CNN na primjeru razbijanja CAPTCHA-e

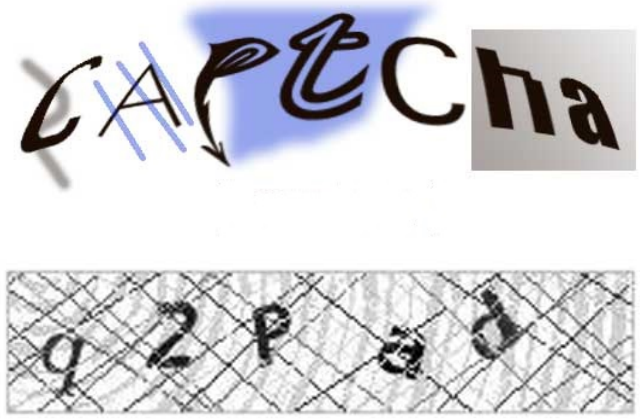
Daniel Jelušić
Sveučilište u Zagrebu
Prirodoslovno matematički fakultet
Diplomski smjer : RiM
Email: djelusic133@gmail.com

Jure Šiljeg
Sveučilište u Zagrebu
Prirodoslovno matematički fakultet
Diplomski smjer : RiM
Email: jures91@gmail.com

***Sažetak*—Razbijanje specifičnog stila CAPTCHA-e u akademske svrhe pomoću konvolucijske neuronske mreže (CNN) je dobro poznat problem. Mi ćemo se fokusirati na CAPTCHA-e duljine 5 ili 6. Na slikama će, također, biti generiran salt & pepper šum zajedno s distorzijom slova. Tehnike strojnog učenja ćemo usmjeriti k OCR-u, dok ćemo šum uklanjati image processing tehnikama u MATLAB-u.**

I. UVOD

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) vrsta je autentikacije "izazov-odgovor" koji se koristi u računarstvu da bi odredilo je li korisnik čovjek ili računalo, s ciljem sprječavanja pristupa zlonamjernim računalnim programima. Proces najčešće podrazumjeva jedno računalo (server), koji traži od korisnika da odradi jednostavan test koji računalo može generirati i ocijeniti. Pretpostavka je da drugo računalo nije u stanju riješiti taj test, pa se svaki korisnik koji unese točan odgovor se smatra čovjekom.



Slika 1: Primjer captchi

CAPTCHA testovi traže od korisnika da unese nekoliko simbola (najčešće slova i/ili brojeva) koji su prikazana na slici, koja je na neki način iskrivljena. Zbog toga se ponekad naziva "obrnuti Turingov test", jer podrazumijeva stroj koji cilja na prepoznavanje ljudi, za razliku od originalnog turingovog testa kojeg izvode ljudi da bi prepoznali računala.

Postoji nekoliko pristupa pri pokušajima da se zaobiđe CAPTCHA:

1) iskorištavanje bugova koji dopuštaju napadaču da posve zaobiđe CAPTCHA test

2) poboljšanje softvera za prepoznavanje znakova

3) korištenje jeftine radne snage za prolaženje testova

4) sirova sila - višestruki uzastopni napadi

Mi ćemo se u našim razmatranjima osloniti na točku 2, ali što zapravo govori točka 3 i što je OCR? Optičko prepoznavanje znakova, obično skraćeno na OCR (Optical Character Recognition), je mehaničko ili elektronsko pretvaranje skeniranih slika rukom, pisaćim strojem ili ispisanog teksta u strojno kodirani tekst. To se pretežno koristi kao oblik unosa podataka iz nekakvog izvornog papirnatog izvora podataka, bili to dokumenti, računi, pisma, ili bilo koji ispisani zapis. To je uobičajena metoda digitalizacije tiskanih tekstova, tako da oni mogu biti elektronski pretraživani, komprimirano pohranjeni, prikazani na mreži, te korišteni u procesima strojnog prevođenja, text-to-speech i u rudarenju podataka. OCR ujedinjuje raspoznavanje uzoraka, umjetnu inteligenciju i računalni vid.

U bilo kojoj vrsti problema gdje se traži prepoznavanje objekata, mogu se izdvojiti barem dvije stavke na koje treba obratiti pažnju. Prva je svakako segmentacija gdje se od nas traži da što bolje izdvojimo podobjekte iz nekog većeg objekta kako bi postali što upotrebljiviji i korisniji u daljnjem istraživanju. Druga i ona nama zanimljivija stavka je kvalitetan algoritam strojnog učenja(OCR) za prepoznavanje tih objekata.

Naš cilj je napraviti što bolji algoritam koji će uspješno prepoznavati CAPTCHA-e određenog stila, te ćemo pokušati da taj algoritam bude generaliziran. Potrebno je spomenuti kako će naš stil biti prilično jednostavan i instrukcionalan. Današnje CAPTCHA-e se formiraju s velikim distorzijama slova i zgusnutim razmještajem istih (otežava segmentaciju, pa i napredniji algoritmi slome zube). Mi se u ovom radu nećemo baviti takvim ekstremnim slučajevima. Za dobiti što bolji algoritam koji pristojno generalizira, smatramo da je potrebno napraviti što bolji OCR algoritam. Algoritam će, inače, svaki char smjestiti u odgovarajuću kategoriju (ukupno 62 kategorije za $0-9$, a-z te A-Z). Samu implementaciju CNN-a smo pronašli na web-u, a glavne stavke našeg CNN-a su da je to neuronska mreža čija se arhitektura sastoji od ulaznog, konvolucijskog/pooling, softmax regresijskog i izlaznog sloja. Nešto više o samoj CNN ćemo reći u nastavku.

II. SKUP PODATAKA

Skup podataka smo, u osnovi, generirali pomoću php skripte koju je bilo potrebno dodatno modificirati za naše potrebe, a ista se sastoji od slika fiksnih dimenizija koje su potom bile reprezentirane na način da stvorimo matricu koja ima dimenzije iste kao i sama slika (naravno, dimenzije podešene), a svako polje matrice odgovara boji piksela ($a_{i,j} \in [0, 1]$).

Podatke smo generirali na više načina. Za početak je ideja bila generirati 60000 CAPTCHA-i duljine 1 pomoću te skripte, te na njima trenirati, a model testirati tako da kreiramo CAPTCHA-u, obradimo sliku (segmentacija + uklanjanje šuma) i u konačnici testiramo. Uz to nas je zanimalo kako bi se naš model ponašao ukoliko bismo trening skup napravili tako da segementiramo CAPTCHA-e, predprocesiramo ih, treniramo model, testiramo model na isti način kao i u prethodnom primjeru (glavni i najbolji način!). Za kraj smo htjeli vidjeti i usporediti kako bi se naš model ponašao na skroz drukčijem trening skupu u čiju svrhu smo uzeli skup char 74k besplatno dostupan na web-u. Za svaki od navedenih slučajeva smo imali skriptu koja je automatski punila txt datoteku s ispravnim labelama našiš CAPTCHA-i.

Napomenuti valja kako je skup za učenje u prvom slučaju bio veličine 50000, dok smo trenirali na 10000, u drugom slučaju je trening skup bio veličine 20000, a testni 5000, a za kraj smo u zadnjem slučaju koristili 10000 primjeraka kod terninga i 1000 kod testa.

(a) Početna captcha

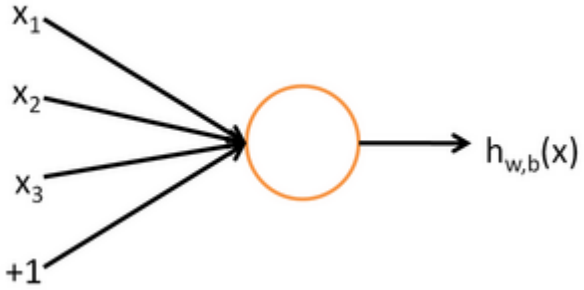
(b) Nakon odstranjivanja šuma

(c) Nakon segmentiranja

Slika 2: Odstranjivanje šuma i segmentacija

III. CNN

Osnovna gradivna jedinica neuronske mreže je neuron. Pokušajmo za početak opisati strukturu mreže, kako radi i koji su nam bitni dijelovi za gradnju modela. Najjednostavnija je, pak, mreža, se sastoji od samo jednog neurona.



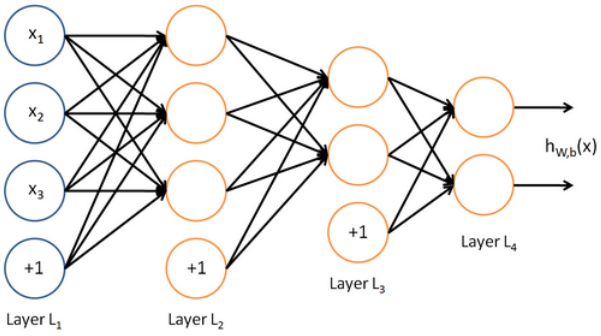
Slika 3: Jedan neuron

Dobro je spomenuti da su ulazni parametri upravo slike reprezentirane matricom fiksne veličine (u ovisnosti o trening/testu kojeg smo radili), težine (W_i) i pripadni bias ($b = 1$). Postupak je takav da se za neuron odrede težine i one se dodijele podacima koji se u tom neuronu računaju. Neuron može davati jedan ili više izlaznih rezultata (ovdje jedan). Izlaz neurona dan je sljedećim izrazom : $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, gdje se $f : \mathbb{R} \mapsto \mathbb{R}$ naziva aktivacijskom funkcijom. Najuobičajeniji oblici takve funkcije su :

1) sigmoidna funkcija - $f(z) = \frac{1}{1+e^{-z}}$,
2) ReLU funkcija - $f(z) = \max(0, z)$

Mi smo u našem radu koristili obje, ali je ReLU dala malo bolje rezultate. Rezultati predstavljaju kako taj neuron iz mreže djeluje na postupak klasifikacije (koristimo mrežu za klasifikaciju).

Sama mreža se najčešće sastoji od više slojeva (layera). Na slici je prikaz neuronske mreže s 4 sloja gdje se najljeviji zove ulazni sloj, najdesniji se zove izlazni sloj, dok se ostali slojevi zovu skriveni (imaju 2 takva - L_2 i L_3).



Slika 4: Prikaz neuronske mreže sa 4 sloja

Opisat ćemo sada glavne korake pri gradnji naše mreže. Zamislimo sljedeću situaciju. Jedan od mogućih načina kako napraviti NN jest tako što izgradimo potpuno povezanu mrežu gdje je svaki neuron prethodnog sloja povezan sa svakim neuronom sljedećeg sloja. Uzmimo sada da želimo propagirati kroz mrežu slike dimenzija 96x96. Takva propagacija je skupa jer imamo oko 10^4 ulaznih jedinica i ako uzmemo u obzir da bismo htjeli naučiti oko 100 značajki, ispada da trebamo

naučiti 10^6 parametara, te će sama propagacija unaprijed i unatrag usporavati izvršavanje za oko 10^2 puta u odnosu na sliku dimenzija 28x28. Jedno jednostavno rješenje takvog problema se nudi u restringiranju broja konekcija između ulaznih jedinica i jedinica u skrivenom sloju. To možemo ostvariti na način da dozvolimo svakoj skrivenoj jedinici (čvoru) da se spoji samo s malim brojem ulaznih jedinica. Ova ideja o "lokalnom spajanju" je inspirirana biologijom i kako su rani vizualni sustavi spojeni u biologiji. Lokalno spajanje se koristi kod konvolucijskih neuronskih mreža sa više konvolucijskih slojeva.

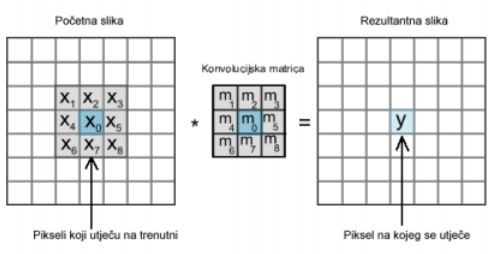
Prirodne slike imaju specifično svojstvo **stacionarnosti**. To znači da su statistički podaci jednog dijela slike gotovo identični statističkim podacima drugog dijela slike (bilo kojeg). To sugerira da značajke koje smo naučili na jednom dijelu slike možemo slobodno primijeniti na neki drugi dio slike te možemo koristiti iste značajke na svim lokacijama.

Da budemo precizniji, ako naučimo značajke na malom dijelu slike (recimo 8x8) uzetog metodom slučajnog odabira, možemo primijeniti tu naučenu značajku bilo gdje na slici! Posebno, možemo uzeti naučene 8x8 značajke i **konvoluirati** ih s većom slikom.

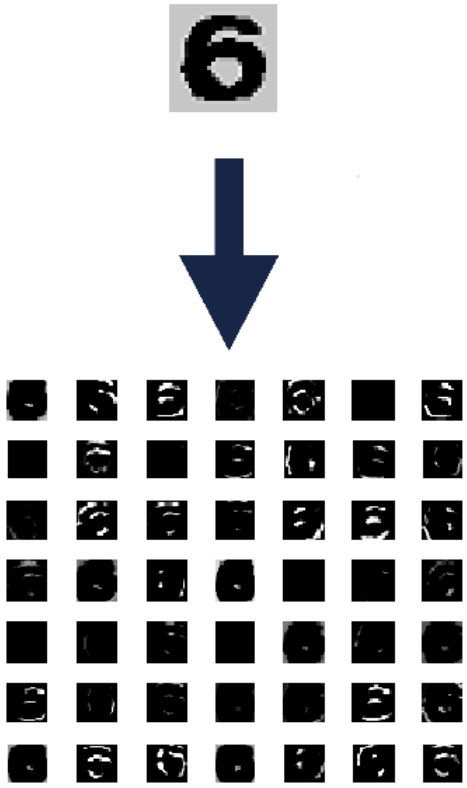
Formalno, ako dobijemo neku veliku sliku x_{velika} s r redaka i c stupaca, prvo treniramo na malim (x_{mala}) podslikama ekstrahiranim s velike slike tako što naučimo k značajki $f = \sigma(W^{(1)}x_{small} + b^{(1)})$ gdje je σ sigmoidna ($\sigma(t) = \frac{1}{1+e^{-t}}$) ili ReLU ($\sigma(t) = \max(0, t)$) funkcija, dok su $W^{(i)}$ težine, a $b^{(i)}$ biasi od vidljivih slojeva prema skrivenim. Za svaku $a \times b$ podsliku (gdje su a, b dimenzije x_m) unutar velike slike, računamo $f_s = \sigma(W^{(1)}x_m + b^{(1)})$ što nam daje $f_{konvoluirano}$, tj. $k \times (r - a + 1) \times (c - b + 1)$ niz konvoluiranih značajki.

Nakon dobivanja određenih značajki, htjeli bismo iste koristiti za klasifikaciju. U teoriji, mogu se koristiti sve ekstrahirane značajke s klasifikatorom (kao što je softmax klasifikator), ali to može biti izazovno za računanje (složenost). Pretpostavimo da smo na početku razmatrali sliku(primjer) veličine 96x96 i da smo naučili 400 značajki preko 8x8 inputa. Nakon konvolucije output je veličine $(96-8+1)*(96-8+1) = 7921$, a budući da imamo 400 značajki, rezultat će se sastojati od $400*89^2$ značajki po primjeru. Učenje kalsifikatora s inputom od oko 3 milijuna značajki može biti nezgrapno, ali također sklono overfittingu.

Za riješiti ovo prvo se treba sjetiti kako smo odlučili upotrijebiti konvoluirane značajke zato što slike imaju stacionarno svojstvo koje implicira da značajke koje su bile korisne na nekom području ostaju (vrlo vjerojatno) korisne i na drugim područjima. Stoga ako želimo opisati veću sliku, jedan prirodan pristup bi bio računanje pojedinih statističkih vrijednosti na različitim lokacijama, pa se tako npr. mogu računati srednje vrijednosti (mean) ili maksimalne vrijednosti (max) određene značajke na određenoj lokaciji u slici. Formalno, nakon što dobijemo konvoluirane značajke, odlučimo se za dimenzije područja (recimo $m \times n$) preko kojeg ćemo raditi pooling. Potom podijelimo naše konvoluirane značajke na područja veličine $m \times n$ koja se međusobno ne preklapaju, te uzimamo max ili mean vrijednosti tih područja da dobijemo



Slika 5: Operacija konvolucije

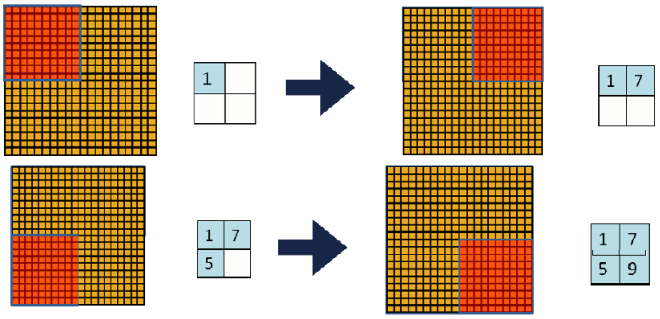


Slika 6: Prikaz konvolucije na jednom primjeru iz testnog skupa

poolirane i konvoluirane značajke. Takve značajke nad kojima je napravljen pooling sada mogu biti korištene za klasifikaciju.

Dva su dobra razloga za korištenje poolinga. Jedan je to što smanjujemo mogućnost overfittinga, a druga prednost je što ovakvim postupcima "sažimanja" smanjujemo dimenziju u usporedbi s korištenjem svih ekstrahiranih značajki (računanje postaje jendnostavnije). Ovakve postupke nazivamo mean pooling ili max pooling (u ovisnosti koju pooling operaciju koristimo, jasno). Sljedeća slika pokazuje kako se može napraviti pooling preko 4 lokacije koje se ne preklapaju :

Dobro i jako poželjno svojstvo poolinga je tzv. translacijska invarijantnost. Za razumjeti ovakvo svojstvo zamislimo serijsko spajanje pooling i konvolucijskog sloja. Postoji 8 smjerova u koje se može translatirati input slika za jedan



Slika 7: Prikaz poolinga

piksel. Ako npr. radimo max pooling pomoću 2x2 područja, 3 od 8 mogućih konfiguracija će reproducirati isti output u konvolucijskom sloju. Za max pooling pomoću 3x3 područja, ovaj broj naraste na 5/8. Ukratko, dobro je ako imamo sliku koju želimo klasificirati i ako nam ne igra ulogu ukoliko je ista malo translatirana lijevo ili desno (i dalje se dobro klasificira).

Sljedeći bitan korak je softmax regresijski sloj. Ovaj model generalizira logističku regresiju za problem klasifikacije gdje labela y može poprimiti više od 2 vrijednosti. U našem slučaju poprima izlazne podatke iz konvolucijskog/pooling sloja. Korisno nam je pošto je cilj napraviti razliku između 62 klase char-ova. U logističkoj regresiji hipoteza izgleda ovako:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}},$$

, a parametari modela, θ, su bili trenirani da minimiziraju cost funkciju. Cost funkcija je važan koncept u procesu učenja koji predstavlja mjeru koliko blizu/daleko je pojedino rješenje od optimalnoga za problem koji rješavamo.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

U softmax regresiji smo zainteresirani za klasifikaciju gdje je broj klasa strogo veći od 2, pa labela y može poprimiti k različitih vrijednosti (umjesto samo dvije). Tada u našem trening skupu $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, imamo da je $y^{(i)} \in \{1, 2, \ldots, k\}$. Za naše potrebe $k = 62$ različite klase.

Za dani input x, želimo da naša hipoteza procjenjuje vjerojatnost da $p(y = j|x) \forall j = 1, \ldots, k$. Npr. želimo procijeniti vjerojatnost da oznaka klase poprimi svaku od k različitih vrijednosti. Tada će naša hipoteza davati za output k - dimenzionalni vektor čija je suma elemenata jednaka 1 davajući nam k procjena vjerojatnosti. Konkretno, naša hipoteza $h_\theta(x)$ poprima oblik:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)}; \theta) \\ p(y^{(i)} = 2|x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

gdje su $\theta_1, \theta_2, \ldots, \theta_k \in \Re^{n+1}$ parametri našeg modela.

Važno bi bilo napomenuti kako se u našoj mreži cost funkcija mijenja tokom propagacije unaprijed, a na izlazu iz softmax sloja njena formula izgleda ovako :

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y^{(i)} = j\right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right] + \\ + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n} \theta_{ij}^2$$

Pri gradnji modela smo koristili backpropagation algoritam (propagiranje greške unatrag) sa stohastičkim gradijentnim spustom[1]. To je dodatna optimizacija kojom aproksimiramo derivacije umjesto direktnog računanja i koja je iznimno važan korak za brže učenje mreže.

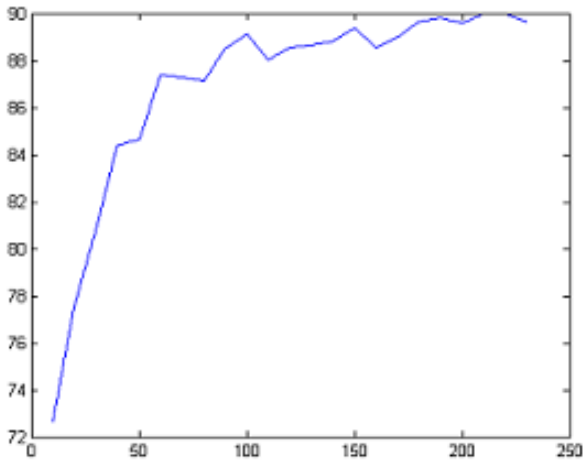
IV. TESTIRANJE I REZULTATI

U ovom odjeljku ćemo pokušati kvantificirati koliko je CNN bio dobar klasifikator za naš problem. Kao evaluator za točnost modela smo koristili k-struku cross validaciju (gdje je bilo $k = 5$ i $k = 6$).

Za samo treniranjem glavne mreže je korištena skripta gdje se koristi 5-fold CV nad specifičnim skupom koji se sastoji od 5000 CAPTCHA-i duljine 5 s defaultnom distorzijom i šumom, vrši se segmentacija pozivom funkcije segmentiranje (očisti šum i segmentira sliku na pojedine char-ove), normaliziraju se slike (za oznake skupa pogledati ovo). Nakon toga se pokrene skripta koja prepoznaje CAPTCHA-e duljine 5. Sama skripta funkcionira tako da učita sve CAPTCHA-e koje želimo testirati i na svakoj od njih pozove funkciju segmentiranje koja predprocesira sliku tako da ukloni šum koristeći funkciju uint8 i svojstva šuma (vrijednosti pojedinog pixela). Potom se slike segmentiraju da izvučemo char-ove (BoundingBox) i sve je testirano sa CAPTCHA-ma veličine 5. Napravi se "bijeli padding" (što se pokaže vrlo važnim) gdje se svaka segmentirana slika pojedinog chara nadopuni bijelim pixelima do veličine 34×34 bez obzira na prethodnu veličinu (važno kod razlikovanja malih i velikih char-ova) te se na kraju resizea na 28×28. Svaka slika se potom normalizira (oduzme se očekivanje i podijeli s varijancom), pretvori u "stvarnu" kategoriju koja odgovara pojedinom char-u (1-62) i za kraj se računa uspješnost.

Za kraj spomenimo još učinak. Probali smo nekliko varijanti treninga i testiranja. Bitno je spomenuti kako je prvotna ideja bila da stvorimo CAPTCHA-e duljine 1 pomoću php skripte (bez predprocesiranja!), na njima treniramo mrežu (6-fold CV), potom generiramo CAPTCHA-e duljine 6, predprocesiramo ih i pokušamo testirati. Međutim, mreža se pokazala dosta osjetljivom na generalizaciju (visok overfitting). Naime, u trening skupu je vrlo mala varijacija između pojedinih slova (bez obzira na distorziju) i kao takvu je mreža nauči. Kod testnog skupa je, također, mala varijacija, ali drukčijeg tipa nego ona iz trening skupa, tako da je postotak prepoznatih bio dosta loš (27%). Slike su izgledale dosta

slično, a opet, očito, dosta različito. Nakon toga smo trening skup napravili od CAPTCHA-i koje su predprocesirane, testni skup na isti način i uspješnost je narasla na oko 75% za CAPTCHA-u duljine 5. Valja napomenuti kako je 5-fold CV (trening i test skup generirani segmentacijom i predprocesiranjem) davao uspješnost u prosjeku od 88.75% ([90.240000000000000,89.420000000000000
,88.500000000000000,87.560000000000000,88]).



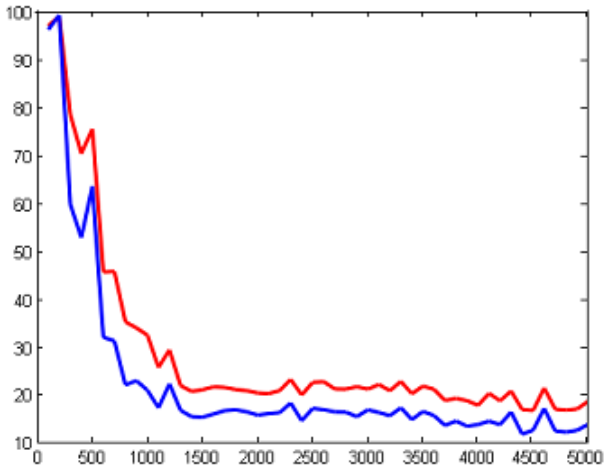
Slika 8: Prikaz točnosti po iteracijama za jednu instancu 5-fold CV

6-fold CV je na trening i test skupu generiranom pomoću skripte davala rezultate u prosjeku od 99.1167%. Radili smo također i sitne varijacije s početnim težinama i pripadnim biasima gdje smo ih generirali normalnom i uniformnom razdiobom i nema neke posebne razlike (90.16% naspram 90.24% u korist uniformne razdiobe). Jedan zanimljivi prikaz krivulje učenja se može vidjeti na sljedećoj slici.

Također smo pokušali trenirati skup 74k na 10k slučajno odabranih primjeraka. Testiranje je pokazalo uspješnost od 84% kod CV, dok je za testiranje CAPTCHA-e loša - ispod 5%. Očekivano je jer je jasno da ćemo bolje rezultate dobiti na kompatibilnijem trening skupu. Uz ovaj glavni dio (5-fold CV s uspješnosti od 75%) smo napravili i prikaz greške ponašanja greške obzirom na iteracije :

V. ZAKLJUČAK

Prilikom samog kodiranja smo dosta koristili vektorizaciju (što je jedna dobra značajka MATLAB-a) koja nam je dodatno pomogla pri dobivanju brzine. Sama CNN se pokazala kao dobar klasifikator koji prilično dobro generalizira obzirom na naša očekivanja, a dodatno su uspješnosti pripomogli CV te backpropagation s gradijentnim spustom. Ova tema je dosta plodna i uvijek se može bolje i više istraživati. Uobičajena varijacija na naš model je dodavanje novih skrivenih slojeva. Primjer za to je Yann LeCunova CNN za prepoznavanje rukom pisanih znamenki (sličan problem) koja ima 5 slojeva. Valja napomenuti kako slični problemi u OCR-u koriste razne



Slika 9: Prikaz krivulje učenja kod skupa generiranog segmentiranim captchama. Crvena krivulja predstavlja grešku na testnom skupu, a plava na trening skupu

pristupe i algoritme uz poprilično dobre rezultate(iako je neka varijacija NN, zapravo, najčešća). Možemo spomenuti kako se uz to koristi još i Linearna regresija[2], Linearni SVM [3], kNN[4] i drugi. Naravno, bilo bi također zanimljivo vidjeti kako bi naša mreža radila da smo koristili tehnike strojnog učenja pri samoj segmentaciji koja je najčešće i najvažnija (pogotovo kod jako zgusnutih slova i velikog šuma).

Za veliki dio implementacije konvolucijske neuronske mreže smo koristili razne tutorijale i opširan opis problema koju je na web-u omogućilo američko sveučilište Stanford. Pri radu smo koristili brojne optimizacije s prethodno navedenih stranica. Našom implementacijom smo uspjeli dobiti pristojna rješenja i vrlo dobre postotke (uspješnost na samo 5 epoha s 5-fold CV-om je bila oko 75 %).

VI. LITERATURA

[0]Tutorial sa Stanforda
[1]SGD
[2]Linearna regresija
[3]Linearni SVM
[4]kNN