UNIVERSITEIT GENT

Multidisciplinair Ingenieursproject - Informatica

Een neuraal netwerk trainen en een getekend cijfer laten herkennen

door

Groep 6

Willem Dendauw
Niels Hauttekeete
Bram Van de Walle
Egon Vanhoenacker
Leonie Van Renterghem

Begeleiders: prof. Veerle Ongenae en mevr. Leen Brouns

Universiteit Gent
Faculteit Ingenieurswetenschappen en architectuur
Opleiding Industriële wetenschappen - Informatica
Academiejaar 2018-2019

Inhoudstafel

Doelstelling

De opdracht van dit project is om in teams van vier of vijf studenten een programma in Python te ontwikkelen dat onder andere een niet-triviaal algoritme implementeert. Hiervoor hebben wij ons gebaseerd op een boek genaamd 'Neural Networks and Deep Learning[1]', geschreven door Michael Nielsen. Dit is een inleidend boek over neurale netwerken. Al in het eerste hoofdstuk wordt een neuraal netwerk gemaakt en getraind. Het doel van dit netwerk is om handgeschreven cijfers te herkennen. Op dat idee zijn wij verder gegaan. Het idee is geworden om een applicatie te schrijven waarin de gebruiker met de muis een cijfer kan tekenen en vervolgens een reeds getraind netwerk kan laten 'raden' welk cijfer de gebruiker heeft getekend. Neurale netwerken zijn zeer interessant om over te leren. Het is echter niet altijd even gemakkelijk om ergens te beginnen met te leren over dit onderwerp. Daarom wilden we naast het tekenen en laten herkennen van een cijfer ook de gebruiker de kans te geven om te leren over dit onderwerp.

Prestatie

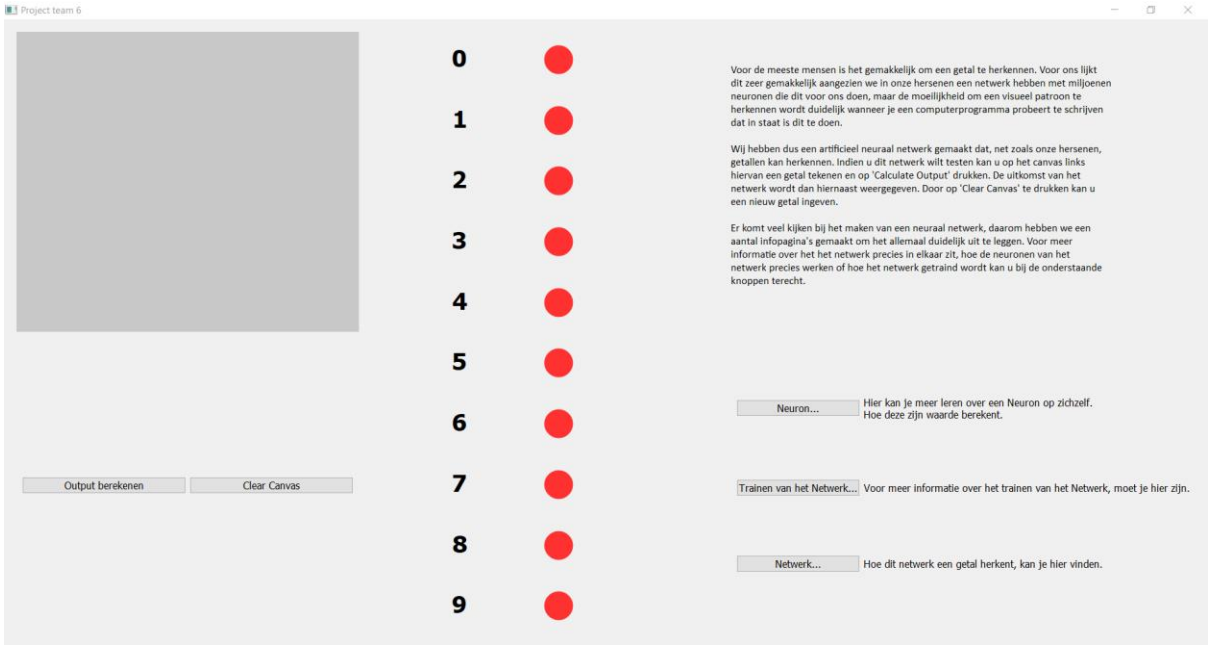
De applicatie is opgebouwd uit hoofdzakelijk twee onderdelen. Enerzijds is er het algoritme, anderzijds is er de GUI[2]. Voor elk onderdeel is een package voorzien. Binnen het package 'Algoritme' zit de module 'Network' wat alle code bevat om een neuraal netwerk aan te maken en te trainen. Het package 'GUI' is onderverdeeld in subpackages en een module 'MainGUI'. Tot slot is er ook nog een Main module die een object van MainGUI maakt. Automatisch wordt het venster van MainGUI geopend. Dit is de start van het programma.

Graphical User Interface

Wanneer de gebruiker de applicatie opent, wordt de hoofdpagina aan hem/haar getoond. Een screenshot van de hoofdpagina is te zien in Figuur 1.

In het oog springen de rode bollen en de bijhorende cijfers. Wanneer een getekend cijfer als input wordt meegegeven aan een getraind netwerk en aan dat netwerk gevraagd wordt welk

[1] http://neuralnetworksanddeeplearning.com/index.html
[2] Graphical User Interface

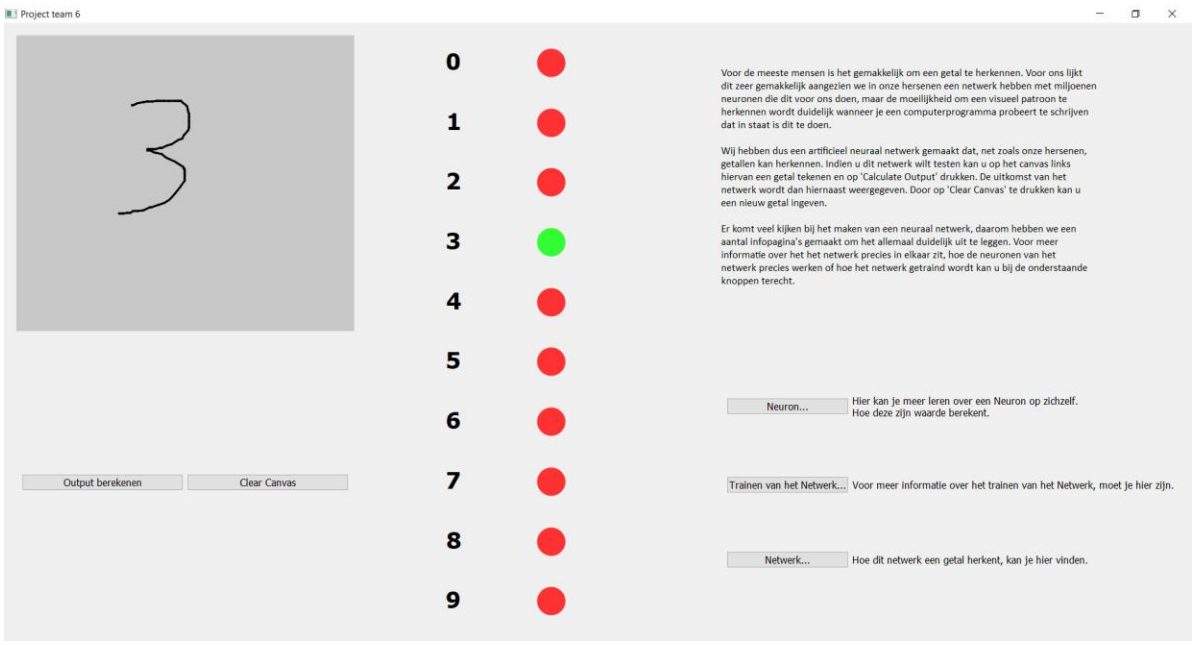


Figuur 1: Hoofdpagina

cijfer werd getekend, dan zal dat netwerk voor elk cijfer een waarde tussen 0 en 1 geven. Als het neuraal netwerk voor een bepaald cijfer een waarde dicht bij 0 geeft, dan wil dat zeggen dat het netwerk denkt dat de gebruiker dat cijfer helemaal niet heeft getekend. In dat geval zal de bol bij dat cijfer rood zijn. Wanneer het netwerk echter denkt dat de gebruiker dat cijfer wel heeft getekend, zal het voor dat cijfer een waarde dicht bij 1 geven en zal de bol groen kleuren. Het netwerk is niet altijd zeker van z'n stuk en soms zal het een waarde tussen 0 en 1 geven. Om ook dit visueel te kunnen voorstellen hebben we ervoor gekozen om met lineaire gradiënten te werken. Wanneer een cijfer een waarde tussen 0 en 1 krijgt, zal de kleur tussen rood en groen liggen.

De gebruiker van de applicatie moet dus een cijfer kunnen tekenen. Hiervoor hebben is er een klasse Painter geschreven. Een object van deze klasse is te zien in de hoofdpagina, namelijk het grijze vierkant linksboven. Hier krijgt de gebruiker de kans om een cijfer te tekenen (zie Figuur 2). Nadat het cijfer getekend is kan het neuraal netwerk, dat in de hoofdpagina al reeds getraind is, bepalen welk cijfer de gebruiker heeft getekend. Het neuraal netwerk krijgt het getekende cijfer als input op het moment dat de gebruiker op de knop 'Output berekenen'

heeft gedrukt. De output van het netwerk wordt visueel voorgesteld op het scherm zoals te zien in Figuur 2. Het neuraal netwerk herkent het getekende cijfer en dit wordt getoond aan



Figuur 2: Hoofdpagina, gebruiker heeft een cijfer getekend en laten berekenen door het neuraal netwerk

de gebruiker door de bol bij cijfer '3' groen te kleuren.

Wanneer de gebruiker een nieuw cijfer wil tekenen moet hij/zij op de knop 'Clear Canvas' drukken. Vervolgens zal het Painter-object zichzelf herstellen naar de oorspronkelijke toestand. Wenst de gebruiker ook nog de gekleurde bollen te herstellen naar de oorspronkelijke toestand, dan moet hij/zij nog eens op de knop "Output berekenen" drukken, de gekleurde bollen zullen allemaal terug rood zijn.

Naast het tekenen en bepalen van cijfers, is er ook een educatieve doelstelling gelinkt aan dit project. Daarvoor moet de rechterkant van de hoofdpagina bekeken worden. De tekst die rechtsboven staat is een inleidende uitleg over het project en hoe de gebruiker zijn weg kan vinden doorheen de applicatie. Rechtsonder bevinden zich drie knoppen: 'Neuron…', 'Trainen van het Netwerk…' en 'Netwerk…'. Wanneer de gebruiker op een van deze knoppen drukt, zal er steeds een venster geopend worden. Elk venster is een infopagina, respectievelijk

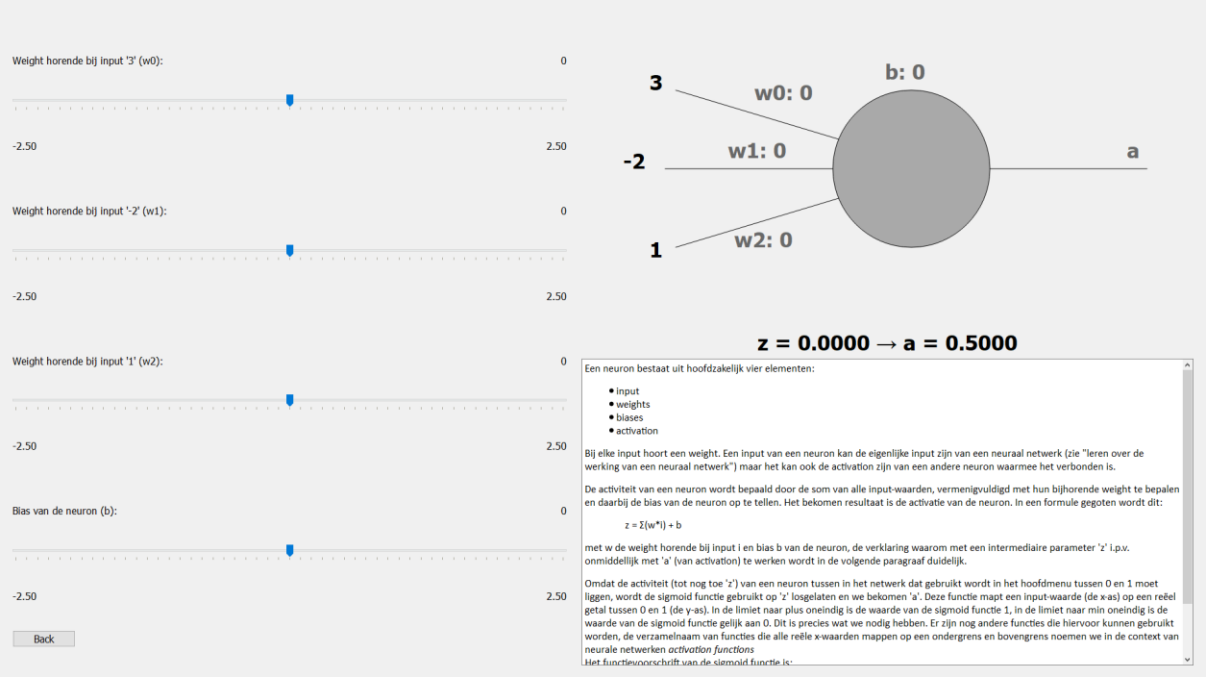
over wat een neuron is, hoe een neuraal netwerk getraind wordt en wat een neuraal netwerk eigenlijk is.

In Bijlage A staat een sequentiediagram van wat er allemaal gebeurt wanneer een gebruiker een cijfer tekent en de output verwacht. In Bijlage B staat een sequentiediagram van een gebruiker die de applicatie sequentieel doorneemt.

In de volgende drie secties gaan we wat dieper in op de drie infopagina's.

Infopagina: Neuron

Zoals eerder vermeld is de applicatie opgesplitst in verschillende packages. In de package 'GUI' zit een subpackage 'Neuron'. Hier bevindt zich alle code van de infopagina over een neuron. Als de gebruiker de infopagina opent, wordt het volgende getoond:



Figuur 3: Infopagina: het neuron

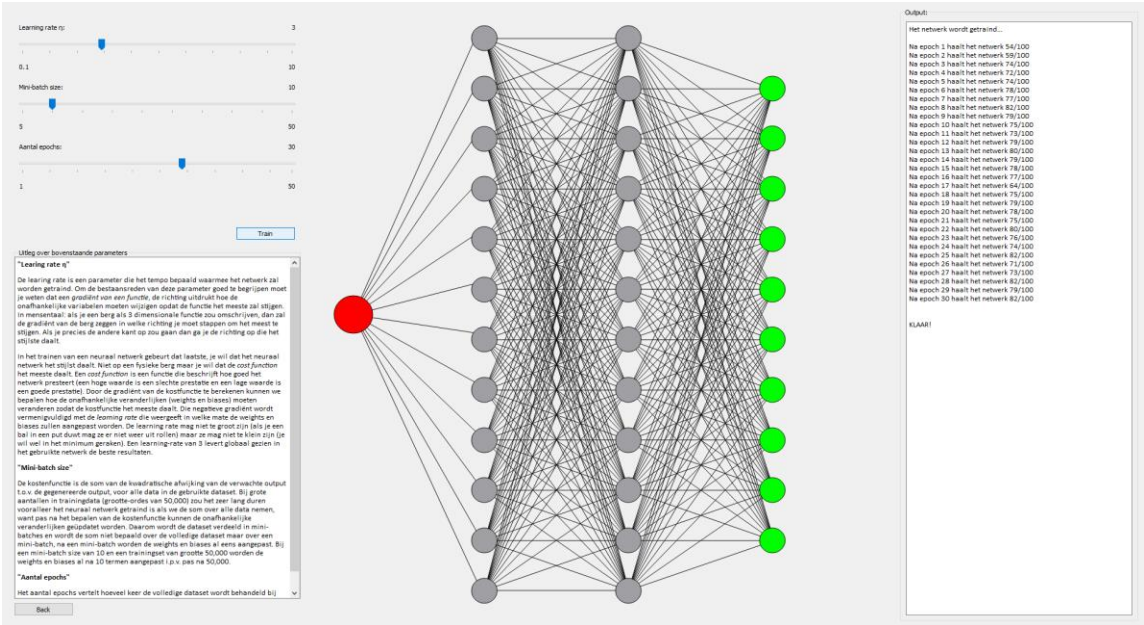
De package Neuron bevat twee klassen, NeuronGUI en NeuronVisualization. Een instantie van NeuronGUI wordt door de MainGUI aangemaakt wanneer de gebruiker op de knop "Neuron…" drukt en de NeuronGUI zal openen. Bij creatie van een NeuronGUI wordt ook een object van NeuronVisualization gemaakt, dat laatste maakt het mogelijk om de een neuron

visueel voor te stellen. Het verloop van de constructie van deze objecten is te volgen in het sequentiediagram in Bijlage C.

Een neuron bestaat uit vier elementen, inputs, weights, een bias en een activation. Bij elke input hoort een eigen weight. De input van een neuron kan een activatie van een ander neuron zijn waarmee het verbonden is, of het kan de input zijn van het neuraal netwerk, hier zou dat dan dus de grijswaarde zijn van een pixel van een cijfer.

Op de NeuronGUI zijn vier sliders aanwezig, zij bepalen de waarden van de drie weights en de bias. De sliders zijn links te zien op Figuur 3. Wanneer de sliders wijzigen zal ook de tekst in de NeuronVisualization wijzigen. Dit wordt mogelijk gemaakt door de methode 'changeParameters()' van NeuronVisualization, waarin de nieuwe waarden van de vier parameters worden meegegeven. Het gebruik van deze infopagina wordt duidelijk gemaakt door de loop in het sequentiediagram in Bijlage C.

Infopagina: Trainen van het netwerk



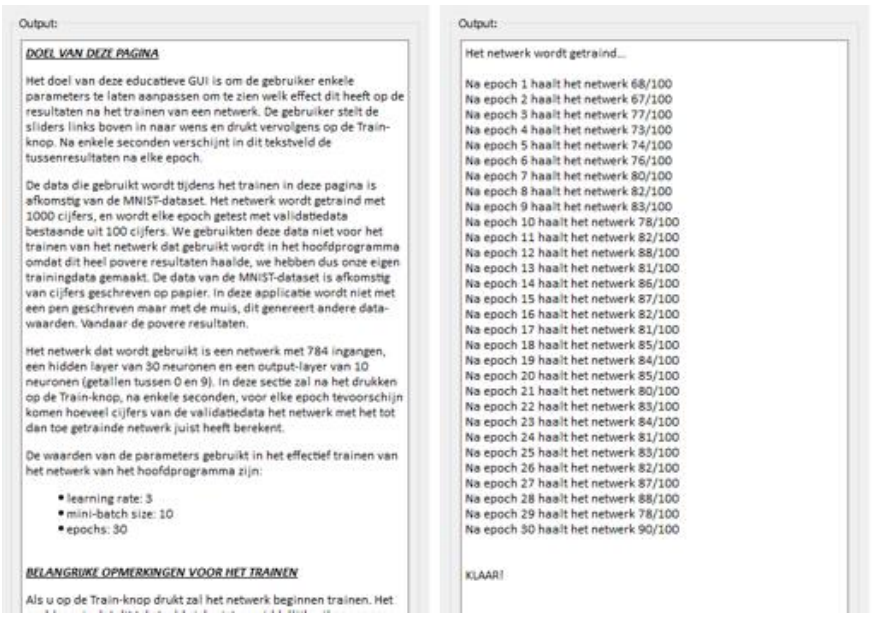
Figuur 4: De TrainingGUI

De TrainingGUI (zie Figuur 4) is een educatieve GUI met als doel de gebruiker te tonen welk effect bepaalde parameters hebben op het trainen van een neuraal netwerk.

De grafische interface bestaat uit 3 delen: aan de linker kant een paneel met 3 QSliders, deze kunnen gebruikt worden om de parameters (learning rate, batch grootte en aantal epochs) in te stellen. Daaronder staat wat meer uitleg over de parameters.
In het midden staat een visuele representatie van een neuraal netwerk met 1 (rode) input node en 10 (groene) output nodes. Dit dient om de gebruiker een beter beeld te geven over wat er achter de schermen gebeurt.

Aan de rechterkant staat een tekstvak waar initieel informatie staat over de GUI samen met een aantal belangrijke opmerkingen over het trainen van het netwerk (zie Figuur 5a). Nadat de gebruiker de sliders heeft ingesteld en het netwerk traint zal dit tekstvak tonen hoe het netwerk getraind werd (zie Figuur 5b). Zo kan de gebruiker zien of de configuratie van de parameters gezorgd heeft voor een optimale training.

In de volgende paragrafen zullen de drie bovenstaande delen van de GUI verder besproken worden en zal de werking van het trainen van het netwerk in meer detail worden uitgelegd.



Figuur 5: (a) Initiële toestand tekstvak (b) Tekstvak na trainen

Er zijn drie parameters die het trainen van het netwerk beïnvloeden de learning rate, de batch grootte en het aantal epochs. Deze drie parameters worden in onderstaande paragrafen beschreven.

“Learning rate η”

De learing rate is een parameter die het tempo bepaalt waarmee het netwerk zal worden getraind. Om de bestaansreden van deze parameter goed te begrijpen moet je weten dat een *gradiënt van een functie*, de richting uitdrukt hoe de onafhankelijke variabelen moeten wijzigen opdat de functie het meeste zal stijgen.

In mensentaal: als je een berg als 3-dimensionale functie zou omschrijven, dan zal de gradiënt van de berg zeggen in welke richting je moet stappen om het meest te stijgen. Als je precies de andere kant op zou gaan dan ga je de richting op die het steilste daalt.

In het trainen van een neuraal netwerk gebeurt dat laatste, je wil dat het neuraal netwerk het steilst daalt. Niet op een fysieke berg maar je wil dat de *cost function* het meeste daalt. Een *cost function* is een functie die beschrijft hoe goed het netwerk presteert (een hoge waarde is een slechte prestatie en een lage waarde is een goede prestatie). Door de gradiënt van de kostfunctie te berekenen kunnen we bepalen hoe de onafhankelijke veranderlijken (weights en biases) moeten veranderen zodat de kostfunctie het meeste daalt. Die negatieve gradiënt wordt vermenigvuldigd met de *learning rate* die weergeeft in welke mate de weights en biases zullen aangepast worden. De learning rate mag niet te groot zijn (als je een bal in een put duwt mag ze er niet weer uit rollen) maar ze mag niet te klein zijn (je wil wel in het minimum geraken). Een learning rate van 3 levert globaal gezien in het gebruikte netwerk de beste resultaten.

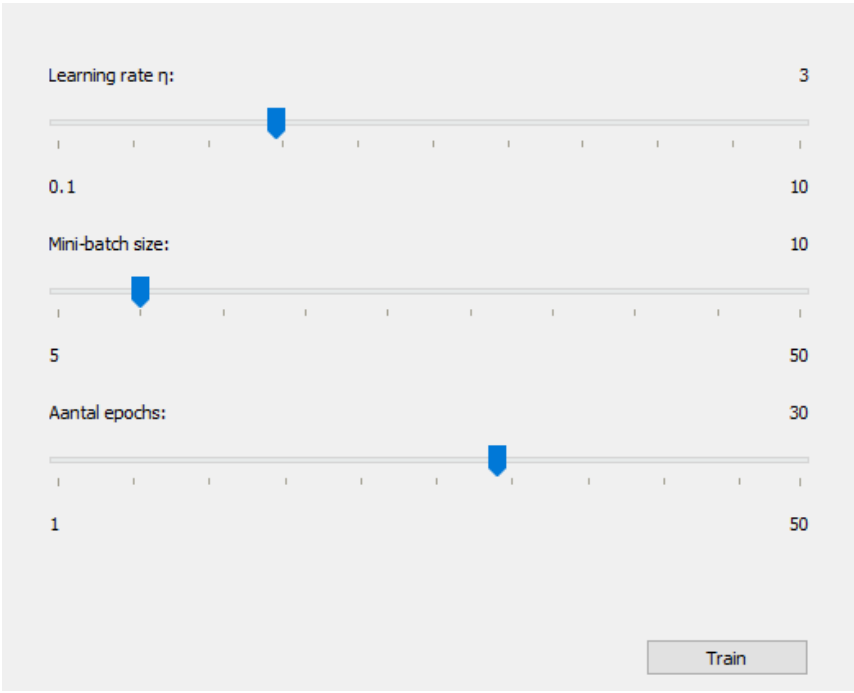
"Mini-batch size"

De kostenfunctie is de som van de kwadratische afwijking van de verwachte output t.o.v. de gegenereerde output, voor alle data in de gebruikte dataset. Bij grote aantallen in trainingsdata (grootteordes van 50,000) zou het zeer lang duren vooraleer het neuraal netwerk getraind is als we de som over alle data nemen, want pas na het bepalen van de kostenfunctie kunnen de onafhankelijke veranderlijken geüpdatet worden. Daarom wordt de dataset verdeeld in mini-batches en wordt de som niet bepaald over de volledige dataset maar over een mini-batch, na een mini-batch worden de weights en biases al eens aangepast. Bij een mini-batch size van 10 en een trainingsset van grootte 50,000 worden de weights en biases al na 10 termen aangepast i.p.v. pas na 50,000.

"Aantal epochs"

Het aantal epochs vertelt hoeveel keer de volledige dataset wordt behandeld bij het trainen. Bij elke epoch wordt de trainingsdata opgesplitst in mini-batches. Na de laatste epoch zit het trainen er op.

Deze parameters kunnen aan de hand van de sliders te zien op Figuur 6 aangepast worden. Zo kan de gebruiker op zoek gaan naar de beste configuratie om het netwerk te trainen.



Figuur 6: Sliders en 'Train'-knop

Wanneer de gebruiker dan op de knop 'Train' (Zie Figuur 6) drukt zal er een nieuw object van de klasse TrainingThread aangemaakt worden. In die klasse wordt een object van Network (Algoritme.Network) aangemaakt dat geïnitialiseerd wordt met random waarden. Vervolgens wordt dit netwerk getraind met parameters die ingesteld zijn door de gebruiker (zie Bijlage E).

Ondertussen zal tijdens iedere epoch nagekeken worden hoe accuraat het netwerk op dat moment is. Dit wordt dan weggeschreven naar het outputvenster aan de rechterkant van de GUI (Zie rechts in Figuur 4).

Om het netwerk te visualiseren is er een hulpklasse NNVisualisatie geschreven. Deze klas is afgeleid van QWidget en kan dus direct gebruikt worden binnen de applicatie. Bij constructie van een object van deze klasse kan de programmeur meegeven hoeveel inner layers er moeten zijn en hoeveel nodes deze layers moeten hebben. Er is altijd 1 input node en 10 output nodes voor deze applicatie maar de inner layers kunnen dus aangepast worden naar behoeven.

Het netwerk wordt getekend door gebruik te maken van de klasse QPainter, deze klasse heeft de mogelijkheden om op een QWidget te tekenen.

Wanneer de TrainingGUI wordt aangemaakt wordt er een object van de klasse NNVisualisatie aangemaakt (zie Bijlage E).

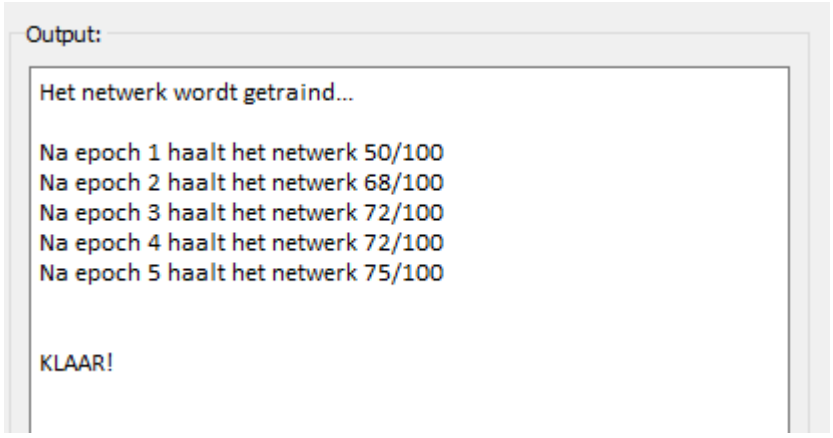
De data die gebruikt wordt tijdens het trainen in deze pagina is afkomstig van de MNIST-dataset. Het netwerk wordt getraind met 1000 cijfers, en wordt elke epoch getest met validatiedata bestaande uit 100 cijfers. We gebruikten deze data niet voor het trainen van het netwerk dat gebruikt wordt in het hoofdprogramma omdat dit heel povere resultaten haalde, we hebben dus onze eigen trainingsdata gemaakt (Zie Aanmaken trainingsdata en testdata). De data van de MNIST-dataset is afkomstig van cijfers geschreven op papier. In deze applicatie wordt niet met een pen geschreven maar met de muis, genereert andere data-waarden. Vandaar de povere resultaten.

Het netwerk dat wordt gebruikt is een netwerk met 784 ingangen, een hidden layer van 30 neuronen en een output-layer van 10 neuronen (getallen tussen 0 en 9). In deze sectie zal na het drukken op de Train-knop, na enkele seconden, voor elke epoch tevoorschijn komen hoeveel cijfers van de validatiedata het netwerk met het tot dan toe getrainde netwerk juist heeft berekent.

De waarden van de parameters gebruikt in het effectief trainen van het netwerk van het hoofdprogramma zijn:

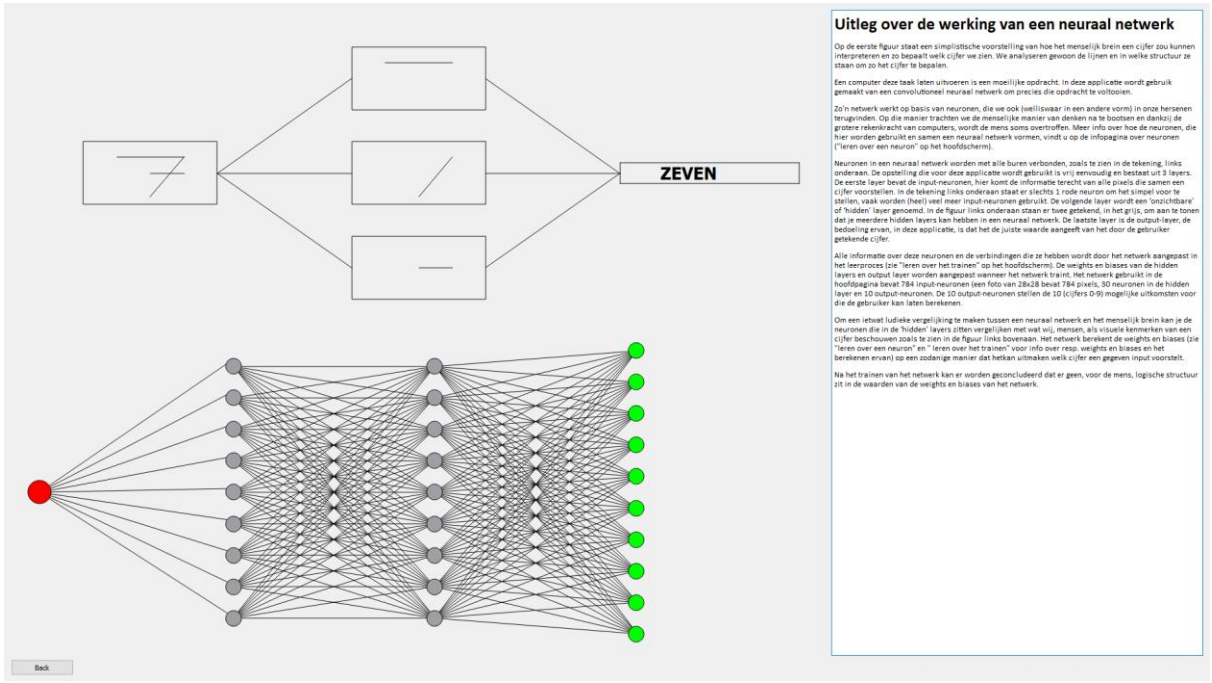
- learning rate: 3
- mini-batch size: 10
- epochs: 30

Na het trainen van een netwerk zal de output weggeschreven worden naar een outputvenster. Deze output zal lijken op wat afgebeeld staat in Figuur 7, afhankelijk van de gekozen parameters.



Figuur 7: De output van het trainen van een netwerk

Infopagina: Netwerk



Figuur 8: De NetworkGUI

De NetworkGUI (Figuur 8) is een educatieve GUI met als doel de gebruiker een vergelijking te tonen tussen het denkproces van de mens (hier simplistisch voorgesteld) en dat van een neuraal netwerk. Daarnaast wordt er besproken hoe een neuraal netwerk is opgebouwd en hoe het werkt. Het sequentiediagram van deze GUI vindt u in Bijlage D.

De GUI is opgebouwd uit drie delen, 2 QWidgets die visueel een voorbeeld geven en een QTextEdit met schriftelijke uitleg. De bovenste QWidget is geïmporteerd uit de klasse HumanCanvas die speciaal voor deze GUI werd aangemaakt. Deze klasse tekent alle hokjes en lijntjes met een QPainter en zet ze in een QWidget. De onderste QWidget is geïmporteerd van de bestaande klasse NNVisualisatie, deze klasse werd oorspronkelijk gemaakt voor de TrainingGUI en hiermee kan je kiezen hoeveel lagen je in je netwerk wil en deze wordt teruggeven als een QWidget.

Op de eerste figuur staat een simplistische voorstelling van hoe het menselijk brein een cijfer zou kunnen interpreteren en zo bepaalt welk cijfer we zien. We analyseren gewoon de lijnen en in welke structuur ze staan om zo het cijfer te bepalen.

Een computer deze taak laten uitvoeren is een moeilijke opdracht. In deze applicatie wordt gebruik gemaakt van een convolutioneel neuraal netwerk om precies die opdracht te voltooien.

Zo'n netwerk werkt op basis van neuronen, die we ook (weliswaar in een andere vorm) in onze hersenen terugvinden. Op die manier trachten we de menselijke manier van denken na te bootsen en dankzij de grotere rekenkracht van computers, wordt de mens soms overtroffen.

Neuronen in een neuraal netwerk worden met alle buren verbonden, zoals te zien is in figuur 8, links onderaan. De opstelling die voor deze applicatie wordt gebruikt is vrij eenvoudig en bestaat uit 3 layers. De eerste layer bevat de input-neuronen, hier komt de informatie terecht van alle pixels die samen een cijfer voorstellen. In de tekening links onderaan staat er slechts 1 rode neuron om het simpel voor te stellen, vaak worden (heel) veel meer input-neuronen gebruikt. De volgende layer wordt een 'onzichtbare' of 'hidden' layer genoemd. In de figuur links onderaan staan er twee getekend, in het grijs, om aan te tonen dat je meerdere hidden layers kan hebben in een neuraal netwerk. De laatste layer is de output-layer, de bedoeling ervan, in deze applicatie, is dat het de juiste waarde aangeeft van het door de gebruiker getekende cijfer.

Alle informatie over deze neuronen en de verbindingen die ze hebben wordt door het netwerk aangepast in het leerproces. De weights en biases van de hidden layers en output layer worden aangepast wanneer het netwerk traint. Het netwerk gebruikt in de hoofdpagina bevat 784 input-neuronen (een foto van 28x28 bevat 784 pixels, 30 neuronen in de hidden layer en 10 output-neuronen). De 10 output-neuronen stellen de 10 (cijfers 0-9) mogelijke uitkomsten voor die de gebruiker kan laten berekenen.

Om een ietwat ludieke vergelijking te maken tussen een neuraal netwerk en het menselijk brein kan je de neuronen die in de 'hidden' layers zitten vergelijken met wat wij, mensen, als visuele kenmerken van een cijfer beschouwen zoals te zien in figuur 8 links bovenaan. Het netwerk

berekent de weights en biases op een zodanige manier dat het kan uitmaken welk cijfer een gegeven input voorstelt.

Na het trainen van het netwerk kan er worden geconcludeerd dat er geen, voor de mens, logische structuur zit in de waarden van de weights en biases van het netwerk.

Algoritme

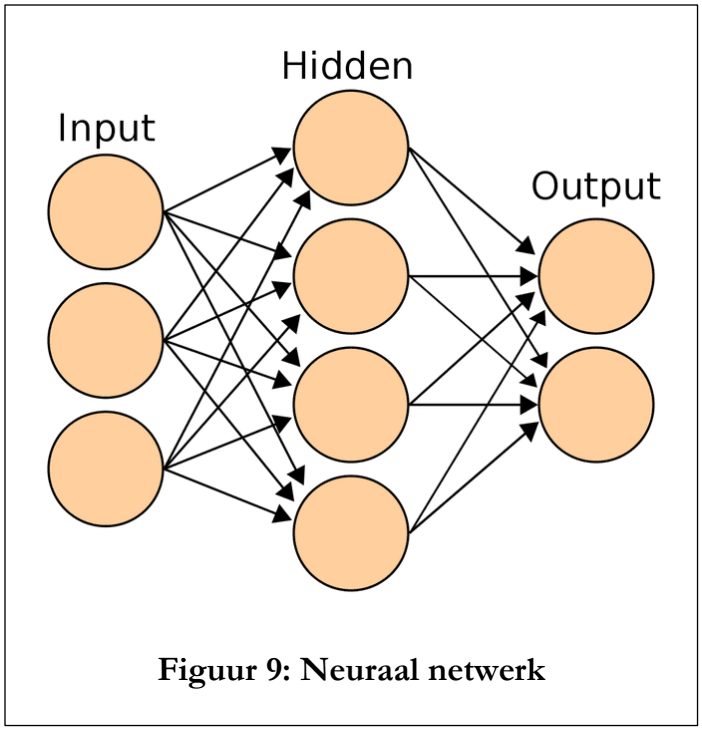
De klasse Network vervult de functie van het neuraal netwerk. Deze klasse bevat ook de algoritmen die het netwerk trainen, zodat het netwerk de getallen kan herkennen. De twee algoritmen die gebruikt worden om het netwerk te trainen zijn gradient descent en backpropagation.

Het netwerk dat in deze applicatie gebruikt wordt bestaat uit 3 lagen met respectievelijk 784, 30 en 10 neuronen. Een vereenvoudigde versie van het netwerk is terug te vinden in Figuur 9. De 784 neuronen van de eerste laag stellen het getekende cijfer voor. Dit is een 28x28 matrix waarvan de waarde aangeeft hoe donker die pixel is, m.a.w. de grijswaarde van de pixel. Voordat het cijfer kan worden meegegeven als input aan het neuraal netwerk moet elke rij achter elkaar gezet worden. De volgende layer, de hidden layer, bestaat uit 30 neuronen, de keuze hiervoor is omdat volgens Michael Nielsen[3] dit het beste netwerk is om de taak te vervullen. Tot slot is er nog een output-layer, bestaande uit 10 neuronen (omdat er 10 cijfers zijn), die de output van het neuraal netwerk is. Wanneer het netwerk een input heeft behandeld zal elke output-neuron weergeven in welke mate het neuraal netwerk het cijfer horende bij die neuron in de meegegeven input herkent. Indien het netwerk zeker is dat hij het cijfer in de input herkent, zal de activatie van de output-neuron dicht bij de 1 liggen. Herkent het netwerk

[3] Michael Nielsen is de author van het boek "Neural Networks and Deep Leanring"

het cijfer niet bij die output-neuron, dan zal de activatie horende bij die neuron dicht bij de 0 zijn.

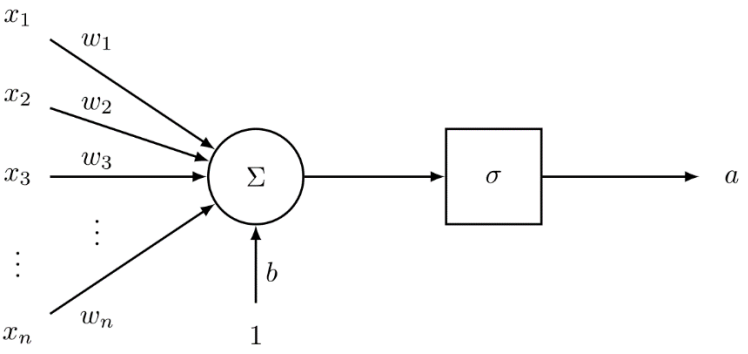
In ons netwerk maken we gebruik van sigmoïd neuronen (Zie Figuur 10). De output van deze neuronen wordt berekend aan de hand van deze formule.



Figuur 9: Neuraal netwerk

$$\frac{1}{1+\exp\left(-\sum_j w_j x_j - b\right)}$$

Hierbij is b de bias van het neuron, w het gewicht en x de output van de vorige laag.



Figuur 10: Sigmoid neuron

Om het netwerk te trainen aan de hand van gradient descent wordt er een kostfunctie berekend.

$$C = -\frac{1}{n} * \sum_j \left[y_j \ln a_j^L + (1 - y_j) * \ln(1 - a_j^L) \right] + \frac{\lambda}{2n} * \sum_w w^2$$

Hierbij is n het aantal training inputs, y de verwachte output, a de de effectieve output, λ de regularisatie factor en w is het gewicht.

Indien het netwerk ver van de verwachte output zit zal de kost groot zijn. De gewichten en biases van het netwerk moeten dus aangepast worden, zodat de kost zo klein mogelijk wordt. Om dit te realiseren wordt het algoritme genaamd 'gradient descent' gebruikt.

Gradient descent verdeelt de trainingsdata in kleine stukken, minibatches genaamd. De reden voor de verdeling van de trainingsdata in minibatches is tijdswinst. Het is namelijk zo dat eerst een som moet berekend worden over de verzameling data waarvan je de kost wil bepalen, alvorens de onafhankelijke veranderlijken (weights en biases van het netwerk) kunnen worden aangepast. Dat zou willen zeggen dat we over de volledige dataset (vaak in grootteorde van 100,000) de som moeten bepalen. Als we de trainingsdata verdelen in minibatches en na elke minibatch de weights en biases al updaten, dan wordt het netwerk binnen een epoch[4] al stap voor stap getraind, in plaats van slechts een keer op het einde van een epoch. Indien we

[4] een epoch is een cyclus doorheen de volledige dataset, zie de sectie over de infopagina over het trainen van een neuraal netwerk

gradient descent gebruiken op minibatches moeten we eigenlijk spreken van een stochastic gradient descent.

Stochastic gradient descent berekent de gradient van de kostfunctie voor elke minibatch. Daarbij wordt backpropagation, het tweede gebruikte algoritme, gebruikt om op een efficiënte manier de partiële afgeleiden $\partial C/\partial w$ en $\partial C/\partial b$ van de kostfunctie berekend. Deze gradiënt zal in kleine stappen in de negatieve richting gevolgd worden, tot het (lokaal) minimum bereikt wordt. Wanneer het minimum bereikt wordt wil dat zeggen dat het verschil tussen de verwachte output en de effectieve output minimaal is, dat is dus precies wat we nodig hebben. Dit proces wordt over een aantal epochs herhaald, zodat de gewichten en biases van het netwerk zeker goed afgesteld zijn.

Klassendiagrammen

De GUI package bestaat uit verschillende subpackages. De Neuron package, de Network package de Trainen package, de Output package en de Painter package zijn hier deel van. Deze hebben allemaal een essentiële rol in het maken van de GUI zelf. Omdat er zoveel onderverdelingen waren, staat de GUI package verdeeld in drie delen. Bijlage H is het eerste deel van de package, deze bevat de Neuron package en de Network package. Het tweede deel is te zien in bijlage I en bevat de Trainen package. Het derde deel bevat de Output package en de Painter package, dit bevindt zicht in bijlage J.

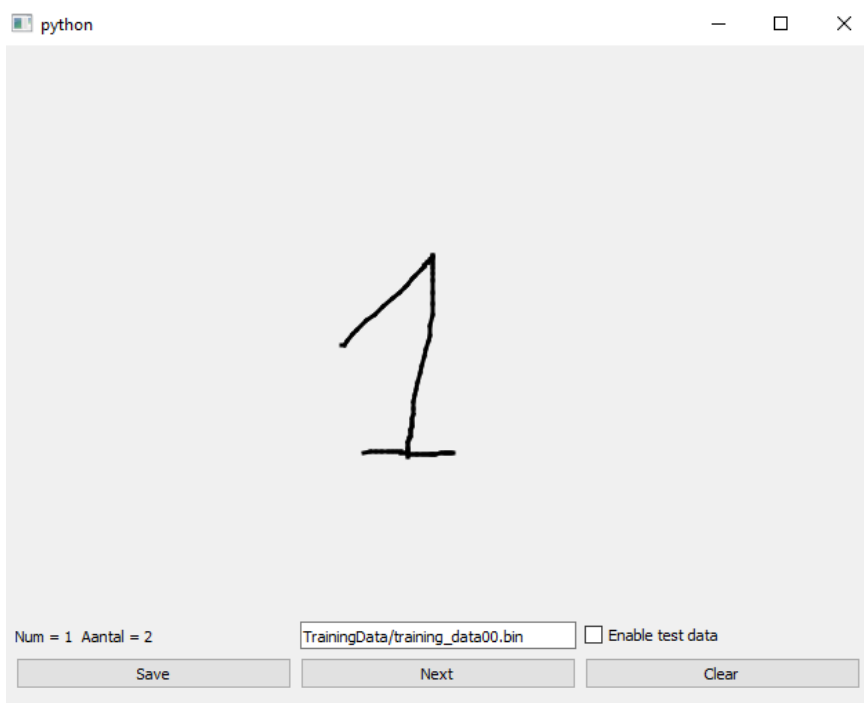
Naast de GUI package is er ook een Algoritme package, te zien in bijlage K. Deze is verantwoordelijk voor het maken van ons eigen netwerk.

De andere analysedocumenten zijn hiervoor al eens aangekaart, of naar verwezen.

Aanmaken van trainingsdata en testdata

Om het netwerk te kunnen trainen en testen is er data nodig omdat zoals eerder vermeld is de MNIST-data onbruikbaar. Daarom is er een hulp applicatie geschreven zodat we zelf data kunnen aanmaken.

Met een keuzehokje kan de gebruiker aangeven of hij/zij ofwel trainingsdata of testdata wil maken. Het is belangrijk van dit juist aan te vinken want trainings- en testdata worden op verschillende manieren opgeslagen. Testdata wordt opgeslagen in de map bestemd voor testdata en de trainingsdata in de map voor trainingsdata.
In het tekstvak moet de gebruiker de bestandsnaam opgeven waar hij/zij de data wil opslaan. Na iedere twee cijfers die de gebruiker tekent wordt er automatisch een back-up gemaakt zodat moest de applicatie crashen de gebruiker niet alles kwijt is.



Figuur 11: Hulpapplicatie voor het aanmaken van data

In Figuur 11 staat de applicatie, deze bestaat uit een tekenveld van de klasse Painter, hierop moet de gebruiker het cijfer tekenen dat aangegeven staat bij 'Num = '. Hiernaast kan de gebruiker het aantal cijfers zien dat hij/zij getekend heeft. Na het tekenen van het cijfer moet de gebruiker op de knop 'Next' drukken om het cijfer op te slaan de voorbestemde instantievariabele. Als de gebruiker een foutief cijfer tekent of vindt dat het cijfer te slordig getekend is dan kan hij/zij het tekenveld leegmaken door op de knop 'Clear' te drukken.

Als de gebruiker klaar is met data maken, drukt hij/zij op de knop 'Save'. Hierna zal de applicatie nog een pop-up tonen die om bevestiging vraagt of er zeker gestopt wil worden.

Het gebruik van deze applicatie is ook te volgen in Bijlage F.

Deze module maakt geen deel uit van het eindproduct van het project maar is essentieel om bruikbare data te bekomen om het netwerk van het eindproduct te doen werken. De weights en biases van het netwerk getraind met de trainingsdata zullen opgeslagen worden en gebruikt worden bij constructie van het Netwerk-object in de MainGUI.

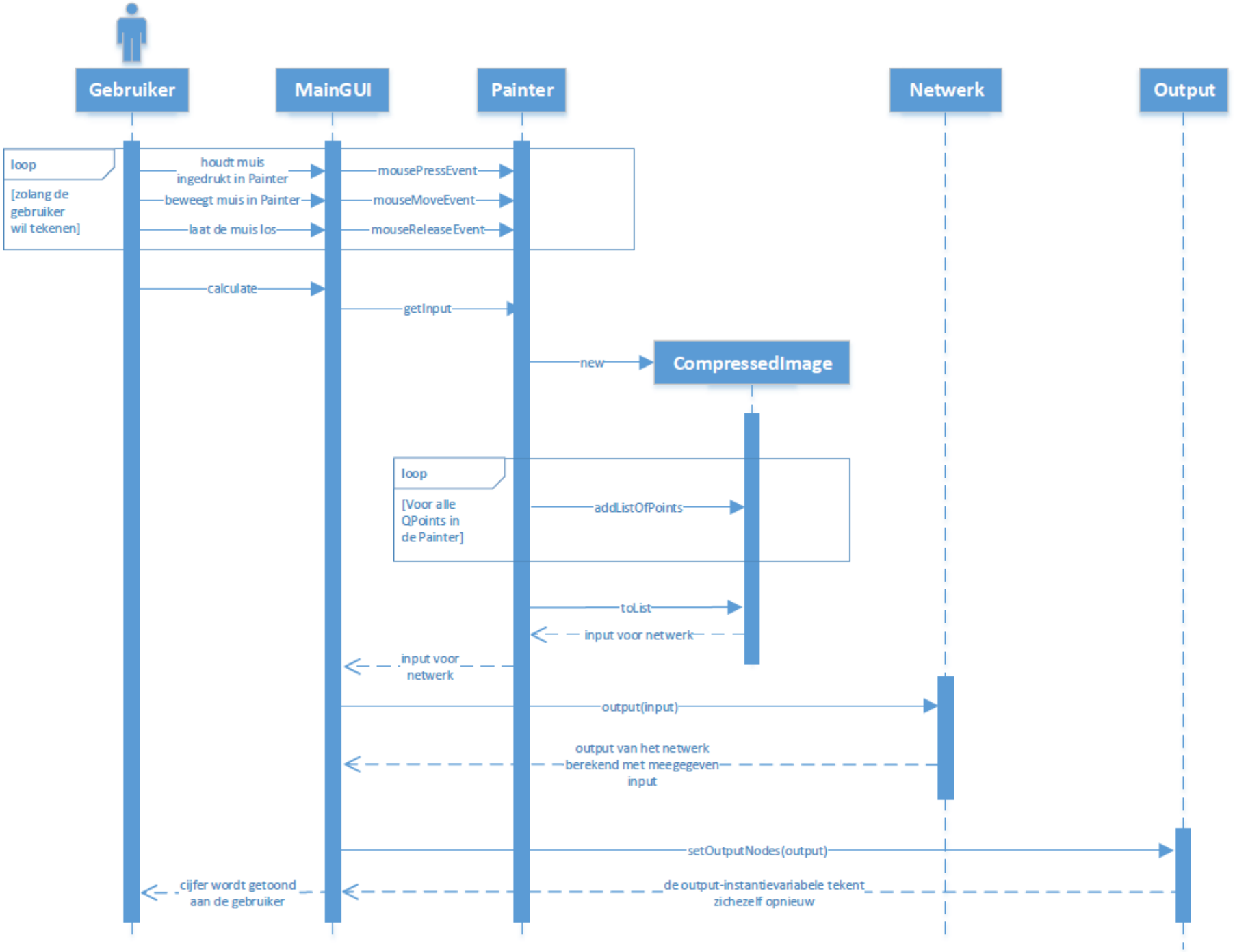
Trainen van het netwerk

Om het netwerk te trainen is er een module geschreven die eerst alle bestanden met trainingsdata zal ophalen uit de map met trainingsdata en de data zal lokaal worden opgeslagen in een variabele die zal meegegeven worden bij constructie van het netwerk. Hierna zal hetzelfde proces plaatsvinden voor de testdata. Er moet minimaal een bestand te vinden zijn in de map voor trainingsdata en ook een in de map voor testdata, indien dit niet het geval is zal de applicatie een error geven.

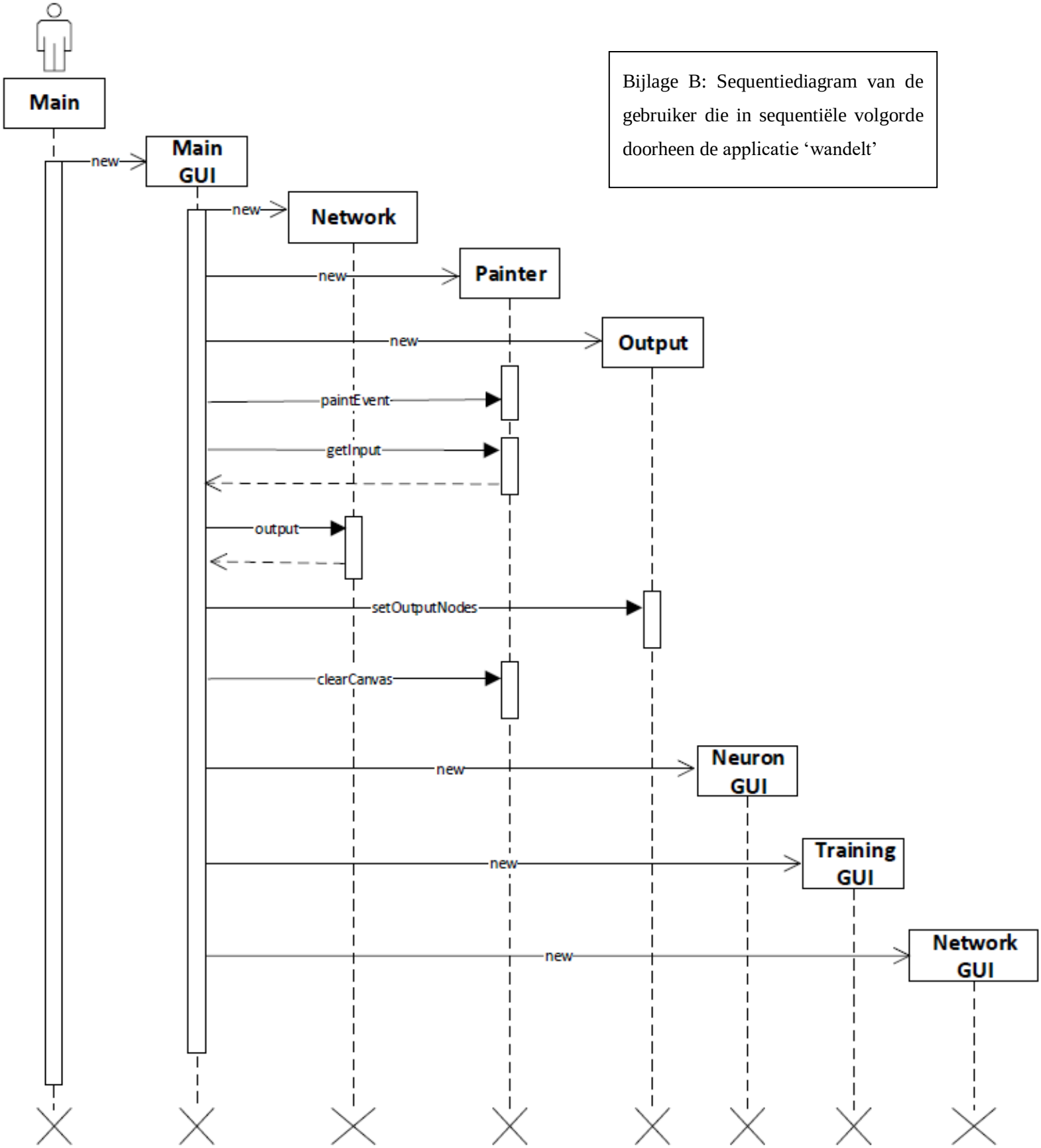
Hierna zal de data gebruikt worden om een object van de klasse Network aan te maken, dit netwerk zal getraind worden en na afloop zullen de parameters van het netwerk opgeslagen worden voor later gebruik in de applicatie.

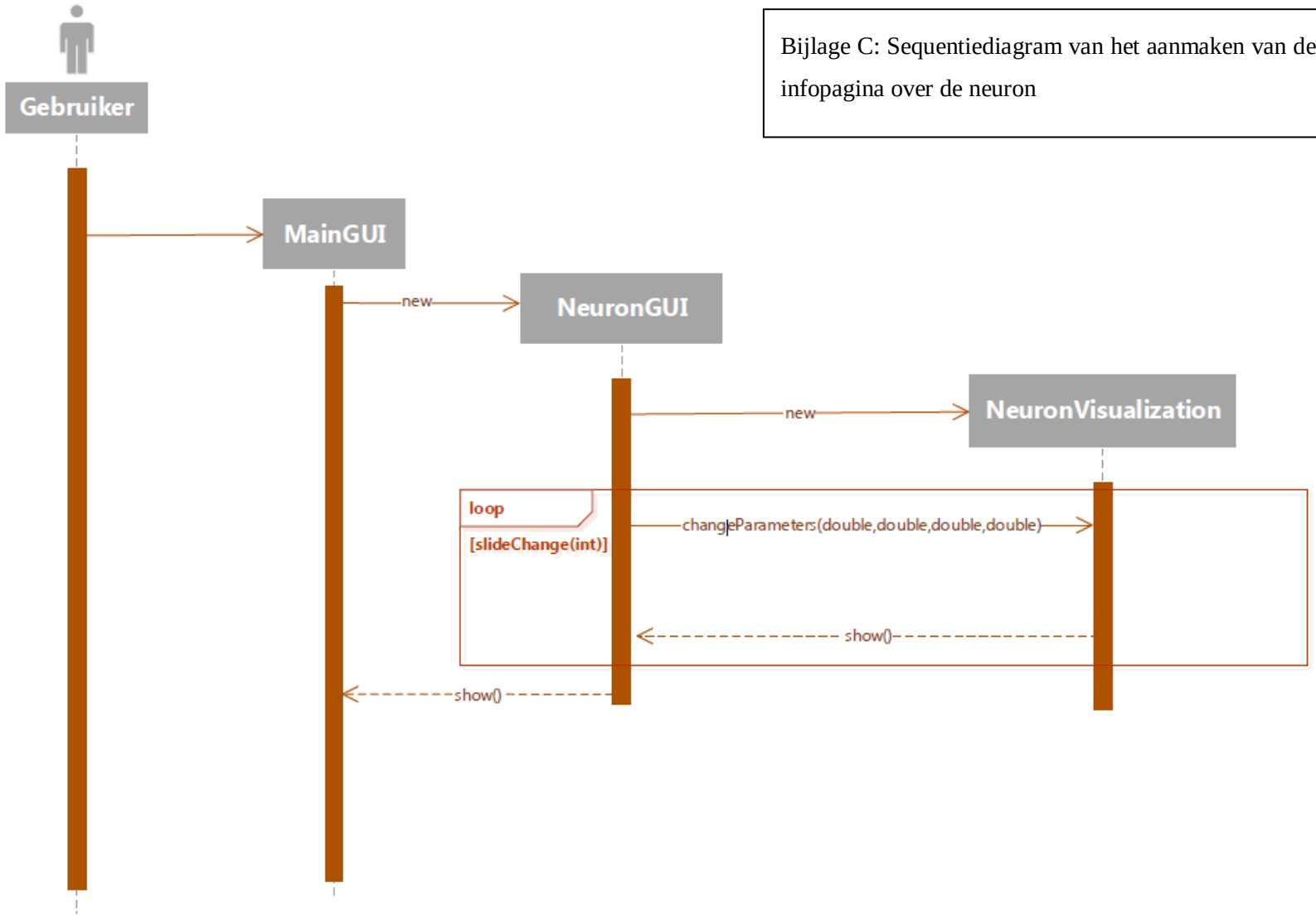
Het verloop van deze klasse is ook te volgen op het activiteitendiagram in Bijlage G.

Bijlage A: Sequentiediagram van de gebruiker die een cijfer tekent en dat getekende cijfer laat bepalen door het getrainde neurale netwerk

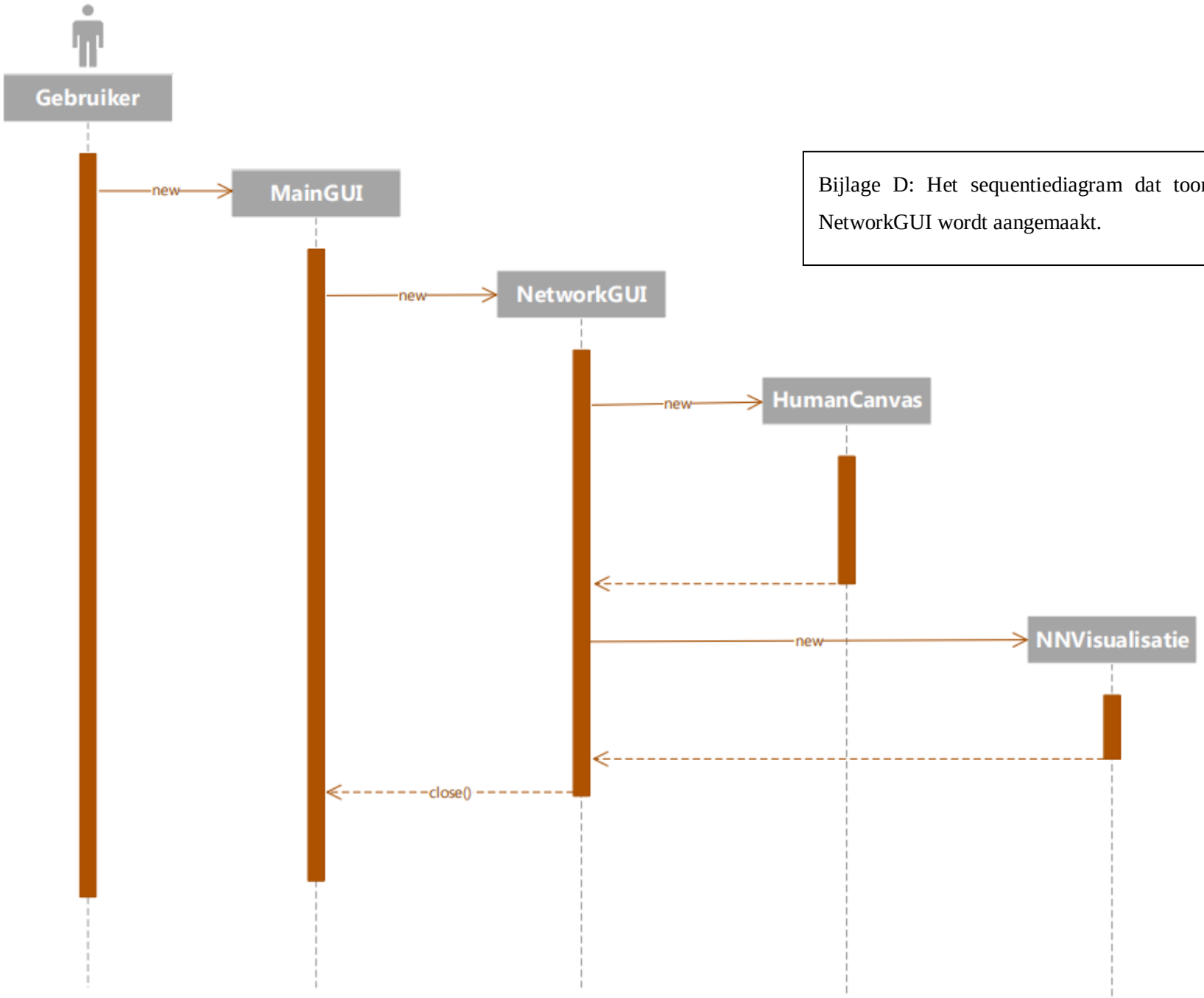


Bijlage B: Sequentiediagram van de gebruiker die in sequentiële volgorde doorheen de applicatie ‘wandelt’

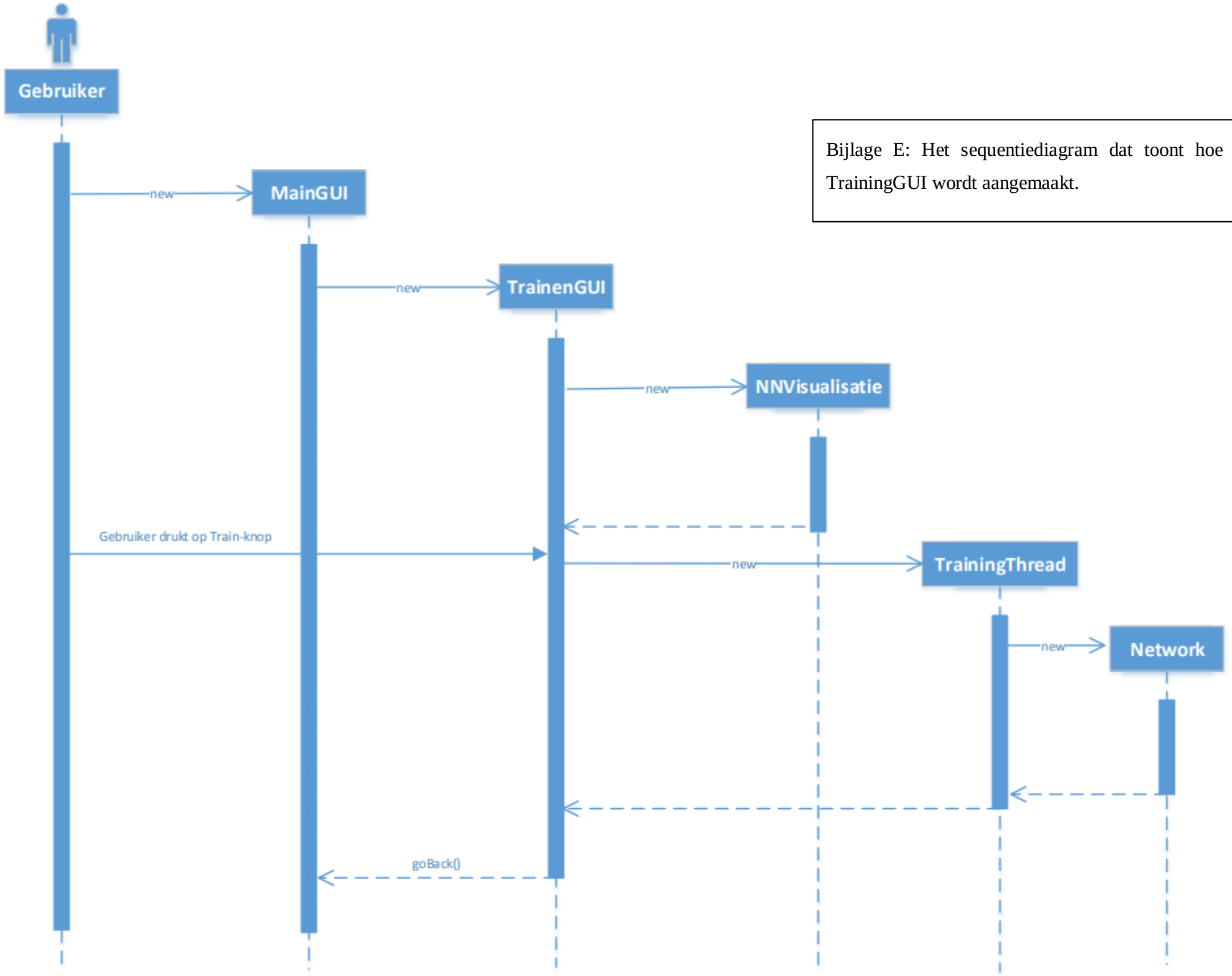




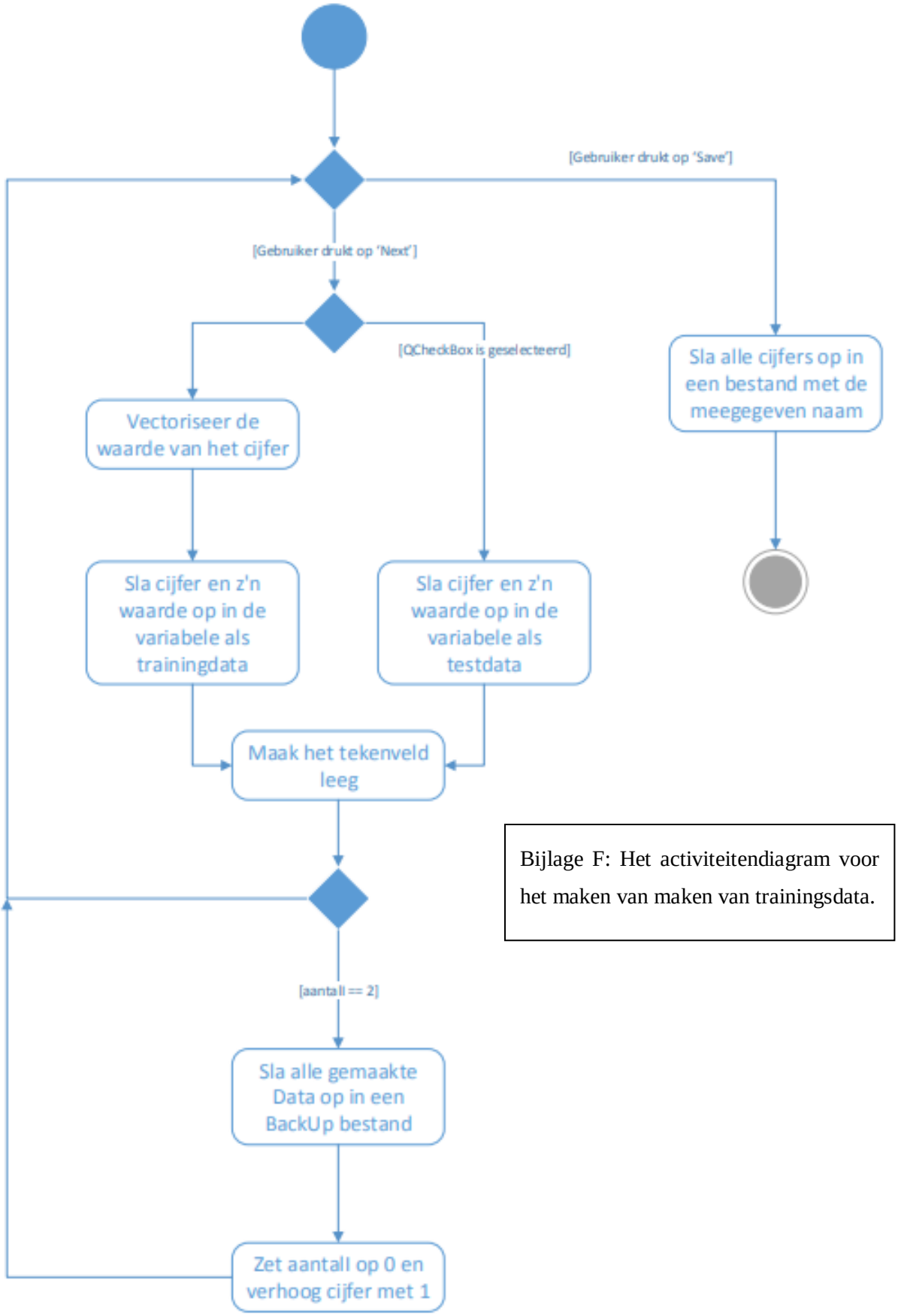
Bijlage C: Sequentiediagram van het aanmaken van de infopagina over de neuron



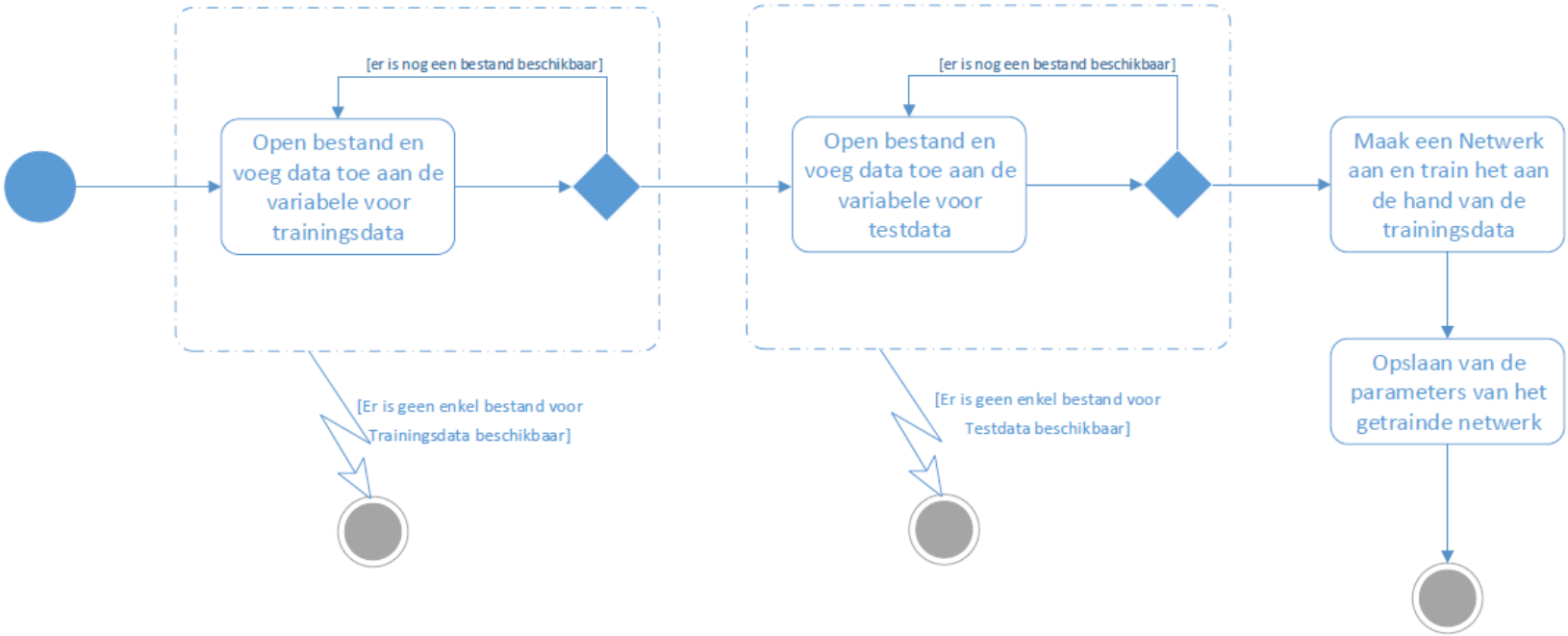
Bijlage D: Het sequentiediagram dat toont hoe de NetworkGUI wordt aangemaakt.



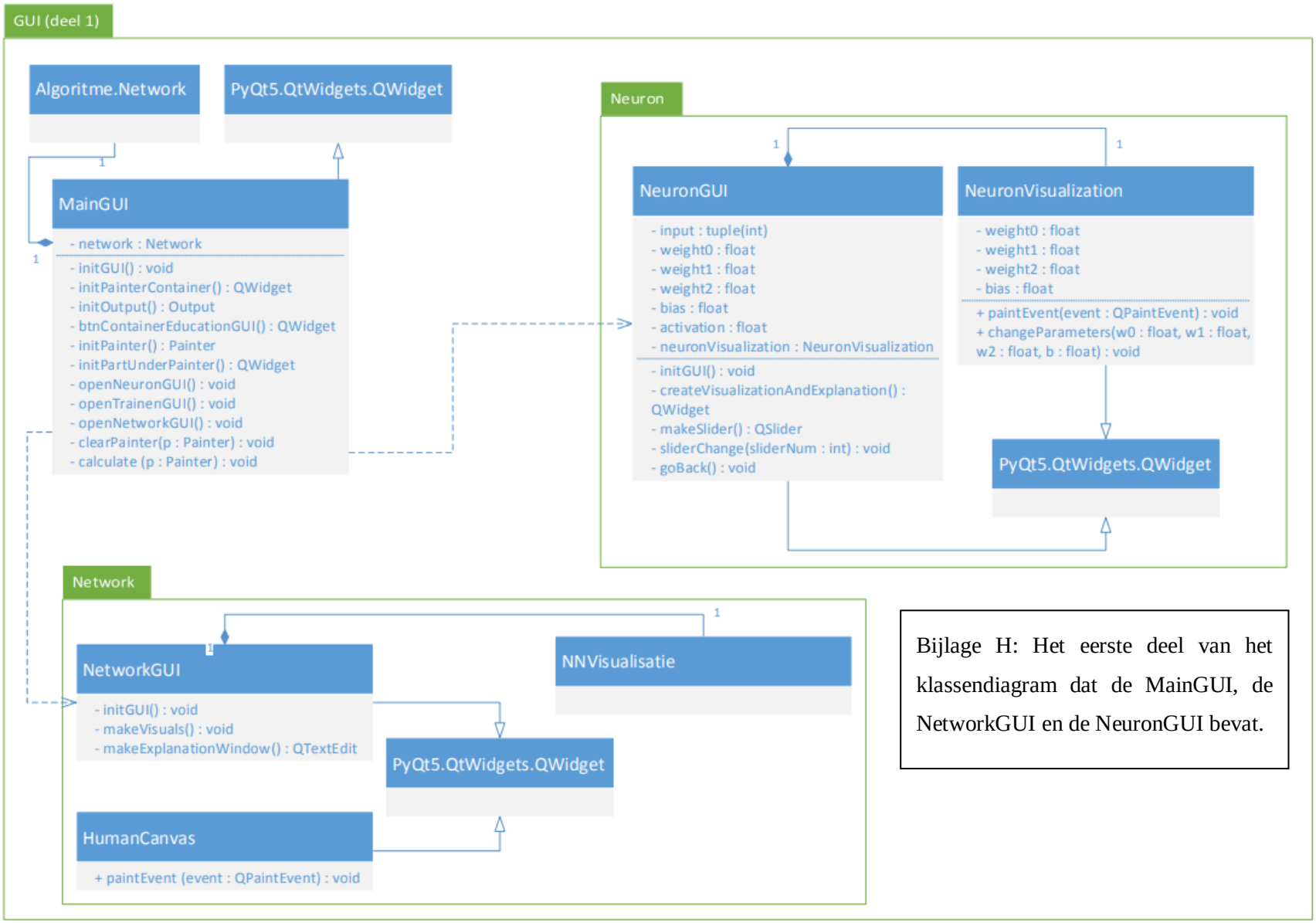
Bijlage E: Het sequentiediagram dat toont hoe de TrainingGUI wordt aangemaakt.



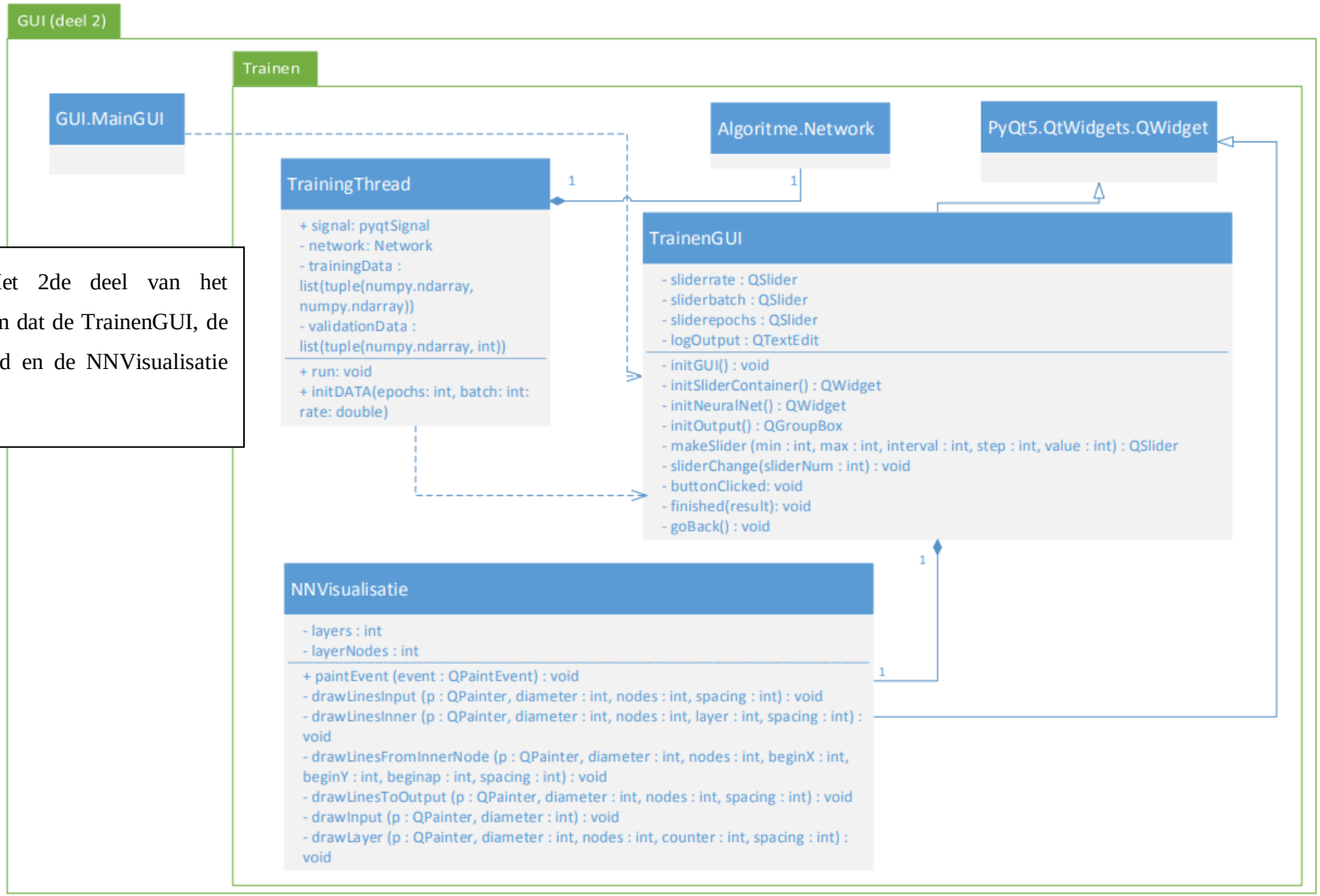
Bijlage F: Het activiteitendiagram voor het maken van maken van trainingsdata.



Bijlage G: Het activiteitendiagram voor het trainen van het netwerk.

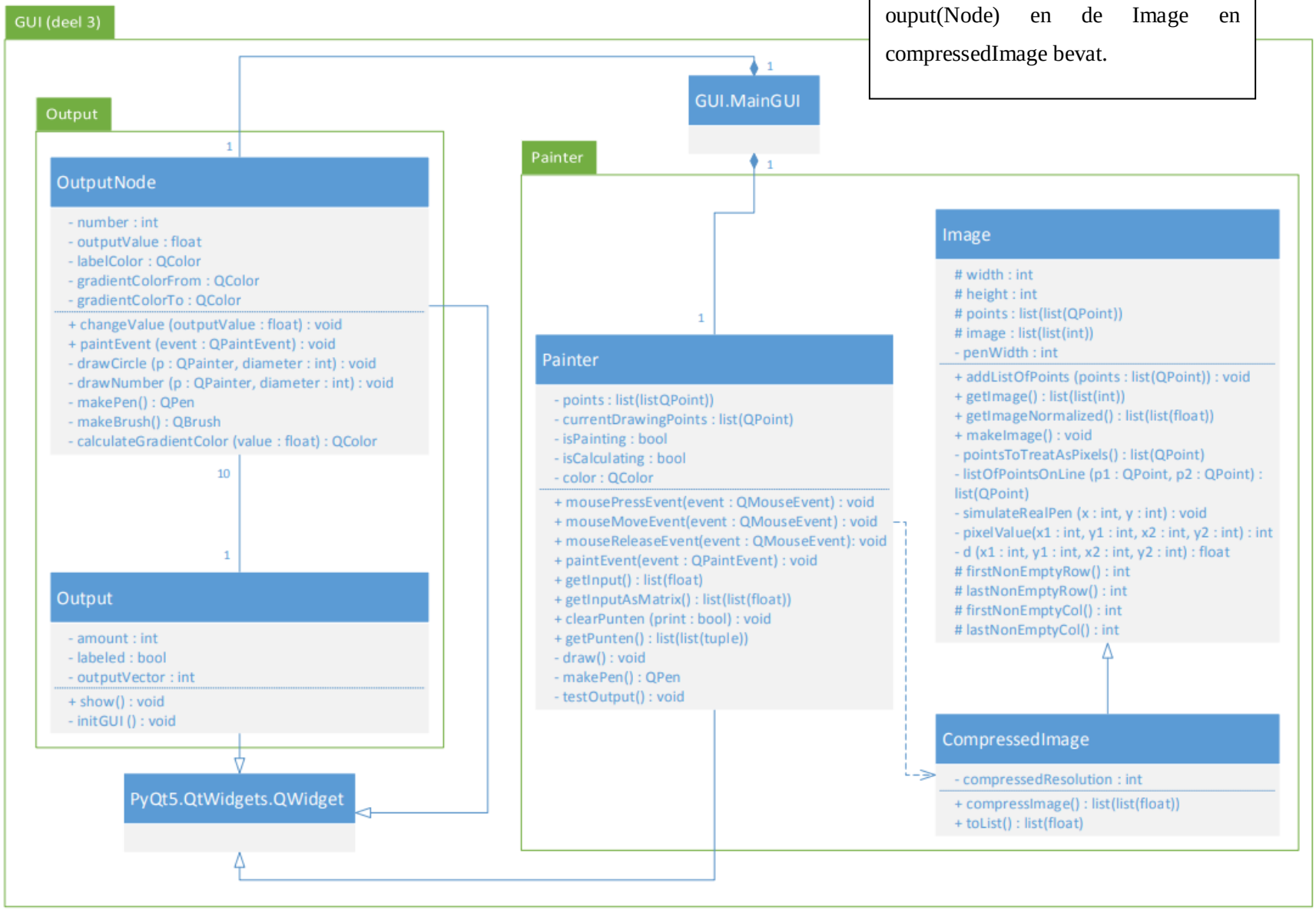


Bijlage H: Het eerste deel van het klassendiagram dat de MainGUI, de NetworkGUI en de NeuronGUI bevat.



Bijlage I: Het 2de deel van het klassendiagram dat de TrainenGUI, de TrainingThread en de NNVisualisatie bevat.

Bijlage J: Het 3de deel van het klassendiagram dat de Painter, de ouput(Node) en de Image en compressedImage bevat.



Bijlage K: Het 4de deel van het klassendiagram dat het netwerk bevat.