CNN, EMINST & ritade tangenter

William Frid

Höst 2023

1 Introduktion

Vid användning av datorer används i regel tre typer av inmatningsmetoder, tangentbort, datormus (eller styrplatta på bärbara enheter) och kamera. Dessa tre har alla utvecklats över en lång tid och kommer i olika varianter och finnesser. Men med dagens stora framsteg inom AI och hårdvara kan vi ta detta längre utan att utveckla nya datortillbehör. Genom att använda de webbkameror vi har och AI-modeller ihop med befintliga algoritmer för bildbehandling och kan vi bland annat ersätta eller utöka våra tangentbord med ritade sådana. Anpassade för just rätt användningsområde.

Denna rapport täcker framtagandet av ett enklare djupt faltningsnät (engelska CNN) för att klassificera tecken (siffrorna 0-9 samt versalerna A-Z [då dessa är mer passande för tangentbord]). Rapporten inkluderar även övergripande arbetet kring att läsa av ett videoflöde samt bildbehandling då detta är avgörande steg i processen (se sektion 4). Modellen i fråga bygger på Tensorflow och dataflödet samt bildbehandlingen sker med hjälp utav OpenCV (mer under sektion 2).[1][2] Notera att denna modell liknar den första typen av RCNN som föreslogs i handlingen "Rich feature hierarchies for accurate object detection and semantic segmentation", men saknar stödet att föreslå ett justera område ihop med klassifisering.[3]

Konceptet med platta tangentbord är ingenting nytt. En redan lanserad produkt av IBM kallas för "Projection keyboard" och använder sig av en laser för att projektera ett tangentbord och avgör med hjälp av algoritmer var fingrarna är baserat på ljusreflektioner med infrarött ljus.[4] Men detta kommer med problemet med ytterligare hårdvarukostnader. Ett problem som potentiellt kan undvikas. Dummare hårdvara, smartare mjukvara.

[1] Tensorflow. https://www.tensorflow.org/ (2024-01-13).

[2] OpenCV. https://opencv.org/ (2024-01-13).

[3] Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, IEEE. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. IEEE Xplore. https://ieeexplore.ieee.org/document/6909475 (2024-01-10).

[4] 2015. Projection keyboard. Wikipedia. https://en.wikipedia.org/wiki/-Projection_keyboard (2024-01-10).

För bäst förståelse av denna rapport rekommenderas att du läser den ihop med kod-filerna som finns i ett GitHub-repository: https://github.com/williamfridh/5TF078-CNN-EMINST-drawn-keyboards

2 Tekniker & flöde

De tre huvudsakliga tekniker som användes ihop med denna rapport var följande:

- Python(> 3.10+) - CNN och program för datorflöde.[5]
- OpenCV(4.8.1.78+) - Datorseende och bildbehandling.
- Tensorflow(2.14+) - Källkodsramverk för maskininlärning och djupinlärning.

Notera att ingen av dessa tekniker är utstickande och att resultat kan replikeras med andra vektyg.

Innan vi går vidare i rapporten kan en förståelse över det slutgiltiga programmets flöda vara av vikt. Med det första steget som är dataflödet. Genom OpenCV kan data hämtas in via en webbkameran och sedan tillsammans med inbyggda funktioner och ytterligare skriven logik klippa ut de ritade tecken utan att inkludera de ritade rutorna (kallat ”region of interest” [ROI]). Dessa rutor kan sen skickas till CNN modellen som klassifiserar datan som sedan presenteras i programmet tillsammans med de detekterade rutorna. Flödet beskrivs tydligt i figur **??**.

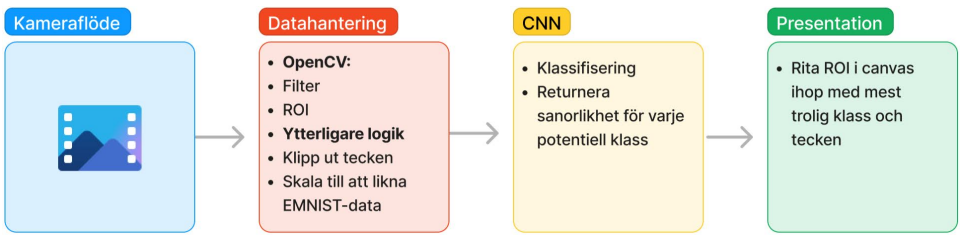

Figure 1: Illustrationen visar en övergripande förklaring av flödet för datan. Hur datan hämtas video en webbkamera, detekteras, bearbetas och sedan klassifiseras samt presenteras.

En djupare förståelse för flödet kommer tillkomma i takt med att rapporten fortlöper.

[5]python. https://www.python.org/ (2024-01-13).

3 CNN

Följande sektion täcker arbetet kring CNN modellen. Så som data, framtagning av struktur, träning och test. För bäst förståelse för flödet rekommenderas att läsa Jupyter Notebook-filen med lokalisering CNN/main.ipynb i GitHub repositoryt (se sista paragrafen under sektion 1) ihop med denna sektion.

3.1 Data

För att träna modellen att kunna klassificera ritade tecken valdes EMNIST (Extended MNIST).[6] En utökad version av MNIST som även inkluderar bokstäver hämtade från NIST Special Database 19 konverterade till $28x28$ i storlek framtagen för att möjligöra ett mer komplext benchmark än MNIST som började nå upp till 99.7% (ohållbart).[7][8] Settet är i sin helhet obalanserat, men det finns en färdig balanserad samling data man kan nyttja för att ens modell inte ska partisk, sämre i ut-stickande fall, ge missledande mätvärden, och så vidare.

Det balanserade settet innehåller total 112800 träningsbilder och 18800 testbilder av storlek $28x28x1$ (fyrkantiga med längd 28 och en färgkanal [gråskala]). Efter att ha filtrerat bort $a - z$ (exklusive de tecken som var lika för båda gemener och versaler) och delat upp den kvarvarande träningsdatan i både träning och validering fanns följande datamängder; träning(69120), validering(17280) och test(14400). EN relativt liten datamängd för detta syfte, men kan duga gott om datan som matas in är mycket tydlig.

Då transformering av stor mängd data kan leda till minnesbrist används "data streams/generators" som både skickade datan till en funktion som transformerar den, och utför "data augmentation" för att generera ytterligare data då EMNIST (balanserat set) är litet. Bildtransformationen som även inkluderar invertering av färger (då test-programmet för dataflöde nyttjar det) finns illustrerat i figur 2.

[6] 2018. EMNIST (Extended MNIST). Kaggle.- https://www.kaggle.com/datasets/crawford/emnist (2024-01-10).

[7] NIST Special Database 19Dataset. NIST. https://www.nist.gov/srd/nist-special-database-19 (2024-01-13).

[8] Gregory Cohen, Saeed Afshar, Jonathan Tapson, André van Schaik, arXiv. 2017. EMNIST: an extension of MNIST to handwritten letters. Cornell University. https://arxiv.org/abs/1702.05373v1 (2024-01-11).

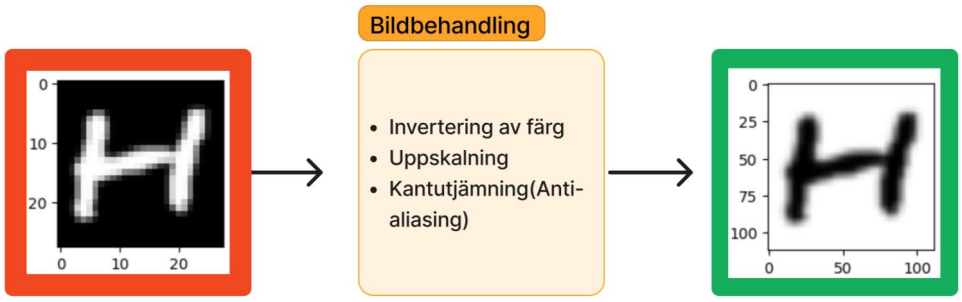

Figure 2: Resultatet av bildbearbetningen av bilder för test, träning och validering som utförs via dataflödet. Invertering av färg sker då dataflödet som används utför detta av praktiska skäl. Uppskaling för att modellen ska kunna ta in större data. Samt kantutjämning för mjukare kanter och mer koncist data.

3.2 Framtagning

Genom att nyttja överförd inlärning kan man optimera arbetet då man nyttjar redan tränade lager. Flertal tester utfördes med Keras Tuner (ihop med mätvärdet "accuracy") för att finna de optimala lagrena att kapa modellerna vid samt hur de skulle se ut efteråt (kompakta lager, normalisering, dropout och så vidare), men detta var ej hållbart på grund av begränsade resurser och träning utföras med hjälp av en GPU av typen RTX 2070 Super.[9] Vid en sådan begränsning tenderar video-minne att ta slut då minneshanteringen är begränsad. Trotts crasher vid körningar hittades ett flertal potentiella kombinationer med Xception och VGG16 som båda är typiska lager att nyttja för en sådan modell.[10][11]

Då modellerna var stora, och trotts att inte alla lager tränades, tog träning av dessa modeller mycket lång tid och tenderade att vara stora i filstorlek ($50 - 500MB+$). Efter träningar som tagit upp mot 6 timmar med träffsäkerhet på cirka 91% tog beslutet att utforska och träna modeller från grunden.

En ny uppsättning med Keras Tuner byggdes med inspiration av VGG16's arkitektur som nyttjar små, men många, $3x3$ faltningslager med faltningskärna på $[1, 2]$.[12] Efter en sökning med 14 försök hittades en modell på enbart $9.40MB$ som nådde hela 90.4% träffsäkerhet efter bara 6 epoker träning. Denna modell är illustrerad i figur 3. Intressant nog bidrog ett svagt "dropout"-lager med ett lågt värde på 0.15 trotts tre stycken normaliseringslager.

[9] KerasTuner. Keras. https://keras.io/keras_tuner/ (2024-01-13).
[10] Xception. Keras. https://keras.io/api/applications/xception/ (2024-01-13).
[11] VGG16 and VGG19. Keras. https://keras.io/api/applications/vgg/ (2024-01-13).
[12] Understanding VGG16: Concepts, Architecture, and Performance. datagen. https://datagen.tech/guides/computer-vision/vgg16/ (2024-01-11).

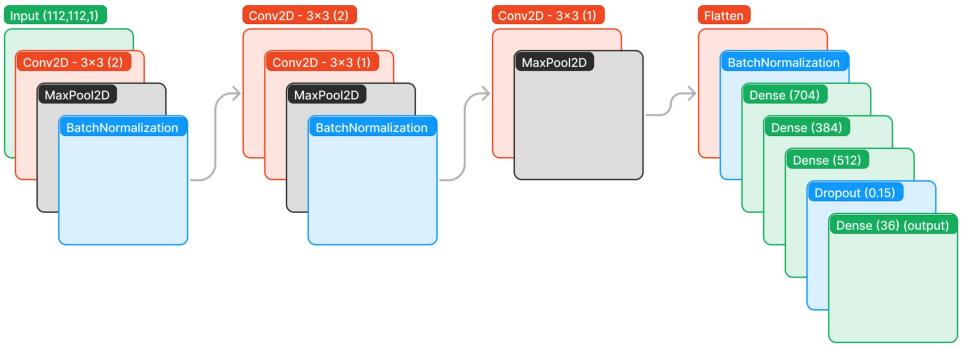

Figure 3: Den slutgiltiga modellen där värdet inom parenteser på faltningsnät representerar faltningskärna.

Med målet att klassifisera ritade tecken effektivt, träffsäkert och träna modellen inom en rimlig tid togs beslutet att bygga modellen för enbart gråskala. Detta innebar enbart en färgkanal istället för tre och däremot betydligt mindre data för modellen att bearbeta. Beslutet grundades i det faktum att datasettet (EMNIST) var gråskala och att användningsområdet inte kräver tre kanaler färg. Det här var någonting som inte hade varit möjligt med förtränade lager då de krävde $3-4$ färgkanaler. Dock skalades träningsdatan upp från $28x28$ till $112x112$ för att träna modellen skulle kunna ta emot mer detaljerad data om sådan skickas eftersom tecken som ska klassificeras kan komma att vara betydligt tunnare skrivna än de i träningssettet (se figur 2).

3.3 Träning & analys

För att träna modellen användes optimeringsmetoden Adam (Adaptive Moment Estimation) då det är en väl beprövad metod som ger snabb konvergens med gott resultat. Andra optimierare så som SGD ihop med Nesterov för mer effektiv konvergering är ett annat potentiellt alternativ, men som troligtvis inte skulle leda till en större förbättring.[13][14] Detta då tester som utförts och dokumenterts i handlingen ”On Empirical Comparisons of Optimizers for Deep Learning” tyder på att Adam och Nesterov presterar relativt lika i slutändan när det kommer till problem som MNIST. Och så EMNIST är mycket likt, är detta ett rimligt beslut.[15] Men ytterligare tester är alltid rekommenderat att utföra.

[13] tf.keras.optimizers.Adam. Tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam (2024-01-13).

[14] tf.keras.optimizers.experimental.SGD. Tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/experimental/SGD (2024-01-13).

[15] Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, George E. Dahl, arXiv. 2019. On Empirical Comparisons of Optimizers for Deep Learning. Cornell University. https://arxiv.org/abs/1910.05446 (2024-01-13).

Ihop med detta nyttjades för mätvärde "accuary" och som förlustfunktion "categorical_crossentropy". Dessa två då "accuary" ger ett tydligt mätvärde på modellen prestanda i vårt fall, och förlustfunktionen är väl passande när det finns flera klasser att välja på. "categorical_crossentropy" kan användas ihop med "softmax"-aktivering i det sista lagret (output) i modellen som modellen enbart ska nämna en potentiell (mest trolig) klass. Men att låta modellen ge skriva ut säkerheten kan även den datan nyttjas i detta fall. Till exempel för att klassificera en "region of interest" (ROI) som inkorrekt ifall ingen klass har en större sannolikhet än 50%.

Vid träning av modellen utfördes 4 träningssessioner med skiftande antal epoker, batch-storlek och inlärningstakt för att uppnå det optimala resultatet (se figur 1). Andra uppsättningar testades, men denna var den snabbaste som nåde det högst nådda resultatet på 92.3% vid test.

Batch-storlek	Epoker	Inlärningstakt
256	32	0.0001
128	32	0.0001
64	64	0.00005
32	128	0.00001

Table 1: Tabell som visar på strukturen av modellens träning.

Under träningen sker de första epoker mycket långsamt och modellen är fast med en "validation accuracy" kring 3%. Dock ökar denna sedan kraftigt upp till kring 90% följt av att träningen fortsätter till nästa steg (se figur 4). Nästa och kommande träningssessioner leder till mindre förbättringar men tillräckligt för att pressa modellen ytterligare 2%. Då modellen nyttjade tidigtstopp (engelska "early stopping") stannade träningen av mycket tidigt samtliga sessioner. Cirka $10 - 20$ epoker in. *Notera att hela träningsflödet finns att se i filen CNN/main.ipynb i GitHub-repositoryn (se sektion 1).*

Då resultatet inte verkade kunna gå högre än ca 92.3% gjordes en utförlig analys för att hitta vilka tecken modellen ej klarade av klassifisera korrekt. För att göra detta på mest effektiva vis nyttjades en "confusion matrix", som är vad det låter som. En matris som visar på var modellen är förvirrad (se figur 5). Figur belyser tydligt några specifika områden som är väntade. Modellen tenderar till exempel att klassifisera båda bokstaven "D" och "O" som siffran "0", samt bokstaven "B" som "R". Även andra tecken blandar den ihop, så som "S" och "5", samt "2" och "Z".

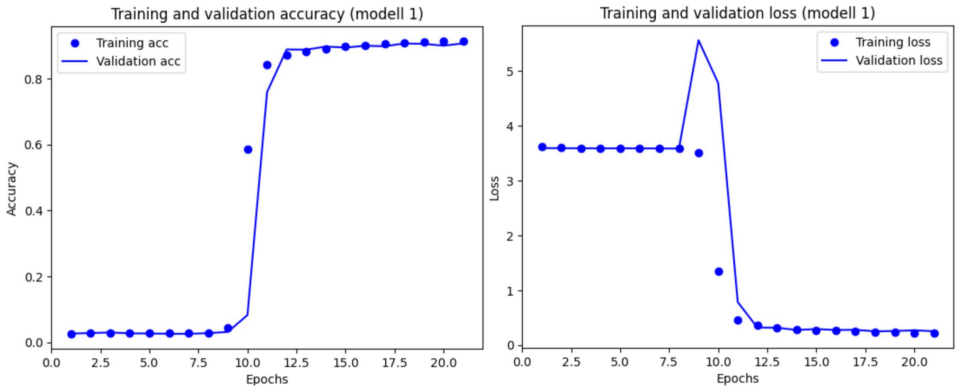

Figure 4: Modellens första träningssession. Notera den plötsliga skillnaden i resultatet och de mycket lika värdena mellan tärning och validering.

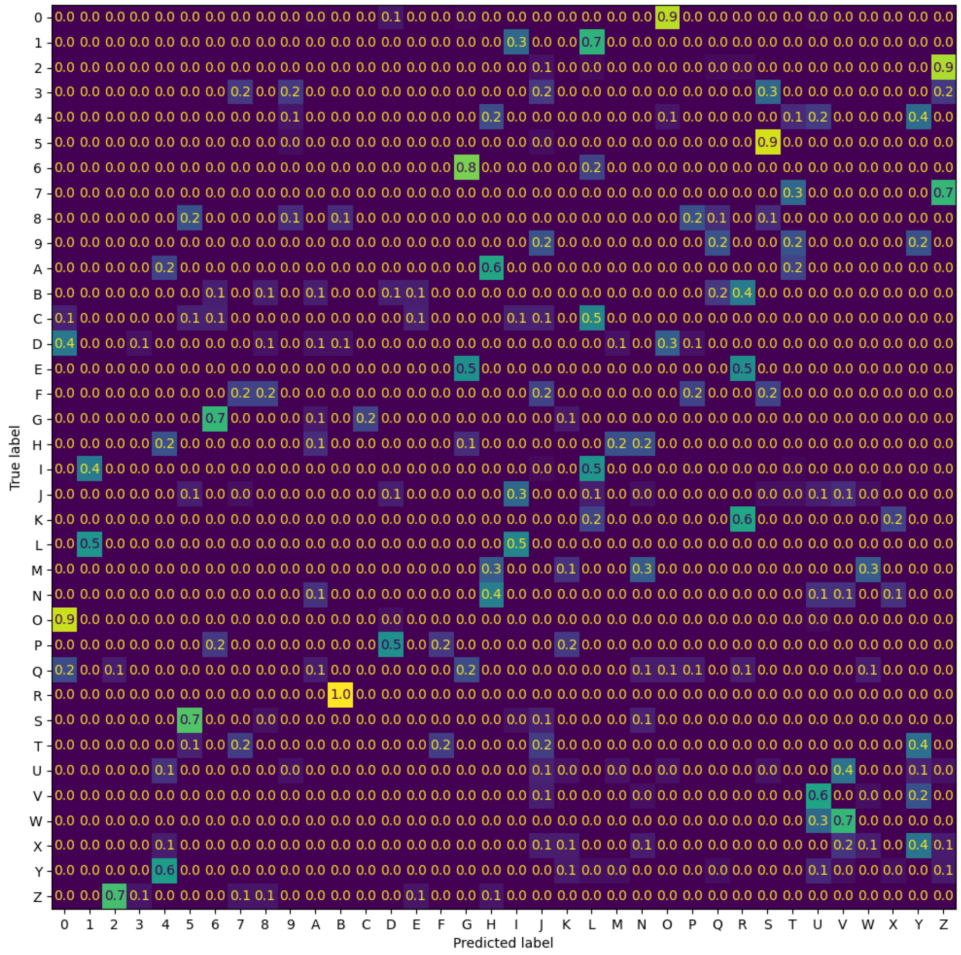

Figure 5: "Confusing matrix" som representerar korrekta och felaktiga gissningar. Mörkare rutor innebär mindre fel, ljusare innebär mer fel. Notera att $[0.0 - 1.0] = [0\% - 100\%]$

Men hur kommer det sig? Genom att tänka efter vilka tecken vi människor tenderar ha svårt att urskilja är dessa felaktiga klassifiseringar helt rimliga. Som människa har vi inte bara tecknet att utgå ifrån, utan även sammanhang. Skulle vi läsa ”1Z345” vet vi att det handlar om siffran ”2” och inte bokstaven ”Z”. För att bevisa att så är fallet (anledningen till den låga träffsäkerheten) kan vi i figur 6 se hur 3 exempel har skrivits ut. Vi ser att de mest typiska felen vi kan tänka oss åtminstone är nära det korrekta.

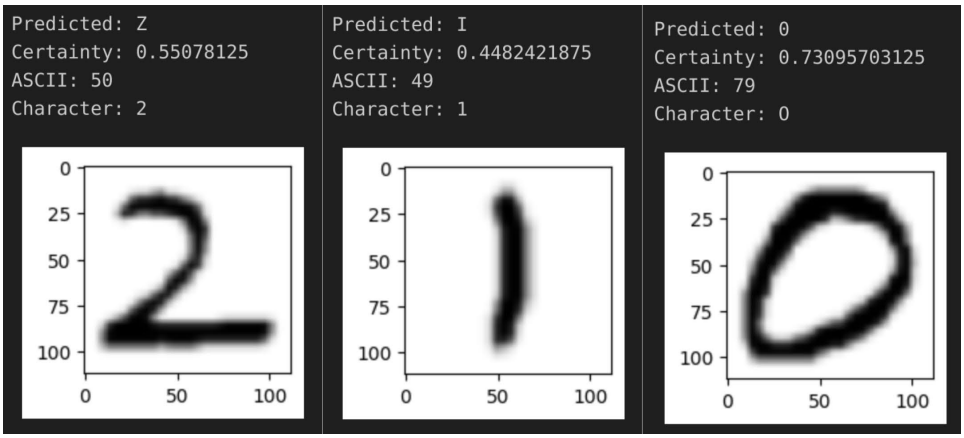

Figure 6: Tecken förkommer i datasettet så otydligt skriva att en människa inte kan klassifisera dem korret. Ännu mindre en simpel AI modell.

4 Program för dataflöde

För att kunna testa principen med ritade tangenter (exklusive spåring av fingrar) behövs ett stadigt dataflöde i form av bilddata. Mer exakt ”region of intereset” som behandlats att likna träningsdatan EMNIST (se fighure 1 och 2). Det stadiga dataflödet skapades genom ett enkelt program (se figur 7) som nyttjade OpenCV för att applicera filter och hitta ROI. De filter som nyttjades har redan nämnts, men ROI kan vara värt att belysa ytterligare.

Det första steget var att låta OpenCV finna hörnen av varje genom kant- och konturdetektion (“The Suzuki and Abe algorithm”). Sedan orienterades dessa korrekt för att kunna användas för att justera perspektivet och skapa en bild som kameran hade varit precis ovanför pappret och inte filmat med en cirka 30 vinkel. Detta ihop med nya höjden som hittades via formeln $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} * (1 + cos(X))$ (där X är kamerans vinkel mätt från plan yta) för att kompensera för bildens förvrängning.

När bildperspektivet är justerat klipps kanterana ner med en pixel åt gången tills kanterna är näst intill helt vita. Detta är en aggresiv metod, men som fungerar sålänge kanterna är ungefär lika breda. Sedan kapas bilden steg för steg från varje håll tills enbart symbolen i mitten är kvar. Nu när symbolen är fri placeras den på en ny canvas med en padding på

20% åt varje håll för att likna MNIST datan.[16] Sist skalas datan sedan till rätt storlek ($112x112$ pixlar) och skickas till CNN-modellen som returnerar klassifiseringen. Klassifiseringen skrivs ut i programmet, både som lista och tillsammans med ROI i videoflödes-rutan.

Under processen sker en rad kontroller för att se till att ROI faktiskt inte saknar data av intresse eller är för liten. Detta kombinerat med justerbara parametrar, sökning i intervall, och matematik för att förhindra dubbletter och falska ROI leder till ett stabilt flöde.

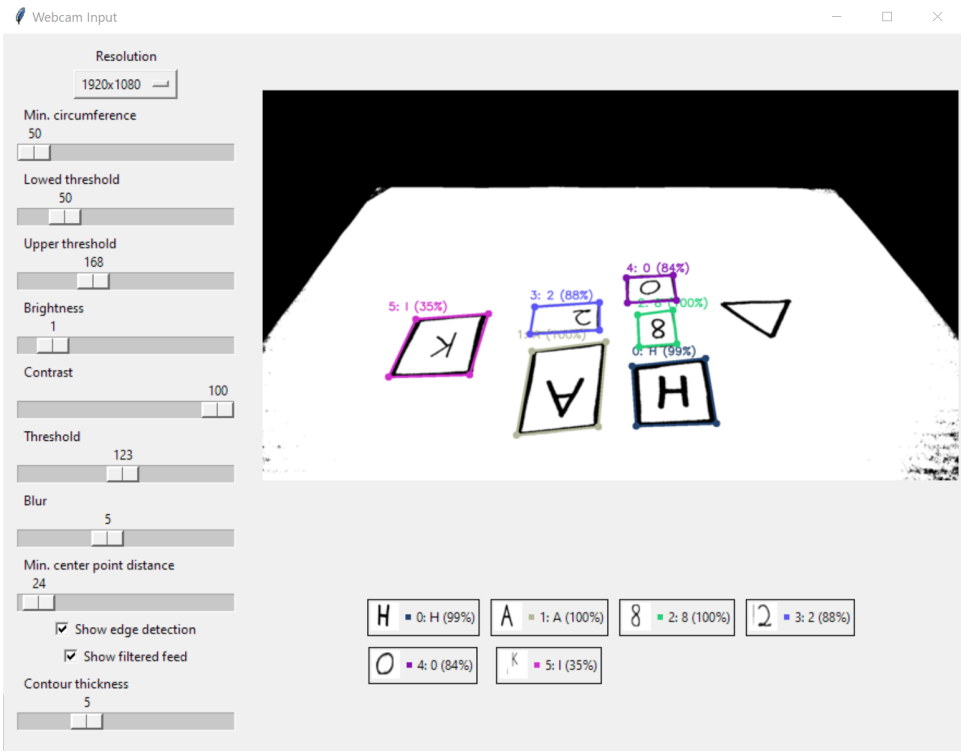

Figure 7: Till vänster finns de parametrar som går att justera som användare, varav de flesta i många fall har mycket lika värden. Till höger är videoflödet som kan vissas med och utan bilder samt utkskriven data. Sist har finns en lista nere till höger som visar alla funna tecken. Notera hur Bokstaven "K" har klassifiserats fel.

Vi ser i figur 7 att den hanterar sin uppgift relativt bra. Men plats för förbättring finns då bokstaven "K" har klippts ut fel. Ett sådant problem kan åtgärdas genom justerad logik för att välja de "bästa" utklippta bilderna samt nyttja CNN-modellens klassificerings-säkerhet som en indikation på att

[16]Rukshan, Pramoditha, Data Science 365. 2022. Acquire, Understand and Prepare the MNIST Dataset. medium. https://medium.com/data-science-365/acquire-understand-and-prepare-the-mnist-dataset-3d71a84e07e7 (2024-01-13).

någonting är inkorrekt och behöver justeras.

I GitHub-repositoryn (se sista stycken under sektion 1) finns en inspelning som visar ett demo under "/Recordings/test_t.mp4".

5 Resultat

Arbetet och testerna som har utförs ihop med denna rapport tyder på att denna typ av inmatning är möjlig att skapa och har potential. Att använda en CNN för att läsa av ritade tangenter ihop med ROI är absolut möjliget och hållbart. De hinder som kvarstår är däremot att lösa symboler som är mycket lika andra (se figur 6). Någonting som potentiellt skulle kunna lösas genom att gruppera siffror och bokstäver separat, eller genom riktlinjer för hur tecken ska ritas (siffran "0" kan t.ex ha ett streck igenom sig eller under).

Den balanserade versionen av EMNIST har visat sig duga gott för att uppnå en tillräckligt hög träffsäkerhet om de tecken som ritar är tydliga nog, men håller inte i längen just på grund av de lika symbolerna. Ett alternativ är att ersätta den otydliga datan i EMNIST för att få en bättre modell med högre träffsäkerhet. Dessutom skulle en kontroll av EMNIST behöva utföras då viss data är rentav oanvändbar (extremt otydlig eller inkorrekt).

Modellen som valdes blev en simpla modell väl passande att tränas på långsammare hårdvara med en sammanlagd träningstid på cirka 60 minuter men med en god träffsäkerhet för den korta tiden. En modell som kan klassificera 36 olika tecken men med hinder av lika symboler.

6 Diskussion

Modellens låga träffsäkerhet på 92.3% kan bero på många faktorer, så som litet dataset och simpel modell. Men det som troligtvis har lett till den största förlusten är datan som är, även för oss människor, förvirrande. Under sektion 3.3 diskuteras bland annat tecken som är lika andra, men inte kvaliten på EMNIST. I figur 8 syns till vänster och höger två symboler som inte ens är rimliga. Samt i mitten har vi en bild på talet "69" som är klassifiserat som siffran "0". Problem som dessa i ett dataset har en kraftig påverkan på modellens utkomst. För att undvika detta skulle data med med unika tecken kunna ha använts. Till exempel data där alla bokstäver skrivs med mer detalj.

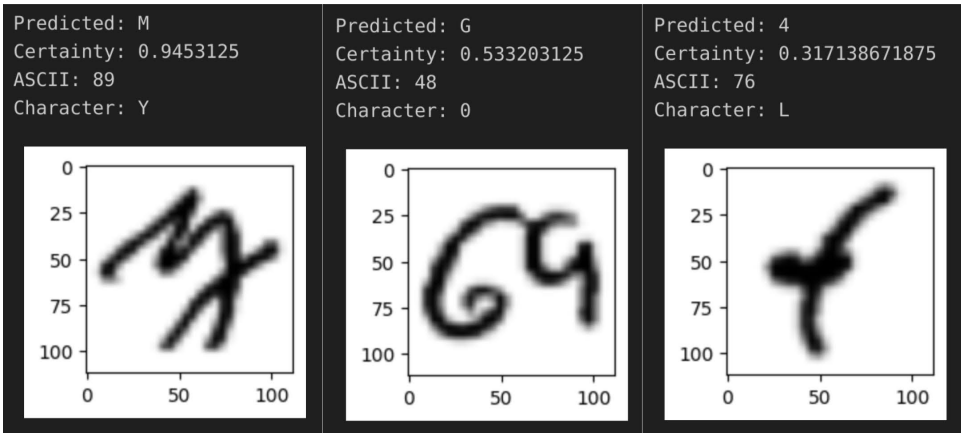

Figure 8: Handstilar så otydliga att datan ej är brukbar, samt förekommande inkorrekt data i samlingen.

En bättre metod för att lösa samma problem som denna uppsättning hade varit att ta fram en mer komplex modell av typen RCNN (riktig), eller varför inte till och med Fast RCNN, Faster RCNN eller YOLO (You Only Look Once). För att kunna göra detta hade annan data behövts. Data i form av bilder med flera symboler och regioner placerade ihop med regionsdata som potentiellt hade kunnat genereras genom program för att kunna experimentera med principen. Program för detta experimenteras redan med och exempel finns publicerade på GitHub.[17] Att träna en model för det här hade däremot krävt en större mängd datakraft och hade troligtvis varit utom vad det fanns resurser för vid skapandet av CNN-modellen som nämnt i rapporten.

[17] 2021. MNIST Object Detection dataset. GitHub. https://github.com/hukkelas/MNIST-ObjectDetection (2024-01-11).