \in \mathbb{R}^{N/K} \times \mathbb{R}^{N/K}$.

Uzmimo iz prijašnjeg primjera izlaz konvolucijskog sloja kao ulaz, znači $X \in \mathbb{R}^4 \times \mathbb{R}^4$, te konstantu sažimanja $K = 2$. Kao izlaz dobivamo $Y \in \mathbb{R}^2 \times \mathbb{R}^2$, koji

računamo kao:

$$Y_{00} = max(X_{00}, X_{01}, X_{10}, X_{11})$$
$$Y_{01} = max(X_{02}, X_{03}, X_{12}, X_{13})$$
$$Y_{10} = max(X_{20}, X_{21}, X_{30}, X_{31})$$
$$Y_{11} = max(X_{22}, X_{23}, X_{32}, X_{33})$$

Ilustracija sloja sažimanja prikazana je na slici 3.2.



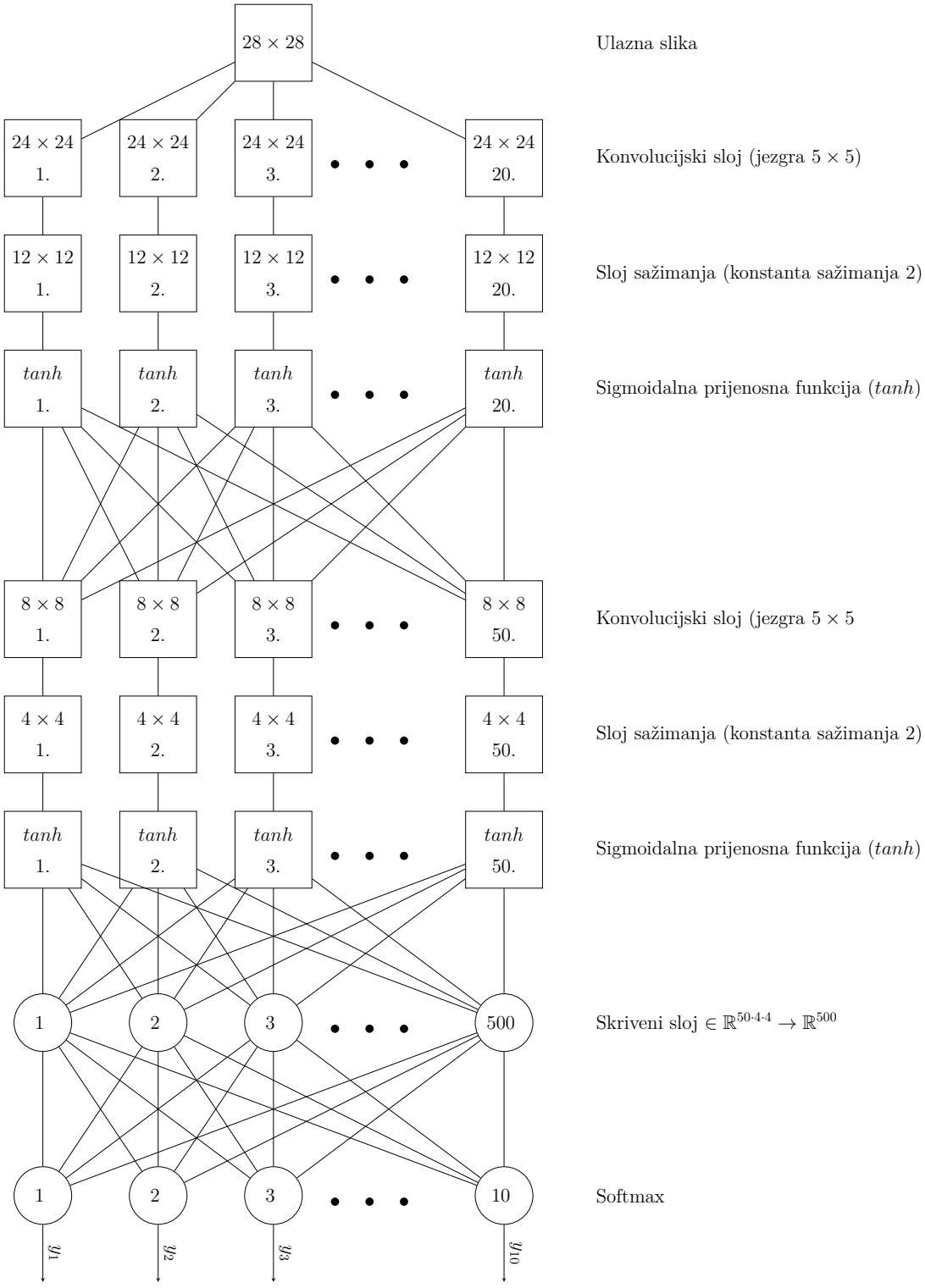
Slika 3.2: Ilustracija sloja sažimanja

Na slici 3.3 prikazana je konvolucijska neuronska mreža za klasifikaciju slika korištena u ovom radu, sadrži:

1. Konvolucijski sloj s 20 jezgri 5×5
2. Sloj sažimanja maksimalnom vrijednovsću s konstantom sažimanja $K = 2$
3. Sigmoidalna prijenosna funkcija ($tanh$)
4. Konvolucijski sloj s 50 jezgri 5×5
5. Sloj sažimanja maksimalnom vrijednovsću s konstantom sažimanja $K = 2$
6. Sigmoidalna prijenosna funkcija ($tanh$)
7. Potpuno povezani skriveni sloj sa sigmoidalnom prijenosnom funkcijom ($tanh$)
8. Softmax sloj

Kao mjera pogreške uzima se negative log likelihood, gdje je $p(y'_i = y_i)$ vjerojatnost koju klasifikator kaže za točnu klasu:

$$L = -\frac{1}{N} \cdot \sum_{i=1}^{N} log(p(y'_i = y_i))$$

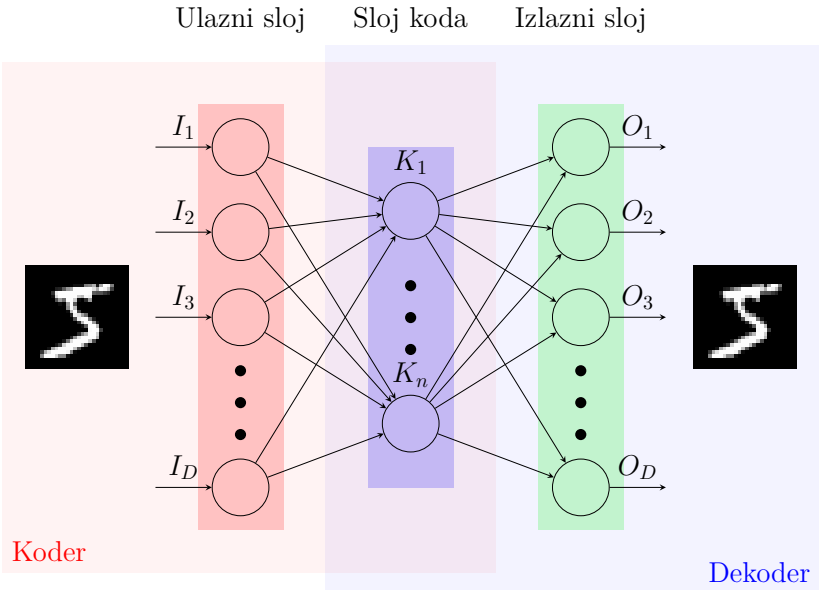


Slika 3.3: Arhitektura za klasifikaciju slika 28×28

4. Autoenkoder

Dosad su u radu objašnjene obične neuronske mreže i konvolucijske neuronske mreže. Njihova zajednička karakteristika je, temeljem ulaznih parametara naučiti "pogađati" izlazne parametre. Kao primjeri su navedeni određivanje cijene stana temeljem podacima o stanu, temeljem slike klasificirati objekt na njoj.

S druge strane, autoenkoderi (engl. *Autoencoders*) čine drugu vrstu neuronskih mreža. Primjer najosnovnijeg autoenkodera prikazan je na slici 4.1.



Slika 4.1: Autoenkoder tijekom učenja

Autoenkoder se sastoji od 2 dijela: koder i dekoder. Uloga kodera je transformacija ulaznih parametara dimenzionalnosti D, tj. $X \in \mathbb{R}^D$ (npr. slika) u kôd manje dimenzionalnosti od same slike, a uloga dekodera je rekonstrukcija slike samo iz kôda. Zajednički dio kodera i dekodera je sloj koji sadrži kod. Izrazi za kodiranje i dekodiranje ulaznih parametara prikazani su u nastavku.

$$K = f(W_{IK} \cdot I + b_{IK})$$
$$O = f(W_{KO} \cdot K + b_{KO})$$

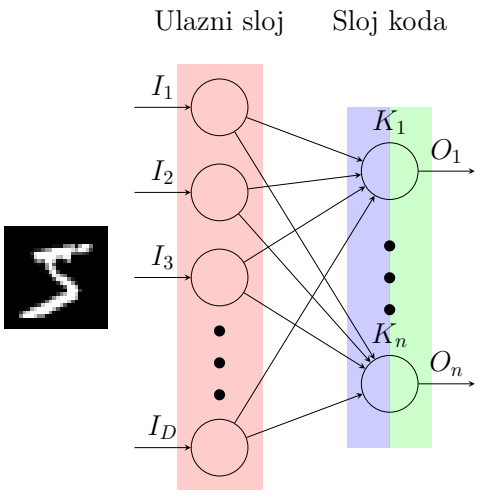
I je vektor ulaznih parametara, K je vektor kôda i O je vektor izlaznih vrijednosti. W_{IK} su težine između ulaznog sloja i sloja koda, b_{IK} je prag između ulaznog sloja i sloja koda, W_{KO} su težine između sloja koda i izlaznog sloja, b_{KO} je prag između sloja koda i izlaznog sloja i f je prijenosna funkcija. Ključno za primijetiti je da dekoder ne koristi ulazne parametre, samo kôd koji je koder izračunao.

Učimo parametre autoenkodera s funkcijom pogreške sličnost između rekonstrukcije (izlaz iz dekodera) i originalne slike (ulaz u koder) koja je prikazana u nastavku.

$$L = -\sum_{k=1}^{N}(I_k \cdot logO_k + (1 - I_k) \cdot log(1 - O_k))$$

N je broj ulaznih podataka, O_k je k-ti izlazni podatak iz dekodera, I_k je ulazni podatak.

Ako nas zanima samo kôd, jednom naučenom autoenkoderu mičemo dekoder. Tako kôd postaje izlaz mreže, kao što je prikazano na slici 4.2.



Slika 4.2: Autoenkoder nakon učenja

Takav autoenkoder bi se mogao koristiti za kompresiju podataka, ili za smanjivanje dimenzionalnosti podataka - što može pomoći u učenju drugih modela,

s obzirom na to da se u takvom kodu nalaze najbitnije značajke.

Postoje i druge varijante autoenkodera, kao što je autoenkoder koji uklanja šum (engl. *Denoising Autoencoder*) - ovdje je razlika ta što se na ulazu nalazi slika sa šumom, a na izlazu originalna slika bez šuma, što upravo znači da učimo parametre mreže koji bi naučili maknuti šum. Posebno zanimljiva vrsta autoenkodera su translatirajući autoenkoderi, koji su opisani u sljedećem poglavlju.

5. Translacijski autoenkoder

Kao što se iz naslova naslućuje, u ovom poglavlju bavimo se drugačijom inačicom autoenkodera. Prvo uvedimo koncept kapsula koje konceptualno možemo gledati kao zasebnu neuronsku mrežu. Ideja je da svaka kapsula nauči prepoznavati dio slike (vizualne entitete). Na primjeru slova abecede, mogli bismo imati jednu kapsulu koja prepoznaje lijevu vertikalnu crtu slova H, drugu kapsulu koja prepoznaje desnu vertikalnu crtu H, i treću koja prepoznaje horizontalnu crtu slova H (i još mnogo drugih za svaki pojedinačni dio svakog slova).

Translacijski autoenkoder je skup od M kapsula, svaka od njih će biti izvedena na sličan način kao autoenkoder. Ideja je da se tijekom treniranja na svaku sliku $X_i \in \mathbb{R}^N$ dimenzionalnosti $N = M \times K$ primijeni dva puta neka translacijska funkcija $T\colon \mathbb{R}^2 \times \mathbb{R}^N \to \mathbb{R}^N$. Time se generira novi skup za treniranje:

$$X_i' = \{x = T(\Delta_x, X_i), t = T(\Delta_t, X_i), \Delta = \Delta_t - \Delta_x\}$$

Kao što se može naslutiti, tijekom treniranja će x biti ulaz, a t izlaz translacijskog autoenkodera, tj. svake kapsule. Ono što je cilj je da svaka kapsula u kôdu sadrži vjerojatnost p da je prisutan vizualni entitet koji kapsula predstavlja, a x i y će označavati relativni pomak vizualnog entiteta u odnosu na cijelu ulaznu sliku. Ovaj autoenkoder će tijekom učenja biti prisiljen naučiti iz slike x rekonstruirati sliku t poznavajući Δ.

Svaka kapsula je zasebni autoenkoder, koji u *koder* dijelu ima jedan skriveni sloj koji se naziva sloj za prepoznavanje (engl. *recognition layer*) i njegove težine se označavaju s H_r. U dekoder dijelu na x i y primijenimo translaciju Δ, dobivši x' i y' (C'). Sada iz C' želimo rekonstruirati sliku, zbog čega u *dekoder* dijelu imamo skriveni sloj koji nazivamo generativni sloj (engl. *generative layer*) i njegove težine označavamo s H_g. Drugi sloj *dekodera* je izlaz kapsule (kod treniranja) koji se dodatno izlaz množi s p. Na kraju, izlaz translacijskog autoenkodera je suma svih

izlaza kapsula. Ili simbolički:

$$H_r = \sigma(W_{xh}x + b_r) \in (0,1)^{N_r} \tag{5.1}$$

$$C = W_{hc}H_r + b_c \in \mathbb{R}^2 \tag{5.2}$$

$$C' = C + \Delta \in \mathbb{R}^2 \tag{5.3}$$

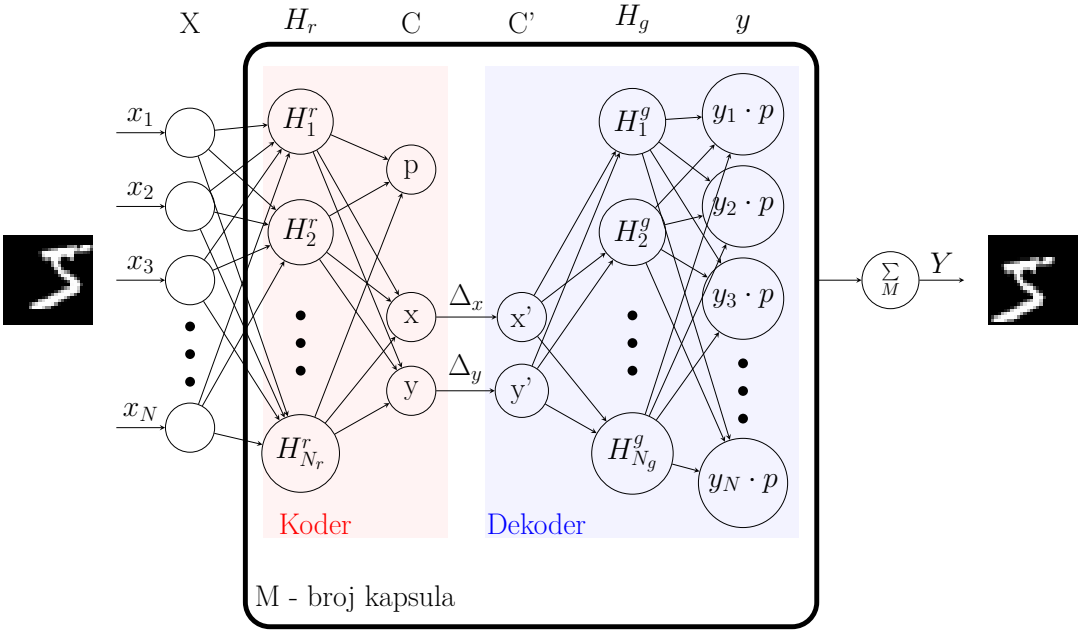
$$p = \sigma(W_{hp}H_r + b_p) \in (0,1) \tag{5.4}$$

$$H_g = \sigma(W_{cg}c' + b_g) \in (0,1)^{N_g} \tag{5.5}$$

$$y = \sigma(W_{hy}H_g \cdot p + b_y) \in \mathbb{R}^N \tag{5.6}$$

$$Y = \sum_{i=0}^{N} y_i \tag{5.7}$$

Gdje W_{xh} predstavlja težine između ulaznog sloja (X) i sloja za prepoznavanje (H_r), W_{hc} i W_{hp} predstavlja težine između sloja za prepoznavanje (H_r) i sloja s kôdom (C), W_{cg} predstavlja težine između sloja s translatiranim kôdom (C') i generativnog sloja (H_g), a W_hy predstavlja težine između generativnog sloja (H_g) i izlaznog sloja (y). Prikaz translacijskog autoenkodera možete vidjeti na slici 5.1.



Slika 5.1: Translacijski autoenkoder tijekom učenja

Mjera greške koju koristimo i tijekom treniranja pokušavamo minimizirati je cross-entropy, koja je prikazana izrazom u nastavku.

$$E = -\frac{1}{N}\sum_{j=1} t_j \cdot log(y_j) + (1 - t_j) \cdot log(1 - y_j) \tag{5.8}$$

Mrežu treniramo gradijentnim spustom i unazadnom propagacijom pogreške, zbog čega nam treba gradijent.

$$\frac{\partial E}{\partial y_j} = -\frac{1}{N} \cdot \left(\frac{t_j}{y_j} + \frac{1 - t_j}{1 - y_j}\right) = -\frac{1}{N} \cdot \frac{t_i \cdot (1 - y_j) - y_j \cdot (1 - t_j)}{y_j \cdot (1 - y_j)} \tag{5.9}$$

$$= \frac{1}{N} \cdot \frac{y_i - t_i}{y_j \cdot (1 - y_j)} \tag{5.10}$$

Korisno je primijetiti da ako u izrazu (5.8) zapišemo y_j kao $\sigma(y_j)$, gradijent postaje jednostavniji:

$$E = -\frac{1}{N} \cdot \sum_{j=1} t_j \cdot log(\sigma(y_j)) + (1 - t_j) \cdot log(1 - \sigma(y_j)) \tag{5.11}$$

$$\frac{\partial E}{\partial y_j} = ... = -\frac{1}{N} \cdot \frac{t_i - \sigma(y_j)}{\sigma(y_j) \cdot (1 - \sigma(y_j))} \cdot \frac{\partial \sigma(y_j)}{\partial y_j} = \frac{1}{N} \cdot (\sigma(y_i) - t_i) \tag{5.12}$$

Pritom koristeći:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x)) \tag{5.13}$$

Idemo redom, pritom se koristimo chain ruleom:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} \tag{5.14}$$

I derivacijama matrica[1]:

$$\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T \tag{5.15}$$

U nastavku slijedi postupak izračuna gradijenata.

$$\frac{\partial E}{\partial W_{hy}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial W_{hy}} \tag{5.16a}$$

$$\frac{\partial y}{\partial W_{hy}} = p \cdot H_g^T \tag{5.16b}$$

$$\frac{\partial E}{\partial b_y} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial b_y} \tag{5.17a}$$

$$\frac{\partial y}{\partial b_y} = 1 \tag{5.17b}$$

[1] `http://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`

$$\frac{\partial E}{\partial W_{cg}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial H_g} \cdot \frac{\partial H_g}{\partial W_{cg}} \tag{5.18a}$$
$$\frac{\partial y}{\partial H_g} = p \cdot W_{hy} \tag{5.18b}$$
$$\frac{\partial H_g}{\partial W_{cg}} = H_g \cdot (1 - H_g) \cdot c'^T \tag{5.18c}$$

$$\frac{\partial E}{\partial b_g} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial H_g} \cdot \frac{\partial H_g}{\partial b_g} \tag{5.19a}$$
$$\frac{\partial H_g}{\partial b_g} = H_g \cdot (1 - H_g) \tag{5.19b}$$

$$\frac{\partial E}{\partial W_{hp}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial W_{hp}} \tag{5.20a}$$
$$\frac{\partial y}{\partial p} = W_{hy} H_g \tag{5.20b}$$
$$\frac{\partial p}{\partial W_{hp}} = p \cdot (1 - p) \cdot H_r^T \tag{5.20c}$$

$$\frac{\partial E}{\partial b_p} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial b_p} \tag{5.21a}$$
$$\frac{\partial p}{\partial b_p} = p \cdot (1 - p) \tag{5.21b}$$

$$\frac{\partial E}{\partial W_{hc}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial H_g} \cdot \frac{\partial H_g}{\partial c'} \cdot \frac{\partial c'}{\partial c} \cdot \frac{\partial c}{\partial W_{hc}} \tag{5.22a}$$
$$\frac{\partial H_g}{\partial c'} = H_g \cdot (1 - H_g) \cdot W_{cg} \tag{5.22b}$$
$$\frac{\partial c'}{\partial c} = 1 \tag{5.22c}$$
$$\frac{\partial c}{\partial W_{hc}} = H_r \tag{5.22d}$$

$$\frac{\partial E}{\partial b_c} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial H_g} \cdot \frac{\partial H_g}{\partial c'} \cdot \frac{\partial c'}{\partial c} \cdot \frac{\partial c}{\partial b_c} \tag{5.23a}$$
$$\frac{\partial c}{\partial b_c} = 1 \tag{5.23b}$$

$$\frac{\partial E}{\partial W_{xh}} = \frac{\partial E}{\partial y} \cdot \left(\frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial H_r} \cdot \frac{\partial H_r}{\partial W_{xh}} + \frac{\partial y}{\partial H_g} \cdot \frac{\partial H_g}{c'} \cdot \frac{\partial c'}{\partial c} \cdot \frac{\partial c}{\partial H_r} \cdot \frac{\partial H_r}{\partial W_{xh}} \right) \tag{5.24a}$$

$$\frac{\partial p}{\partial H_r} = p \cdot (1 - p) \cdot W_{hp} \tag{5.24b}$$

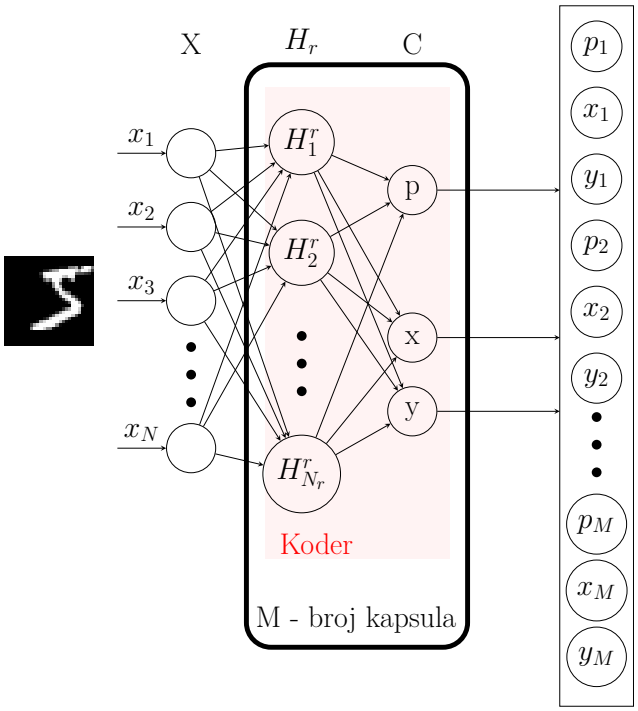
$$\frac{\partial H_r}{\partial W_{xh}} = H_r \cdot (1 - H_r) \cdot x \tag{5.24c}$$

$$\frac{\partial c}{\partial H_r} = W_{hc} \tag{5.24d}$$

$$\frac{\partial E}{\partial b_r} = \frac{\partial E}{\partial y} \cdot \left(\frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial H_r} \cdot \frac{\partial H_r}{\partial b_r} + \frac{\partial y}{\partial H_g} \cdot \frac{\partial H_g}{c'} \cdot \frac{\partial c'}{\partial c} \cdot \frac{\partial c}{\partial H_r} \cdot \frac{\partial H_r}{\partial b_r} \right) \tag{5.25a}$$

$$\frac{\partial H_r}{\partial b_r} = H_r \cdot (1 - H_r) \cdot x \tag{5.25b}$$

Jednom kada naučimo autoenkoder, možemo maknuti *dekoder* dio i onda dobijemo translacijski autoenkoder prikazan na slici 5.2.



Slika 5.2: Translacijski autoenkoder nakon učenja

Ono što dobijemo je, za svaku kapsulu vjerojatnost da je prisutan vizualni entitet kojeg ona predstavlja, i x, y koji predstavljaju relativni pomak u odnosu na sliku.

6. Raspoznavanje i lokalizacija translacijskim autoenkoderima

Glavna tema ovog rada je lokalizacija objekata na slici, što ovo poglavlje čini najbitnijim dijelom rada. Zbog načina na koji radimo klasifikaciju, bitna nam je mogućnost klasifikacije slika, pa ćemo u poglavlju 6.1. pojasniti kako možemo izlaze translacijskog autoenkodera iskoristiti za klasifikaciju. Nakon toga, u poglavlju 6.2 ćemo objasniti dva algoritma za lokalizaciju: raspoznavanjem slikovnih okna fiksnih dimenzija i houghovom transformacijom.

6.1. Raspoznavanje translacijskim autoenkoderima

Zamislimo da radimo klasifikaciju slika na kojima je rukom nacrtano jedno slovo. Kada bismo takve slike koristili tijekom učenja translacijskog autoenkodera, za pretpostaviti je da bismo imali kapsule koje bi prepoznavale dijelove slova (vertikalna crta, horizontalna crta itd.). Kada bi se aktivirale dvije kapsule koje predstavljaju vertikalnu crtu, mogli bismo uspoređivati njihovu udaljenost. Kada bi ona bila relativno mala, mogli bismo misliti da je na slici prisutno slovo H, U ili N, a kada bi bila malo veća, na slici bi moglo biti prisutno slovo M ili W. U situaciji da je udaljenost mala i još je aktivirana kapsula koja predstavlja horizontalnu crtu, i njezina lokacija je na sredini - mogli bismo znati da se radi o slovu H.

Ovakvo razmišljanje ukazuje na to da bismo mogli iskoristiti parametre pozicija $x_i, y_i \in \mathbb{R}$ koje dobijemo za svaku kapsulu, zajedno s vjerojatnosti da je vizualni identitet prisutan $p_i \in (0, 1)$ te uspoređujući njihov odnos zaključiti o kojoj se znamenci radi. Definirajmo elemente matrice udaljenosti između kapsula i i j kao:

$$d_{ij} = \sigma_x^2 \cdot (x_i - x_j + b_{ij}^x)^2 + \sigma_y^2 \cdot (y_i - y_j + b_{ij}^y)^2 \tag{6.1}$$

Parametri σ_x i σ_y predstavljaju parametre koliko je bitan odnos kapsula i i j, a b_{ij}^x i b_{ij}^y parametri udaljenosti između pojedine kapsule. Zatim definirajmo ulaz u *softmax* klasifikator kao:

$$f_{ij} = \frac{p_i \cdot p_j}{1 + d_{ij}} \tag{6.2}$$

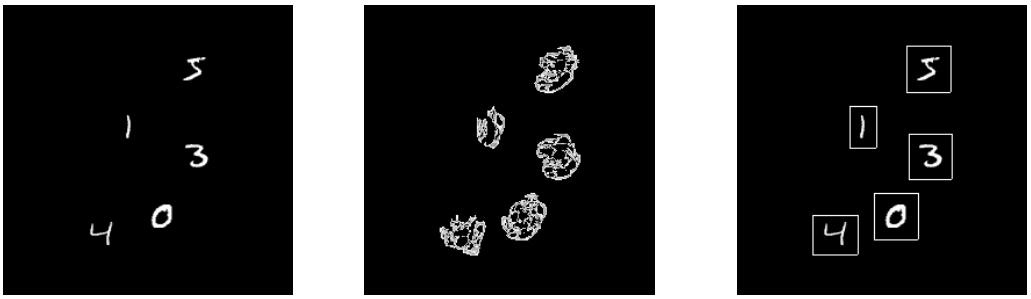
I izlaz iz softmax funkcije koristimo kao izlaz neuronske mreže.

6.2. Lokalizacija

Način na koji radimo lokalizaciju temelji se na principu slikovnih okana. Ideja je da napravimo prolaz kroz sliku, pomičući se po slikovnom oknu fiksne širine s fiksnim razmakom između okana. U nastavku su prikazane dvije metode za lokalizaciju: raspoznavanjem slikovnih okana i houghovom transformacijom izlaza translacijskog autoenkodera.

6.2.1. Lokalizacija raspoznavanjem slikovnih okana

Za svako slikovno okno, izračunamo izlaz klasifikatora: klasa k i vjerojatnost za klasu p_k. Rezultat prolaza slike 6.1 (a) možete vidjeti na slici 6.1 (b). Ono što možete primijetiti je da su se napravili grupe piksela u okružju svake znamenke, pa je ideja naći sve grupe i njima odrediti lokaciju znamenke, što je prikazano na slici 6.1 (c). U nastavku je prikazana glavna funkcija za lokalizaciju.



(a) (b) (c)

Slika 6.1: Primjer lokalizacije: (a) sirova slika, (b) lokacije prozora nađenih znamenki i (c) označene znamenke

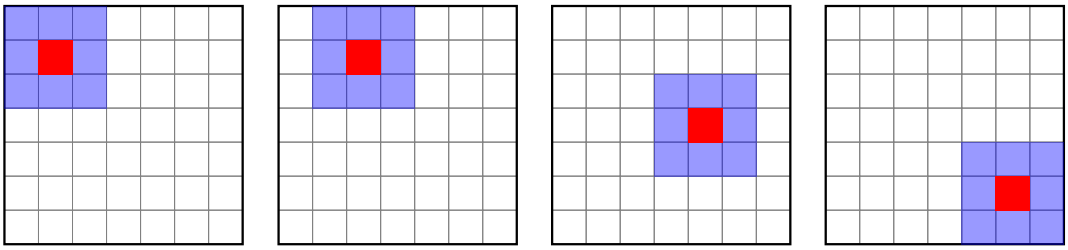
```
1  def localization(image, net, window = 28, step = 1, space = 4):
2    classes, probabilities = classify(image, net, window, step)
3    clusters = clustering(image, probabilities, step + space)
4    result = draw_rectangles(image, clusters)
5    return result
6
7  # Vraca dvije matrice. Prva matrica na poziciji (x, y) sadrzi
8  # klasu slikovnog okna oko (x, y) dimenzija window x window,
9  # a druga matrica vraca vjerojatnost vracene klase. Metoda
10 # se krece po slikovnim oknima u razmaku step.
11 def classify(image, net, window, step)
12
13 # Vraca listu koja sadrzi liste svih koordinata unutar jedne grupe.
14 def clustering(image, probabilities, space)
15
16 # Na originalnu sliku dodaju se pravokutnici oko svake grupe.
17 def draw_rectangles(image, clusters)
```

Izvorni kod 6.1: Glavna metoda za lokalizaciju

Vidimo da se ovaj algoritam za lokalizaciju sastoji od dva koraka. Sažeta implementacija prvog koraka je u nastavku, a prikaz nekoliko koraka je dan na slici 6.2 za $window = 3, step = 1$ i sliku dimenzija 7×7.

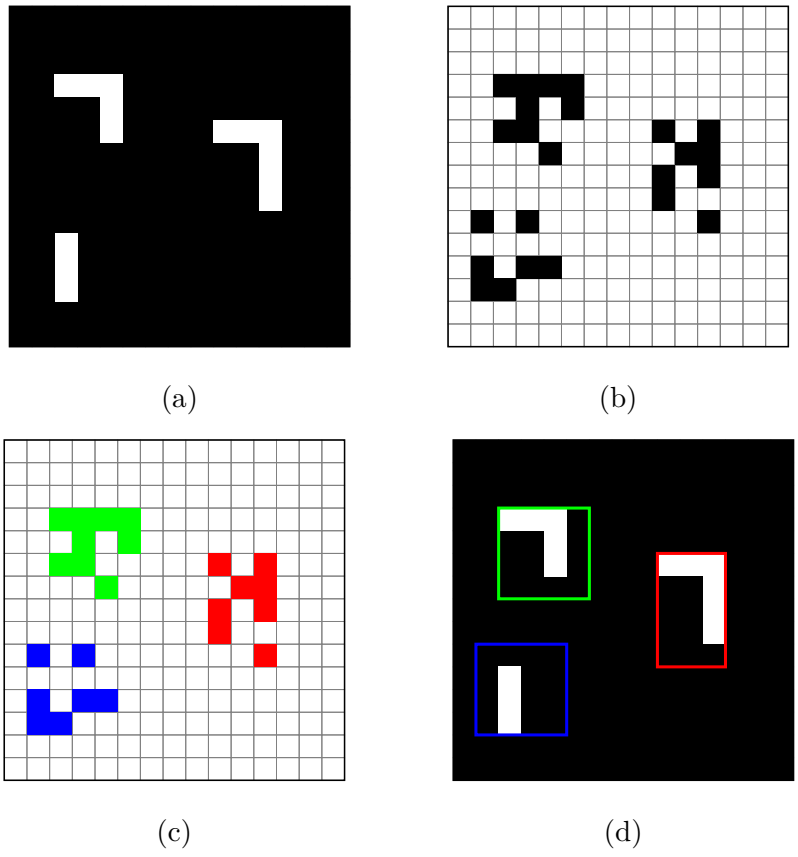
```
1  def classify(image, net, window, step):
2    rows, cols = dimension(image)
3    classes = empty_matrix(rows, cols)
4    probabilities = empty_matrix(rows, cols)
5
6    for row in range(0, rows - window, step):
7      for col in range(0, cols - window, step):
8        # sredina prozora
9        x, y = row + window / 2, col + window / 2
10
11       # izvuceno slikovno okno iz originalne slike
12       patch = image[row : row + window, col : col + window]
13
14       # dobij od mreze klasu i vjerojatnost za nju, i spremi
15       classes[x, y], probabilities[x, y] = net(patch)
16
17   return classes, probabilities
```

Izvorni kod 6.2: Prvi korak lokalizacije



Slika 6.2: 1., 2., 14. i 25. (zadnji) koraci prvog dijela algoritma (metoda *classify*)

Sada kada imamo matricu *probabilities*, iskoristimo je da bismo našli grupe unutar slike, oko kojih ćemo onda nacrtati pravokutnik i time završiti s lokalizacijom. U nastavku možete vidjeti jednostavniju implementaciju drugog dijela lokalizacije, a na slici 6.3 možete vidjeti ilustraciju cijele lokalizacije.



Slika 6.3: Ilustracija algoritma za lokalizaciju: (a) sirova slika, (b) vjerojatnosti za svaki prozor, (c) nađene grupe (svaka grupa je svoje boje) i (d) oko svake grupe je dodan pravokutnik

```
1  # Vraća True ako je koordinata (x, y) unutar slike, i nije
2  # u listi visited
3  def valid(image, x, y, visited)
4
5  # Vraća susjedstvo koordinate (x, y): svaki (x', y') za koji
6  # vrijedi: |x - x'| <= space i |y - y'| <= space
7  def neighbourhood(x, y, space)
8
9  # Vraća True ako je koordinata (x, y) već unutar neke grupe.
10 def partOfGroup(x, y, groups)
11
12 # Proširuje se oko (x, y) koristeći algoritam pretraživanja u širinu
13 def expand(probabilities, (x, y), space):
14   rows, cols = dimension(probabilities)
15   queue = []      # Sluzi kao red u algoritmu - za posjetiti
16   visited = []   # Rezultat
17
18   queue.push((x, y))
19   visited.push((x, y))
20   while len(queue) > 0:
21     (x', y') = queue.pop()
22     for xy in neighbourhood(x, y, space):
23       # Ako je valjana koordinata i vjerojatnost je pozitivna
24       if valid(image, x, y, visited) and probabilities[x, y] > 0.0:
25         queue.push((x, y))
26         visited.push((x ,y))
27
28   return visited
29
30 # Vraća grupe na slici.
31 def clustering(image, probabilities, space):
32   rows, cols = dimension(image)
33
34   result = []
35
36   for row in range(0, rows):
37     for col in range(0, cols):
38       if not partOfGroup(row, col, result):
39         group = expand(probabilities, (row, col), space)
40         if len(group) > 0:
41           result.append(group)
42
43   return result
```

Izvorni kod 6.3: Algoritam za grupiranje

Konceptualno, grupiranje se svodi na algoritam pretraživanja u širinu za svaki piksel. Ideja je za svaki piksel za koji ne znamo kojoj grupi pripada proširiti se rekurzivno na susjedne piksele. Proširujemo se samo na one piksele na kojima je vjerojatnost p pozitivan broj. Gornja implementacija dodatno dozvoljava proširivanje na sve susjede koji su unutar fiksnog razmaka.

6.2.2. Lokalizacija translacijskim autoenkoderom

U prethodnom odjeljku provodimo lokalizaciju klasificijom slikovnih okana. Bolja ideja je iskoristiti parametre p, x i y koje dobijemo iz translacijskog autoenkodera, te metodom glasanja odredimo središte znamenke. Ovakva metoda naziva se Houghova transformacija.

Ideja je proširenje algoritma koji je opisan u prethodnom poglavlju. Umjesto da neko slikovno okno čije je središte (x_o, y_o) šaljemo u klasifikator i dobijemo klasu k i vjerojatnost p_k, prosljeđujemo slikovno okno translacijskom autoenkoderu. Time dobivamo vrijednosti $p_1, x_1, y_1, p_2, x_2, y_2, \ldots, p_N, x_N, y_N$ (gdje je N broj kapsula). U odnosu na korak koji smo radili kada klasificiramo slikovna okna: `probabilities[`x_o`, `y_o`] = p`, sada za svaku kapsulu (p_i, x_i, y_i) radimo: `probabililities[`x_o `-` x_i`, `y_o `-` y_i`] += 1`, tj. glasamo za $(x_o - x_i, y_o - y_i)$.

Ideja u ovom radu je malo drugačija, te je podijeljena u više koraka:

1. Za svako slikovno okno
   (a) Sakupi glasove unutar slikovnog okna
   (b) Ignoriraj sve pozicije gdje je broj glasova $<= 3$
   (c) Pribroji dobivene glasove rezultatnim glasovima
2. Sve rezultantne glasove skaliraj na raspon vrijednosti [0, 1], što nam predstavlja varijablu `probabilities` iz metode `classify`
3. Ostatak algoritma je isti kao u prethodnom poglavlju

7. Programska izvedba

Za cijelu programsku izvedbu korišten je programski jezik *Python 2.7*. Korištena je i numerička biblioteka *NumPy*, koja (između ostalog) sadrži velik broj korisnih funkcija za rad s matricama. Primjer koda prikazan je u nastavku.

```
1  import numpy as np
2  rng = np.random
3
4  # 400 primjera dimenzionalnosti 784
5  N = 400
6  features = 784
7
8  # Generirajmo podatke
9  X = rng.randn(N, features)
10
11 # Nasumicne tezine - primjera radi
12 w = rng.randn(features)
13
14 # Izracunaj Y = Xw
15 Y_ = np.dot(X, w)
```

Izvorni kod 7.1: Primjer korištenja biblioteke *Numpy*

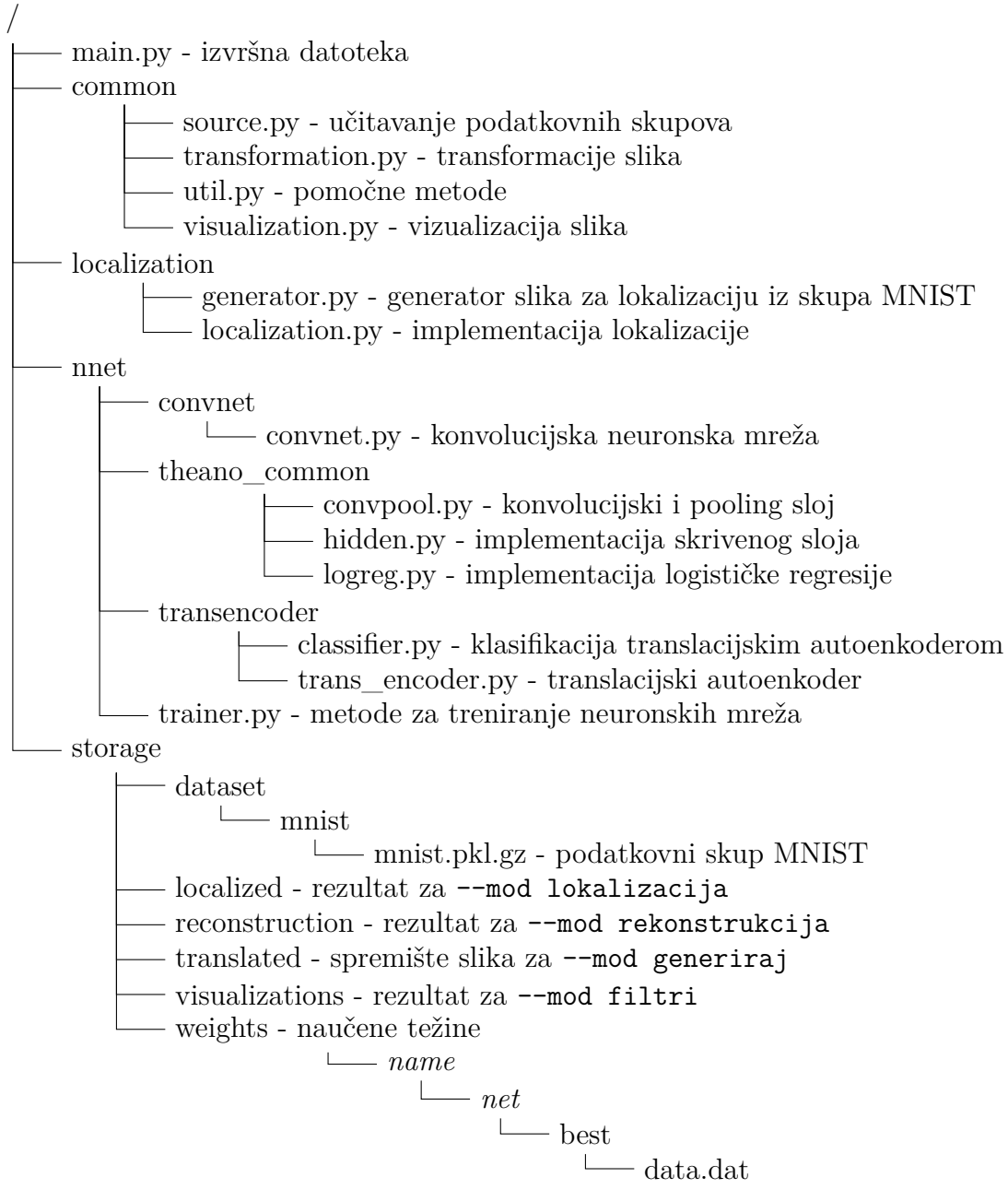
Za većinu vizualizacija korištena je biblioteka Matplotlib, koja je rađena po uzoru na Matlabove mogućnosti vizualizacije podataka.

Radi lakoće implementacije i brzine izvođenja, za implementaciju konvolucijske neuronske mreže korištena je biblioteka *Theano*. Posebnost ove biblioteke je simboličko definiranje funkcija, što omogućava biblioteci mogućnost samostalnog računanja gradijenta - bez da ga mi eksplicitno moramo definirati. Primjer učenja linearne regresije $f(X) = Xw + b$ dan je u nastavku.

```
1  import numpy as np
2  import theano
3  import theano.tensor as T
4  rng = numpy.random
5
6  # 400 primjera dimenzionalnosti 784
7  N = 400
8  features = 784
9
10 # Generiraj podatke
11 X, Y = rng.randn(N, features), rng.randn(N) * 10
12
13 # Definiraj simboličke varijable
14 x = T.matrix("x")
15 y = T.vector("y")
16 w = theano.shared(rng.randn(features), name="w")
17 b = theano.shared(0., name="b")
18
19 output = T.dot(x, w) + b
20 cost = (output - y) ** 2
21
22 # Funkcija koja racuna gradijente
23 gw, gb = T.grad(cost, [w, b])
24
25 # Funkcija koja azurira tezine
26 train = theano.function(
27             inputs=[x,y],
28             outputs=[output, cost],
29             updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))
30
31 # Funkcija koja izracuna izlaz za predani ulaz
32 model = theano.function(inputs=[x], outputs=output)
33
34 for i in range(10000):
35     output, error = train(X, Y)
36     print ("{}. iteracija, greska {}".format(i, error))
```

Izvorni kod 7.2: Primjer korištenja biblioteke *Theano*

Na slici 7.1 prikazana je struktura direktorija. Možemo vidjeti da u direktorij `storage` spremamo podatkovne skupove, rezultate lokalizacije, rekonstrukcije, generirane translatirane slike, vizualizacije filtara i naučene težine neuronskih mreža.



Slika 7.1: Struktura direktorija

7.1. Pokretanje

Cijelo tekstualno sučelje prema korisniku implementirano je kroz datoteku `main.py`. Cijela implementacija sastoji se od više dijelova:

1. Učenje neuronske mreže:

   Ove naredbe će pokrenuti učenje odabrane neuronske mreže. Na ekranu će

se ispisivati trenutno stanje (broj iteracija, trenutna pogreška mreže). Treniranje se bilo kada može prekinuti prekidom programa (`CTRL + C`), kada se trenutne težine mreže spremaju u direktorij `storage/weights/ime_mreže`.

   (a) Konvolucijska neuronska mreža:
   `python main.py --name convnet --net convnet --mod treniraj --skup mnist`

   (b) Translacijski autoenkoder:
   `python main.py --name transencoder --net transencoder --mod treniraj --skup mnist`

   (c) Klasifikacija pomoću translacijskog autoenkodera:
   Preduvjet za ovu naredbu je naučen translacijski autoenkoder (naredba iznad).
   `python main.py --name transencoder_classifier --net transencoder_classifier treniraj --skup mnist`

2. Vizualizacija filtara mreže:
   Ove naredbe će učitati spremljene težine odabrana neuronske mreže i u direktorij `storage/visualizations/ime_mreže` spremiti slike koje vizualiziraju filtre mreže. Preduvjet za ove naredbe je naučena mreža.

   (a) Filtri konvolucijske neuronske mreže:
   `python main.py --name convnet --net convnet --mod filtri`

   (b) Filtri translacijskog autoenkodera:
   `python main.py --name transencoder --net transencoder --mod filtri`

3. Prije izvršavanja lokalizacije nad skupom *MNIST*, želimo generirati veće slike koje sadrže više manjih znamenki iz skupa *MNIST*, nasumično razmještene na slici. Naredba u nastavku generira 100 slika dimenzija 250×250, gdje svaka slika sadrži 5 nasumično odabranih znamenki iz skupa *MNIST*.
   `python main.py --mod generiraj --skup mnist --M 100 --K 5 --size 250`

4. Ove naredbe će učitati spremljene težine odabrane neuronske mreže i 100 slika generiranih gornjom naredbom.
   Nakon toga će pokrenuti lokalizaciju i za svaku sliku će spremiti 4 slike u direktorij `storage/localization/ime_mreže`:

- `XY_raw.png` - originalna slika
- `XY_probs.png` - vjerojatnost za klasu za svaki prozor
- `XY_class.png` - boja klase za svaki prozor (legenda boja je gore lijevo, od 0 do 9)
- `XY_local.png` - rezultat lokalizacije

Potrebne naredbe su u nastavku.

(a) Lokalizacija koristeći konvolucijsku neuronsku mrežu (klasifikacija):
```
python main.py --name convnet --net convnet --mod lokalizacija
--lokalizacija klasifikacija --N 100
```

(b) Lokalizacija koristeći translacijski autoenkoder (Hough):
```
python main.py --name transencoder_classifier
--net transencoder_classifier --mod lokalizacija
--lokalizacija hough --N 100
```

8. Rezultati

U ovom poglavlju će biti opisan podatkovni skup korišten u ovom radu, dobiveni rezultati klasifikacije konvolucijskom neuronskom mrežom i translacijskim autoenkoderom te dobiveni rezultati lokalizacije klasifikacijom slikovnih okana i houghovom transformacijom izlaza translacijskog autoenkodera.

8.1. Podatkovni skup - MNIST

Skup MNIST (engl. *Mixed National Institute of Standards and Technology*) je skup rukom pisanih znamenki od 0 do 9. Mali dio skupa prikazan je na slici 8.1. Na ovom skupu se najčešće evaluiraju metode za klasifikaciju slika, zbog čega na stranici originalnog skupa[1], autori održavaju popis dobivenih rezultata za razne klasifikatore. Skup sadrži 60000 slika za treniranje i 10000 slika za testiranje. Slike su dimenzija 28×28, centrirane i raspon vrijednosti je $[0, 255]$.

Slika 8.1: 100 nasumično odabranih znamenki iz skupa MNIST

[1] `http://yann.lecun.com/exdb/mnist/`

S obzirom na to da je odabran Python kao glavni jezik korišten u ovom radu, koristio sam inačicu podatkovnog skupa[2] koja je njemu prilagođena. Ovakva inačica se jednostavno može učitati uz pomoć koda u nastavku.

```
1  import cPickle
2  import gzip
3
4  def loadMnist(path):
5      f = gzip.open(path, 'rb')
6      return cPickle.load(f)
7
8  # odvojeni skupovi za treniranje, validaciju i testiranje
9
10 # skup za treniranje sadrži 50 000 slika, skup za validiranje
11 # 10 000 slika i skup za testiranje 10 000 slika
12 train_set, valid_set, test_set = loadMnist('mnist.pkl.gz')
13
14 # svaki skup sadrži par (slike, labele)
15 X_train, y_train = train_set
```

Izvorni kod 8.1: Primjer učitavanja podatkovnog skupa MNIST

Osim što je prilagođena Pythonu, ova inačica donosi određene promjene nad originalnim skupom:

- slike su normalizirane na raspon vrijednosti $[0, 1]$, što pomaže u učenju mreže
- originalni skup za treniranje je podijeljen na 50000 slika za treniranje i 10000 slika za validaciju

8.2. Konvolucijska neuronska mreža

Konvolucijska neuronska mreža čija je arhitektura opisana i skicirana u 3. poglavlju učena je nad skupom za treniranje (50000 slika) i testirana nad skupom za testiranje (10000 slika).

Možemo vidjeti i kako se konvolucijska neuronska mreža ponaša ako:

- Učimo i testiramo nad originalnim skupovima

[2] `http://deeplearning.net/data/mnist/mnist.pkl.gz`

– Učimo na originalnom skupu, a testiramo nad slikama nad kojima je primijenjena translacija $\Delta = \mathcal{N}(\mu, \sigma)^2 = \mathcal{N}(0, 3)^2 \in [0, 3]^2$

– Učimo i testiramo nad slikama nad slikama nad kojima je primijenjena translacija $\Delta = \mathcal{N}(\mu, \sigma)^2 = \mathcal{N}(0, 3)^2 \in [0, 3]^2$

Tablica u nastavku prikazuje grešku nad skupom za testiranje u sve tri opisane situacije.

Bez translacija	0.91%
Translacije samo na skupu za testiranje	34.6%
Translacije i na skupu za učenje i na skupu za testiranje	2.69%

Tablica 8.1: Pogreške konvolucijske neuronske mreže nad skupom za testiranje

Primjećujemo velik utjecaj translacije kada učimo na originalnim slikama, a testiramo na translatiranim. Ipak, ako i učimo na translatiranim slikama, možemo vidjeti da mreža djelomično uspije naučiti klasificirati takve slike. Svih 91 netočno klasificiranih slika bez translacija prikazane su na slici u nastavku.

Slika 8.2: Netočno klasificirane znamenke

8.3. Translacijski autoenkoder

Translacijski autoenkoder učen je s parametrima:

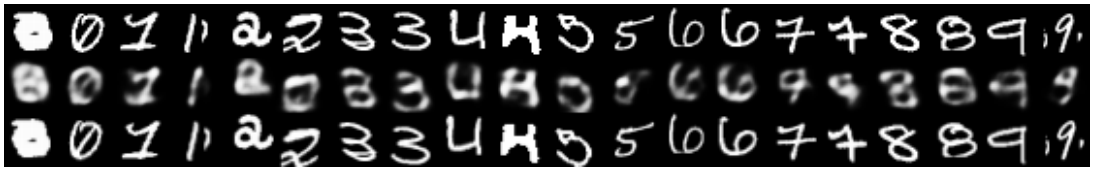
- Broj kapsula: 100
- Broj neurona u sloju za prepoznavanje (po kapsuli): 10
- Broj neurona u generativnom sloju (po kapsuli): 10
- Faktor učenja: $\alpha = 0.01$
- Parametri normalne distribucije (za translaciju): $\mu = 0$, $\sigma = 3$

Ovakva mreža nakon 500 iteracija nad 50000 slika iz skupa MNIST nauči parametre modela s rekonstrukcijskom pogreškom:

$$L = \frac{1}{50000} \cdot \sum_{x=1}^{28} \sum_{y=1}^{28} (Y_{xy} - Y'_{xy})^2 = 14.77$$

Na slikama 8.3 i 8.4 vidimo po dvije slike za svaku znamenku na kojima je rekonstrukcijska pogreška minimalna / maksimalna.

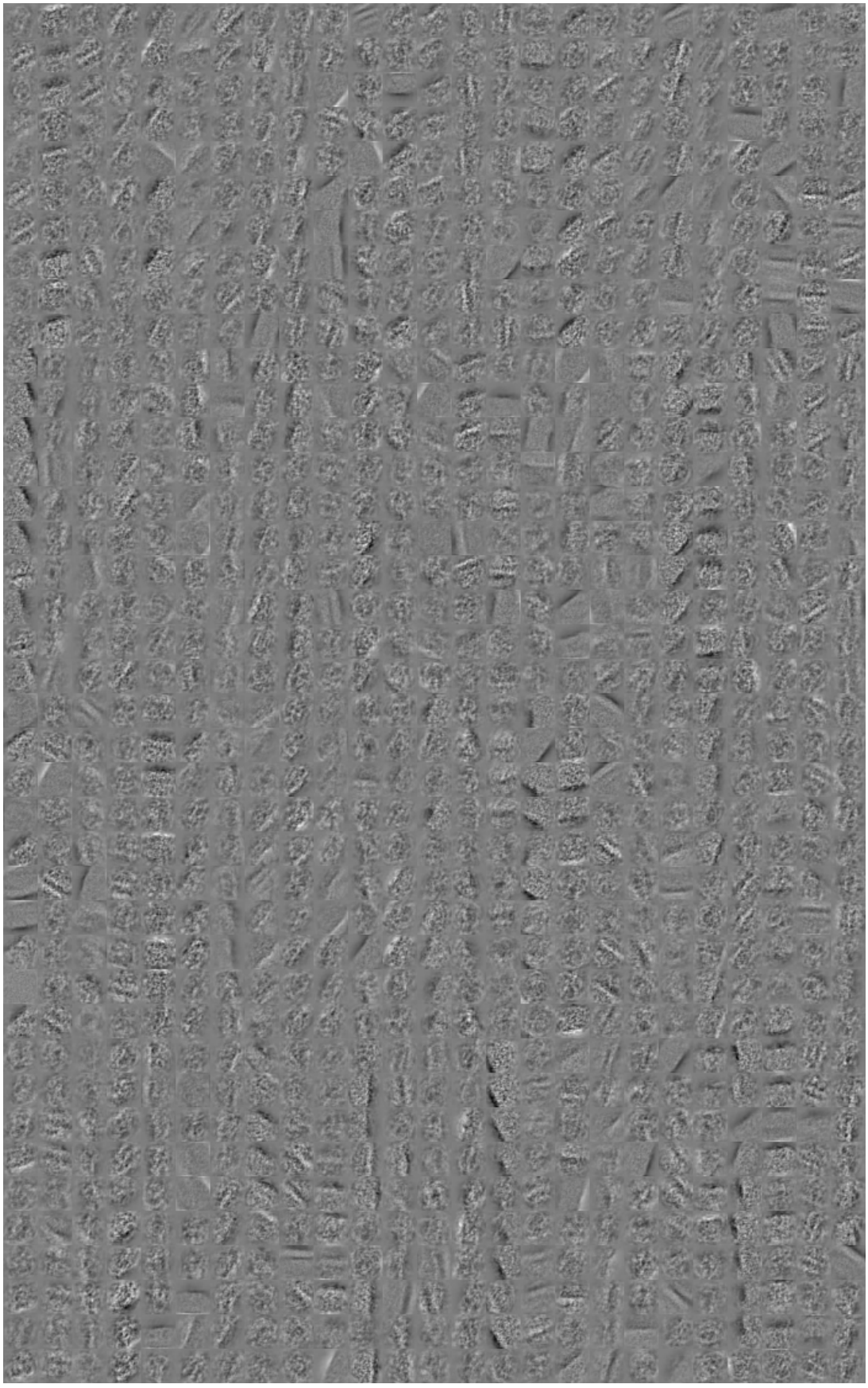
Slika 8.3: Znamenke s najmanjom rekonstrukcijskom pogreškom
Prvi red: ulazna slika, **Drugi red**: rekonstruirana slika, **Treći red**: izlazna slika



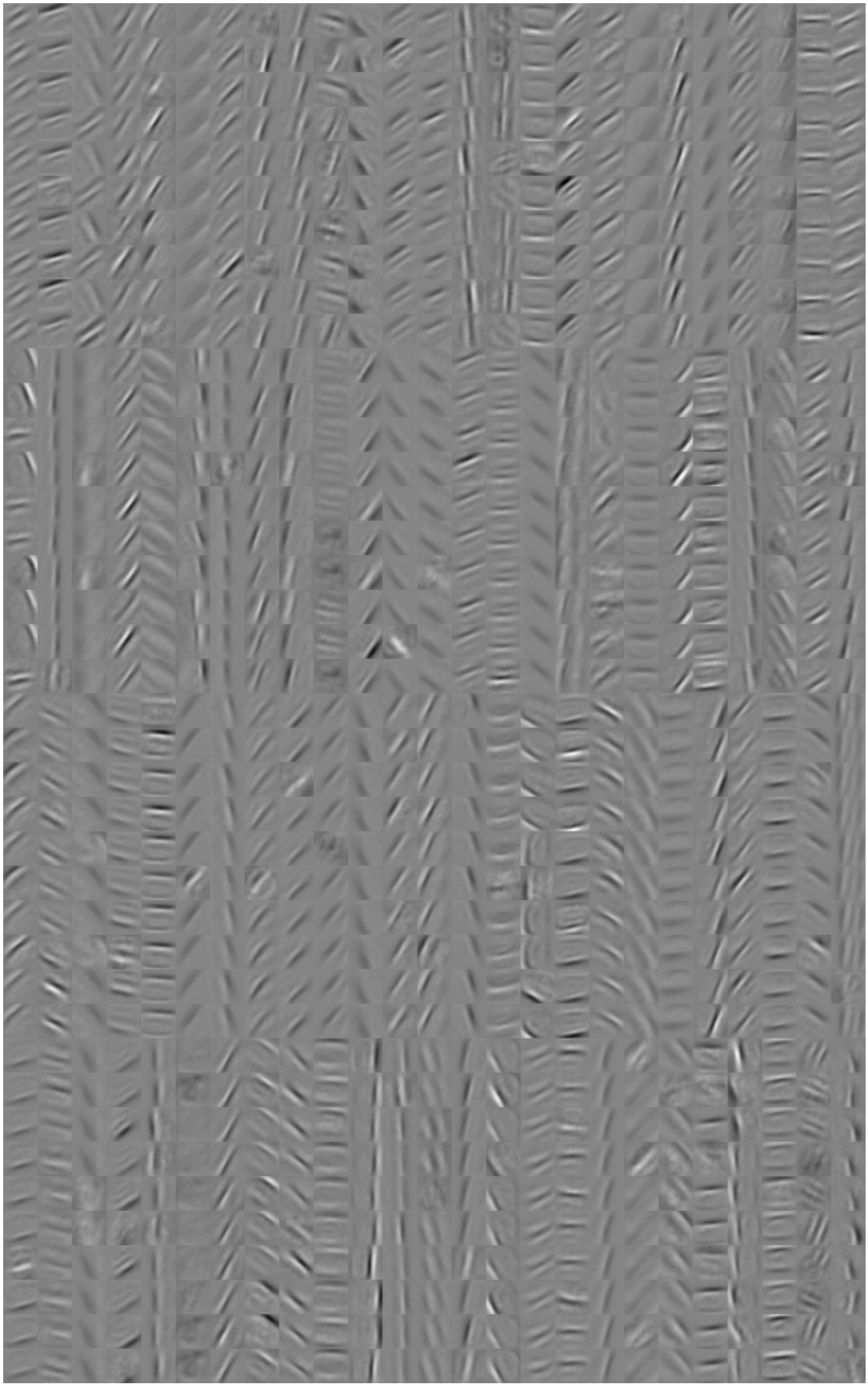
Slika 8.4: Znamenke s najvećom rekonstrukcijskom pogreškom
Prvi red: ulazna slika, **Drugi red**: rekonstruirana slika, **Treći red**: izlazna slika

Greška klasifikacije objekata ovisi o primijenjenoj translaciji nad slikama, i kreće se između 70% i 95%.

Dodatno, na slikama 8.5 i 8.6 vidimo naučene težine u sloju za prepoznavanje (W_{xh}) i generativnom sloju (W_{hy}).



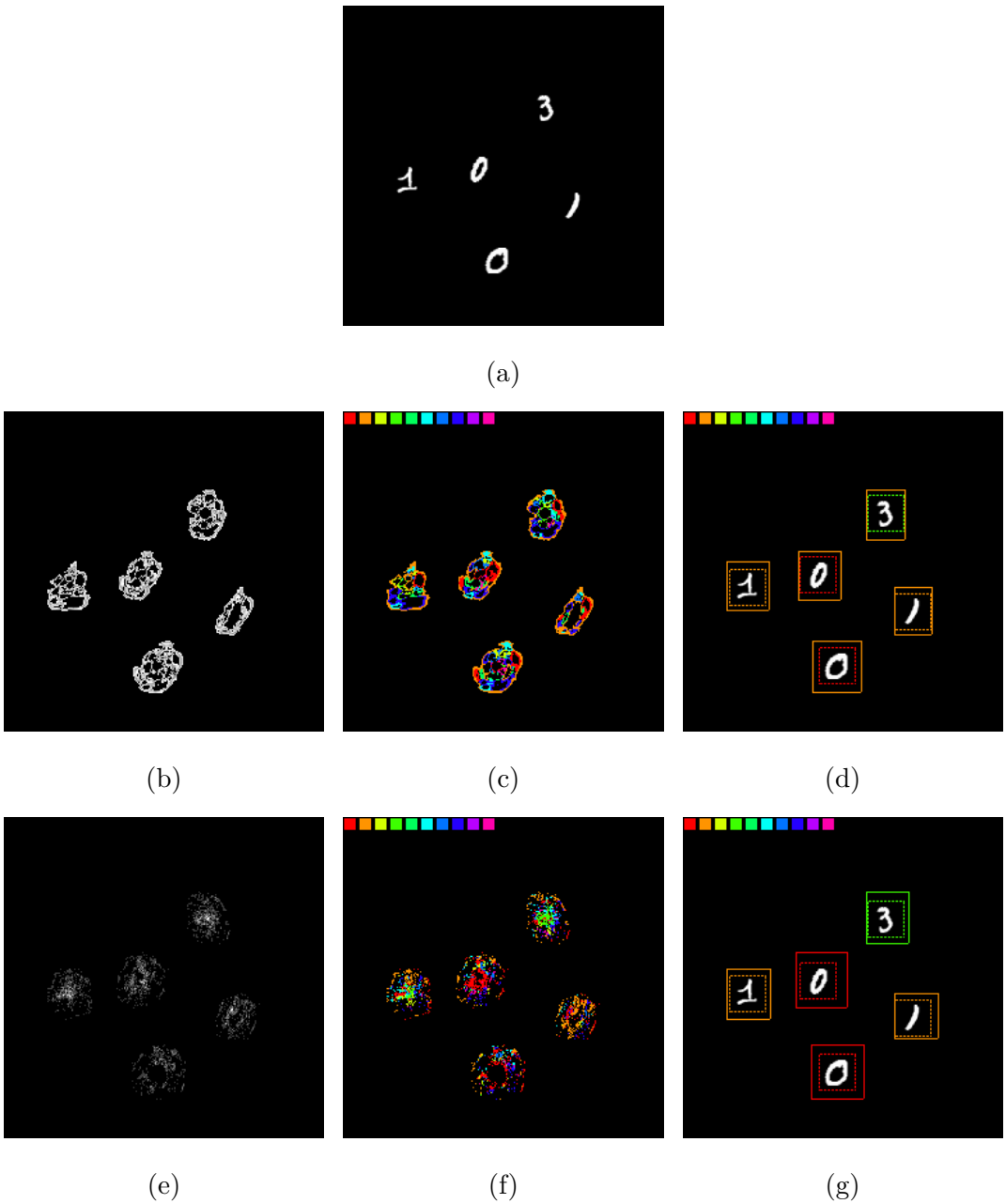
Slika 8.5: Težine sloja za prepoznavanje (W_{xh})



Slika 8.6: Težine generativnog sloja (W_{hy})

8.4. Lokalizacija

Na slici u nastavku su prikazani rezultati lokalizacije koristeći oba opisana algoritma u 6. poglavlju.



(a) (b) (c) (d) (e) (f) (g)

Slika 8.7: Prvi red: (a) originalna slika, **Drugi red**: rezultati dobiveni klasifikacijom slikovnih okana (konvolucijska neuronska mreža), **Treći red**: rezultati dobiveni translacijskim autoenkoderom i Houghovom transformacijom
(b) (e) Vjerojatnosti klase slikovnog okna, (c) (f) Klasa slikovnog okna, (d) (g) iscrtkano je točno označena znamenka, puna crta je rezultat lokalizacije (na vrhu je legenda boja za znamenke 0-9)

Rezultati u drugom redu dobiveni su klasifikacijom slikovnih okana pomoću konvolucijske neuronske mreže, a rezultati u trećem redu dobiveni su Houghovom metodom i translacijskim autoenkoderom.

Ono što primjećujemo je da prva metoda daje visoke vjerojatnosti i kada prozor pokriva samo dio znamenke, dok druga metoda najveće vjerojatnosti ima oko sredine znamenke, što je u skladu s očekivanjem. Ipak, ono što bismo htjeli postići s drugom metodom je da translacijski autoenkoder izračuna točan pomak (x, y) u sredinu znamenke za sva slikovna okna, što vidimo da nije slučaj. Razlog tome je što je translacijski autoenkoder učen na male translacije $\Delta = \mathcal{N}(0, 3)^2$, a učenje kod većih translacija dovodi i do veće rekonstrukcijske pogreške.

Dodatno, klase slikovnih okana kod prve metode ne možemo iskoristiti za klasifikaciju cijelog lokaliziranog dijela, jer i kod manjih translacija konvolucijska neuronska mreža griješi u klasifikaciji. S druge strane, kod druge metode vidimo da je dobar dio središnjim slikovnih okana točno klasificiran, zbog čega te rezultate možemo koristiti i za klasifikaciju. Ovo je vidljivo u matricama konfuzije (engl. *confusion matrix*) prikazane u nastavku.

	0	1	2	3	4	5	6	7	8	9
0	0	68	0	0	0	0	0	0	0	0
1	0	39	0	0	0	0	0	0	0	0
2	0	28	0	0	0	0	0	0	0	0
3	0	58	0	0	0	0	0	0	0	0
4	0	41	0	0	0	0	0	0	0	0
5	0	11	0	0	0	0	0	0	0	0
6	0	62	0	0	0	0	0	0	0	0
7	0	47	0	0	0	0	0	0	0	0
8	0	32	0	0	0	0	0	0	0	0
9	0	64	0	0	0	0	0	0	0	0

Tablica 8.2: Matrica zabune za lokalizaciju klasifikacijom slikovnih okana

	0	1	2	3	4	5	6	7	8	9
0	63	1	0	0	0	0	0	1	0	0
1	0	68	0	0	0	0	0	0	0	0
2	21	0	4	0	0	0	0	0	0	0
3	29	2	0	11	0	2	0	12	0	0
4	20	14	0	0	0	0	0	7	0	0
5	3	8	0	0	0	0	0	0	0	0
6	30	10	0	0	0	0	23	0	0	0
7	0	15	0	0	0	0	0	31	0	0
8	27	0	0	0	0	3	0	0	0	0
9	4	7	0	0	0	0	0	29	0	22

Tablica 8.3: Matrica zabune za lokalizaciju houghovom transformacijom

Možemo odrediti preciznost lokalizacije kao srednju vrijednost preciznosti lokalizacije za svaku sliku pojedinačno, koju izračunavamo izrazom u nastavku.

$$A = \sum_{i=1}^{K} \frac{P(O_D \cap O_T)}{P(O_D \cup O_T)}$$

K je broj znamenki na slici, O_D je detektirano slikovno okno, O_T je točno slikovno okno, P je površina. Tablica u nastavku prikazuje preciznost oba algoritma za lokalizaciju.

	Preciznost
Klasifikacija slikovnih okana	0.248
Houghova transformacija	0.241

Tablica 8.4: Preciznost lokalizacije

Bitno je promotriti i koliko slikovnih okana algoritam za lokalizaciju označi, a zapravo nisu znamenke (engl. *false positive, FP*), te koliko ne označi, a znamenke su (engl. *false negative, FN*). Ovo je prikazano u tablici u nastavku.

	FN	FP
Klasifikacija slikovnih okana	20	0
Houghova transformacija	33	10

Tablica 8.5: Ukupan broj propuštenih (FN) i krivih (FP) oznaka

9. Zaključak

U okviru ovog rada ostvarena je implementacija jednostavne konvolucijske neuronske mreže za klasifikaciju znamenki iz skupa MNIST, te translacijski autoenkoder i klasifikacija znamenki uz pomoć translacijskog autoenkodera. Drugi dio rada opisuje dvije izvedbe lokalizacije: klasifikacijom slikovnih okana i houghovom transformacijom. Dobiveni rezultati pokazuju da je konvolucijska neuronska mreža nadmočnija za klasifikaciju slika, ali je pokazan problem translacije kod konvolucijske neuronske mreže. Dobivena je i slična preciznost lokalizacije objekata na slici, ali je pokazana dominacija translacijskog autoenkodera i houghove transformacije za klasifikaciju lokaliziranog objekta.

U budućem radu brzina klasifikacije i lokalizacije bi se mogla poboljšati korištenjem grafičkog procesora. Dodatno, mogu se uvesti potpune afine transformacije (rotacija, skaliranje, smik). Može se prilagoditi postupak lokalizacije za realne slike, kao što su slike iz skupa FER-MASTIF.

Literatura

Filip Krnjić. *Raspoznavanje objekata dubokim neuronskim mrežama*, 2009. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/krnjic09bs.pdf.

David Stutz. *Understanding Convolutional Neural Networks*, 2015. URL https://github.com/davidstutz/seminar-convolutional-neural-networks/blob/master/paper/seminar.pdf.

Stanford University. Cs231n: Convolutional neural networks for visual recognition, June 2015. URL http://cs231n.github.io/.

Vedran Vukotić. *Raspoznavanje prometnih znakova neuronskim mrežama*, 2014. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/vukotic14ms.pdf.

Sida Wang. *Learning to Extract Parameterized Features by Predicting Transformed Images*, 2011. URL http://nlp.stanford.edu/~sidaw/home/_media/papers:thesis.pdf.

Dalbelo Bašić B. i Šnajder J. Čupić M. *Umjetne neuronske mreže, Službeni materijali s predmeta Umjetna inteligencija*, 2012. URL http://www.fer.unizg.hr/_download/repository/UI_12_UmjetneNeuronskeMreze%5B1%5D.pdf.

Duboke neuronske arhitekture za lokalizaciju objekata

Sažetak

Konvolucijske neuronske mreže postižu vrhunske rezultate na području klasifikacije slika. Međutim, te uspjehe nije jednostavno preslikati na područje lokalizacije objekata u slikama. U ovom radu razmatramo algoritme za lokalizaciju objekata u slikama koristeći konvolucijske neuronske mreže. Osim toga, razmatramo translacijski autoenkoder kao rješenje problema koji konvolucijska neuronska mreža ima s translacijom. U konačnici, razmatramo algoritam za lokalizaciju koristeći translacijski autoenkoder. Prikazani su i komentirani dobiveni rezultati klasifikacije slika u skupu MNIST, kao i lokalizacije slika iz istog skupa.

Ključne riječi: neuronske mreže, strojno učenje, računalni vid, lokalizacija objekata, klasifikacija slika, konvolucijske neuronske mreže, translacijski autoenkoder, kapsule

Deep neural architectures for object localization

Abstract

Convolutional neural networks achive excellent results in the field of image classification. However, this success is not easily mapped to the field of object localization. In this paper, we consider algorithms for object localizations using convolutional neural networks. Furthermore, we consider translational autoencoder as a solution for problems convolutional neural network has with translations. Finally, we consider the algorithm for object localization using translational autoencoder. Results of image classification and object localizations of images in MNIST dataset are shown and discussed.

Ključne riječi: neural networks, machine learning, computer vision, object localization, image classification, convolutional neural networks, translational autoencoder, capsules