Универзитет „Св. Кирил и Методиј“ - Скопје

Факултет за информатички науки и компјутерско инженерство

CAPTCHA Breaker

Проектна задача по предметот

Машинска Визија

Ментор:

Д-р Андреа Кулаков

Изработиле:

Ангела Маџар, 181010

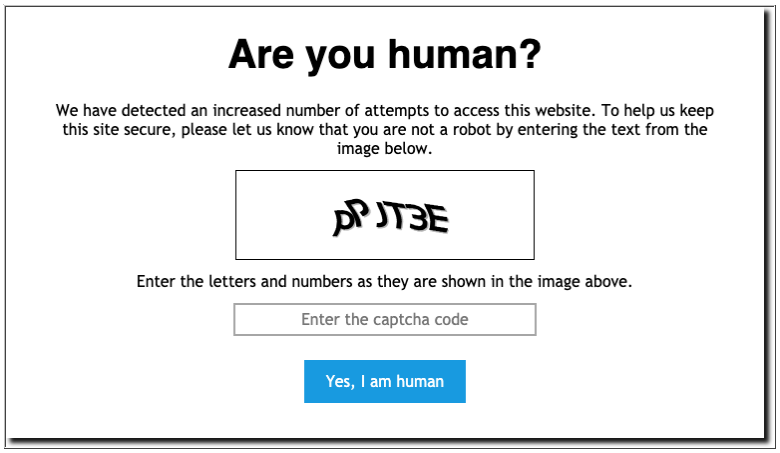
Петар Поповски, 181007

Содржина

1. Вовед

1.1. Што е CAPTCHA?

*“CAPTCHA (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part) е тип на безбедносна мерка позната како challenge-response автентикација.” - Google*



Слика 1. Пример за CAPTCHA

CAPTCHA помага да се заштитат корисниците од спамови и декрипција на нивните лозинки, со тоа што бара од нив да завршат едноставен тест што докажува дека корисникот е човек, а не компјутер (бот програма).

CAPTCHA тестот се состои од два едноставни дела:

- случајно генерирана низа од букви и/или броеви кои се појавуваат како искривена слика
- поле за текст

За да се помине тестот и корисникот да докаже дека е човек, едноставно треба да ги внесе знаците што ги гледа на сликата во полето за текст (Слика 1.).

Captchas имаат различни примени за безбедност, приватност за да ги заштитат веб-страните од каква било штета. Некои од областите каде што се користи CAPTCHA вклучуваат:

- Ублажување на спам за коментари
- Онлајн анкета

- Веб регистрација
- Заштита на доверливи информации на веб-страните
- Паметни картички
- Филтрирање СМС спамови
- Спречување на фишинг
- Спречување на VoIP спам
- Онлајн игри
- Спречување на роботи во социјалната мрежа
- Search engine ботови

1.1.1. Историја

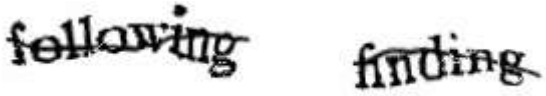
Терминот CAPTCHA е измислен во 2003 година од Луис фон Ан, Мануел Блум, Николас Џ. Хопер и Џон Лангфорд кои сакале да најдат начин да ги филтрираат огромните армии на спам-ботови кои се преправаат дека се луѓе. Најчестиот тип на CAPTCHA (прикажан како верзија 1.0) првпат бил измислен во 1997 година од две групи кои работеле паралелно (Слика 2.).

На просечниот човек му требаат приближно 10 секунди за да реши типична CAPTCHA.

Слика 2. CAPTCHA верзија 1.0

1.1.2. reCAPTCHA

reCAPTCHA е бесплатен сервис во сопственост на Google кој претставува понова верзија на CAPTCHA системот. Оригиналната верзија бара од корисниците да дешифрираат тешко читлив текст или да одговараат на слики (Слика 3.).



Слика 3. reCAPTCHA верзија 1

Во 2013 година, reCAPTCHA започнува да спроведува анализа на однесувањето на корисникот на прелистувачот за да предвиди дали корисникот е човек или бот без да побара од него да реши CAPTCHA проблем. Следната година, Google започнал да го применува новото reCAPTCHA API, со „NO CAPTCHA reCAPTCHA“, каде што корисниците за кои системот смета дека се со низок ризик, односно не се спам/бот програми, треба само да кликнат на едно поле за избор за да го потврдат својот идентитет. Во секој случај, доколку системот не е сигурен за идентитетот на корисникот, може да се прикаже reCAPTCHA.

Во 2017 година, Google воведува нова „невидлива“ reCAPTCHA, каде што верификацијата се случува во позадина и воопшто не се прикажуваат никакви CAPTCHA задачи доколку се смета дека корисникот е со низок ризик, односно не е спам/бот програма.

Во овој проект нема да се разгледуваат проблеми поврзани со reCAPTCHA од верзија 2 (Слика 4.).

Слика 4. reCAPTCHA верзија 2

1.2. Што е CAPTCHA Breaker?

CAPTCHA Breaker е програма која е напишана со цел да го помине CAPTCHA/reCAPTCHA тестот без потреба од човечка интервенција. CAPTCHA Breaker програмата може да е корисна за различни намени како што се **web crawling**, **автоматизација**, **маркетинг,**

оптимизација и многу повеќе. Имено, при некои од овие наведени примери, доколку е потребно, CAPTCHA Breaker програмата може да се интегрира како функција во веќе постоечки апликации со цел да помогне во решавањето на CAPTCHA задачите. Имено, постојат различни пристапи за да се направи ова, но ние во овој труд решивме тоа да го направиме со помош на библиотеки за машинска визија како што е OpenCV и CNN невронски мрежи (Convolutional Neural Network) кои ќе ги разгледаме подетално во точка 2.3.

1.3. Опис на проектната задача

Идејата на овој CAPTCHA Breaker проект ја навестува самото име - автоматско решавање на CAPTCHA предизвиците. Тоа го правиме етапно во неколку чекори. Како влез на моделот се испраќа CAPTCHA слика.

Најпрвин, се користат техники на Машинска Визија за отстранување на секој шум во позадина на CAPTCHA сликите, со цел да се препознаат карактерите во неа за да се поделат на посебни слики.

Следно, секоја CAPTCHA слика се дели на 5 посебни слики во зависност од карактерите во неа. Секоја од тие 5 нови слики се зачувува во посебен фолдер обележан според карактерот на сликата (Слика 5.). На овој начин градиме ново податочно множество кое се состои од секој еден карактер, вклучувајќи ги цифрите и големите латинични букви. Ваквото податочно множество брои приближно 2500 слики кои потоа ги користиме за тренирање и тестирање на моделот.

Слика 5. Новото податочно множество

Тренирањето на моделот со Конволуциска невронска мрежа се прави со 80% од сликите од новото податочно множество, а се тестира со 20% за да се провери точноста на моделот.

Накратко, процесот на класифицирање на секој карактер од CAPTCHA сликата е следен: при влез на сликата, истата се претпроцесира и дели на 5 различни слики во зависност од карактерите. Потоа, секој карактер се испраќа на моделот, кој како таков го предвидува карактерот и ја запишува неговата вредност како string. За крај, го добиваме посакуваниот резултат како низа од 5 карактери.

Сите чекори се детално објаснети во делот Секција *2. Методологија*.

1.4. Податочно множество

Податочното множество кое го користевме за тренирање и тестирање на моделот за CAPTCHA Breaker е јавно достапно на овој линк. Истото се состои од 501 CAPTCHA слики кои се именувани (лабелирани) според текстот на самата слика. Сликите се состојат од искривени карактери кои се комбинации од цифри и големи латинични букви. Со цел да е отежнато читањето за компјутерот, на сликите е додаден шум и дополнителни криви линии (Слика 6.). Секоја слика се состои од точно 5 карактери.

Слика 6. Пример слика од податочното множество

2. Методологија

2.2. Претпроцесирање

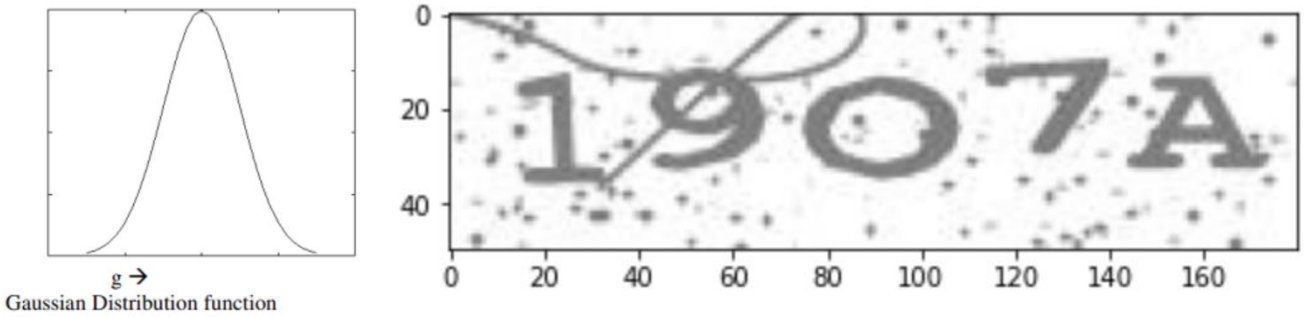
Претпроцесирањето е чекор во кој податоците се прават полесни за интерпретирање. Кај визуелните податоци, односно сликите, целта на претпроцесирањето е да се подобри квалитетот на сликата за истата да може да се анализира на поефикасен начин. Може да се отстранат непосакувани дисторзии, а да се потенцираат значајни карактеристики на сликата.

2.2.1. Справување со шум

Шумот во сликите е присуство на артефакти кои не потекнуваат од оригиналната содржина на сликата. Генерално, може да се дефинира како случајна варијација на информација за светлина или боја во сликата. Честопати, шумот се појавува во зрнеста структура насекаде низ сликата.

Постојат различни типови на шум, а еден од нив е Гаусовиот шум односно случаен шум. Се дефинира како статистички шум со густина на веројатност еднаква на нормалната дистрибуција. Има униформна распределба на сликата.

При набљудување на сликите од податочното множество, може да се забележи случаен, речиси рамномерно распределен шум во позадина на карактерите (Слика 7.).

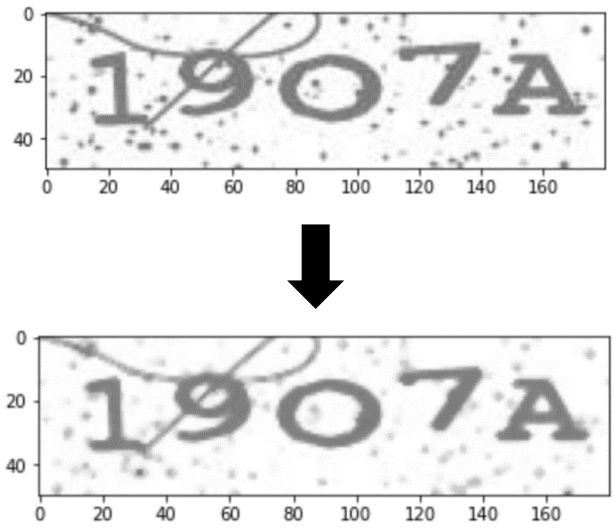


Слика 7. Одредување на типот на присутниот шум

Median Blur Филтер

Median филтерот е познат по добрите перформанси за специфични типови на шум, како што се Гаусовиот и случаен шум. Функционира така што централниот пиксел на соседството M x M се заменува со средната вредност на соодветниот „прозорец“. Од голема важност е да се напомене дека пикселите кои содржат шум имаат вредност која значајно се разликува од средната вредност на околните пиксели. Оваа идеја е и суштината на употребата на Median филтерот, кој може да се справи со вакви проблеми на шум.

Овој филтер е применет врз сликите од CAPTCHA податочното множество за истите да се измазнат и да се подготват за понатамошно претпроцесирање (Слика 8.)

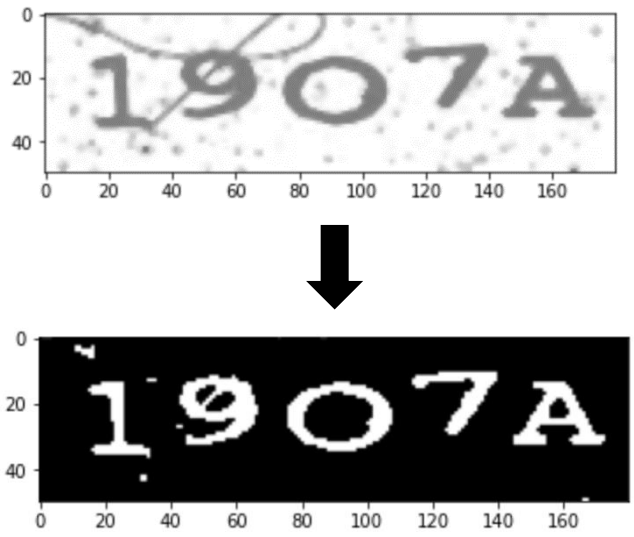


Слика 8. Слика пред и по примена на Median Blur филтер

Binary Threshold

Оваа техника се користи како начин на селекција на областите од интерес на сликата, притоа игнорирајќи ги деловите кои немаат важност во конкретната намена. Важи за едноставен, но ефикасен метод за сегментација на слики. При thresholding, се врши конверзија на слика со боја или слика со сиви тонови (grayscale) во бинарна (црно-бела) слика, така што се споредува вредноста на секој пиксел со зададената гранична вредност. Доколку вредноста на пикселот е помала од граничната (thresholding), пикселовиот интензитет добива вредност 0.

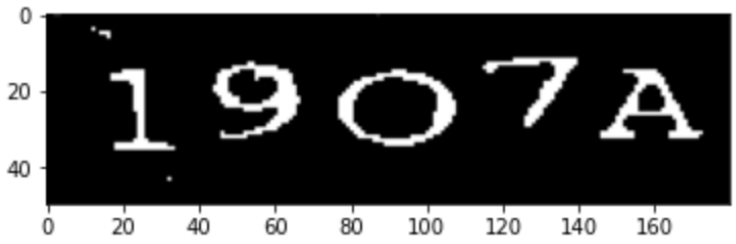
Во конкретниот случај, карактерите имаат помал интензитет од позадината, па затоа беше соодветно да се употреби **inverse binary thresholding** (Слика 9.). Така, доколку вредноста на пикселот е поголема од гранчната, добива вредност 0 (позадината добива црна боја). Останатите пиксели добиваат вредност 255 (карактерите добиваат бела боја). Граничната вредност беше одредена со повеќекратни обиди и визуелна инспекција.



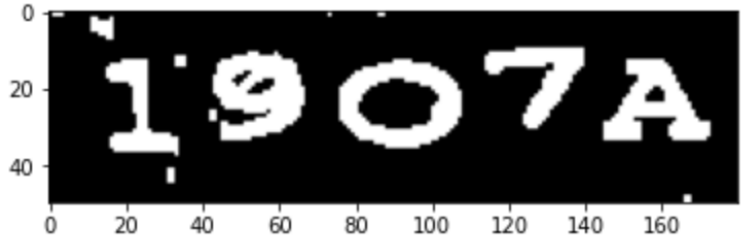
Слика 9. Слика пред и по примена на Inverse Binary Thresholding

Dilate и Erode Филтри

Морфолошките филтри се користат за заострување на сликата, а Dilate и Erode филтрите се двата основни морфолошки оператори. Dilate го избира најсветлиот пиксел во околината и додава пиксели на границата на објектите во сликата, додека Erode го избира најтемниот пиксел, отстранувајќи пиксели од границата на објектите. Во претпроцесирањето на сликите од податочното множество, покрај тоа што експериментиравме со двата филтри, експериментиравме и со нивна последователна употреба. Употребада на Erode по Dilate се нарекува **Closing**, а обратната употреба се нарекува **Opening**. Сепак, заклучивме дека Dilate филтерот најдобро ја подготвува сликата за понатамошно претпроцесирање. Резултатите од овие два филтри може да се погледнат на Слика 10. и Слика 11.



Слика 10. Ефект на Erode филтерот

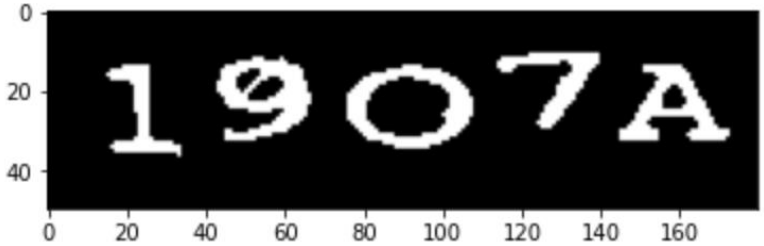


Слика 11. Ефект на Dilate филтерот

Детекција на поврзани компоненти

На Слика 10. и Слика 11. Воочливо е натамошно присуство на преостанат шум. Регионите на преостанат шум се значително помали од регионите на карактерите. За да се отстранат ваквите малечки делови на шум, користевме метод за детекција на поврзани компоненти. Поврзана компонента е множество на поврзани пиксели со специфично својство V. Два пиксели p и q се поврзани доколку патот од p до q има својство V.

За детекција на поврзани компоненти ја искористиме функцијата **ConnectComponentsWithStats** од OpenCV. На Слика 12. е прикажан ефектот на оваа функција. Воочливо е дека нејзината употреба за справување со минимален преостанат шум резултира со успех.

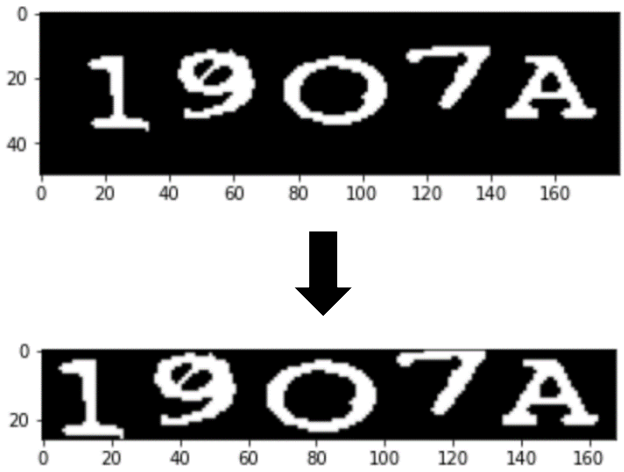


Слика 12. Ефект на ConnectComponentsWithStats

Елиминација на непотребна позадина

За да сме сигурни дека сме се ослободиле од потенцијален преостанат шум кој би можел да се наоѓа по рабовите на сликата, одлучивме да ја отстраниме непотребната позадина околу карактерите. За да го направиме ова, набљудувавме на која позиција од сликата се поставени карактерите и заклучивме дека најчесто, сите карактери започнуваат по 10-тиот пиксел, а завршуваат пред 180-тиот пиксел на x-оска.

Резултатот од ваквиот процес е прикажан на Слика 13.



Слика 13. Процес на отстранување на непотребна позадина

2.2.2. Детекција и поделба на карактери

Поделба на карактерите со Canny Edge Detector

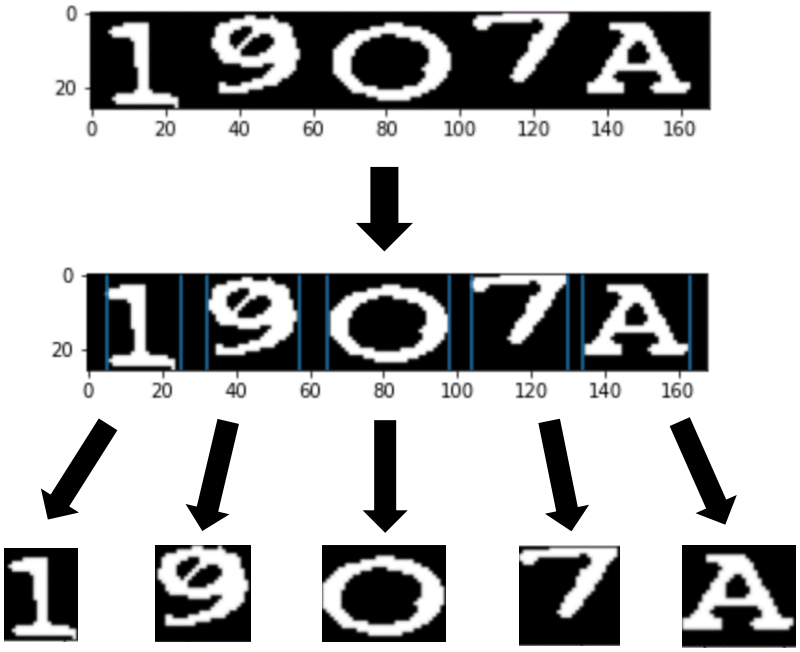
Идејата во оваа проектна задача е првично да се детектираат карактерите на сликата. Потоа, сликата да се подели на под-сликички во кој е присутен по еден карактер и да се истренира модел врз основа на така добиените слики. При влез на нова и непозната captcha слика, излезот се одредува врз основа на поединечна класификација на секој карактер од таа слика.

За детекција на рабовите, односно почетокот и крајот на секој карактер во сликата, користевме **Canny Edge Detector**. Оваа техника се користи за екстрахирање на корисни структурни информации од објекти присутни на сликата, значително намалувајќи го количеството на информација за обработка.

Се карактеризира со:

- **Добра детекција:** можност да детектира и означи рабови онаму каде се присутни
- **Добра локализација:** минимално растојание помеѓу детектираните рабови и вистинските рабови
- **Јасен одговор:** еден детектиран раб за секој вистински раб

Во нашиот случај, Canny Edge Detector се користеше за детекција на вертикални линии во кои сите пиксели имаат вредност 0, односно имаат црна боја. На овој начин ги одредивме координатите на x-оската на секој карактер присутен во сликата. Потоа, го „отсековме“ секој карактер врз основа на вака добиените координати и го зачувавме како посебна слика во соодветен фолдер (Слика 14.). Овој процес резултираше со 36 различни фолдери, односно категории, што е соодветно на бројот на уникатни карактери во лабелите на сликите (Слика 5.).



Слика 14. Детекција и поделба на карактери со Canny Edge Detector

2.3. CNN невронски мрежи

2.3.1. Невронски мрежи

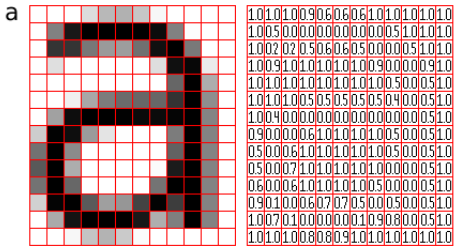
Невронски мрежи или Вештачки невронски мрежи се компјутерски и математички модели инспирирани од биолошките невронски мрежи од кои се состои човечкиот мозок. Минималните пресметковни елементи на вештачката невронска мрежа се нарекуваат *неврони*. Невронските мрежи обично се состојат од три или повеќе слоеви: *влезен* слој, *скриен* слој (или слоеви) и *излезен* слој. Во некои случаи не се земаат предвид влезните и излезните слоеви, а за бројот на слоеви во мрежата се зема бројот на скриени слоеви.

Важна карактеристика на невронската мрежа е нејзината способност да учи од зададени примери од податочното множество. Невронската мрежа се тренира на голем број примери кои се состојат од влезно-излезни парови. Во проблемите со препознавање објекти (object detection), таков пар ќе биде влезната слика и лабелата што одговара на неа - името на објектот. Тренирањето за невронска мрежа е итеративен процес кој постепено ја намалува грешката при предвидувањето. Овој процес се состои од чекори наречени епохи, од кои секоја ги прилагодува „тежините“ на невронската мрежа, односно параметрите на скриените слоеви на мрежата. На крајот од процесот на тренирање, перформансите на невронската мрежа обично се доволно добри за да се изврши задачата за која е тренирана, иако често е невозможно да се најде оптималниот сет на параметри што совршено ги препознава сите влезови.

2.3.2. Што се конволуциски невронски мрежи (CNN)

Конволуциските невронски мрежи се тип на длабоки невронски мрежи (Deep Neural Networks) кои се состојат од повеќе скриени слоеви и се специјализирани за обработка на податоци што имаат топологија слична на мрежа (grid/matrix), како што е Слика 15.

Дигитална слика е бинарна репрезентација на визуелни податоци. Содржи серија пиксели распоредени на мрежен начин и вредности на пиксели за да означи каква боја и со колкав интензитет треба да биде секој пиксел (Слика 15.).



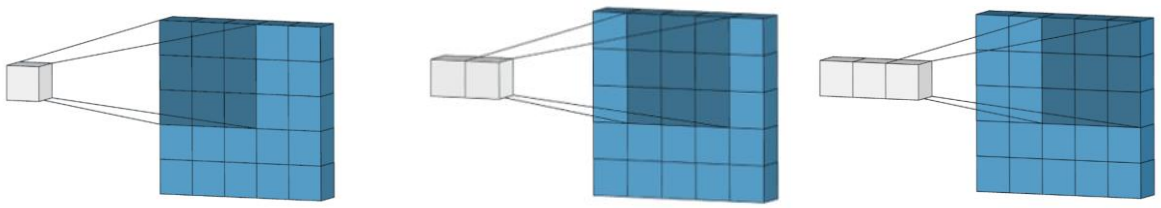
Слика 15. Репрезентација на слика како мрежа од пиксели

Архитектура на CNN
CNN обично имаат три слоја:

- **конволуциски** слој
- **pooling** слој
- **целосно поврзан** слој (Fully connected layer)

Конволуцискиот слој е столбот при градење CNN и го носи главниот дел од пресметковното оптоварување на мрежата.

Овој слој пресметува скаларен производ помеѓу две матрици, каде што едната матрица е збир на параметри што може да се научат, инаку познати како кернел, а другата матрица е ограниченият дел од приемното поле, во нашиот случај – дел од сликата. Јадрото, односно кернелот е просторно помал од сликата, но е подлабок. Ова значи дека, ако сликата е составена од три (RGB) канали, висината и ширината на јадрото ќе бидат просторно мали, но длабочината се протега до сите три канали.



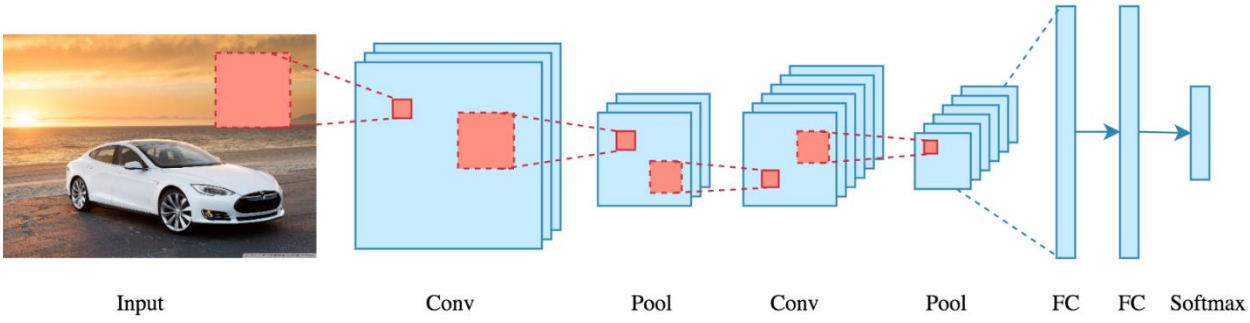
Слика 16. Дел од анимација при конволуција

Тривијалните слоеви на невронска мрежа користат множење на матрица со матрица од параметри што ја опишуваат интеракцијата помеѓу влезната и излезната единица. Ова значи дека секоја излезна единица е во интеракција со секоја влезна единица. Сепак, конволуциските невронски мрежи имаат *sparse* интеракција. Ова се постигнува така што јадрото е секогаш помало од влезот, односно сликата во случајов.
Што значи ова?
Сликата може да има милиони или илјадници пиксели, но додека ја обработуваме/изминуваме со помош на кернелот, можеме да откриеме значајни информации кои се од десетици или стотици пиксели. Ова значи дека треба да складираме помалку параметри кои, не само што ги намалуваат потребите за меморија на моделот, туку и ја подобруваат статистичката ефикасност на моделот. Новата матрица која се формира со лизгање на кернелот/филтерот преку сликата и пресметувањето на скаларниот производ истовремено се нарекува **Feature Map.**

Ако пресметувањето на една карактеристика во просторна точка (x1, y1) е корисно, тогаш треба да биде корисно и во некоја друга просторна точка, да речеме (x2, y2). Тоа значи дека за едно дводимензионално парче, т.е., за создавање на една активациска мапа, невроните се ограничени да користат ист сет на тежини. Во традиционалната невронска мрежа, секој елемент од тежинската матрицата се користи еднаш, а потоа никогаш не се ,,посетува" пак, додека конволуциската мрежа има заеднички параметри, т.е., за добивање излез, тежините што се применуваат на еден влез се исти како тежината што се применува на друго место (Слика 16.).

Pooling слојот се извршува после конволуцијата за да ја намали димензионалноста на секоја feature map. Ова ни овозможува да го намалиме бројот на параметри и пресметки во мрежата, со што контролираме да не дојде до overfitting.
CNN користи *max-pooling*, во кое дефинира просторно соседство што значи го зема најголемиот елемент од feature мапата во тој прозорец. По слојот за здружување, нашата мрежа станува инваријантна за мали трансформации, изобличувања и транслации во влезната слика.

По конволуциските и pooling слоевите, додаваме неколку **целосно поврзани (fully connected)** слоеви за да ја заокружиме архитектурата на CNN. Излезот од слоевите на конволуција и pooling претставуваат high-level карактеристики на влезната слика. Fully connected слоевите ги користат овие карактеристики за класификација на влезната слика во различни класи врз основа на тренирачкото множество.



Слика 17. Целосна архитектура на CNN за предвидување на класата на објект (автомобил)

Гледајќи ја големата слика (Слика 17.), CNN архитектурата исполнува две главни задачи: екстракција на карактеристики/features (конволуција + pooling) и класификација (целосно поврзани слоеви). Општо земено, колку повеќе чекори на конволуција имаме, толку покомплицирани карактеристики нашата мрежа ќе може да научи да препознава.

Областите на примена на конволуциските невронски мрежи не се ограничени само на препознавање објекти. Тие се широко користени за препознавање на говор и аудио, обработка на читања од различни типови сензори или за сегментирање сложени повеќеслојни слики (како што се сателитски карти) или медицински слики (X-зраци или слики од MRI).

3. Тренирање

Оваа секција содржи дефиниција на моделот кој ќе се тренира со 80% од креираното податочно множество. Конволуциските Невронски Мрежи се соодветен избор кога станува збор за препознавање на слики.

Нашиот модел е составен од **два конволуциски слоеви** и **два целосно поврзани** (fully-connected) **слоеви**. Првиот конволуциски слој очекува **29 x 28 x 1** тензор на слика, каде 29 и 28 се висина и ширина на сликата, додека 1 е бројот на канали на сликата (бидејќи нашите слики се grayscale, бројот на канали е еднаков на 1).

Но, зошто токму 29 x 28? Pytorch конволуциските мрежи очекуваат слики со фиксна големина, односно влезните слики мора да имаат иста висина и ширина. За да ги сведеме сите добиени слики од индивидуални карактери на иста димензија, пресметавме *средна висина и ширина на сите слики*. Пред да ги испратиме на конволуциската мрежа, ја променивме нивната димензија во соодветно добиените вредности, односно 29 x 28.

Параметрите на првиот целосно поврзан слој се исто така важни. Прима тензор со влезна големина **25 * 24 * 3**. Секогаш кога поминуваме низ конволуциски слој, се намалува големината на влезниот тензор соодветно на големината на кернелот (јадрото), во отсуство на падинг. Така, примена на 3 x 3 кернел го намалува влезниот тензор за 1 одгоре, оддолу, одлево и оддесно. Затоа, излезот од првиот конволуциски слој е **27 x 26 x 3**, а од вториот конволуциски слој е **25 * 24 * 3.**

На Слика 18. е прикажана архитектурата на нашата невронска мрежа.

```
# training a convolutional neural network with parameters that correspond to the input images
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # convolutional layer (sees 29x28x1 image tensor)
        self.conv1 = nn.Conv2d(1, 3, kernel_size=3)
        # convolutional layer (sees 27x26x3 tensor)
        self.conv2 = nn.Conv2d(3, 3, kernel_size=3)
        # linear layer (25 * 24 * 3 -> 108)
        self.fc1 = nn.Linear(25*24*3, 108)
        # linear layer (108 -> 36)
        self.fc2 = nn.Linear(108, 36)
    def forward(self, x):
        # sequance of convolutional layers with relu activation
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        # flatten the image input
        x = x.view(-1, 25*24*3)
        # 1st hidden layer with relu activation
        x = F.relu(self.fc1(x))
        # output-layer
        x = self.fc2(x)
        return x
```

Слика 18. Архитектура на Конволуциската Невронска Мрежа

Во текот на тренирањето, дефиниравме loss функција и optimizer за ажурирање на тежините на моделот. **CrossEntropyLoss** е соодветна функција за проблеми на мулти-класификација, имајќи предвид дека имаме 36 различни категории на карактери. За оптимизација користевме **SDG**, т.е. **Stochastic Gradient Descent**.
Тренирањето вклучува 5 чекори (Слика 19.):

- **optimizer.zero_grad()** – Pytorch ги акумулира градиентите на последователни backward passes за кои треба да се анулираат градиентите во секоја итерација
- **net(inp)** – прави feed forward на batch (size = 6) низ слоевите и ги зема излезните резултати
- **criterion(outs, lab)** – ги користи лабелите и излезните резултати за пресметка на загубата (loss) врз основа на дефинираната loss функција
- **loss.backward()** – го пресметува градиентот за секој параметар
- **optimizer.step()** – ги ажурира параметрите врз основа на градиентите

```
# specify loss function (categorical cross-entropy)
criterion = nn.CrossEntropyLoss()

# specify optimizer
optimizer = torch.optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

n_epochs = 70

for epoch in range(n_epochs):
    running_loss = 0.0
    for i, (inp, lab) in enumerate(dloader, 0):
        inp = inp.to(device)
        lab = lab.to(device)

        # clear the gradients
        optimizer.zero_grad()

        # forward pass
        outs = net(inp)

        # batch loss
        loss = criterion(outs, lab)

        # backward pass
        loss.backward()

        # perform optimization(parameter update)
        optimizer.step()

        running_loss += loss.item()

        if i % 100 == 99:
            print('[%d, %d] loss: %f' % (epoch + 1, i+1, running_loss))
            running_loss = 0.0

print('finished')

# save the trained model
torch.save(net, 'C:\\Users\\angelamadjar\\Desktop\\model.pt')
```

Слика 19. Тренирање на моделот

Бројот на епохи и големината на batch беа одредени по повеќекратно пробување со

различни вредности. Вредноста на loss-функцијата во почетните и крајните епохи е прикажана на Слика 20.

```
[1, 100] loss: 358.213955
[2, 100] loss: 356.438898
[3, 100] loss: 348.904238
[4, 100] loss: 249.887382
[5, 100] loss: 113.525102
...
[66, 100] loss: 0.183065
[67, 100] loss: 0.501343
[68, 100] loss: 0.393992
[69, 100] loss: 0.551126
[70, 100] loss: 0.130518
finished
```

Слика 20. Вредност на Loss-function

4. Тестирање

Тестирањето на перформансите на моделот ги вклучува истите трансформации користени во процесот на тренирање. Разликата е податочното множество кое се користи, односно преостанатите 20% од добиените слики на карактери кои се непознати за истрениранот модел.

Како метрика за перформансите на моделот избравме **Accuracy** бидејќи суштински станува збор за проблем на класификација.

Во сегментот на код прикажан на Слика 21. **net** варијаблата го содржи моделот кој го истрениравме. Бидејќи немам потреба од back-propagation при тестирање, се користи **torch_grad** за да се забрза времето на извршување. Кодот проверува дали моделот точно ги класифицира карактерите од сликите и пресметува accuracy. Прикажана е и вредноста на accuracy на моделот која изнесува 91.11%.
Битно е да се потенцира дека добиената вредност на accuracy важи за класификација на поединечен карактер. Оттука, може да претпоставиме дека за една captcha слика со 5 карактери, моделот ќе има accuracy = 91.11 ^ 5.

```
total = 0
correct = 0

net = torch.load('C:\\Users\\angelamadjar\\Desktop\\model.pt').to(device)
with torch.no_grad():
    for i, (inp,lab) in enumerate(dloader,0):
        inp = inp.to(device)
        lab = lab.to(device)

        # forward pass
        outs = net(inp)
        # convert output scores to predicted class
        _, pred = torch.max(outs,1)
        correct += (pred == lab).sum().item()
        total += lab.size(0)

print('Accuracy: %f %%' % (100*correct/total))
```

```
Accuracy: 91.111111 %
```

Слика 21. Тестирање на моделот

5. Резултати и заклучок

Иако денешните верзии на CAPTCHA и reCAPTCHA се софистицирани за да избегнат нивно пробивање, ова беше интересен предизвик од областа на Машинска визија. Резултатите кои ги добивме со тренирање и тестирање на моделот кој го изработивме се задоволителни, односно **точноста** изнесува **91.1%**, додека **loss функцијата** при тренирање достигна вредност од **0.130518**. На Слика 22. е прикажана практична примена на моделот.

```
break_captcha("C:\\Users\\angelamadjar\\Documents\\Downloads\\archive (1)\\teltest2\\9H2PW.png")
```

```
'9H2PW'
```

```
break_captcha("C:\\Users\\angelamadjar\\Documents\\Downloads\\archive (1)\\teltest2\\02KKK.png")
```

```
'02KKK'
```

Слика 22. Практична примена на моделот

Како што програмите од типот на овој проект стануваат пософистицирани, CAPTCHA предизвиците ќе треба да се носат со идните проблеми за нивно детектирање. Некои експерти сугерираат дека captcha проверките во иднина може целосно да се заменат со биометриски проверки - како брзо скенирање на очите.

Идни насоки за подобрување на моделот би биле користење на поголемо податочно множество или пак аугументација на веќе постоечкото, со што моделот би бил поотпорен на покомплексен шум во CAPTCHA сликите. Дополнително, користењето на пософистициран метод за детекција на објекти на слика би придонел за поголема точност на моделот.

6. Референци

1. https://medium.com/spidernitt/image-preprocessing-why-is-it-necessary-8895b8b08c1d
2. https://analyticsindiamag.com/a-guide-to-different-types-of-noises-and-image-denoising-methods/
3. https://datacarpentry.org/image-processing/07-thresholding/
4. https://www.sciencedirect.com/topics/engineering/morphological-filter
5. https://geekflare.com/captcha-solving-services-api/
6. https://en.wikipedia.org/wiki/CAPTCHA
7. https://www.mentalfloss.com/article/81927/surprisingly-devious-history-captcha
8. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20Convolutional%20Neural%20Network%2C%20also,binary%20representation%20of%20visual%20data.
9. https://www.codementor.io/@james_aka_yale/convolutional-neural-networks-the-biologically-inspired-model-iq6s48zms
10. https://www.tsohost.com/blog/6-reasons-why-your-website-needs-a-captcha-form