Bericht 45

Version vom 16.07.2023

Gruppe Montagsmaler 45:

- Bastian Huszar
- Bircan Sahin
- Julian Vögel
- Lucas Wewetzer

Aufgabenstellung

Wir haben das Projekt “Montagsmaler” für uns ausgesucht. Im Rahmen des Projekts sollten wir eine App mit GUI in Java schreiben, mit der man auf einer Zeichenfläche einen Gegenstand zeichnen können sollte. Diese Zeichnung sollte dann von einem eigens dafür entwickelten Neuronalen Netz erkannt werden. Die gesamte Projekt sollte nach dem Model-View-Controller-Pattern gestaltet sein.

Aufgabenaufteilung

Zur Vorbereitung auf das Projekt haben wir erst einmal grob die verschiedenen Kompetenzen verteilt. So einigten wir uns darauf, dass Lucas und Bastian zusammen hauptverantwortlich für die GUI sind, während Bircan und Julian für die Entwicklung des Neuronalen Netzes zuständig sind.

Projekt-Design (nach MVC-Pattern)

Um die Zuständigkeiten innerhalb des Projektes und die Interaktion von GUI und Neural Network besser unterteilen zu können, haben wir uns nach erstem Einlesen und Rumprobierenuns etwas länger zusammengesetzt per Discord und uns genau angeschaut, wie das Projekt genau organisiert sein sollte und kamen zu einem übersichtlichen Ergebnis.

Erst einmal haben wir als Gruppe festgelegt, wie die GUI ungefähr aussehen sollte: Wir entschieden uns für eine Zeichenfläche, auf der man mit gedrückter linker Maustaste mit der Maus einen Strich machen kann. Daneben sollte es ein paar Buttons geben, mit denen man die App steuern kann, unter anderem: einen Clear-Button und einen Recognize-Button (erkennt das Gezeichnete). Das Erkannte wird dann in einer Textbox angezeigt. Die GUI ist aufteilt in View- (mit main function) und Controller-Teil (mit Buttons), hat aber auch noch weitere Dateien und Ressourcen für eine funktionierende GUI.

Für das Neuronale Netz, um auch gut Ergebnisse zu erhalten, haben wir uns als Gruppe entschieden, Zeichnungen von 5 möglichst unterschiedlichen Gegenstände zu erkennen. Auch wollten wir ein Hidden Layer nutzen, haben uns aber zunächst nicht festgelegt auf eine genaue Neuronenzahl des Hidden Layers. Allerding bekam das Input Layer 28*28=784 Neuronen, da wir unser Neuronales Netz (im Folgenden auch NN genannt) mit Trainingsdaten von Google Quickdraw trainieren wollten, deren Bilder ein Format von 28*28 Pixeln haben (Die Grau-Bittiefe beträgt übrigens 8 Bit). Das Output Layer bekam 5 Neuronen für die letztlichen Wahrscheinlichkeiten der einzelnen zu erkennenden Gegenstandszeichnungen. Das Neuronale Netz übernimmt den Model-Teil im MVC-Pattern.

Die Gegenstände, für die wir uns entschieden haben, sind: eine Krawattenfliege, eine Wolke, ein Briefumschlag, die Sonne und ein T-Shirt.

Damit die Interaktion zwischen den einzelnen Komponenten auch gut klappt, brauchten wir entsprechende Hilfsmethoden bzw. -funktionen: Nach Drücken des Recognize-Buttons sollte ein repräsentatives Bild ähnlich zu den Trainingsdaten aus der Zeichung erstellt werden. Dafür sollte Image-Cropping und Downscaling betrieben werden, sodass die Zeichnung mitting in einem 28x28-Bild dargestellt wird. Da die Trainingsdaten nur Graustufen-Pixel mit einer Bittiefe von 8 Bit haben, entschieden wir uns, entsprechend das gezeichnete und zu erkennende Bild in ein Bitmap-Bild (mit gleichen Daten-Rahmenbedingungen, also auch mit Grau-Bittiefe von 8 Bit) zu formatieren, sodass am Ende eine repräsentative Datei vorhanden war. Diese drei Teile übernahm die GUI-Gruppe.

Jetzt hatten wir im Projekt-Design auch ein gutes Bild vom Gezeichneten, passend zu den Trainingsdaten. Um die Bilddaten dem Neuronalen Netz und damit der finalen Auswertung zuzuführen, musste das Bild in ein Java-Double-Array konvertiert werden. Diesen Teil übernahm innerhalb der NN-Gruppe Bircan.

Da jetzt nun fast alles geplant war, konnten wir an die handfeste Entwicklung rangehen.

Neuronales Netz

Nachdem wir uns nach ausgiebiger Recherche einfachheitshalber für ein Neuronales Netzwerk mit einem Hidden Layer entschieden, startete Bircan mit der Implementierung des Neuronalen Netzes und Julian kümmerte sich um die Matrix-Klasse für das Neuronale Netzwerk.

Zunächst schrieb Bircan ein kleines Programm um die Trainingsdaten von Google’s Quickdraw (.npy files) in Arrays umzuwandeln, sodass wir in Java damit weiterarbeiten konnten. Dann machte sie sich an die Implementierung des eigentlichen Neuronalen Netzwerkes und erstellte hierfür eine neue Klasse NeuralNetwork. Zusammen in Absprache mit Julian entschieden sie sich, das Neuronale Netzwerk mit Batches zu realisieren, da es effektiver und schneller beim Training ist. Anfänglich wurden die Attribute und Methoden gewählt, die das Netzwerk enthalten soll, welche wie folgt umgesetzt wurden: Das Netzwerk besteht aus einer Eingabeschicht (input layer), einer versteckten Schicht (hidden layer) und einer Ausgabeschicht (output layer). Der Konstruktor NeuralNetwork erstellt ein neues Neuronales Netzwerk mit gegebener Größe für die Eingabe-, Ausgabe- und versteckten Schichten. Die Größe einer Schicht bestimmt die Anzahl der Neuronen in der jeweiligen Schicht. Des Weiteren wird eine Lernrate (learningRate) festgelegt, die bestimmt, wie schnell das Netzwerk während des Trainings lernt. Die Methode initializeWeights initialisiert die Gewichte und Verzerrungen (biases) für das Netzwerk. Diese Werte werden zu Beginn zufällig gesetzt, in diesem Fall mit Werten aus einer normalverteilten Zufallsvariablen (Gauß-Verteilung). Wir entschieden uns für tanh als Aktivierungsfunktion des Netzwerks. Da wird die Hyperbolische Tangens-Funktion (tanh) genutzt. Die Methode tanhDerivative berechnet die Ableitung der Tanh-Funktion. Sie wird benötigt für den Backpropagation-Schritt im Trainingsprozess. Die Methode softmax wird für die Ausgabeschicht verwendet und stellt sicher, dass die Ausgabewerte des Netzwerks Wahrscheinlichkeiten zwischen 0 und 1 darstellen, derenSumme genau 1 ist. Die Methode forward führt einen Vorwärtsdurchlauf durch das Netzwerk durch. Sie nimmt die Eingabe, multipliziert sie mit den Gewichten, addiert die Verzerrungen und wendet die Aktivierungsfunktion an. Die Methode train implementiert das Training des Netzwerks. Hierbei wird die Technik des stochastischen Gradientenverfahrens mit Minibatches angewandt. Dabei werden die Eingaben in kleine Batches unterteilt und die Netzwerkgewichte basierend auf dem durchschnittlichen Fehler des Batches aktualisiert. Im Detail besteht das Training aus einem Vorwärtsdurchlauf (forward pass), in dem die Ausgabe des Netzwerks basierend auf den Eingabedaten berechnet wird. Anschließend wird der Fehler zwischen dieser Ausgabe und dem tatsächlichen Zielwert berechnet (der sog. "Fehler" oder "Verlust"). Dieser Fehler wird dann im Backpropagation-Schritt verwendet, um die Gewichte und Verzerrungen des Netzwerks anzupassen. Hierbei werden auch die Ableitungen der Aktivierungsfunktionen genutzt.

Nach der Implementierung des Netzwerkes kümmerte sich Bircan um verschiedene Klassen und Funktionen von der Unterstützung des Trainings bis hin zur Schnittstelle zwischen GUI und NN. Julian behielt weiterhin den großen Überblick und unterstützte die Gruppe bei der Planung und Organisation. Unter anderem erstellte Bircan eine Klasse NeuralNetworkIO, um während des Trainings die vier weights- und biases-Matrizen in einer network_matrices.csv-Datei abzuspeichern und für das nächste Training aufrufen zu können. Außerdem schrieb sie eine Training Klasse, worin am Ende das Netzwerk trainiert wurde und eine TestNetwork-Klasse, womit das Netzwerk nach einem Trainingsabschnitt immer getestet wurde und prozentual die Genauigkeit des Netzwerkes ausgegeben wurde. Getestet wurde dabei immer auf 100.000 neuen unbekannten Bildern, mit denen nicht trainiert wurde.

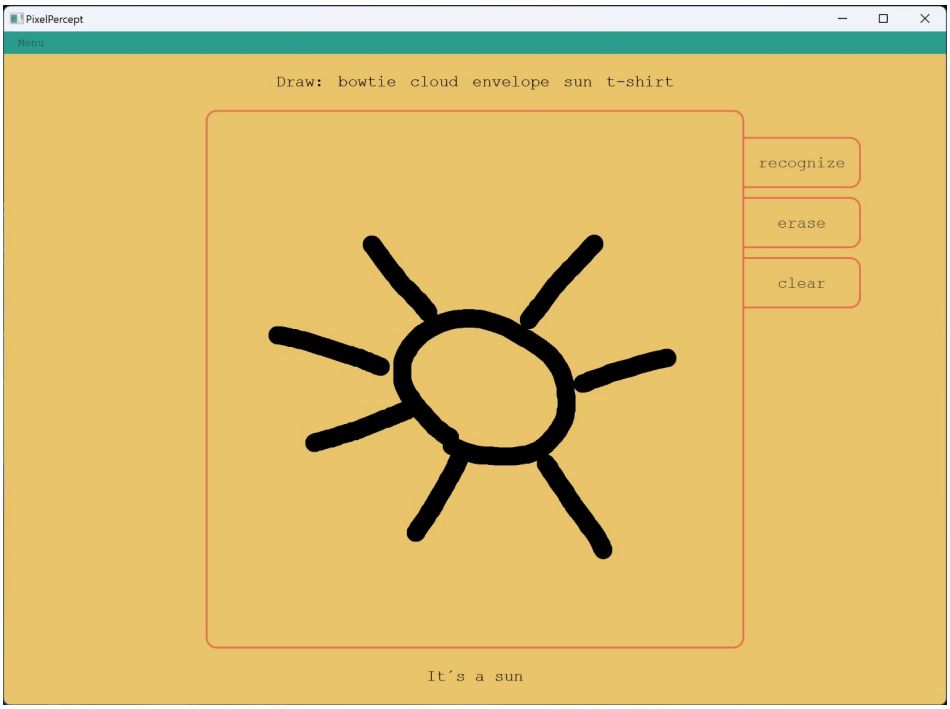
Beim Training an sich musste sehr viel hin und her ausprobiert werden und zwischendurch viele Kopien des Netzwerkes mit NeuralNetworkIO erstellt werden, um am Ende auf das beste Ergebnis zu kommen. Da es Ziel des Trainings ist, auf ein lokales bzw. globales Minimum (man kann leider nicht wissen, was man am Ende erreicht) in der Funktion zu kommen, hat Bircan versucht sich künstlich diesem zu nähern. Zunächst startete sie mit einer kleinen Batchsize, einer geringen Epochenanzahl und einer recht hohen learningRate (0,01), um große Schritte Richtung Minimum zu machen. Außerdem wählte sie am Ende nach mehreren Tests eine Neuronenanzahl von 1024 für das Hidden Layer, da dies sich in unserem Fall als am besten ergeben hat. Schritt für Schritt wurden nun die Batchsize und Epochenanzahl ein wenig erhöht, wohingegen die learningRate immer ein Stückchen verringert wurde. Dadurch wurden die Schritte zum Minimum immer kleiner und kleiner und das Netzwerk näherte sich dem Minimum immer weiter, ohne es aus Versehen zu überspringen. Zwischendurch wurden immer wieder Kopien erstellt, da man manchmal auch wieder zurück zu einer vorherigen Kopie musste. Des Weiteren wurde das Netzwerk nach einer bestimmten Epochenanzahl getestet, um mitverfolgen zu können, dass das Netzwerk sich insgesamt (bis auf kleine natürliche Schwankungen) verbessert und dem Minimum nähert. Am Ende erreichten wir eine Genauigkeit von ca. 96-97% und beendeten das Training aus dem Grund, dass es zeitlich immer aufwendiger wurde, da man anfangs sehr schnell gute Fortschritte macht, am Ende es jedoch sehr lange dauert für kleine Fortschritte. Darüber hinaus waren wir der Meinung, dass unser Netzwerk in der gegebenen Zeit nicht mehr sehr viel besser werden kann, da es ein sehr simples Neuronales Netzwerk mit nur einem Hidden Layer ist.

Nach dem erfolgreichen Training kümmerte sich Bircan um die Schnittstelle zwischen NN und GUI. Die Klasse BMPToArrayConverter nimmt das von der GUI nach dem Zeichnen erstellte picture.bmp und wandelt es in ein Array um. Folglich wird das Neuronale Netzwerk mit den aus dem Training gespeicherten Matrizen (network_matrices.csv) initialisiert und das BMP-Array in die forward-Methode gefüttert, wodurch wir ein Array mit 5 Werten (output) erhalten. Dabei steht jeder Wert für die Wahrscheinlichkeit des jeweiligen Motivs (Reihenfolge der Motive im Array wie folgt: [bowtie, cloud, envelope, sun, t-shirt]). Die höchste Wahrscheinlichkeit wird letztlich gewählt und auf der Oberfläche ausgegeben.

GUI

User-Interface und Designentscheidungen

Das User-Interface besteht aus einer Zeichenfläche die sich in der Mitte des Fensters befindet. Rechts von der Zeichenfläche befinden sich drei Buttons. Mit diesen Buttons kann der Benutzer Gemaltes auf der Zeichenfläche löschen, zwischen den Funktionen Zeichen/Radiergummi umstellen und die recognize-Funktion aufrufen, welche den Prozess der Bilderkennung durch das Neuronale Netz auslöst. Die App startet direkt im Hauptfenster und der User kann sofort auf die Funktionen der App zugreifen. Auf ein „Starten“-Button oder ähnliches haben wir bewusst verzichtet um die Benutzung des Programms für den User so einfach und zeiteffektiv wie möglich machen.

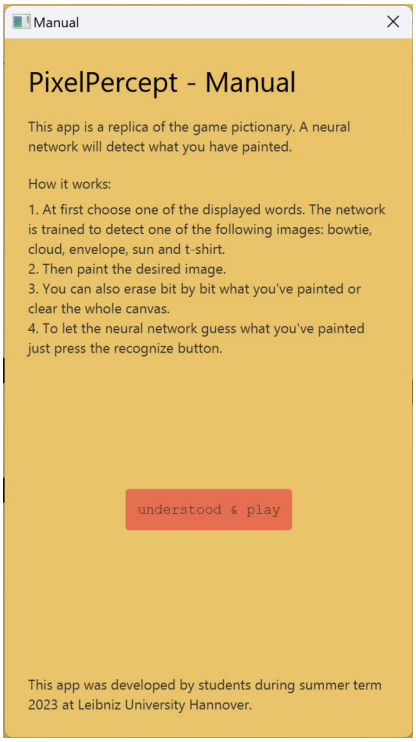


Main-Window der Applikation PixelPercept

Die Farben in denen das Programm gestaltet ist sollten zueinander passen und ansprechend aussehen. Hierzu haben wir mit Unterstützung des Internets und etwas herumprobieren für uns passende Farben gewählt.

Sollte der Benutzer das Programm aus Versehen schließen, würde gegebenenfalls Gemaltes gelöscht werden ohne das es erkannt wurde. Um dies zu vermeiden haben wir eine Abfrage eingebaut in welcher der Benutzer nochmal gefragt wird ob er das Programm wirklich schließen will.

Außerdem haben wir uns dafür entschieden ein User-Manual zu erstellen, welches über einen separaten Menupunkt zu erreichen ist. Hier wird dem User die Bedienung des Programms erklärt. Das Manual wird in einem neuen Fenster geöffnet. Während das Manual-Fenster geöffnet ist, ist die Interaktion mit dem Hauptprogramm für den User nicht möglich. Das Manual-Fenster muss zuerst geschlossen werden.



User-Manual der Applikation PixelPercept

Beim Entwickeln der Zeichenfunktion ist uns aufgefallen, dass beim schnellen Ziehen der Maus über das Canvas unser Programm keinen durchgängigen Strich zeichnete. Es entstand zwar ein Strich, dieser war jedoch von Lücken durchzogen. Dies lag daran, dass wir zuerst das Canvas mit einem kleinen Punkt bemalten, wenn die Maus über das Canvas gezogen wurde. Wird die Maus nun zu schnell über das Canvas gezogen entstehen Lücken im Strichverlauf. Dieses Problem haben wir behoben, indem wir mittels einer Schleife alle Punkte die erkannt wurden jeweils mit einer Linie verbinden. So entsteht eine durchgezogene Linie ohne Lücken.

Beim Entwickeln des Designs haben wir darüber nachgedacht wie das Design des Programms mit seinen geforderten Funktionalitäten am besten strukturiert werden kann. Software ist unserer Meinung nach gelungen wenn sie intuitiv und effizient (keine überladenen Seiten, keine unnötigen Mausklicks) zu benutzen ist. Dies war in unser PixelPercept Applikation einfach umzusetzen, da unsere App nicht sonderlich komplex ist. Wir haben jedoch gelernt, dass das Design und die Führung des Benutzers durch die Applikation eine Bedingung für ein erfolgreiches Programm sein kann.

Verarbeitung des Gezeichneten und Verbindung mit Backend

Aufbauend auf der grundlegenden Planung der GUI, welche das Teilstück des Projektes der Mensch Computer Interaktion repräsentiert, wurden zunächst einige Arbeitsgrundlagen geschaffen, woraufhin verschiedene Iterationen der GUI folgten. Zuerst war es nur möglich ein Bild zu malen und dann auch zu löschen was mit Userevents realisiert wurde. Dann folgte die Option das gemalte Bild im Bitmap-Format zu speichern und anzusehen, was die Problemanalyse stark vereinfacht.

Die Funktion des Image-Croppings in Kombination mit dem Downscaling ermöglichen die Anwendung der gezeichneten Bilder auf das Neuronale Netz, wobei diese Voraussetzung durch das 28*28 Pixel-Format der Trainingsdaten bedingt ist.

Das Image-Cropping hat die Funktion den für die Auswertung relevanten Teils des Canvas auszuschneiden. Dafür wird ein Quadrat um die Zeichnung gelegt, wobei die Zeichnung stets in der Mitte des Quadrats liegt. Dies ermöglicht bessere Resultate beim Downscaling, da nicht bemalte Teile des Bildes beim Downscaling ignoriert werden können, wodurch mehr der 28*28 Pixel die eigentliche Zeichnung darstellen, was den Informationsverlust bei diesem Schritt minimiert. Damit das Quadrat die Maße des Canvas nicht überschreiten kann, falls zum Beispiel in einer Ecke gemalt wird, ist es erforderlich, dass der Teil, welcher auf dem Canvas bemalt wird, stets von ausreichend weißer Bildfläche umschlossen ist, was zuvor gewährleistet werden muss.

Beim Downscaling auf 28*28 Pixel haben wir auf die Funktionen von Graphics2d zurückgegriffen. Der Vorteil von Graphics2d besteht darin, dass keine Externe Library von Drittanbietern zusätzlich heruntergeladen werden muss, um das Programm zu nutzen, da Graphics2d bereits zur Standartversion von Java gehört. Um möglichst gute Ergebnisse zu erzielen ist es ratsam, neben dem Image-Cropping, eine ausreichend große Pinselgröße zu wählen und groß genug zu Zeichen, da dadurch mehr Informationen über die Zeichnung weitergegeben werden und der Informationsverlust bei Reduktion der Pixel weniger Signifikant ausfällt.

Vom Backend (KI) werden Wahrscheinlichkeiten für die jeweiligen Bilder zurückgegeben. Auf Basis dieser Wahrscheinlichkeiten wird denn in Textform ausgegeben welches Bild das Neuronale Netzwerk erkannt hat.

Wie startet man PixelPercept?

Um die App zu starten, muss man unter dem Pfad “PixelPercept/src/main/java/de/uni_hannover/pixelpercept/view” mit IntelliJ die Datei “PixelPerceptApplication.java” öffnen und auf den grünen Playbutton in der App drücken. Nach ein wenig Warten launched die App und sie sehen das beschriebene Fenster.

Zusammenarbeit

Innerhalb der Gruppe lief es absolut reibungslos. Jedes Gruppenmitglied hat aktiv und motiviert seinen eigenen, qualitativen Beitrag zum Projekt geleistet. Die anfängliche, grobe Kompetenzeinteilung blieb auch in etwa bestehen, doch die letztlichen Zuständigkeiten haben sich später etwas konkreter abgezeichnet. Lucas hat vor allem die Grafische Benutzeroberfläche designed, Bastian hat viel rund um GUI-NN-Interaktion entwickelt, Bircan hat das Neuronale Netz trainiert und die entsprechenden Daten dafür angepasst und Julian hatte neben NN-Konzeption zusammen mit Bircan den großen Überblick über das Projekt(-Design).

Fazit / Was haben wir gelernt?

Nachträglich besser hätten wir kaum etwas gemacht. Insgesamt haben wir so gezielt an unseren Aufgaben rumgefeilt und eine schöne App entwickelt.

Bastian und Lucas haben aufgrund ihrer Arbeit an der GUI viel gelernt in Bezug auf Frontend-Entwicklung, Bircan und Julian haben dagegeben durch ihre NN-Entwicklung viel darüber gelernt. Alle bis auf Bircan (sie konnte es schon) haben dazu noch Git gelernt. Die gegebenen Lernressourcen waren gut, wir mussten allerdings noch hier und da etwas weiter recherchieren.

Das Wichtigste war aber, dass wir gelernt haben, einerseits wie man ein etwas größeres Projekt in Teamarbeit realisiert, wie man Aufgaben delegiert und wie die einzelnen Java-Klassen in Package-Strukturen mit einander interagieren.

Das Arbeitspensum und die verfügbare Zeit waren angemessen, insbesondere weil die Vorlesungen früher für die Gruppenphase aufgehört haben.

Insgesamt fanden wir das Projekt interessant und lehrreich.