Profielwerkstuk David Schep V6B Begeleider: E.P. Izeboud

Mens Vs. Computer

Waarom de mens niet meer te onderscheiden is van de computer.

14 Februari 2017

Inhoudsopgave

Ontwerpprobleem

Iedereen herkent het, onderaan een inlogpagina. Eén klein plaatje met een hele partij onleesbare letters. Hierbij is het idee dat wij die letters moeten herkennen en in moeten vullen. Wat is het nut hiervan eigenlijk? Is dit de enige oplossing? Kunnen wij, mensen, echt niks beters bedenken?

Deze plaatjes worden Captcha's genoemd; Computer Automated Public Turing Tests to tell Computers and Humans Apart. Deze naam betekent dat het doel van een captcha is om mensen en computers van elkaar te onderscheiden. Hoe dit gedaan moet worden, is al een lange tijd een grote en moeilijk te beantwoorden vraag. Na veel onderzoek en hard werken hebben onderzoekers ontdekt dat mensen erg goed zijn in het herkennen van patronen. Wanneer een mens een boom ziet, weet hij dat het een boom is. Als een mens een stuk tekst leest, weet hij van elke letter exact welke het is. Deze laatste ontdekking heeft men vooral gekozen om computers van mensen te onderscheiden. Computers zijn namelijk niet zo goed in het herkennen van plaatjes of het herkennen van letters. Voor computers zijn letters maar rijen van pixels.

Het volgende probleem was om deze ontdekking op een praktische manier toe te gaan passen. Dit bleek nog niet zo eenvoudig. De oplossing moet natuurlijk geautomatiseerd worden, met een computer. Een computer moet dus iets maken wat een andere computer niet kan herkennen. Met dit in gedachte is men tot de oplossing gekomen om de gebruiker een plaatje te laten zien. In dit plaatje wordt door een computerprogramma een rij letters gezet. Deze letters moeten zo op dit plaatje staan dat ze niet te veel op gewone letters lijken maar wel genoeg dat de mens nog kan herkennen welke letter dit is. Het computerprogramma neemt dus de originele letters en vervormt deze door bijvoorbeeld de pixels van de letters uit te rekken of in te krimpen. Hiernaast worden nog veel meer methodes gebruikt om captcha's onleesbaar te maken voor computers, elk met een specifiek doel. Met deze methode zijn captcha's jarenlang geweldig geweest in het onderscheiden van mensen en computers. De laatste tijd is dit anders geworden. Computers zijn slimmer geworden, de rekenkracht is met enorme stappen vooruitgegaan en nu heeft zelfs een eenvoudige Casio rekenmachine meer rekenkracht dan de computer waarmee vroeger de Apollo-missies zijn berekend. Door deze recente ontwikkelingen en ook door al het onderzoek dat gedaan wordt op het gebied van fotoherkenning en patroonherkenning, hebben computers hierin een extreme vooruitgang gemaakt. Met deze vooruitgang zouden computers captcha's toch met gemak op moeten kunnen lossen?

Dit is het probleem dat ik uiteen wil zetten; de oude vertrouwde captcha's zijn niet effectief meer, ze kunnen ons niet meer beschermen tegen de oneindige hoeveelheid bots die hackers op ons af sturen. Met dit profielwerkstuk ga ik bewijzen dat captcha's verledentijd zijn door een script te schrijven dat met een eenvoudige computer Captcha's met een hoge precisie kan oplossen.

Programma van eisen

Om te bewijzen dat captcha's niet veilig meer zijn is het belangrijk dat het script een captcha moet kunnen oplossen. Het percentage goed is hierbij niet belangrijk. Bij captcha's krijg je namelijk oneindig veel pogingen. Vaak een captcha fout hebben en dan uiteindelijk de captcha goed hebben wordt nog steeds gezien als een mens zijn.

Dit script wordt gericht op één soort captcha aangezien elke captcha anders is en het onmogelijk gaat zijn een script te maken om elke verschillende soort op te lossen. Daarom heb ik gezocht voor de meest gebruikte. Na een hele tijd onderzoeken heb ik gevonden dat de meest gebruikte captcha de SecurImage (Drew Philips, 2005 – 2015) captcha is.

Ontwerpvoorstel

Het programma moet dus captcha's op kunnen lossen. Specifiek de SecurImage captcha. Zie bron 1. Bij het schrijven van het programma ga ik zo veel mogelijk van de code zelf bedenken. Ik ga zo min mogelijk externe hulp gebruiken. Voor sommige aspecten van het script gaat het echter essentieel zijn om code van andere mensen te hergebruiken, maar grotendeels wil ik zelf op de code komen.

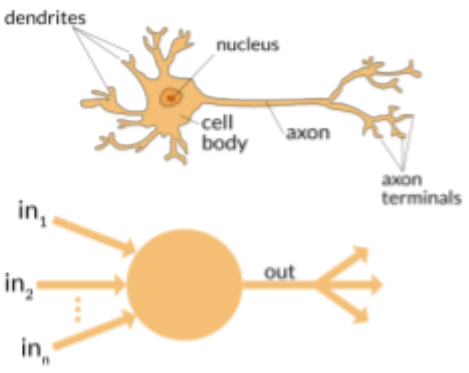
*Bron SEQ Bron * ARABIC 1 SecurImage captcha*

Voor het oplossen van deze captcha moeten enkele moeilijkheden overwonnen worden. Het doel bij het oplossen van een captcha is om de computer de letters te laten herkennen. Om dit zo moeilijk mogelijk te maken heeft de SecurImage captcha vele beveiligingen, elk met een specifiek doel. Voor het oplossen moeten wij goed beredeneren welke moeilijkheden en problemen wij tegen gaan komen. Het eerste probleem wat wij tegen komen, is het herkennen van de vervormde letters. De letters zijn niet simpelweg op één lijn geplaatst en dat maakt het moeilijk om de letters van links naar rechts te lezen. De oplossing hiervoor is om de letters los van elkaar te lezen. Dit is dan ook direct de eerste stap. Namelijk de letters segmenteren om deze los van elkaar te gaan herkennen. Bij het segmenteren komen wij ook enkele problemen tegen. Namelijk lijnen, ruis en vervorming van de letters. Deze 3 problemen worden bij het uiteindelijke ontwerp verder uit gewerkt.

Het volgende probleem waar wij tegen aan lopen, is dat de gesegmenteerde letters moeilijk te herkennen zijn. Ze hebben altijd een andere vorm en staan nooit mooi op een lijntje. Computers zijn altijd al slecht geweest in herkennen van patronen. Toch is het mogelijk om patronen te herkennen. Dit kan een computer doen door middel van een neuraal netwerk.

Neurale netwerken

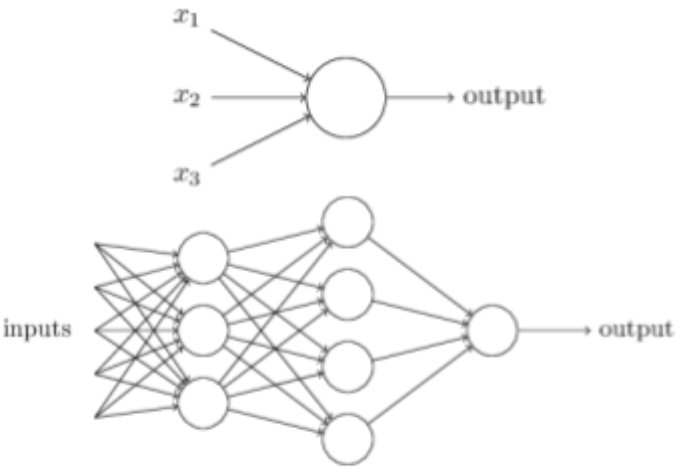
Wat is een neuraal netwerk? Een neuraal netwerk is een soort simulatie van de hersenen. De eenvoudigste versie van een neuraal netwerk is een perceptron. Een perceptron simuleert één enkele menselijke neuron. De perceptron krijgt inputs net zoals een menselijk neuron deze krijgt. Een neuron evalueert dan de inputs en geeft dan een zekere output. Zie Bron 2. Bij een perceptron is dit alles, hiermee kan het hele eenvoudige problemen oplossen maar als wij een systeem willen waarmee wij moeilijkere problemen kunnen oplossen en patronen kunnen herkennen, hebben wij een perceptron netwerk nodig, ook wel een neuraal netwerk genoemd. Dit netwerk bestaat uit meerdere perceptrons die elk meerdere inputs krijgen, deze evalueert en dan outputs doorgeeft aan andere perceptrons. Zie Bron 3. Voordat een neuraal netwerk patronen kan herkennen, moet het netwerk getraind worden. Bij het trainen van een netwerk wordt gekeken naar met welke waardes de perceptrons de inputs evalueert. Elke perceptron heeft een zeker gewicht op de inputs. Als een perceptron inputs binnenkrijgt dan beslist hij de output op de waarde van de inputs en de waarde van de gewichten op die inputs. Bij het trainen van een perceptron worden deze



Bron 2 Menselijke neuron vs. Perceptron (Berger)

gewichten aangepast totdat bij een bepaalde input de perceptron de gewenste output geeft. Bij het trainen van een neuraal netwerk worden van alle perceptrons berekend wat de gewenste waardes zijn. Er zijn vele manieren voor het trainen van neurale netwerken. De methode hier toegepast, heet de *backpropagation method*. Bij backpropagation wordt vanuit de laatste perceptron, de output perceptron, teruggerekend wat de waardes in de perceptrons in de laag ervoor zouden moeten zijn. Dit gaat zo door alle lagen van perceptrons heen, totdat de input perceptrons bereikt zijn. Bij het trainen van een neuraal netwerk wordt deze stap honderden keren herhaald totdat een neuraal netwerk steeds beter de data kan onderscheiden.

Voor het herkennen van patronen is een backpropagation neuraal netwerk dus een goede oplossing. Na het segmenteren van de letters in de captcha, wordt een netwerk getraind op het herkennen van de letters. Bij het trainen wordt een netwerk van 900 input perceptrons, 200 verborgen perceptrons en ongeveer 50 output perceptrons gebruikt. Het aantal input perceptrons staat gelijk aan het aantal pixels die de uiteindelijke letters zullen hebben. Het aantal output perceptrons hangt af van het aantal verschillende letters en cijfers dat de captcha gebruikt. Het aantal verborgen perceptrons is een arbitrair gekozen getal, dit maakt niet heel veel verschil.



Bron 3 Perceptron vs. Perceptron netwerk(Nielsen)

Bij het correct trainen van het neurale netwerk is veel voorbeeld data nodig. Letters die uit de captcha zijn gehaald en waarvan bekend is wat de letter is. Door dan het netwerk voor een lange tijd te laten trainen op de verzamelde data zal het netwerk steeds beter letters uit nieuwe data weten te herkennen.

Ontwerp

Alle programmeercode is te vinden op mijn persoonlijke Github:
https://github.com/DavidEncrypted/captcha

Ook zijn daar alle vorige versies te vinden, tijdens het programmeren zijn vaak verschillende versies opgeslagen zodat goed te zien is hoe het script steeds uitgebreider en beter is geworden.

Allemaal is het geschreven in python. Dit was de correcte keuze omdat python een erg handige manier heeft van code hergebruiken die andere mensen geschreven hebben.

Het ontwerp bestaat uit 3 verschillende stappen:

1. Segmentatie
2. Neuraal netwerk trainen
3. Neuraal netwerk gebruiken

<u>Stap 1: Segmentatie</u>

Bij de segmentatie stap wordt de captcha gescand en worden de letters los van elkaar opgeslagen. Bij het ontwerpen van deze captcha heeft men vele beveiligingen geïmplementeerd om deze stap zo moeilijk mogelijk te maken. De eerste beveiliging is ruis in de vorm van losse groepjes pixels door de captcha heen. De tweede beveiliging is lijnen door de letters heen. De derde beveiliging is letters vervormen om het zo moeilijk mogelijk te maken voor programma's om de letters puur op vorm te herkennen.

Voor elke beveiliging zijn dit de oplossingen:

a) Ruis

Bron 4 Captcha voor en na ruis-verwijdering (SecurImage)

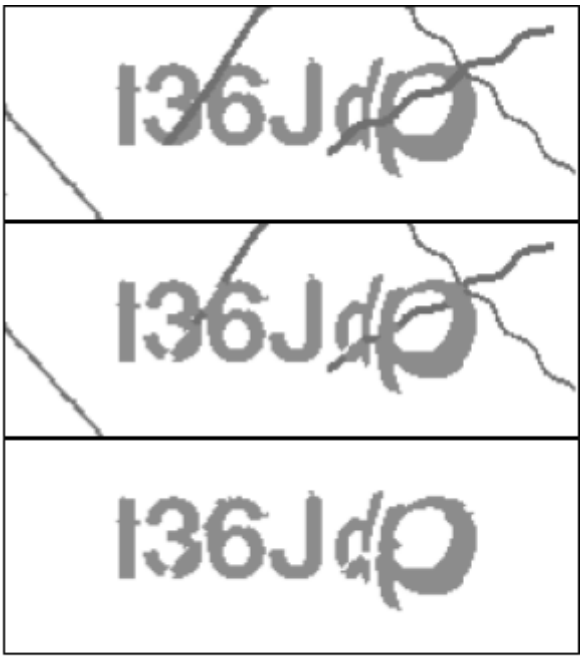
De stappen om ruis te verwijderen:

1. Verzamel alle groepjes van losse pixels kleiner dan 15:
    a. Van de omringende pixels:
        i. Als het grootste gedeelte van deze pixels wit is:
            1. Maak alle pixels in de groep wit
        ii. Als het grootste gedeelte van deze pixels onderdeel zijn van de letters:
            1. Maak alle pixels in de groep letterkleur

Dit is geïmplementeerd in de functie *remove_groups_touching_white()* Zie appendix A.

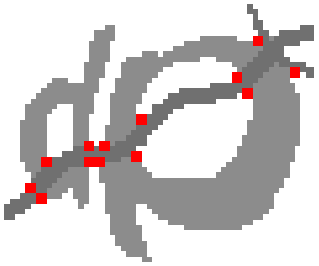
b) Lijnen

De stappen om lijnen te verwijderen:

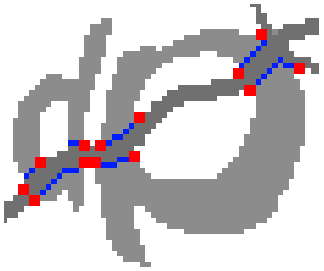


Bron 5 Captcha voor en na lijn-verwijdering (SecurImage)

1. Verzamel alle hoekpunten die lijn, letter en achtergrondpixels raken (Lijnletterachterpunten):



2. Verzamel alle punten waarbij de lijnpixels en letterpixels elkaar raken:



3. Als deze punten op meer dan 3 pixels afstand van de lijnletterachterpunten staan, worden de letterpixels de lijn in gegroeid:

4. Hierna worden alle lijnpixels verwijderd en zijn alleen de letterpixels over:

Dit is geïmplementeerd in de functie *grow_letter_into_line().* Zie Appendix A.

c) Vervorming

Het vervormen maakt het moeilijk om letters op vorm te herkennen. Om dit te omzeilen wordt een neuraal netwerk gebruikt. Dit wordt verder uitgewerkt in het onderdeel Neuraal netwerk trainen.

Wanneer de lijnen en het ruis verwijderd zijn, moeten alle pixels die bij één enkele letter horen verzameld worden. Aangezien het lijn verwijderen niet perfect is, blijven er vaak nog gaten in de letters over. Dit maakt het moeilijk om gemakkelijk alle letters los van elkaar te verzamelen. Om toch de letters goed te verzamelen, worden eerst van alle pixels die elkaar raken groepen gemaakt. Als er in totaal 6 groepen met pixels zijn hebben wij alle letters los van elkaar verzamelt maar als het aantal groepen groter is dan 6 moeten de dichtstbijzijnde groepen bij elkaar gevoegd worden. Dit gebeurt door vanuit het centrum van de pixelgroepen naar de dichtstbijzijnde pixel van een andere groep te zoeken. Bij deze zoektocht ligt een extra gewicht op verticale afstand. Als een pixel dus verticaal 5 pixels van het centrum af ligt en een andere pixel horizontaal 3 pixels, zal de pixelgroep toch bij de pixelgroep met de afstand van 5 pixels verticaal worden gevoegd. Want wanneer de letters worden doorgesneden door lijnen, zitten de pixelgroepen veel vaker boven elkaar dan naast elkaar. Het bij elkaar voegen van pixelgroepen wordt herhaald totdat er nog maar 6 pixelgroepen over zijn. Dit zijn de segmenten, elk een losse letter. Deze segmenten worden in de volgende stappen gebruikt bij het verder oplossen van de captcha.

Stap 2: Neuraal netwerk trainen

Een volledig neuraal netwerk schrijven is erg ingewikkeld en overbodig aangezien anderen dit al vaak hebben gedaan. Voor dit project is de 'Fast Artificial Neural Network Library' gebruikt, ook wel '*FANN*' genoemd (Nissen, 2003 - 2017). Deze library heeft de backpropagation methode volledig geïmplementeerd. Om het netwerk uiteindelijk goed te kunnen gebruiken, moet het wel verteld worden wat de structuur moet zijn van het neurale netwerk. Nog een paar andere insignificante eigenschappen van het netwerk heb ik daarom aangepast. Dit is allemaal geïmplementeerd in het *train_ann.py* script. Zie Github.

a) Dataset verzamelen

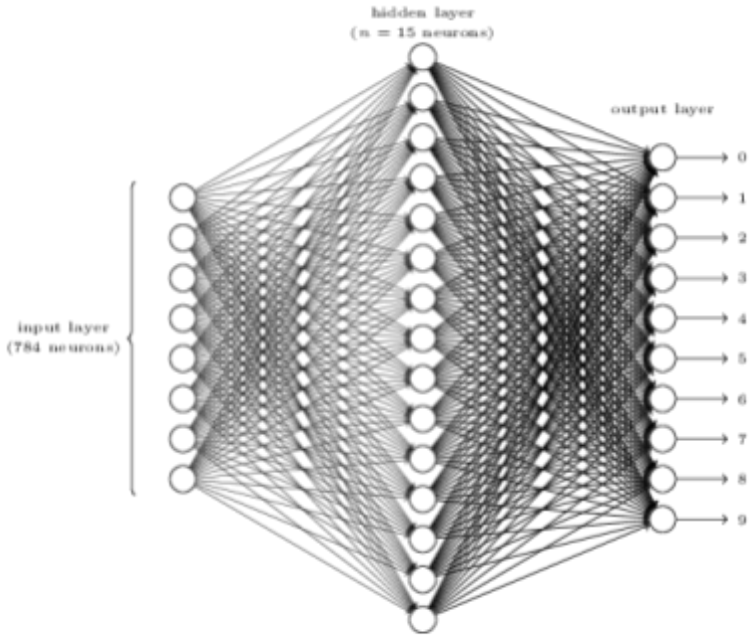
Bij het trainen van een neuraal netwerk moet het netwerk wel data hebben om op te trainen. Dit bleek nog een moeilijk probleem te zijn. Aangezien er een vrij groot netwerk wordt gebruikt van 900 input neuronen moet het netwerk kunnen trainen op extreem veel data. De vereiste is om minstens 10000 geclassificeerde letters te hebben. Dit is het minimum van ongeveer 200 voorbeeld letters per letter. Elke letter meer dan dit minimum zal het netwerk preciezer maken. Hoe meer hoe beter. Het eerste plan om data te verzamelen was om met de hand heel veel letters te classificeren en dan dit voor een paar dagen achter elkaar te doen en misschien nog andere mensen te vragen om daarmee te helpen. Na dit geprobeerd te hebben, bleek het dat dit absoluut niet zou werken en veel te veel tijd zou gaan kosten.

Er was dus een andere oplossing nodig. Het programma dat de captcha's genereert staat op de SecurImage website zodat iedereen het kan implementeren in zijn website. De eerste verwachting was dat het script niet zelf de captcha's genereert maar dat die ergens op een server van SecurImage zouden worden gegenereerd. Dit zou vele malen veiliger zijn en zou het onmogelijk maken voor mensen met slechte bedoelingen om automatisch een dataset te genereren. Deze verwachting was fout. Na het goed bekijken van de source code van SecurImage ben ik erachter gekomen dat de captcha's direct in het programma gemaakt worden. Dit betekent dat ik het programma van SecurImage kon downloaden, deze op mijn computer kon zetten, vervolgens op een persoonlijke server kon zetten en zelf captcha's kon gaan genereren. Niet alleen heb ik dit gedaan maar ik heb ook de volledige sourcecode van het programma doorgenomen en op een paar punten aangepast zodat ik zelf aan het programma kon meegeven welke code ik wilde en hoeveel letters ik wilde. Na dit op mijn server gezet te hebben, kon ik simpelweg vanuit mijn script vragen naar een captcha van de letter w en dan kreeg ik een captcha met alleen de letter w. Of ik kon vragen naar een captcha met 4 keer de letter A en dan kreeg ik een captcha met 4 keer de letter A. Dit was extreem handig om een dataset te genereren. Door vanuit mijn script heel vaak één enkele letter op te vragen, deze letter daarna uit de captcha op de segmenteer methode te halen en tenslotte alleen deze letter op te slaan, kon ik snel een dataset bouwen.

Door dit programma meerdere nachten aan te laten staan en alleen maar losse letters te laten verzamelen, heb ik een dataset kunnen creëren van 170000 letters, elke letter perfect geclassificeerd. De grote van deze dataset was 1 Gigabyte aan verschillende letters.

b) Trainen

Bij het trainen van het neurale netwerk wordt het netwerk met 900 input, 200 verborgen en 47 output perceptrons getraind op de dataset van 170000 letters. Dit trainen heeft 5 uur geduurd en aan het einde had het netwerk een error van nog maar 5 %. Dit betekent dat als het netwerk een nieuwe set met letters krijgt om op te lossen, hij er nog maar 1 op de 20 fout heeft. Een volledige captcha bestaat uit 6 letters daarom is het oplossingspercentage van het neurale netwerk 0.95^6 = +/- 0.74 => 74%. Een prachtige uitkomst die niet van tevoren te verwachten was.



Bron 6 Voorbeeld neuraal netwerk (Nielsen)

Stap 3: Neuraal netwerk gebruiken

Na het netwerk getraind te hebben, moet het toegepast worden. Bij deze stap komen de vorige twee stappen samen. Om captcha's op te kunnen lossen, moet een captcha verkregen worden. Dit kan van een website waar de captcha staat die opgelost moet worden maar dit kan ook vanuit een grote testgroep van captcha's. Deze testgroep is verzameld door het programma van SecurImage aan te passen op een manier die ervoor zorgt dat er een captcha gegenereerd wordt met de code die ik aan hem door geef. Op deze manier heb ik een testgroep weten te creëren van 10000 verschillende captcha's. Om deze captcha's op te lossen met mijn programma moet ik enkele stappen doorlopen. Als eerste moet de captcha geladen worden. Daarna moet de captcha gesegmenteerd worden. De segmenten die hieruit komen, gaan vervolgens door het neuraal netwerk heen die de uiteindelijke letters geeft. Dit is allemaal geïmplementeerd in het script *test_cap.py.* Zie Github. Na het testen met dit script, is het uiteindelijke slagingspercentage 53%.

Conclusie

Het script heeft nu een slagingspercentage van 53%. Dit betekent dat het script ongeveer de helft van alle captcha's die dit SecurImage programma genereert, kan oplossen. Dit feit alleen al heeft bewezen dat de captcha van SecurImage niet meer veilig is en dat deze captcha niet meer gebruikt moet worden. Zoals eerder vertelt, maakt het bij captcha's niet uit of iemand de captcha de eerste keer fout heeft, je zult altijd een nieuwe captcha krijgen om het opnieuw te proberen. Als wij dus bedenken dat dit niet uit maakt, is het doel dat het script met enig percentage slagingskans de captcha's kan oplossen met gemak gehaald. Het is zelfs veel beter geworden dan nodig is om dit te bewijzen.

Na het kritisch kijken naar het script wordt duidelijk dat dit slagingspercentage nog steeds omhoog kan. Een van de grootste problemen momenteel met het script is dat het niet tegen letters kan die tegen elkaar aan geplakt zijn. Dit gebeurt minstens 1 in de 15 keer. Dat betekent dat het slagingspercentagegetal met 6.67% kan toenemen als hiervoor een oplossing geïmplementeerd wordt. Naast deze simpele verbetering zijn er nog talloze kleine verbeteringen die het slagingspercentage kunnen verbeteren.

Als ik dit resultaat vergelijk met het programma van eisen, zie ik dat het script voldoet aan alle eisen en dat het sommige eisen zelfs overtreft.

Appendix A Belangrijke functies

```
def remove_groups_touching_white(self):
    groups = self.get_groups(20)
    for group in groups:
        totWhi = 0
        for (x, y) in group:
            for (s, t) in ((x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)):
                if (s > 0 and s < (self.size[0]) and t > 0 and t < (self.size[1])):
                    if (self.getpixel((s, t)) == 255):
                        totWhi += 1
        if (totWhi > 1):
            for (x, y) in group:
                self.putpixel((x, y), 255)
        else:
            for (x, y) in group:
                self.putpixel((x, y), 140)
```

```
def get_line_letter_background_points(self):
    llbpoints = []
    letpoints = self.collect_pixels([140])[0]

    for (s, t) in letpoints:
        whi = 0
        lin = 0
        for (v, w) in ((s + 1, t), (s - 1, t), (s, t + 1), (s, t - 1),
                       (s - 1, t - 1), (s + 1, t - 1), (s - 1, t + 1), (s + 1, t + 1)):
            if (v > 0 and v < (self.size[0] - 1) and (w > 0) and
                                                w < (self.size[1] - 1)):
                if (self.getpixel((v, w)) == 255):
                    whi += 1
                elif (self.getpixel((v, w)) == 112):
                    lin += 1
        if (whi > 0 and lin > 0):
            llbpoints.append((s, t))
    return llbpoints
```

```
def get_line_letter_points(self):
    llpoints = []
    linepoints = self.collect_pixels([112])[0]
    llbpoints = self.get_line_letter_background_points()

    for (s, t) in linepoints:
        whi = 0
        let = 0
        for (v, w) in (
                (s + 1, t), (s - 1, t), (s, t + 1), (s, t - 1),
                (s - 1, t - 1), (s + 1, t - 1), (s - 1, t + 1),
                (s + 1, t + 1)):

            if (v > 0 and v < (self.size[0] - 1) and (w > 0) and
                                                    w < (self.size[1] - 1)):
                if (self.getpixel((v, w)) == 255):
                    whi += 1
                elif (self.getpixel((v, w)) == 140):
                    let += 1

        if (whi == 0 and let > 1):
            bdc = 10000
            for (xc, yc) in llbpoints:
                dx = abs(s - xc)
                dy = abs(t - yc)
                dc = (dx * dx) + (dy * dy)
                if (dc < bdc): bdc = dc
            if (bdc > 2):
                llpoints.append((s, t))

    return llpoints
```

```
def grow_letter_into_line(self, distance, llpoints):
    for (rx, ry) in llpoints:
        lineCoords = []
        runt = 0
        if (self.getpixel((rx, ry)) == 112):
            lineCoords.append((rx, ry))
            temp = 0
            while (len(lineCoords) > 0 and runt < 10):
                runt += 1
                (x, y) = lineCoords[0]
                for (s, t) in ((x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)):
                    if (s >= 0 and s < (self.size[0]) and t >= 0 and
                                                        t < (self.size[1] - 1)):
                        if (self.getpixel((s, t)) == 112):
                            dx = abs(s - rx)
                            dy = abs(t - ry)
                            dc = (dx * dx) + (dy * dy)
                            if (dc < distance):
                                lineCoords.append((s, t))
```

```
                                    if runt > 4:
                                        temp = ((s, t), (x, y))
                    self.putpixel((x, y), 140)
                    lineCoords.pop(0)
```

Bronvermelding

Nissen, Steffen (2003 – 2017). Fast Artificial Neural Networks. Geraadpleegt op *http://leenissen.dk/fann/wp/*

Nissen, Steffen (2003). Implementation of a Fast Aritificial Neural Network Library (FANN)

Philips, Drew (2005 – 2015). *SecurImage PHP Captcha.* Geraadpleegt op *https://www.phpcaptcha.org*

Afbeeldingen

Bron 1:
Philips, Drew (2017). SecurImage Captcha. Gegenereerd op 10 Februari 2017, met *https://www.phpcaptcha.org*

Bron 2:
Berger, Christoph (2017). Menselijke neuron vs. Perceptron [Illustratie]. Geraadpleegd op 12 Februari 2017, van https://appliedgo.net/perceptron/

Bron 3:
Nielsen, M. A. (2015). Perceptron vs. Perceptron netwerk [Illustratie]. Geraadpleegd op 13 Februari 2017, van http://neuralnetworksanddeeplearning.com/chap1.html

Bron 4:
Philips, Drew (2017). Captcha voor en na ruis-verwijdering. Gegenereerd op 13 Februari 2017, met *https://www.phpcaptcha.org*

Bron 5:
Philips, Drew (2017). Captcha voor en na lijn-verwijdering. Gegenereerd op 10 Februari 2017, met *https://www.phpcaptcha.org*

Bron 6:
Nielsen, M. A. (2015). Voorbeeld neuraal netwerk [Illustratie]. Geraadpleegd op 13 Februari 2017, van http://neuralnetworksanddeeplearning.com/chap1.html