Математичка гимназија

МАТУРСКИ РАД

- из програмирања -

Препознавање писаних математичких једначина

Ученик: Филип Бајрактари, IVa

Ментор: мр Стефан Спалевић

Београд, јун 2022.

Садржај

1 Увод **2**

2 Метод **3**
2.1 Богаћење базе података 3
2.2 Обрада слике 4
2.2.1 Претварање слике из *RGB* у *Grayscale* 4
2.2.2 Уклањање шумова са слике 5
2.2.3 Бинаризација 5
2.3 Сегментација 6
2.4 Потенцијални проблеми приликом сегментације 8

3 Класификација карактера **9**
3.1 Конволуционе неуралне мреже 9
3.1.1 Конволуциони слој 9
3.1.2 *Pooling* слој 10
3.1.3 Потпуно повезан слој 11
3.1.4 *Backpropagation* метод 12
3.1.5 Хиперпараметри 14
3.2 Груписање 15

4 База података **17**

5 Резултати **18**

6 Завршна дискусија **20**

Литература **20**

1 Увод

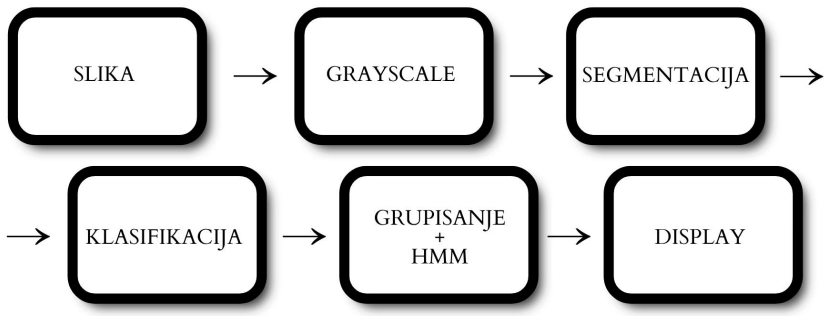
Како је технологија почела нагло да се развија током прошлог века, постала је јефтинија самим тим и приступачнија за велику већину људи. Данас имамо паметне телефоне и личне рачунаре који нам омогућавају да скоро све информације добијемо у веома кратком временском интервалу у свега пар кликова. С тога се средином прошлог века јавио проблем дигитализације докумената из архива и библиотека. Било је логично да није могуће прекуцати све што је човечанство икада створило, па су људи досетили да то реше прављењем одређеног алгоритма који ће то урадити уместо њих. За разлику од текста, проблем препознавања писаних математичких једначина је знатно тежи. Првенствено због количине могућих знакова који треба да се дигитализују (код препознавања текста имамо око 30-ак слова и десетак интерпункцијских знакова, док број математичких симбола долази и до пар стотина). Такође, речи за разлику од једначина немају вишедимензионалну угњежденост као што су степен и индекс. То представља проблем зато што сваки знак једначине мора се прво издвојити, а затим у програму класификовати. Због тога је препознавање једначина тренутно један од најтежих проблема којим се поље компјутерске визије бави.

Из оправданих разлога, на овом проблему се ради последњих неколико деценија. Хеуристика која стоји иза математичких закона је баш то што није дозвољавало научницима да пронађу довољно једноставан алгоритам који ће компјутер знати да изврши као и довољно сложен да препозна и неке комплексније математичке изразе. У референтним радовима су представљене до сада најуспешније методе. Једна од њих је конволуциона неуронална мрежа која се користи за препознавање симобла као и подебљање тренинг симбола ради додатног побољшања прецизности самог система.

Иако није коришћена унапред истренирана неуронална мрежа, тежиште истраживања је било на сегментацији математичких симбола користећи две угњеждене рекурзивне функције и на хеуристици за груписање тих симбола, уз поређење истог метода али овог пута са мрежом истренираном на подебљаним симоблима. Систем у другом случају је достигао значајно већу тачност што је и било за очекивати јер дебљина карактера није униформна већ зависи од стила писања појединца.

2 Метод

Комплексност проблема повлачи и сложеност алгоритма за његово извођење. Поступак је приказан на слици 1. Поступак се састоји од превођења *RGB* слике у *Grayscale*, сегментације, класификације и груписања. Два кључна корака овог алгоритма су сегментација и поновно груписање симбола у *latex* код. Да би се извршила сегментација потребно је прво припремити слику: превести *RGB* структуру слике у *Grayscale* слику и уклонити шумове. Уклањање шумова је можда један од најважнијих подалгоритама, јер уколико остану неки предњи (*foreground*) пиксели, пиксели који носе информацију, алгоритам ће их класификовати као да су неки од могућих математичких симбола и на тај начин деградирати коначан исход.



Слика 1: Блок шема алгоритма за препознавање математичких једначина

Такође, коришћено је подебљавање симбола из базе података. У бази података која је коришћена за тренинг неуралне мреже симболи су један пиксел дебљине. Пре свега, из искуства знамо да се стилови писања знатно разликују. Разликују се и оловке којима пишемо што доводи до тога да је потребно направити систем који ће бити инваријантан на стилове и различите записе истог симбола. За решавање проблема инваријантности коришћена је функција *dilate* из библиотеке *OpenCV* са кернелом облика

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

на тај начин је обезбеђено да када кернел буде итерирао по матрици подебља сваки пиксел тачно једном.

2.1 Богаћење базе података

Главни циљ овог дела пројекта је надоградња постојеће базе података. Начин на који то радимо не треба да буде произвољан, треба да буде

такав да заправо обезбеди суштински другачије слике из којих ће наша неуралне мрежа учити. У пракси су најкоришћеније две методе: коса исправка (*slant correction*) и еластичне трансформације (такође постоје и афине трансформације али њима се нећемо бавити зато што је кроз неколико научних радова показано да имају занемарљив утицај на квалитет неуралне мреже).

Главна поента косе испрваке је да обезбедимо да наш систем буде инваријантан на различите стилове писање. Дефинисаћемо косину као угао у смеру казаљке на сату (*clockwise*) између вертикалне осе и доминантног правца датог карактера. Метода која се доста користи у пракси је Метода вертикалне пројекције. За почетак, помоћу оператора смицања (*shear operator*) се може направити сет који се састоји од различитих вештачки искошених слика. Ово се ради из разлога да уколико је угао косине α онда та иста слика накривљена за негативну вредност угла α је заправо исправљена слика.

Сада се поставља једно врло важно питање. Зашто би нам еластичне трансформације биле корисније од афиних и зашто би нам уопште требале било какве трансформације тог типа? Испоставља се да еластичне деформације заправо одлично уче неуралну мрежу на случајне, неконтролисане осцилације мишића руку настале инерцијом. У афине трансформације спада ротација, транслација и хомотетија односно скалирање. Ниједна од ових трансформација неће суштински променити карактер. Зато су нам потребне еластичне трансформације које ће у свим правцима да развуку одређени карактер. Један од начина на који то може да се имплементира је да за свкаки пиксел на целој слици помоћу функиције *random* одредимо за колико ћемо дати пиксел да транслирамо дуж обе осе. Померања су једино могућа на места суседних пиксела. На тај начин бисмо удвостручити целокупну базу података и обогатити је карактеристикама које иначе не бисмо могли да прикупимо. Међутим, како база података симбола и база података писаних математичких једначина нису синхронизоване било је потребно извршити подебљање базе података симбола (подврста еластичних деформација) како би се извршила веродостојна мерења.

2.2 Обрада слике

Процес обраде слике обухвата све оне промене и модификације на слици које чине слику погоднијом за даљу обраду. Овај део можемо поделити на три процеса: претварање *RGB* слике у *Grayscale* слику (слика која садржи искључиво нијансе сиве), уклањање шумова и бинаризација слике.

2.2.1 Претварање слике из *RGB* у *Grayscale*

Уколико бисмо радили са *RGB* сликом морали бисмо да за сваки пиксел на слици водимо рачуна о количини црвене, зелене и плаве боје. Зато поједностављујемо слику тако што ћемо три вредности заменити једном вредности, тј. *Grayscale* вредности. Формула коју ћемо за то користити је:

$$Y = 0.999R + 0.587G + 0.114B,$$

где су Y, R, G, B редом ознаке за количину сиве, црвене, зелене и плаве боје.

2.2.2 Уклањање шумова са слике

Како је хардвер нешто што је човек створио, логично је да неће увек радити у потпуности као што је замишљено. Грешке хардвера се називају шумови. Због развоја у самој електроници те грешке су све мање и ређе, али су и даље присутне и могу правити проблеме приликом даље обраде. Постоје више врста шумова као што су со и бибер шум, импулсни шум, Гаусов шум и многи други. Како је Гаусов шум најучесталији на сликама, бавићемо се само његовим изоловањем док ћемо остале сматрати занемарљивим.

Начин на који ћемо да извршимо уклањање шумова је крајње једноставан: заменићемо боју одређеног пиксела са усредњеном вредности других пиксела из његове средине. За то је потребно да пронађемо сличне пикселе. Они не морају да се граниче са датим пикселом. Узмимо за пример шаре које се понављају. Зато је важно да скенирамо пропорционално велики део слике како бисмо пронашли што је више могуће пиксела који подсећају на наш оригинални пиксел. Два пиксела личе уколико су им вредности приближно једнаке. Да бисмо то одрадили користићемо функцију из библиотеке *OpenCV* која се зове $fastNLMeansDenoising$.

2.2.3 Бинаризација

Бинаризација је процес у коме се свакој вредности *Grayscale* пиксела додељује једна од две могуће вредности, као што и име алудира, те вредности су 0 или 255. У нашем случају пиксели који буду имали вредност 0 биће предњи (*foreground*) пиксели, док они коју буду имали вредност 255 биће задњи (*background*) пиксели. Да бисмо то урадили потребно је да за сваки пиксел знамо праг на основу којег ћемо му доделити адекватну вредност. Праг или граничну вредност можемо изабрати тако да буде јединствена и унапред одређена, али постоје и паметније методе које ће нам помоћи да оптимизујемо нашу бинаризацију.

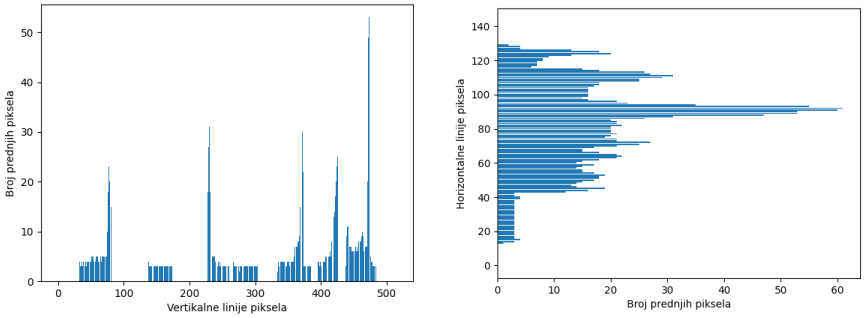
Две методе које су се до сада показале најбоље у пракси су метода адаптивне граничне вредности (*adaptive thresholding*) и Отова метода. Обе методе имају своје предности као и своје мане. Отова метода има већи проценат успешности уколико се ради са бимодалним сликама. Бимодалне слике су слике које на свом графику зависности броја пиксела од њихове боје имају 2 превоја (*peaks*). Нажалост, у нашем случају где осветљеност слике неће бити савршена, потребни су нам локалне граничне вредности. Зато ћемо користити адаптивну методу. Користићемо функцију *adaptive Threshold* из библиотеке *OpenCV*, а као адаптивну методу ћемо користити $adaptiveThreshGaussianC$.

2.3 Сегментација

Посматрајмо обичну неуралну мрежу. Она на свом излазу има одређен број неурона, где сваки неурон представља вероватноћу са којом неурална мрежа може да процени да ли се одређени објекат налази на слици или не. Посматрајмо сада пример математичких израза. Скуп свих потенцијалних формула је одређен свим могућим комбинацијама математичких симбола што препознавање целих израза чини немогућим, али је зато скуп математичких симбола коначан па се зато извршава сегментација.

У претходном делу је наговештена важност уклањања шумова, а у овом делу ће бити и објашњења. За почетак, уведимо нумеричку методу која се назива хистограмска пројекција која нам служи за испитивање да ли се неки део једначине налази у датој врсти или колони слике где се врста и колона слике односи на врсту и колону пиксела који чине дату слику. У датом експерименту су коришћене две хистограмске методе: вертикална и хоризонтална. Вертикална хистограмска метода рачуна број предњих пиксела за сваку колону на слици. Аналогно се дефинише и хоризонтална хистограмска пројекција.

Слика 2: Пример математичког израза



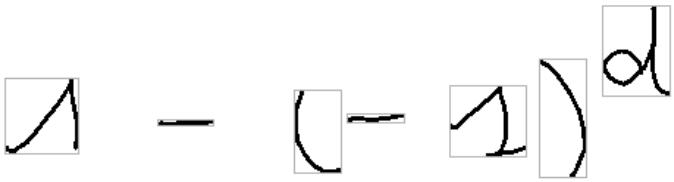
Слика 3: Вертикална и хоризонтална пројекција

Посматрајмо пример математичког израза са слике 2. Уколико погледамо њену хоризонталну хистограмску пројекцију, на слици 3 можемо закључити да се на слици 2 налази само један математички израз јер се на

датом дијаграму уочава један и само један брег (низ повезаних редова на слици таквих да сваки ред садржи барем један предњи пиксел). Слично се добија посматрајући вертикалну хистограмску пројекцију. Може се уочити низ од седам брегова који директно одговара адекватним симболима у математичком изразу. У овом примеру су биле довољне по једна хоризонтална и једна вертикална пројекција да бисмо пронашли све симболе са слике.

Наравно, то не мора да буде генерални случај са математичким изразима. На примеру разломка у било којој једначини се може јасно уочити да је након хоризонталне и вертикалне пројекције успешно сегментирана већина симбола али бројилац, разломачка црта и именилац неког разломка ће бити рачунару представљени као један блок слике. Интуитивно се јавља идеја да уведемо додатни корак, нову хоризонталну пројекцију. Такво решење ће унапредити алгоритам, али је могуће да идаље или бројилац или именилац садржи нови математички израз. Узмимо за пример математички израз $\frac{3}{7}y$. Хоризонтална пројекција ће нам издвоји цео израз док ће вертикална пројекција расчланити израз на два дела: $\frac{3}{7}$ и y. Нажалост, први део израза је и даље неки израз па је потребно поново извршити хоризонталну прокецију како бисмо издвојили 3, $-$ и 7. Из овог примера можемо закључити да је потребно наизменично рекурзивно позивати хоризонталну и вертикалну пројекцију.

Теоријски, овакав низ може да се настави у бесконачност. Из простог разлога, коначан број хоризонталних и вертикалних пројекција би била апроксимација проблема, зато је у овом експерименту искоришћена могућност рекурзивног програмирања у програмском језику *Python*. Као и за сваку рекурзију потребно је дефинисати основни случај, случај када се рекурзија прекида. Прва идеја која се јавља је да прекинемо рекурзију када не можемо да добијемо нове мање блокове слике, тј. када се на графику оба хистограма јавља само један превој (пеак). То би радило у лабораторијски савршеним условима, под претпоставком да ће сваки симбол бити калиграфски написан што не мора да буде случај. Уколико узмемо израз a^y можемо приметити да је вероватно да ће неки део y прећи у вертикалну пројекцију од a. У том случају рекурзија би се зауставила иако сегментација није завршена. Као исправак за овај проблем, на крају сваке рекурзије се изводи $flood\ fill$ алгоритам што је могуће само зато што се сваки пиксел граничи са још четири пиксела (не рачунајући пикселе на ивици слике), тј. тако бинаризовану слику можемо представити као бестежински граф са дисјунктним компонентама.. $Flood\ fill$ алгоритам претвара блок матрице слике у бестежински неусмерени граф и на њему одређује број повезаних компоненти и паралелно са тим маркира пикселе са слике са одговарајућим редним бројем компоненте графа. Алогритам је имплементиран користећи функцију *label* из библиотеке *SciPy*.



Слика 4: Пример сегментације

2.4 Потенцијални проблеми приликом сегментације

Проблеме можемо поделити у две групе. Првој групи припадају преклапања између симбола. Она настају као последица брзине писања. Такви симболи су *log*, *sin*, *cos*,.. Такође, ту могу бити било која нежељена преклапања између било која два симбола, зато што ми за предпоставку узимамо да ће сви симболи бити међусобно одвојени, што генерално не мора да буде случај. Другој групи припадају симболи који се пишу одвојено, стога не можемо да знамо да ли дати фрагмент припада једном или другом симболу или је симбол за себе. Такви су симболи $\pm$ или $=$.

3 Класификација карактера

3.1 Конволуционе неуралне мреже

CNN је једна од класа дубоких неуралних мрежа и један од многобројних алгоритама дубоког учења који нам омогућава да помоћу учивих тезхина (*weight*) и склоности (*bias*) идентификујемо одренђене објекте, тј. да омогућимо рачунару да има исту перцепцију и моћ запажања коју ми људи имамо. Главна разлика између *CNN* и других неуралних мрежа је присуство конволуције. Конвулуција је математичка операција над две функције и за продукт има трећу функцију. У случају *CNN*, конволуција нам служи да од улазних података, помоћу одређених филтера и кернала направимо мапу одлика. На примеру са бројевима те одлике могу бити одређена закривљења или строги оштри углови,...

Може се направити одређена аналогија са нама. Док смо још мали, родитељи нас уче тако што нам показују објекте и изговарају њихова имена. На тај начин повезујемо предмете са именицом коју чујемо. Нажалост, нећемо увек све научити из прве, зато су наши родитељи ту да нас исправе како не бисмо следећи пут направили исту грешку. Такав начин учења у компјутерској науци се назива надгледано учење, зато што тачно знамо шта резултат треба да буде. У случају тачног одговора наша мрежа ће бити награђена, док ће у супротном бити кажњена.

Компјутерски нервни систем је врло сличан нашем. Састоји се од мреже која садржи огроман број неурона. На основу импулса које прими неурон и значаја сваког импулса, неурон одлучује да ли да се активира и да проследи импулс даље кроз мрежу или да аксон остане затворен и на тај начин спречи проток. Укратко, сваки неурон можемо да посматрамо као скуп улаза, скуп тежина и активациону функцију. Та међусобна повезаност неурона класификује *CNN* у групу $feedforward$ неуралних мрежа.

Постоје различите врсте неурона, и оне зависе од стадијума (слоја - *layer*) мреже у којој се налазе. Три основна слоја у свакој неуралној мрежи су улазни, скривени и излазни слој, стим што имамо само један улазни и изланзи слој, док скривених слојева можемо имати више у зависности од структуре мреже и типа проблема. У случају *CNN* то су конволуциони, активациони, *pooling* и потпуно повезан слој.

3.1.1 Конволуциони слој

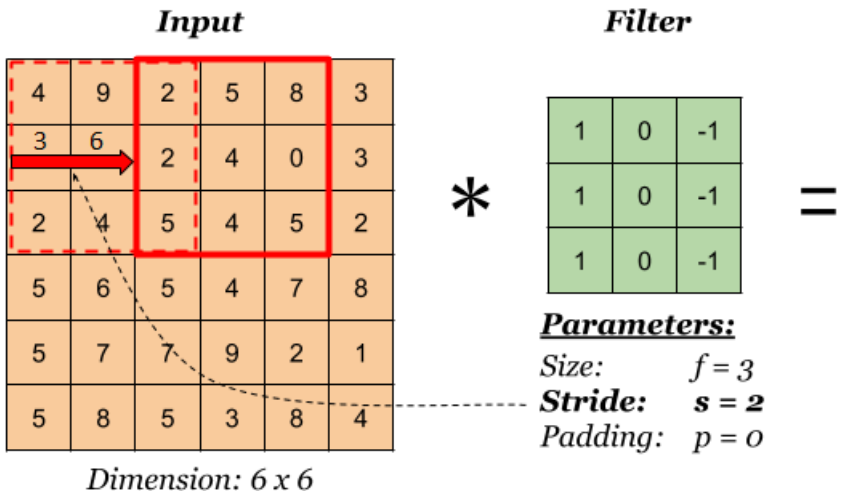
Конволуциони слој је најважнији слој у овом моделу и он служи за грубу детекцију на слици, тј. за проналажење одговарајућих шаблона који ће касније олакшати саму класификацију. Карактеристика овог слоја је присуство филтера (кернала) који садржи одређене тежине. На почетку су те тежине насумично додељене али их мрежа учи кроз епохе тиме побољшава перформансе.

Објаснићемо принцип када је на улазу слика димензија $W \times W \times 1$ (узели смо 1 зато што на улаз долази *grayscale* слика па нису потребна три канала

за RGB). Нека имамо D различитих филтера, са кораком S. Сваки филтер појединачно прелази преко матрице улаза и рачуна производ, по специфичном правилу множења матрица. Свако поље прве матрице се множи са одговарајућим пољем друге матрице и као производ оваквог множења матрица се добија скалар. Затим рачуна збир елемената производа и уписује га у одговарајуће поље матрице излаза, и тако за сваки од датих филтера. На излазу ће се добити тродимензионална матрица $W_1 \times W_1 \times D$, где се W_1 рачуна по формули:

$$W_1 = (W - F + 2P)/S + 1,$$

где је F димензија кернала, S корак, тј. за колико ће кернал да се помери у лево по свакој итерацији и P ширина оквира око слике који се састоји од нула. Главна улога ширине оквира је да одржава димензије улаза. Нова матрица представља улаз за следећи слој.



Слика 5: Конволуција

Потребно је још додатно напоменути да активациони слој који смо помињали можемо сврстати као други део конволуционог слоја. Активациони слој није ништа друго него обична функција. Како вредности у новој матрици могу варирати, потребно их је пустити кроз одређену функцију која ће да нормализује те вредности. Углавном желимо да сместимо све те вредности у опсег од 0 до 1, јер на излазу желимо вероватноћу/сигурност којом можемо да идентификујемо одређени објекат на слици. Наравно, постоје и друге врсте активационих функција са различитим опсезима као што је $ReLU$ функција коју ћемо ми користити у нашој неуралној мрежи.

3.1.2 *Pooling* слој

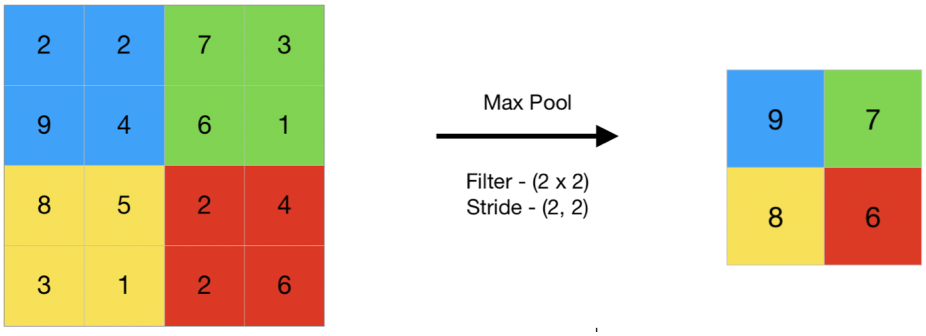
Димензије слике могу бити велике, самим тим у мрежи ће постојати огроман број неурона што ће драстично успорити процес учења. Зато нам

је потребан одређен слој у нашој мрежи који ће смањити димензије матрице, а у исто време сачувати све битне информације, тј. неће пореметити класификацију. Други разлог увођења овог слоја је то што на излазу треба да имамо вектор од n елемената (n представља број различитих предмета која наша мрежа учи да препозна), другим речима треба слику некаоко да спакујемо у вектор. Сваки члан тог вектора представља вероватноћу да се тај објекат налази на слици. На пример ако је $v[3] = 0.92$, то значи да објекат који одговара редном броју 3, има вероватноћу од 92% да се налази на слици.

Постоје две основне врсте овог слоја: средњи *pooling* (*average pooling*) и максималан *pooling* (*max pooling*). Слично као и код конволуционог слоја и овде имамо филтер само овог пута без тежина. Ако су улазне димензије матрице $W_1 \times W1 \times D_1$ и просторна димензија филтера F са кораком S, димнзије производа су $W_2 \times W_2 \times D_2$, где се W_2 и D_2 рачунају по формули:

$$W_2 = (W_1 - F)/S + 1$$

$$D_2 = D_1$$



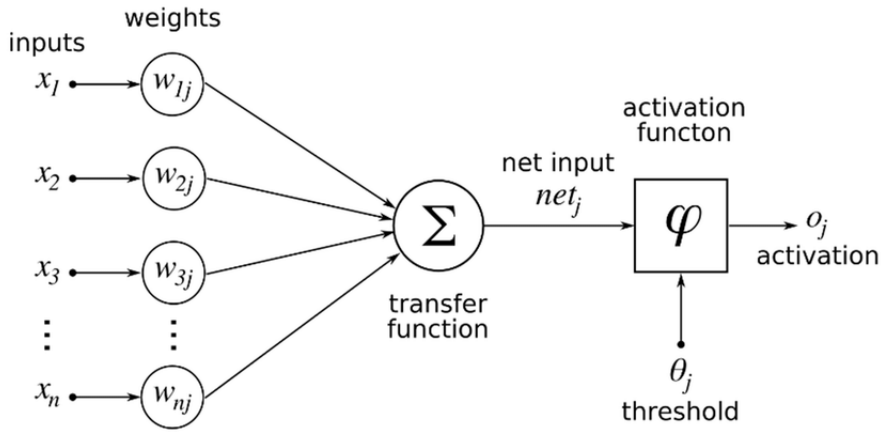
Слика 6: *Pooling* слој

3.1.3 Потпуно повезан слој

Попуно повезани слој спада у групу класификационих слојева. То значи да се он налази на крају саме мреже и њен излаз је и излаз наше мреже. Као што смо већ рекли конволуциони и *pooling* слој служе за проналажење одређених карактеристика на слици, затим се оне пуштају кроз потпуно повезани слој, где се те карактеристике даље обрађују.

Структура ове подмреже је таква да је сваки неурон из једног слоја преко одређених тежина повезан са сваким неуроном из претходног и следећег слоја. Вредности тих тежина мрежа учи кроз епохе. Када слика са улаза прође кроз последњи *pooling* слој, она није истог тополошког облика као жељени излаз, потребно је да се изравна, претвори у низ, где сваки елемент тог низа представља по један неурон на првом слоју. Неуроне на следећем

слоју добијамо као суму неурона са првог слоја помножени одговарајућим тежинама праћен одговарајућом склоности сваког неурона (*bias*).



Слика 7: Пример једног неурона

Због истог разлога као и у конволуционом слоју, вредности сума које се прослеђују у нови слој може драстично да варира, зато нам је потребна активациона функција која ће да нам обезбеди излаз жељеног облика. У пракси, најбоље се показала $Softmax$ активациона функција због њене могућности рада са више различитих класа као и да нормализује излаз за сваку класу у опсег од 0 до 1. Због тог ралога ћемо је и ми користити.

Архитектуру коју ћемо користити је иста као она која се користила у референтном раду и њена структура је $[Conv - Relu - Pool] \times 2 - [FC - Relu] \times 2 - Softmax$. Величина филтера је F, док је укупан број филтера 32. Користићемо 2×2 *max pooling* са кораком дужине 2.

3.1.4 *Backpropagation* метод

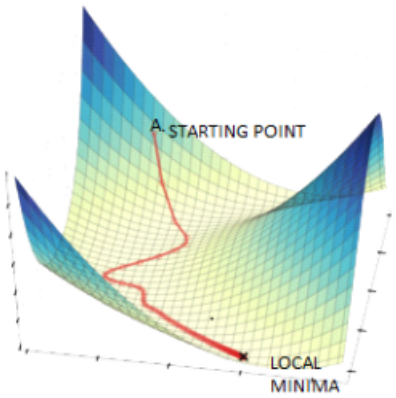
Остало је још само да објаснимо како ће то наша мрежа да учи. Користићемо алгоритам који се назива *backpropagation*. Раније смо утврдили да CNN спада у $feedforward$ класу, тј. излаз из једног слоја служи као улаз у други. То нам омогућава да нашу мрежу посматрамо као неусмерени тезхински граф, а како је граф неусмерен он нам омогућава кретање у оба смера.

Како бисмо објаснили овај метод потребно је да уведемо неке ствари. Прво ћемо дефинисати *cost function*. Она нам служи да нам покаже колико је наш одговор далеко од траженог. Постоје различите врсте, али ће се у истраживању користити квадратна *cost* функција, која је дефинисана као:

$$C = \frac{1}{2n} \sum_x ||y(x) - a^L(x)||,$$

где је $y(x)$ излаз из мреже у облику n-точланог вектора, док је $a^L(x)$ жељени излаз (изведен из етикете коју имамо за сваки податак) и где је n број слика коју је мрежа до сада обрадила.

Како је наш циљ да излаз буде што ближи траженом, логично је да желимо да минимализујемо нашу *cost* функцију. Један од начина да се то уради је помоћу *gradient descent* методе. Она се заснива на проналажењу локалног минимума за било коју вредност. Разлог због кога тражимо локални минимум је тај што када бисмо тражили глобални били би нам потребни суперрачунари, јер димензије уноса функције су јендаке броју неурона што је реда 10^{12}. С тога, рачунање извода такве функције би одузело превише времена. Зато тражимо парцијалне изводе, јер је нама битно да само дивергирамо ка том минимуму, никако од њега. Другим речима покушавамо да нађемо било који вектор по ком бисмо могли да се крећемо под условом да наша потенцијална енергија увек опада. Због великог броја слика које су предодређене за тренинг можемо да трвдимо да ћемо у неком тренутку достићи тај минимум. С тога, потребно је да нађемо било коју промену тежине или промену склоности неког неурона, а да се при томе C ($cost\ function$) смањи.



Слика 8: Градијент опадања

Потребно је да још дефинишемо четири основне једначине (четири основна постулата) *backpropagation* метода. Оне су нам важне из једног простог разлога што математички темељно објашњавају цео процес:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

$$\delta^l = ((\omega^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial \omega_{jk}^{l}} = a_k^{l-1}\delta_j^l$$

Прва једначина нам омогућава да израчунамо грешке свих неурона на последњем слоју. Те вредности ћемо користити да бисмо заправо израчунали грешке неурона на свим осталим слојевима помоћу друге једначине. Када смо израчунали све грешке, из последње две једначине је заправо лако израчунати стопу промене тежина као и склоности сваког неурона појединачно. Отуд и име самог метода, израчунавамо грешке крећући се од послеђенг слоја ка првом.

3.1.5 Хиперпараметри

Хиперпараметри, за разлику од обичних параметара као што су тежине и склоности, су познати унапред и мрежа њих не учи. Постоји мноштво хиперпараметара међу којима су нама најзначајнији *dropout*, стопа учења, *momentum*, број епоха, величина гомиле (*batch size*) и стопа опадања учења.

Dropout се користи како бисмо смањили $overfitting$, тј. смањили грешке неуралне мреже на тренутно невиђене примере. Овај метод ради на принципу да током тренинга насумично изабрани излази из одређених слојева бивају игнорисани. Дефинишимо вероватноћу да одређени излаз неће бити игнорисан са p. Такође, потребно је и да пре него што започнемо са тестирањем скалирамо све тежине за p, зато што ће *dropout* током тренинга натерати мрежу да појача тежине као последица недостатка информација током сваке итерације. P у скривеним слојевима треба да буде између 0.5 и 0.8, док у улазном слоју треба да буде већи од 0.8.

Стопа учења је коефицијент на основу кога одређујемо за колико ћемо да променимо одређену тежину или склоност дуж њихових градијената. Мале стопе учења успоравају процес учења али зато конвергирају глатко, док високе стопе учења убрзавају процес учења али не морају нужно да конвергирају. Стопа учења може бити константна или адаптивна. Код константне, стопа учења се не мења са временом, док код адаптивне, стопа учења се мења кроз итерације. Ми ћемо користити адаптивну *step decay* методу, која се заснива на принципу да након N епоха скалира стопу учења за одређени коефицијент k:

$$lr = lr_0 \times k^{\frac{epoch}{N}},$$

где је *epoch* редни број епохе у којој се тренутно налазимо, а lr_0 почетна стопа учења. Уобичајено је да на сваких 10 епоха преполовимо стопу учења. Иако су хиперпарамети унапред дефинисани потребно их је изабрати врло пажљиво како бисмо остварили што је могуће бољу прецизност. Алгоритам који ћемо за то користити се зове *Grid search*. *Grid search* је процес који претражује све могуће вредности над целим простором хиперпараметара тако да неуралн мрежа достигне највећу тачност.

Моментум нам служи да одредимо правац следећег корака на основу информације о прошлом кораку. Математички има структуру нехомогене дифернцијалне једначине првог реда и служи нам да спречимо настајање осцилација током *gradient descent*-а. Формула по којој се рачуна моментум је:

$$\omega_{t+1} = \omega_t - \alpha m_t$$
$$m_t = \beta m_{t-1} + (1-\beta)\frac{\partial C}{\partial \omega_t},$$

где је β коефицијент моментума и његова вредност углавном износи око 0.9.

Број епоха је број пута колико ће целокупна тренинг база података бити остављена неуралној мрежи на тренинг. Број епоха треба да се повећава све док тачност валидације не крене да пада па чак и када тачност током тренинга расте, на овај начин спречавамо $overfitting$.

Величина гомиле (*batch size*) је заправо број тренинг примерака који ће се пустити у исто време неуралној мрежи на тренирање. Градијент за сваки излазни неурон рачунамо као усредњену вредост свих k градијената за тај одређен неурон из једне гомиле. За вредност k (величину гомиле) узимамо да буде 32.

3.2 Груписање

Иако смислено препознавање свих симбола са слике и њихово груписање у једначину представља лак посао за човека, испоставља се да и није баш најједноставнији процес за рачунар из два разлога. Први је то што се не пишу сви симболи из само једног покрета (без одвајања оловке са папира), а други је то што већина једначина има више од једне димензије, тј. постоје степени, индекси, разломци,...

Као што смо до сада више пута рекли математичке једначине се одликују вишедимензионалном угњежденошћу и зато за сваки карактер који пуштамо кроз неуралну мрежу памтимо растојање у односу на осу која пролази кроз центар масе претходног симбола. На основу те линије одређујемо да ли је тренутни симбол наставак математичког израза или је степен или индекс претходног симбола. Ширину и висину симбола проналазимо помоћу методе повезаних компоненти где памтимо најсевернији, најјужнији, најзападнији и најисточнији пиксел сваког карактера и као њихову разлику рачунамо висину и ширину. Уколико оса претходног симбола пресеца садашњи симбол кажемо да је садашњи симбол наставак математичког израза. Уколико се оса претходног симбола налази испод или изнад садашњег симбола кажемо да је садашњи симбол степен или индекс претходног симбола. Процес је реализован тако што сваки пут када идентификујемо нови симбол и његов положај додајемо његов адекватан *latex* запис на стирнг који чува досадашњи израз.

Што се тиче проблема неповезаних симбола ситуација је знатно тежа. Уопштења ради, претпоставимо да је једначина једног степена, тј. основна линија сече сваки карактер једначине. У наставку су исписани потенцијални искази које треба кориговати:

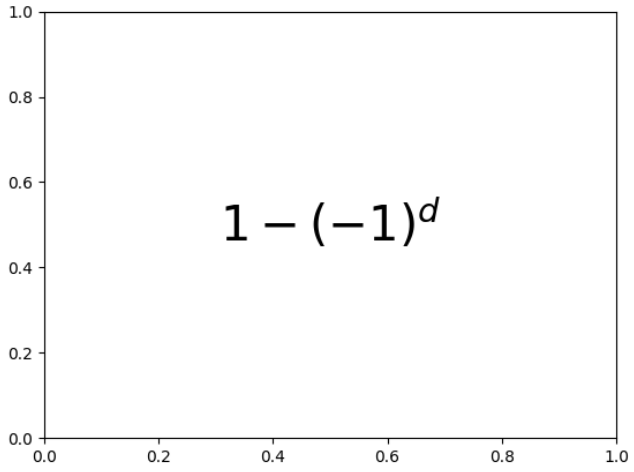
1. знак једнакости, у случају да наиђемо на два знака минус чији су центри у опсегу овог другог (по x-оси) и разликују се по y-оси, можемо да их спојимо и да гледамо на њих као на знак једнакости, јер знамо да је низ од два минуса неправилан математички запис и не може се појавити ни у једном изразу.

2. тригонометријске функције, ако бисмо наишли на низ карактера као што је c, o, s могли бисмо да их спојимо у cos, слично важи и за tan, log,...

3. sin, за синус имамо специфичан случај зато што, у већини случајева, i не можемо да напишемо без подизања оловке са папира, па би у том случају класификациони низ изгледао као s, $|$, n. У том случају бисмо овакву комбинацију подразумевали као да је sin.

4. $\leq$ и $\geq$, уколико би постојао низ знакова $<$, $-$ или $>$, $-$, редом би се класификовали као $\leq$ и $\geq$. Важи и обрнут редослед.

5. $\pm$, уколико би постојао подниз елемената $+$ и $-$, или обрнуто, били би спојени у знак $\pm$.



Слика 9: Коначан изглед математичког израза након груписања

4 База података

За базу података математичких једначина ћемо користити базу података која се користила током $CROHME$-овог такмичења 2011 године. Једначине су дате у облику $InkML$ фајла и подељене су у две групе: тренинг и тест. Једначине из обе групе су лабелисане, тј. зна се тачан садржај сваког фајла. За базу података математичких симбола користићемо $Kaggle$-ову базу података која садржи jpg фајлове.

$InkML$ је по структури сличан $HTML$. Користи се за чување писаног текста у електронској форми, где су сви пиксели који треба да буду обојени задати координатама. Свака једначина је представљена помоћу $< ink >$ елемента. Унутар једначине, сваки замах је представљен помоћу одвојеног $< trace >$ елемента, чији садржај представља $x - y$ кординату сваког места који је врх оловке додирнуо током времена.

```
<ink>
   <trace>
     10 0, 9 14, 8 28, 7 42, 6 56, 6 70, 8 84, 8 98, 8 112, 9 126, 10 140,
     13 154, 14 168, 17 182, 18 188, 23 174, 30 160, 38 147, 49 135,
     58 124, 72 121, 77 135, 80 149, 82 163, 84 177, 87 191, 93 205
   </trace>
   <trace>
     130 155, 144 159, 158 160, 170 154, 179 143, 179 129, 166 125,
     152 128, 140 136, 131 149, 126 163, 124 177, 128 190, 137 200,
     150 208, 163 210, 178 208, 192 201, 205 192, 214 180
   </trace>
   <trace>
     227 50, 226 64, 225 78, 227 92, 228 106, 228 120, 229 134,
     230 148, 234 162, 235 176, 238 190, 241 204
   </trace>
   <trace>
     282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290 129,
     291 143, 294 157, 294 171, 294 185, 296 199, 300 213
   </trace>
   <trace>
     366 130, 359 143, 354 157, 349 171, 352 185, 359 197,
     371 204, 385 205, 398 202, 408 191, 413 177, 413 163,
     405 150, 392 143, 378 141, 365 150
   </trace>
</ink>
```

Слика 10: Пример $InkML$ фајла

5 Резултати

Први део истраживања се односи на разлику у прецизности система приликом класификације математичких једначина написаних помоћу свих осамдесет и два могућа симбола из коришћене базе података и математичких једначина које у себи не садрже одређене симболе из исте базе података.

Табела 1

!	(	)	+	,	−	=	[	]	{	}	0
1	2	3	4	5	6	7	8	9	A	α	|
b	β	C	$\cos^*$	d	δ	$\div^*$	e	$\exists$	f	$\forall$	/
G	γ	$\geq$	$>$	H	i^*	$\in$	∞	$\int$	j^*	k	l
λ	$\ldots^*$	$\leq$	$\lim^*$	log	$<$	M	μ	N	$\neq$	o	p
ϕ	π	$\pm$	$\prime$	q	R	$\rightarrow$	S	σ	$\sin^*$	$\sqrt{x}$	$\sum$
T	$\tan^*$	θ	$\times$	u	v	w	x	y	z		

У табели су представљени сви симболи који се налазе у бази података математичких симбола која је коришћена у овом експерименту, док су звездицом означени они симболи који нису коришћени у другом делу истраживања.

Иако је прецизност неуралних мрежа тренираним са подебљаним и без подебљаних симбола приближно једнака, 98% и 97% редом, запажа се драстична разлика у прецизности приликом препознавања целокупних математичких једначина. Најпре, у оригиналној бази података дебљина сваког симбола је искључиво један пиксел. То би значило да једначине морају бити усликане са велике удаљености како би оригиналне једначине учинили истањеним што у бази података писаних математичких једначина није случај. Једначине су у тој бази тако представљене да заузимају пропорционално велики део слике. У том случају можемо да очекујемо да су једначине увећане и стога подебљане. Удвостручивањем базе података симбола и подебљавањем за тачно један пиксел, омогућавамо неуралној мрежи да буде непроменљива на дебљину симбола, самим тим и инваријантна на раздаљину са које се слика једначина. Мала разлика у прецизности самих неуралних мрежа се огледа у броју тренинг примера из којих је учила неуралне мрежа. Како су се обе мреже тренирале на три епохе, ниједна од њих није достигла засићеност зато што се није приметио пад у прецизности током све три епохе, самим тим мрежа са више тренинг примера се показала бољом. Како су

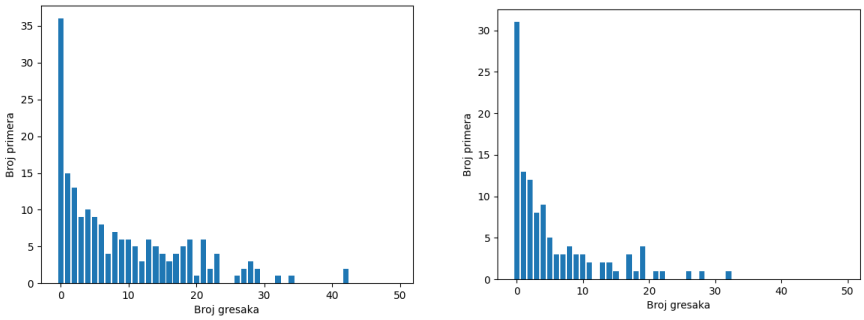
Табела 2

Резултати мерења		
подебљање/симболи	сви симболи	одабрани симболи
са подебљањем	17.29%	29.90%
без подебљања	8.45%	13.17%

обе вредности врло високе, не можемо да кажемо да је ралика у 1% имала утицај на разлику у прецизности целог система.

Други део истраживања се односи на то како одређени симболи праве разлику у коначној прецизности система. Из целокупне базе података је избачено осам симбола који се често користе приликом писања математичких израза и који због њихове комплексности у *latex* запису представљају потенцијалне проблеме. У оба случаја (са и без подебљања симбола) се примећује драстична разлика у тачности система. Разлог томе је чињеница да тригонометријске ознака као и *lim* могу да се напишу на више различитих начина. На пример, неки људи пишу *sin* тако да су им сва три слова спојена, неки раздвајају реч на *si* и *n*, а неки пишу сва три слова одвојено. Уколико неки људи пишу ситније могуће је да *sin* споје са '('. У том случају неуронална мрежа неће знати тачно да препозна израз *sin(* зато што не постоји у бази података симбола. Такође, важно је нагласити да преклапање симбола не мора нужно да буде повезано са тригонометријским ознакама. Проблем преклопљених симбола је и највише заслужан за релативно ниске резултате, зато што рачунар не може да разликује да ли низ испреплетаних линија припада једном симболу или неком скупу симбола.

Последњи део истраживања се односи на процентуалну тачност препознавања. У прва два дела смо посматрали искључиво коначна решења и поредили да ли излаз модела у потпуности одговара унапред познатој етикети. У овом делу се користило Левенштајново растојање како би се поредило колико се излази система разликују од задате тачне вредности, тј. Левенштајново растојање рачуна најмањи потребан број измена на излазном *latex* коду како би он по саставу одговарао адекватној етикети.



Слика 11: Левенштајново растојање

Са датих графика се може приметити да иако је генерална тачност релативно ниска, број примера експоненцијално опада са повећањем броја грешака по примеру. Такође, просечна процентуална тачност препознавања за један математички израз је 75% што показује да би се коришћењем овог система драстично убрзала поменута дигитализација.

6 Завршна дискусија

Тестирањем неуралних мрежа над малим узорцима уочавају се мале разлике у њиховој прецизности. То се објашњава јединственим карактеристикама сваког математичког симбола што омогућава неуралној мрежи прецизну класификацију. У свим експериментима је коришћена конволуциона неурална мрежа са два $conv - pooling$ слоја и три потпуно повезана слоја. Активациона функција је $ReLU$. У раду су извршена три различита мерења. Прво мерење се односи на разлику у класификационој прецизности система у односу на то да ли су током тренинга неуралних мрежа коришћени подебљани симболи или нису. Систем са неурлном мрежом која је тренирана са подебљаним симболима се показао дупло боље приликом класификације математичких једначина него систем са неуралном мрежом која је тренирана са симболима константне дебљине. Овакви резултати су оправдани јер не можемо да очекујемо да су стилови писања различитих људи исти као ни оловке које су коришћене. Друго мерење се односи на класификациону прецизност система у односу на то који су математички симболи коришћени у једначинама. Утврђено је да симболи који имају сложену просторну структуру као и/или компликован *latex* запис смањују прецизност оба система. Овакав резултат је у сагласности са резултатима постигнутим у референтном раду. У овом раду је показано користећи Левенштајново растојање да је прецизност система на нивоу појединачних математичких израза 75% што показује да би коришћење овог система драстично убрзало поменуту дигитализацију. Такође, примећено је да је главни проблем приликом класификације математичких једначина постојање преклопљених симбола. Стога, што се тиче даљих унапређења, развојем алгоритма који би ефективно раздвајао два преклопљена симбола бисмо могли да повећамо класификациону прецизност система.

Литература

[1] C. Lu and K. Mohan, *Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Network*, 2015.

[2] J. C. P. Patrice Y. Simard, Dave Steinkraus, *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis.*

[3] *Recognition and Solution for Handwritten Equation Using Convolutional Neural Network*, 2018.

[4] *Levenshtein distance.* https://en.wikipedia.org/wiki/Levenshtein_distance.

[5] *OpenCV.* https://opencv.org/.

[6] *SciPy.* https://scipy.org/.