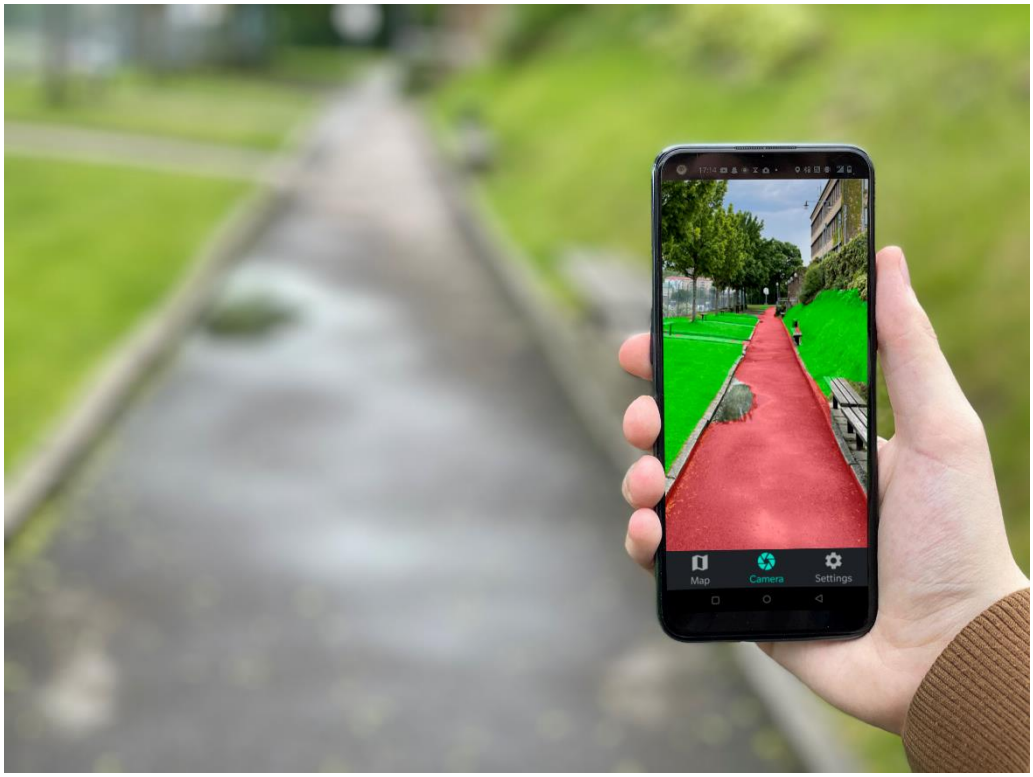
CHALMERS



Navigeringssystem för personer med synnedsättning med hjälp av datorseende, GPS och Text-till-tal

Implementerat för Android mobiltelefoner

Examensarbete inom Data- och Informationsteknik

Philip Axenhamn
Andreas Greppe

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

EXAMENSARBETE

Navigeringssystem för personer med synnedsättning med hjälp av datorseende, GPS och Text-till-tal

Implementerat för Android mobiltelefoner

Philip Axenhamn
Andreas Greppe

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2021

Navigeringssystem för personer med synnedsättning med hjälp av datorseende, GPS och Text-till-tal
Implementerat för Android mobiltelefoner

Philip Axenhamn
Andrea Greppe

Examinator: Arne Linde

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag:
Bild på segmenteringen i applikationen.

Institutionen för Data- och Informationsteknik
Göteborg 2021

SAMMANFATTNING

Utomhusnavigering i okända miljöer kan vara besvärlig för personer med synnedsättning. Hjälpmedel som exempelvis ledarhund, supportpersonal samt käpp har begränsningar och därför finns ett behov av alternativa lösningar. I detta projekt var därför syftet att skapa en applikation som kombinerar semantisk bildsegmentering och GPS i ett fullständigt navigeringssystem. Målet är att synnedsatta ska kunna navigera utomhus med detta system. Metoden gick dels ut på att utveckla en semantisk bildsegmenteringsmodell utifrån en egenskapad datamängd. Denna modell används sedan för att ta fram information om omgivningen. Ett GPS-system implementerades för att skapa vägbeskrivningar. Båda delarna kombinerades till ett gemensamt system som kan användas för navigering utomhus. Återkoppling sker med text-till-tal genom att omvandla informationen från båda delarna till tal. Den slutgiltiga applikationen blev en prototyp av navigeringssystemet som var tillräcklig för testning och evalueringen. Den semantiska segmenteringsmodellen uppnådde en precision på 89 % för valideringsmängden. Det blev tydligt att ett system som bygger på bildsegmentering och GPS skulle vara tillräckligt för att implementera ett praktiskt navigeringssystem för personer med synnedsättning. Men att en utökad datamängd för bild segmenteringen skulle behövas samt vidareutveckling av implementationen.

Nyckelord: utomhusnavigering, semantisk bildsegmentering, GPS, text-till-tal.

ABSTRACT

Outdoor navigation in an unfamiliar environment can be difficult for people with visual impairments. Assistive devices such as guide dogs, support staff and canes have limitations and therefore there is a need for alternative solutions. In this project, the aim was to create an application that combines semantic image segmentation and GPS in a complete navigation system. The goal is for the visually impaired to be able to navigate outdoors with this system. The method was based on developing a semantic image segmentation model based on a self-created dataset. This model is then used to produce information about the user's surroundings. A GSP system was implemented to produce navigation directions. Both parts were then combined into a complete system that can be used for outdoor navigation. Feedback is done with text-to-speech by converting the information from both subsystems into speech. The final application developed was a prototype of the navigation system which was sufficient for testing and evaluation. The semantic segmentation model achieved an accuracy of 89 % for the validation set. It became clear during the project that a system based on image segmentation and GPS would be sufficient to implement a practical navigation system for the visually impaired. However, an increased amount of data for image segmentation would be needed as well as further development of the implementation.

Keywords: outdoor navigation, semantic image segmentation, GPS, text-to-speech.

FÖRORD

Denna rapport är vårt examensarbete på Chalmers tekniska högskola i Göteborg. Examensarbetet omfattar 15 högskolepoäng och har genomförts av Philip Axenhamn och Andreas Greppe under vårterminen 2021. Under arbetet har Sakib Sistek varit vår handlare. Han har hjälpt oss med att forma idéer samt givit oss råd och återkoppling. Vi vill därför tacka Sakib för hans stöd under detta arbete.

Terminologi och förkortningar

Aktiveringsfunktion	(eng. *Activation function*) Matematisk funktion som används inom det dolda lagret i ett artificiellt neuronnät för att aktivera en nod.
Algoritm	En algoritm inom maskininlärning är en procedur som bearbetar data för att skapa en modell.
Artificiell intelligens (AI)	(eng. *Artificial intelligence*) Förmågan hos dataprogram att efterlikna mänsklig intelligens.
Artificiella neuronnät (ANN)	(eng. *Artificial neural network*) En struktur baserad på människans hjärna som består av ett antal noder och lager.
Datorseende	(eng. *Computer vision*) Bearbetning och analys av bilder.
Djupinlärning (DL)	(eng. *Deeplearning*) En subkategori av maskininlärning där träning av en modell sker i flera lager.
Faltningsnätverk	(eng. *Convolutional neural network, CNN*) Ett artificiellt neuronnät som används inom bildigenkänning och består av flera faltningslager.
Filter	Ett filter i ett faltninglager är en matris med multiplikatorer som appliceras på pixlarna i en bild för att framhäva kännetecken.
Förlust	Inom artificiella neuronnät är förlusten ett värde som beskriver hur nära det uppskattade värdet från modellen är det faktiska värdet.
Förlustfunktion	(eng. *Loss function*) En funktion som avgör hur förlustvärdet för en modell beräknas.
Inlärningshastighet	(eng. *Learning rate*) Hastigheten som modell anpassar sig efter data. Kan även beskrivas som steglängd som tas i stokastisk gradientnedstigning.
Interpretator	(eng. *Interpreter*) En funktion som används för att tolka TensorFlow Lite modeller.
Kanaler	(eng. *Channels*) Inom bildbehandling är kanaler ett värde som beskriver vilka färger bilden hanterar.
Mask	Används för att markera var varje klass befinner sig i en bild.
Maskininlärning (ML)	(eng. *Machine learning*) En subkategori till artificiell intelligens där en modell tränas för att sedan lösa ett problem.
Modell	En modell inom maskininlärning är det som fås ut av träningen från en algoritm.
Semantisk bildsegmentering	(eng. *Semantic image segmentation*) En teknik som innebär att man för varje pixel i bilden, anger vilken klass pixeln tillhör.
Skip-koppling	(eng. *Skip-Connection*) Koppling mellan nedskalnings- och uppskalningsdel i ett artificiellt neuronnät.
Sobel-kärna	En matris som används för att framhäva horisontella eller vertikala linjer i en bild.
Stokastisk gradientnedstigning	(eng. *Stochastic gradient descent, SGD*) En metod som används för att minimera förlustfunktionen.
Text-till-tal (TTS)	(eng. *Text-to-speech*) Omvandlar text till muntligt tal.
Underanpassning	(eng. *Underfitting*) När en modell inte anpassats tillräckligt efter träningsmängden.
Överanpassning	(eng. *Overfitting*) När en modell anpassats för mycket efter träningsmängden.

Bilder och figurer

Alla bilder och figurer är skapade av författarna.

Innehållsförteckning

1
Inledning

I Sverige finns ungefär 100 000 synskadade inskrivna i syncentralen. Dessa personer har ett behov av olika hjälpmedel som exempelvis ledarhund och käpp för att kunna navigera i sin vardag [1]. Att utbilda en ledarhund kostar omkring 200 000-350 000 kr [2] och i många fall är det lång väntetid för synskadade att införskaffa sig en ledarhund. Dessutom förekommer det att personer med synnedsättning är allergiska mot pälsdjur och kan därför inte använda ledarhund som hjälpmedel. Ofta används käpp tillsammans med ledarhund för att kunna upptäcka hinder på vägen och avläsa markeringar. I miljöer där markeringar saknas, kan navigeringen bli besvärlig och därför finns det behov av andra lösningar. Ett potentiellt verktyg för en sådan lösning är användning av en applikation i mobiltelefonen. Idag finns stödfunktioner som hjälper synnedsatta att använda mobiltelefoner. I Android finns exempelvis funktionaliteten kallad *"talkback"* som gör att alla knappar och objekt på skärmen kan läsas upp genom ett tryck och aktiveras genom dubbeltryck [3]. Detta gör det enkelt för personer med synnedsättning att använda applikationer och navigera på Android mobiltelefoner. Mobiltelefoner har blivit tillräckligt kraftfulla för att använda tekniker inom datorseende och artificiell intelligens. I detta examensarbete vill vi därför skapa en applikation för Android-mobiltelefoner som kan underlätta vardagen för synskadade med hjälp av tekniker inom datorseende, text-till-tal (TTS) och GPS.

1.1 Syfte

Syftet med detta arbete är att skapa en applikation för Android-mobiltelefoner som hjälper synskadade att navigera utomhus. Genom att använda metoder inom datorseende och telefonens inbyggda GPS, ska användaren få relevant återkoppling genom text-till-tal. Denna information skall underlätta navigering mellan en start- och slutpunkt.

1.2 Mål

Huvudmålet i detta arbete är att utveckla en applikation som stödjer utomhusnavigering för personer med synnedsättning. Detta mål kan delas upp i följande delmål:

- Utveckla en applikation som implementerar GPS-navigering och ger återkoppling till användaren i form av text-till-tal.
- Skapa en bildsegmenteringsmodell utifrån en egen datamängd och evaluera modellen i praktisk användning.
- Implementera funktionalitet för att tolka utdata från modellen i applikationen.
- Kombinera bildsegmentering och GPS-navigeringen för att skapa ett navigeringssystem som ger återkoppling från båda system.
- Skapa ett enkelt GUI som är anpassat för personer med synnedsättning och som är kompatibelt med *”talkback”*.
- Testa hur det sammankopplade navigeringssystemet fungerar i praktiken.

1.3 Frågeställning

Frågeställningen som undersöks i detta arbete är hur artificiell intelligens kan kombineras med traditionella navigeringsverktyg som exempelvis GPS för att hjälpa personer med synnedsättning att navigera i utomhusmiljöer.

1.4 Avgränsningar

- Applikationen kommer vara en prototyp eftersom resurser och tid för att utveckla en fullständig produkt saknas.
- Applikationen kommer endast fungera på telefoner som stödjer Android.
- Arbetet kommer endast innefatta mjukvaruutveckling, ingen hårdvara utvecklas.
- Applikationen kommer enbart vara på engelska.
- Bildsegmenteringsmodellen kommer enbart tränas på egen framtagen datamängd.
- Applikationen kommer inte kunna upptäcka hinder på vägen.
- Endast kameran samt GPS kommer utnyttjas från mobiltelefonen eftersom många mobiltelefoner saknar mer avancerade sensorer som exempelvis LiDAR.

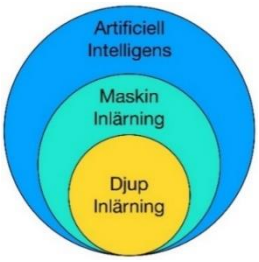
2
Teknisk bakgrund

I detta avsnitt beskrivs viktiga begrepp och tekniker som används inom arbetet. Denna del täcker bland annat: artificiell intelligens, applikationsutveckling i Android, bildanalys samt information om tidigare forskning.

2.1 Artificiell intelligens, Maskininlärning och Djupinlärning

Artificiell intelligens (AI) är en term inom datavetenskapen där datorprogram utför uppgifter som förknippas med intelligent beteende, såsom förmågan att resonera, dra slutsatser och lära sig från tidigare erfarenheter. Några exempel där AI har en betydande roll är inom sjukvården, bilindustrin och energi- och tillverkningsprocesser. Termen innefattar flera subkategorier och i detta avsnitt kommer maskininlärning och djupinlärning beskrivas på en grundläggande nivå.

Maskininlärning är en subkategori av AI, se figur 1, där algoritmer tränas för att hitta mönster och kännetecken från en datamängd. Den färdigtränade algoritmen, även kallad modell, kan efter träning användas för att genomföra estimering på annan data än vad modellen är tränad på. För att modellen ska kunna förbättras inom tränings-processen krävs en rekursiv mekanism. Detta innebär att modellens prestation måste evalueras för att sedan kunna användas för fortsatta träningen [4].



Figur 1. Subkategorierna av Artificiell intelligens

Maskininlärning kan ytterligare delas in i tre kategorier: väglett lärande (eng. *Supervised Learning*), icke-väglett lärande (eng. *Unsupervised Learning*) och förstärkningslärande (eng. *Reinforcement Learning*). I detta arbete utnyttjades enbart väglett lärande som innebär att modellen tränas med märkt data. Varje datapunkt är kopplad med en markering (eng. *label*) som kan ses som "det korrekta svaret". En modell inom väglett lärande kommer leta efter mönster i datan som korresponderar med markeringen. Efter träningen kan modellen estimera vilken markering som tillhör indatan. Skillnaden mellan estimeringen och det korrekta svaret kan under träningen användas för att förbättra modellen [5].

Ofta delas datamängden upp i två olika delar: träningsmängd och valideringsmängd. Träningsmängden innehåller data som algoritmen tränas med och valideringsmängden används för att evaluera träningen. Genom att jämföra det estimerade värdet från modellen med data från valideringsmängden, är det möjligt att anpassa modellen så den presterar bättre [4]. Träningsdatan brukar vara omkring 80 % av hela datamängden. Valideringsmängden består av den resterande delen. Anledningen till att träningsmängden är större än valideringsmängden är för att mer träningsdata ofta ger upphov till bättre presterande modeller med avseende på precision i förutsägelserna.

Djupinlärning är en subkategori av maskininlärning som tillåter modellering av mer komplexa problem som exempelvis bildigenkänning. Till skillnad från maskininlärning, utnyttjar djupinlärning en nätverksstruktur för att bestämma vilka kännetecken som skall tas fram från datan. I maskin-inlärning är extraktionen av kännetecken handgjord men i djupinlärning anpassas den under träningen och kan fortsätta gå djupare i fler extraktioner. Ofta är djupinlärning mer komplex vilket medför att

träningen kan ta längre tid. För att påskynda träningen, är det därför vanligt att utnyttja en GPU. I djupinlärning krävs även mer data för att modellen ska kunna ge ett pålitligt resultat [6].

2.2 Artificiella neuronnät

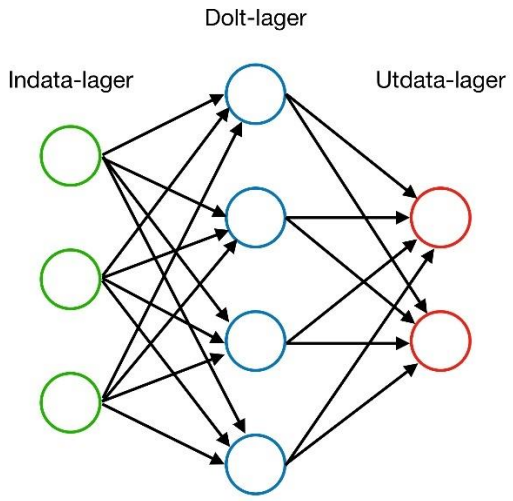
Artificiella neuronnät (ANN) är en del av ett datasystem som är designat att efterlikna biologiska neuronnät. Nätverket består av ett antal noder som är förgrenade med varandra i flera skikt, även kallat lager. Likt människans hjärna, triggas varje nod av en signal som medför att andra noder i nätverket aktiveras. I ett ANN har noder inom ett lager endast kopplingar till noder i närliggande lager. Fallen där alla noder mellan två lager är kopplade, kallas för fullt kopplade lager (eng. *fully connected layer*) [7].

2.2.1 Strukturen av ett ANN

Ett artificiellt neuronnät kan delas upp i 3 olika delar:

- Indata-lager
- Dolt-lager
- Utdata-lager

I figur 2 nedan, visas en generell uppbyggnad av ett artificiellt neuronnät i form av en riktad graf där noderna är i färgerna grön, blå och röd. Det första lagret kallas för indata-lagret. Storleken och utseendet av detta lager är beroende på vilken typ av data som nätverket tränas med. Det nästa lagret kallas för det dolda lagret som befinner sig mellan indata-lagret och utdata-lagret. I detta lager sker omvandlingar av indatan genom att tillämpa vikter och använda en aktiverings-funktion som utgång. Slutligen har varje nätverk ett utdata-lager som förutsägelserna kan tolkas från. Precis som indata-lagret, beror strukturen här på vilken typ av data som modellen ska producera. I ett klassificerings-problem, brukar utdatan vara ett värde mellan 0-1 för varje klass beroende på vald aktiveringsfunktion.



Figur 2. Ett artificiellt neuronnät med 3 noder i indata-lagret, 4 noder i det dolda-lagret och 2 noder i utdata-lagret.

2.2.2 Aktiveringsfunktion

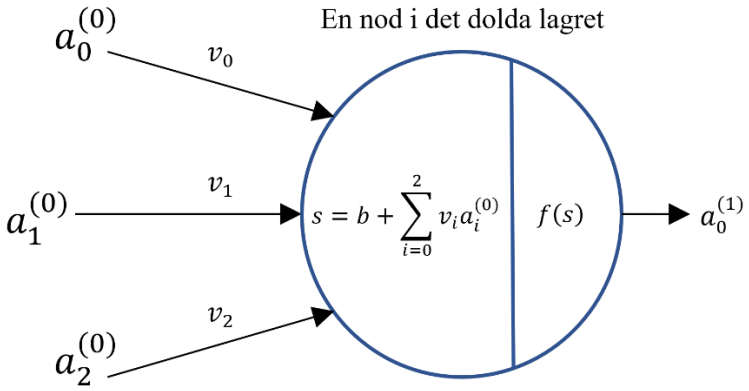
En aktiveringsfunktion (eng. *activation function*) är en matematisk funktion som transformerar indatan från en nod till en mer användbar representation av datan. Indatan för en nod är en summering av vikterna som förs vidare till aktiveringsfunktionen som i sin tur ger ett värde mellan en övre- och lägre gräns [8]. Det finns många olika typer av aktiveringsfunktioner och valet av aktiveringsfunktion beror främst på vad nätverket ska användas till och vad det är för typ av problem som ska lösas. I detta arbete användes följande 2 olika typer av aktiveringsfunktioner: *Relu* och *Softmax*.

Aktiveringsfunktionen Relu, även kallat *Rectified Linear Unit*, är en linjär aktiveringsfunktion som används inom de dolda lagarna i nätverket. Denna funktion omvandlar negativa värden till 0 och ger samma värde som det inmatade värdet om värdet är positivt [8].

Vid klassificering av flera klasser används aktiveringsfunktionen Softmax. Funktionen returnerar ett värde mellan 0-1 för varje klass där den totala summan av värdena för alla klasser blir 1 [8].

2.2.3 Vad är en nod?

I ett artificiellt neuronnät består en nod av en summering av vikter från föregående lager, en bias samt en aktiveringsfunktion. En nod i det dolda lagret kan visualiseras enligt figur 3 nedan.



Figur 3. En nod i det dolda lagret som tar in data från 3 noder i föregående lager.

Utdatan från alla noder i föregående lager som är förgrenade med noden i det dolda lagret, multipliceras med dess korresponderande vikt. I figur 3 har noden en koppling med 3 andra noder i föregående lager och utdatan från dessa representeras av: $a_0^{(0)}, a_1^{(0)}, a_2^{(0)}$. Vikterna för respektive nod är: v_0, v_1, v_2. En summering av alla produkter beräknas och en bias, b, adderas därefter. Det sista steget är att omvandla datan genom en aktiveringsfunktion, f, och utdatan, $a_0^{(1)}$, förs sedan vidare till en nod i nästa lager.

Vikterna har en avgörande roll i hur snabbt arkiveringsfunktionen triggas medan bias används för att antingen påskynda eller fördröja aktiveringen [7]. En jämförelse kan göras med räta linjens ekvation där lutningen motsvarar vikten och skärningen i y-axeln motsvarar bias.

2.2.4 Förlustfunktion

En förlustfunktion (eng. *loss function*) är en funktion som används för att evaluera träningen för ett artificiellt neuronnät. Värdet från funktionen representerar hur väl modellen anpassar sig efter datan. Ett lågt förlustvärde indikerar på att modellen ger en förutsägelse med hög träffsäkerhet under träningen. Målet med att träna ett nätverk blir därför att minimera värdet av förlustfunktionen under träningsprocessen. Efter varje epok kommer värdet av funktionen beräknas baserat på en jämförelse med det förutsagda värdet och det faktiska värdet. Det finns flera metoder för att beräkna detta värde och vilken metod som används beror på användningsområdet av nätverket [8].

Förlustfunktionen *cross entropy* kan användas för att mäta prestandan av modeller som använder probabiliteten mellan 0-1 som utdata. För klassificeringsproblem som hanterar fler än två klasser, används en version av cross entropy kallad *categorical cross entropy* [9].

En annan förlustfunktion som är vanligt förekommande inom segmenteringsproblem är *Dice-förlust* (eng. *Dice-loss*). Denna funktion är bättre anpassad till de pixel-exakta estimeringarna som segmentering innebär. *Dice koefficienten* är ett tal mellan 0-1 och beräknas enligt formeln nedan:

$$Dice\ koefficient = \frac{2|A \cap B|}{|A| + |B|}$$

Funktionen beräknar snittet mellan masken från modellens förutsägelse (A) och den korrekta masken (B), multiplicerar det med 2 och delar täljaren med summan av antalet pixlar för båda maskerna. Värdet som fås av denna funktion vill man maximera eftersom ett högre värde indikerar att masken från modellen är lik masken från träningsdatan. Dice-koefficienten kan sedan användas som förlustfunktion [10]. Viktigt att notera är att ju högre Dice-koefficient, desto bättre är modellen på att estimera maskerna. Förlustfunktionen kan därför beräknas enligt följande:

$$Dice\ förlust = 1 - Dice\ koefficient$$

Denna förlustfunktion kan sedan användas för att evaluera träningen av ett artificiellt neuronnät i bildsegmenterings-ändamål.

2.2.5 Stokastisk gradientnedstigning – Optimeringsfunktioner

En viktig del i hur en modell ”lär” inom djupinlärning är det underliggande optimeringsproblemet som kan definieras genom en förlustfunktion. Detta förlustvärde används för att evaluera hur bra inlärningsprocessen för modellen fungerar. Målet med optimeringen blir därför att uppnå ett så lågt förlustvärde som möjligt. Den grundläggande algoritmen för att optimera detta värde är stokastisk gradientnedstigning (eng. *Stochastic gradient descent, SGD*) som bygger på att hitta den minsta gradienten av en funktion för att hitta minimipunkten. Genom denna minimipunkt kan det minimala värdet hittas [4].

I efterhand har det framkommit förbättringar av SGD som presterar bättre än den grundläggande SGD funktionen. En av dessa kallas för *Adam* (*Adaptive Moment Estimation*). Denna metod bygger vidare på 2 tidigare optimeringsfunktioner, *rmsprop* och *momentum* [11]. Funktionen Adam är känd för att fungera väl på en stor utsträckning av applikationer inom djupinlärning och är anledning till att den används i detta arbete.

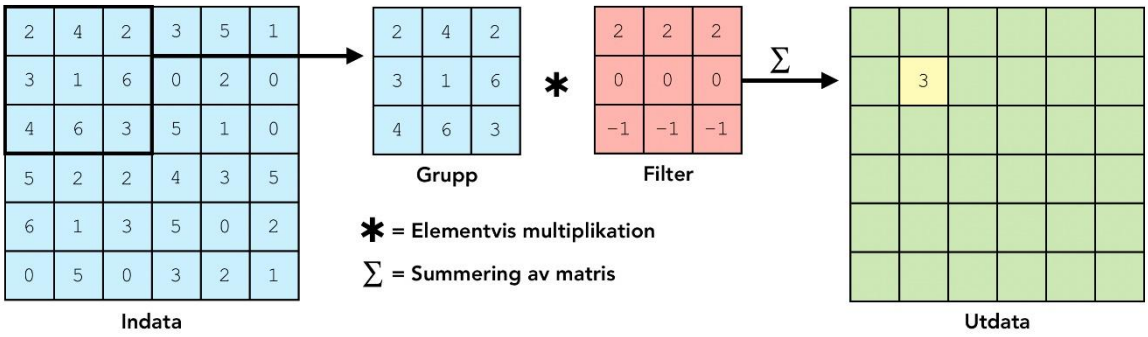
Optimeringsfunktioner utnyttjar en parameter kallad inlärningshastighet som bestämmer med vilken hastighet modellen anpassas efter indatan. Inlärningshastigheten minskas av optimeringsfunktionen under träningen. Trots detta har den grundläggande inlärningshastigheten stor påverkan på träningen. Ett för högt eller lågt värde kan resultera i överanpassning respektive underanpassning och utvecklaren av nätverket måste ta fram ett värde som är anpassad till modellen [4].

2.2.6 Regularisering

Regularisering är en teknik som används för att motarbeta vad som kallas för över- och underanpassning inom djupinlärning. Överanpassning är när modellen anpassar sig för mycket efter träningsdatan vilket leder till att modellen presterar sämre när nya data introduceras. Underanpassning är när modellen inte anpassas tillräckligt efter träningsdatan. För att träna en modell som ger pålitliga estimeringar, krävs en balans i modellens anpassning till datan. Det finns flera metoder för att implementera regularisering i ett artificiellt neuronnät. Två av dessa metoder är *Dropout* och *Early-stopping*. Metoden Dropout bygger på att en bestämd mängd av datan i nätverket nollställs. Detta hjälper modellen att undvika för hög anpassning efter träningsdatan. Early-stopping innebär att träningen slutar tidigt när överanpassning inträffar [4]. Detta kan upptäckas genom att förlusten från valideringsmängden ökar, trots att förlusten av träningsmängden minskar.

2.2.7 Faltningslager

Faltningsnätverk (eng. *convolutional network*) bygger på flera faltningslager (eng. *convolutional layer*) som filtrerar olika delar/detaljer av en bild. När ett faltningslager skapas, specificeras hur många av dessa filter som skall användas. Antalet filter är ofta ett tal med bas 2 exempelvis 16, 32 eller 64 och brukar öka djupare in i nätverket [7]. Ett filter är en matris bestående av tal som multipliceras med varje pixel-värde i gruppen. I figur 4 nedan visas ett exempel på ett filter med storleken 3x3 (röd matris) som appliceras på en grupp av pixlar med samma storlek (blå matris) från indatan.

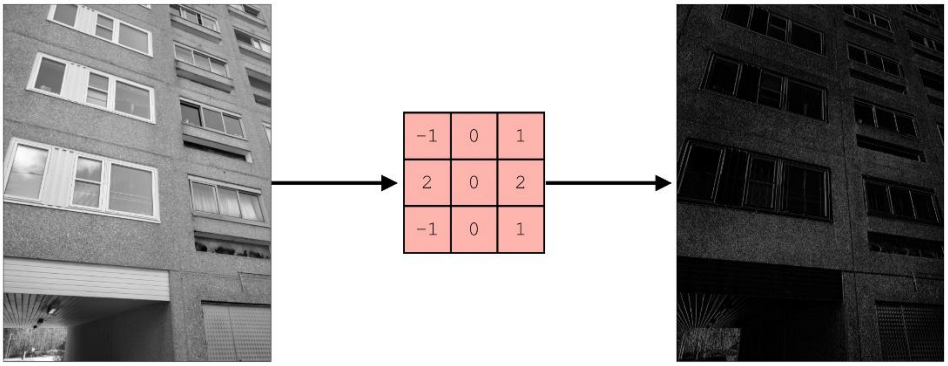


Figur 4. Exempel som visar hur ett filter appliceras på en grupp pixlar från indatan. Notera att endast pixeln med värdet 1 från gruppen får ett nytt värde. I detta exempel blev det nya värdet 3 som visas i matrisen för utdata (grön matris).

Varje tal i filtret multipliceras med värdet som är på samma plats i matrisen från bilden. Därefter summeras alla värden och detta blir det nya värdet för pixeln. Från exemplet i figur 4 ovan, beräknas det nya pixelvärdet enligt följande:

$$\begin{aligned} Nytt\ pixelvärde &= 2*2 + 4*2 + 2*2 \\ &+ 3*0 + 1*0 + 6*0 \\ &+ 4*-1 + 6*-1 + 3*-1 = 3 \end{aligned}$$

Efter att alla pixlar har genomgått samma procedur kommer detaljer visas tydligare i den resulterande bilden, även kallad *feature-map*. Ett exempel på detta visas i figur 5 nedan:



Figur 5. Pixlarna i bilden till vänster filtreras med filtret. Detta resulterar i att vertikala linjer framhävs i bilden. Resultatet kan ses till höger i figuren.

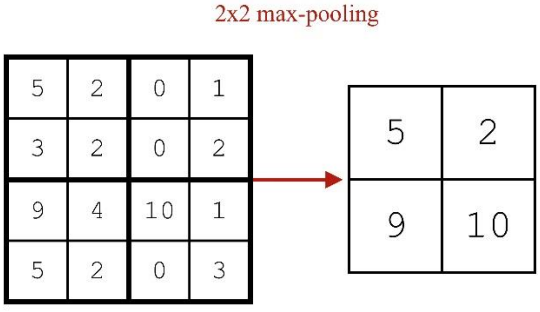
Filtret i detta exempel gör att vertikala linjer i bilden framstår extra tydligt. Hela proceduren med att gå igenom alla pixlar och applicera ett filter på varje pixel kallas för särdragsextraktion (eng. *feature extraction*) [7]. Antalet filter och faltningslager ett nätverk har, skiljer sig och beror på vad modellen ska användas till.

2.2.8 Transponerat faltningslager

Det finns flera metoder för att implementera uppskalning i ett artificiellt neuronnät. En av dem bättre presterande metoderna för detta är genom användning av ett transponerat faltningslager. Ett transponerat faltningslager definieras bland annat av storleken på kärnan, antal filter, steg (eng. *strides*) och fyllnad (eng. *padding*). I denna typ av faltningslager används steg för att förstora bilderna från särdragsextraktionen [12].

2.2.9 Pooling

För att kontrollera storleken på faltningsnätverk samt förtydliga delar av bilden, används en teknik som kallas för *pooling*. Det finns flera typer av denna teknik, exempelvis *min-pooling* och *max-pooling*. Denna operation kan appliceras på matriser med olika dimensioner, exempelvis 2D eller 3D. Principen bygger på att datan delas in i mindre grupper där en specifik del väljs för att representera hela gruppen. Exempelvis i max-pooling, väljs det maximala värdet i gruppen. Detta minskar storleken samtidigt som det bidrar till att de mest prominenta delarna av datan förtydligas [7]. Figur 6 nedan visar ett exempel på hur max-pooling med storleken 2x2 framhäver de maximala värdena för varje grupp.



Figur 6. Exempel på max-pooling med storlek 2x2.

2.2.10 Skip-kopplingar

Skip-kopplingar (eng. skip-connections) bygger på att utdatan från ett lager i nedskalnings-delen av ett nätverk kopplas till indatan från ett lager i uppskalnings-delen. Detta leder till att detaljer som tagits fram i lager från nedskalningsdelen kan återanvändas i uppskalnings-delen. Tekniken används i U-net för att sammanfoga utdatan från lagrena i båda delarna. För att implementera skip-kopplingar i nätverket, skapas antingen *add-layer* eller *concatenate-layer* [7]. I detta arbete används concatenate-layer för att skapa skip-kopplingar.

2.3 Verktyg för utveckling av modeller

I detta avsnitt beskrivs verktyg som kan användas för att skapa en egen modell. Dessa är Python, Tensorflow, Keras och Google colab.

2.3.1 Python

Python är ett högnivåprogrammeringsspråk med öppen källkod som ofta används inom maskininlärning [13]. Python tillåter användandet av externa biblioteket som exempelvis Tensorflow och Keras för att implementera ytterligare funktionalitet.

2.3.2 Tensorflow

Tensorflow är ett bibliotek med öppen källkod, utvecklat av Google. Biblioteket kan användas för att skapa samt exekvera modeller inom maskininlärning. Det kan med fördel användas i samverkan med bibliotek som exempelvis Keras [4]. Google har även skapat biblioteket Tensorflow lite (tflite) för mobila enheter. Eftersom mobiltelefoner inte har lika hög prestanda som exempelvis stationära datorer, krävs att modellen komprimeras. Biblioteket kan användas för att utföra denna konvertering och har även funktioner som tillåter exekvering av den komprimerade modellen direkt på mobiltelefonen. En tflite-modell lagras med datatypen *flatbuffers* i syfte att minska storleken på modellen [14]. Nackdelen med denna konvertering är att modellen presterar sämre med avseende på precisionen i förutsägelserna. Däremot exekveras modellen betydligt snabbare jämfört med en ”standard” Tensorflow modell. För att exekvera en tflite-modell, används en interpretator som stödjer flera plattformar och programmeringsspråk som exempelvis Java, Swift och C++ [14].

2.3.3 Keras

Keras är ett bibliotek som innehåller byggblock för att konstruera artificiella neuronnät. Det finns två olika typer av modeller i Keras: funktionella och sekventiella. I sekventiella modeller definieras lager i ordning och kopplas direkt i den ordningen. Denna struktur är mer begränsad men enkel att använda. I funktionella modeller kopplas istället lagerna till en variabel som ger utvecklaren kontroll över kopplingen. Detta tillåter skapandet av mer avancerade modeller.

Keras innehåller flera funktioner som används för att implementera lager i ett artificiellt neuronnät. Nedan beskriv Keras-funktionerna som används inom detta arbete.

- Conv2D - Detta lager skapar ett faltningslager med bland annat en parameter som bestämmer antal filter som ska användas.
- Dropout - Implementerar regulariseringsmetoden dropout.
- MaxPooling2D - Implementerar max-pooling.
- Concatenate - Detta lager används för att skapa skip-kopplingar. Funktionen slår samman två lager till ett gemensamt lager.
- Conv2DTranspose - Detta lager används för uppskalning av datan.

Vidare information om dessa funktioner finns i Keras API [15].

Efter att strukturen för nätverket har skapats i Keras, kompileras modellen med funktionen *compile*. I denna funktion specificeras optimeringsfunktionen samt förlustfunktionen för modellen. Därefter kan modellen tränas med Keras funktionen *fit*. Denna funktion tar in hyperparametrar som exempelvis batch-storlek, antal epoker, träningsmängden samt valideringsmängden [16]. I samma funktion kan även *callbacks* definieras. I Keras kan callbacks exempelvis utnyttjas till att spara vikterna som ger upphov till bäst resultat under träningen [17]. Detta innebär att om träningen leder till att modellen försämras, kan dem sparade vikterna användas för att skapa modellen [16].

2.3.4 Google Colab

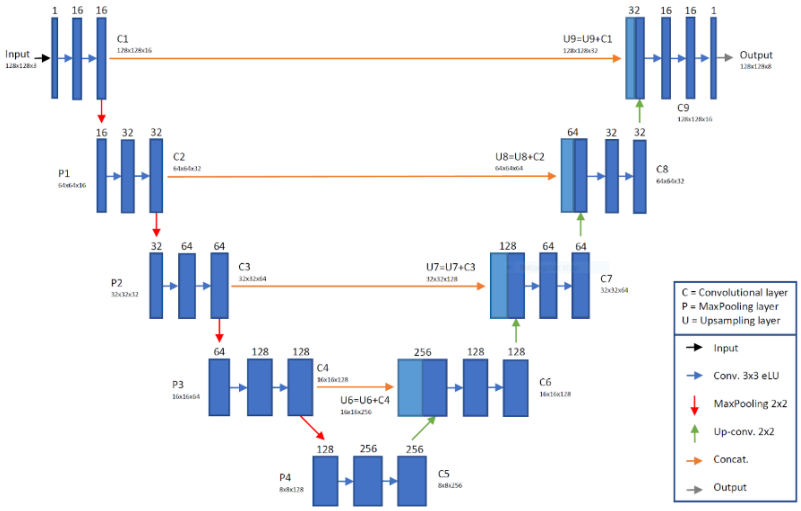
Google Colab är utvecklat av Google research och tillåter utvecklare att skriva samt exekvera Python-kod direkt i en webbläsare. Det är även möjligt att få tillgång till Googles GPU:er vilket gör träning av modeller betydligt snabbare [18].

2.4 Semantisk bildsegmentering

Semantisk bildsegmentering är en teknik som bygger på pixelvis klassificering av olika objekt/klasser i en bild. Till skillnad från objektidentifiering, ger semantisk bildsegmentering en mer detaljerad uppdelning av objekten i en bild eftersom varje pixel representeras av en klass. Tekniken används exempelvis inom sjukvården, robotindustrin och för tolkning av satellitbilder [7]. För att implementera bildsegmentering i detta arbete användes en arkitektur kallad U-net.

2.4.1 U-net

U-Net är en arkitektur som används för semantisk bildsegmentering och konstruerades av Olaf Ronneberger, Philipp Fisher och Thomas Brox år 2015. Enligt deras rapport visade sig arkitekturen vara överlägsen den dåvarande bästa metoden vid ISBI-utmaningen för segmentering av neurala strukturer i elektronmikroskopiska staplar. Nätverket gav både hög träffsäkerhet i förutsägelserna och hög hastighet. Namnet U-net härstammar från att strukturen ser ut som bokstaven U (se figur 7 nedan). Arkitekturen kan delas upp i två delar: en nedskalningsdel och en uppskalningsdel. Strukturen i U-net är symmetrisk vilket innebär att antalet faltningslager i nedskalningsdelen måste vara lika med antalet faltnigslager i uppskalningsdelen [19]. Figur 7 nedan, visar ett exempel på arkitekturen där varje block motsvarar en *”feature map”*.



Figur 7. Exempel på en U-net arkitektur som behandlar bilder med formen (128, 128, 3). Antalet klasser i detta exempel är 8.

Det första lagret i nedskalningsdelen kallas för indatalagret (svart pil i figur 7). Här bestäms storlek samt antal kanaler för bilderna som ska matas in i modellen. Exempelvis kan det vara 128x128x3, dvs en bild med 128x128 pixlar och 3 kanaler (RGB). I nedskalningsdelen sker särdragsextraktioner i ett flertal faltningslager (pilar i blå). Mellan två par av faltningslager befinner sig ett max-pooling-lager som minskar storleken på utdatan från föregående lager. Efter att datan passerat flaskhalsen (eng. *bottleneck*) av nätverket, påbörjas uppskalnings-processen. Detta sker genom att använda ett transponerat faltningslager. I uppskalnings-delen sker även en sammanfogning av faltningslagerna från båda delarna som är på samma nivå. Efter kontrakteringen, utförs ytterligare särdragsextraktioner med två faltningslager [19]. Utdata-lagret i U-net hanterar samma bildstorlek som indata-lagret med skillnaden att datan har formen (128, 128, antal klasser). Datan kan sedan användas för att skapa masker av varje klass. För en mer utförlig förklaring av U-net, se bilaga 1.

2.5 Förbehandling av data & datautökning

Kvalitén på träningsdatan för en modell påverkar inlärningsprocessen och det är därför viktigt att förbehandla datan innan den används för att träna modellen [20]. Förbehandlingen i detta arbete bestod främst av normalisering och standardisering av datan. Det är dessutom fördelaktigt om den befintliga datamängden kan utökas eftersom mer data ofta leder till bättre presterande modeller.

2.5.1 Normalisering av data

Normalisering av data är en teknik som används för att ändra värden i datamängden till samma intervall. I detta arbete är bilderna sparade som 8-bitars heltalsvärden i intervallet 0-255. För att normalisera datan (bilderna i detta fall) divideras bilderna med 255 så intervallet istället blir 0-1 [21]. En fördel med normaliseringen är att inlärningsprocessen sker i högre takt. Om datan inte normaliseras kan nätverket bli instabilt då stora värden bidrar till stora gradienter och små till att gradienten kan försvinna helt vilket resulterar i att modellen slutar att lära. Det är därför väsentligt att datan normaliseras innan träningen inleds [21].

2.5.2 Standardisering av bildstorlek

För dem flesta artificiella neuronnät krävs att storleken på indatan är uniform, detta gäller exempelvis för faltningsnätverk. Som en process av förbehandling av data behöver bilderna konverteras till en gemensam bildstorlek [21].

2.5.3 Datautökning

För att uppnå hög precision i förutsägelserna från en modell, krävs oftast tusentals bilder. En större datamängd med varierad data leder till att modellen kan användas i mer generella situationer. Datautökning (eng. *data augmentation*) är en teknik som kan utnyttjas till att utöka storleken på en datamängd utan att inhämta mer data. I detta arbete bygger tekniken på att skapa nya bilder genom att modifiera bilderna från datamängden. Fördelen med datautökning är dessutom att de modifierade bilderna inte behöver annoteras på nytt, vilket sparar tid [22]. Exempel på modifikationer är: bildrotering, horisontell vändning, ökning/minskning av ljusstyrka, ändrad skala på bilden och förändring i skärpa. I detta arbete användes följande funktioner från biblioteket *imgaug* [23] för att utöka datamängden:

- *Fliplr* - vänder på bilden från höger till vänster.
- *Multiply* - multiplicerar alla pixlar i bilden med ett specifikt värde vilket gör att bilden blir ljusare eller mörkare beroende på värdet.
- *Affine* - transformerar bilden. Kan användas för bildrotation och bildomskalning.
- *Gaussianblur* - gör bilden suddig med en Gaussisk kärna [23].

2.6 Bildanalys

I detta avsnitt ges en kort beskrivning av biblioteket OpenCV och två av dess funktioner som används för vidare analys av maskerna från bildsegmenteringen.

2.6.1 OpenCV

OpenCV är ett bibliotek med öppen källkod för datorseende och maskininlärning. Biblioteket stödjer flera språk som exempelvis Python, Java och C++. Det är även möjligt att använda OpenCV på plattformar som Windows, Linux och Android. Biblioteket innehåller mer än 2500 algoritmer för datorseende [24]. Inom detta arbete användes OpenCV i Android-applikationen främst för att analysera utdatan från segmenteringsmodellerna med hjälp av funktionerna *Canny* och *HoughLines*.

2.6.2 Funktionen Canny

OpenCV innehåller funktionen Canny som implementerar *Canny kantdetekteringsalgoritm*. Denna algoritm är utvecklad av John F. Canny och går ut på att hitta regioner i en bild som har en kraftig förändring av färg eller intensitet [25]. Vanligtvis är bilder i RGB-format vilket innebär att de har 3 kanaler med pixelvärden mellan 0-255. Eftersom det är många värden som behöver bearbetas, är det vanligt förekommande att bilderna omvandlas till gråskala-format innan kantdetektering inleds.

Det första steget i algoritmen består i att reducera eventuellt brus i bilden eftersom bruset kan orsaka att oönskade kanter framkommer. Det vanligaste tillvägagångssättet för brusreducering är att använda ett *Gaussiskt filter* bestående av en kärna (matris) som kan variera i storlek. Storleken på kärnan består av endast udda tal, exempelvis (3x3) och påverkar hur väl filtret reducerar brus. Kärnan appliceras på varje pixel och nya värden beräknas genom elementvis multiplikation [25].

Efter brusreduceringsstadiet, filtreras bilden med två *Sobel-kärnor* för att få första derivatan i både horisontal- och vertikalled, noteras G_x och G_y enligt nedan.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Efter filtreringen kan både magnituden av gradienten G och riktningen θ för varje kant beräknas enligt formlerna nedan:

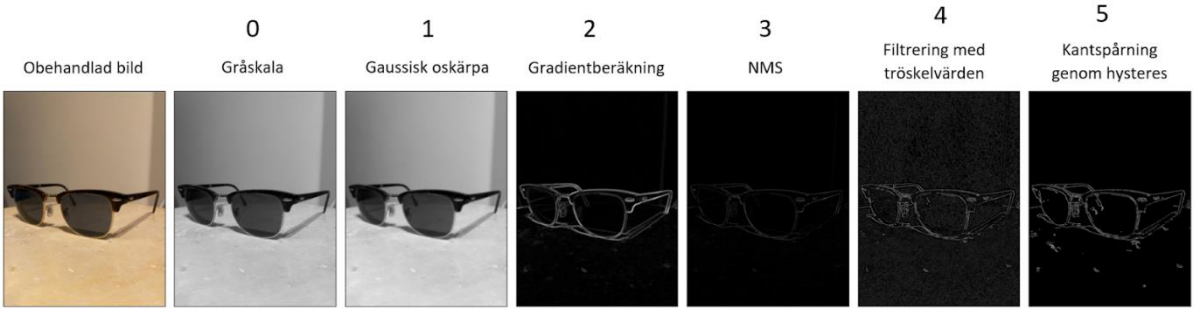
$$G = \sqrt{{G_x}^2 + {G_y}^2} \qquad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Varje kant efter beräkningen av gradienterna är relativt tjock men idealet är att få ut tunna linjer för varje kant i bilden. För att åstadkomma detta används en teknik som kallas för *non-maximum suppression* som i huvudsak går ut på att ta ut pixlar från gradient-matrisen som har störst värde i sitt område i gradientens riktning.

För att ytterligare filtrera bort irrelevanta kanter som finns kvar på grund av brus och färgvariation, tas kanter bort med låg gradient baserat på ett visst tröskelvärde. Det innebär att om gradienten för en kant-pixel är högre än detta tröskelvärde, anses pixeln ha en hög gradient och tas med i nästa steg.

Däremot om gradienten är lägre än tröskelvärdet, kommer kant-pixeln nollställas och inte tas med som en kant i nästa fas.

Den sista delen i Canny kantdetektering går ut på att filtrera ut kanter med pixlar som har lägre gradienter men som också inte är sammankopplade med pixlar med högre gradienter. Ytterligare två tröskelvärden introduceras, max- och min-värden som avgör vilka kanter som är sammankopplade med pixlar med hög gradient. Figur 8 nedan visar alla delar i Canny kantdetektering.



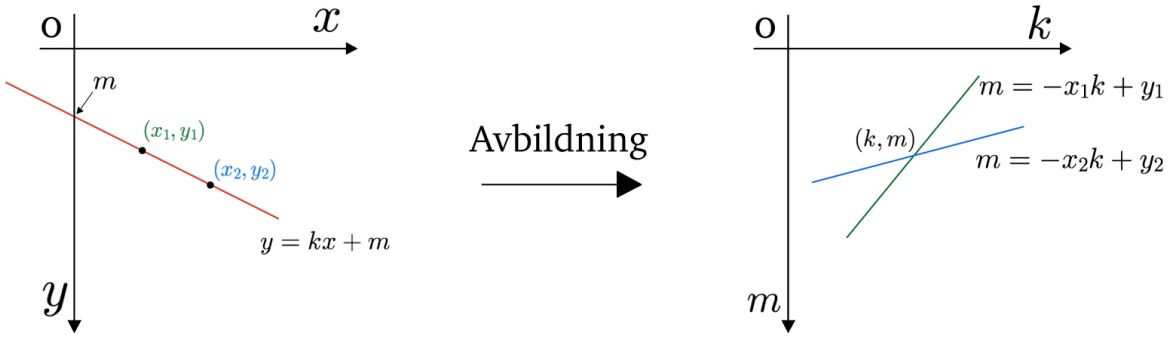
Figur 8. Alla delar i algoritmen för Canny kantdetektering.

2.6.3 Funktionen HoughLines

HoughLines är en algoritm som kan användas för att upptäcka raka linjer i en bild. Algoritmen appliceras efter att bilden har genomgått någon form av kantdetektering, exempelvis Canny. För att förstå hur algoritmen fungerar, är det väsentligt att ha förståelse för hur det traditionella sättet att representera en linje kan avbildas i Hough rummet. En linje kan representeras i två olika system:

- Kartesiskt koordinatsystem
- Polärt koordinatsystem

I det kartesiska koordinatsystemet beskrivs en rät linje med räta linjens ekvation, $y = kx + m$. För att beräkna lutningen krävs att minst två punkter på linjen är kända. Till vänster i figur 9 nedan, visas hur en linje kan representeras i ett kartesiskt koordinatsystem.



Figur 9. Linjen i det kartesiska koordinatsystemet avbildas till Hough rummet.

För att avbilda linjen till Hough rummet som visas till höger i figuren, bryts m ut från rätalinjens ekvation. I detta rum, ses istället punkternas koordinater som konstanter medan m och k är variabler. Det innebär att punkter från det kartesiska koordinatsystemet som är på linjen, kommer representeras av linjer i Hough rummet. Punkten (k, m) motsvarar skärningspunkten för linjerna i Hough rummet. Notera att denna punkt är en beskrivning av linjen i det kartesiska koordinatsystemet. Fördelen med avbildning till Hough rummet är att det blir mindre beräknings krävande då endast punkten (k, m) behövs för att beskriva linjen. Ett problem med att representera linjen i form av räta linjens ekvation är att för vertikala linjer går m och k mot oändligheten. Av denna anledning representeras linjer i Hough linjedetektering med polära koordinater.

Låt $x = r\cos(\theta)$, $y = r\sin(\theta)$ då gäller följande:

$$k = \frac{dy}{dx} = \frac{\frac{dr}{d\theta}\sin(\theta) + r\cos(\theta)}{\frac{dr}{d\theta}\cos(\theta) - rsin(\theta)} = \left(\frac{dr}{d\theta} = 0\right) = -\frac{\cos(\theta)}{\sin(\theta)}$$

$$y = kx + m = rsin(\theta) = -\frac{\cos(\theta)}{\sin(\theta)}x + m = \frac{-rcos^2(\theta)}{\sin(\theta)} + m$$

$$m = rsin(\theta) + \frac{rcos^2(\theta)}{\sin(\theta)} = \frac{rsin^2(\theta)}{sin(\theta)} + \frac{rcos^2(\theta)}{\sin(\theta)} = \frac{rsin^2(\theta) + rcos^2(\theta)}{sin(\theta)} = \frac{r}{\sin(\theta)}$$

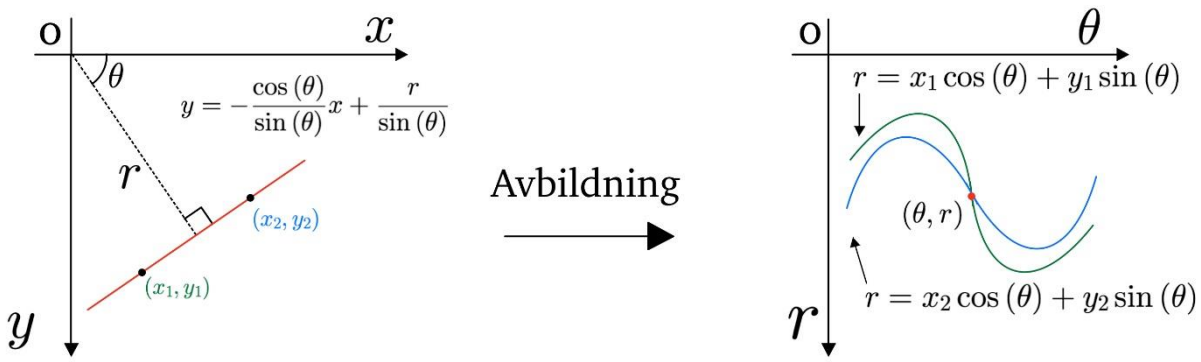
Rätalinjens ekvation kan nu skrivas i polär form enligt följande ekvation:

$$y = -\frac{\cos(\theta)}{\sin(\theta)}x + \frac{r}{\sin(\theta)}$$

För att avbilda linjen i polär form till Hough rummet, bryts r ut från ekvationen ovan. I Hough rummet är x och y konstanter medan vinkeln θ och längden r är variabler enligt formeln nedan.

$$r = x\cos(\theta) + y\sin(\theta)$$

Funktionen beskriver en punkt på linjen i form av en sinusvåg. Skärningen mellan 2 eller flera sinusvågor är en beskrivning av linjen i Hough rummet. Detta illustreras i figur 10 nedan.



Figur 10. Det polära koordinatsystemets avbildning i Hough rummet.

För att detektera linjer med Hough transformering är det väsentligt att ange omfånget för vinkeln θ och längden r. Ett vanligt värde på omfånget för vinkeln, θ, är 180 grader och för längden r: [-i, i] där i är längden på diagonalen för bilden. Efter att bilden har genomgått någon form av kantdetektering, kan Hough linjetransformering appliceras. Algoritmen går ut på att iterera igenom alla pixlar i bilden, om pixeln tillhör en kant, kontrolleras alla möjliga värden för vinkeln θ . För varje vinkel θ, beräknas det korresponderande värdet för r. Om flera pixlar ger upphov till samma värden för θ och r, har en linje upptäckts. Hur många av dessa pixlar som behövs för att en linje ska upptäckas, kan bestämmas med ett tröskelvärde. Slutligen omvandlas den upptäckta linjen tillbaka till formen $y = kx + m$, och linjen kan sedan ritas ut på bilden [26].

2.7 Bild- och dataformat

I detta arbete användes 6 olika typer av bild- och dataformat i Android-applikationen. Denna del av rapporten avser att ge en kort beskrivning av dessa.

2.7.1 YUV

YUV-formatet, YCbCr eller YUV_420_888 består av 3 värden likt RGB-formatet. Den första bokstaven (Y) står för luminans (ljusstyrka) och bestämmer hur ljus varje pixel är. Bokstäverna U och V representerar färginformationen för varje pixel. U (Cb) anger hur blå pixeln är medan V (Cr) står för hur röd pixeln är. I RGB befinner sig färgerna i kanaler medan i YUV är färgerna representerade i koordinater. En fördel med detta är att man skiljer på pixelns ljusintensitet och färg vilket möjliggör att operera endast på färgerna respektive ljusstyrkan. I RGB format är detta inte möjligt då färgerna är kopplade med pixelns intensitet. Varje värde (Y, U och V) i YUV-formatet är sparat i varsitt plan [27]. I Android-applikationen fås bilderna från kameran i detta format.

2.7.2 NV21

NV21-formatet är samma som YUV-formatet med enda skillnaden att NV21 har ett plan för luminans och ett plan för färginformationen [27]. Detta format används som ett mellansteg i konvertering från YUV till JPEG.

2.7.3 JPEG

JPEG-formatet är ett av dem vanligaste bildformaten och används i detta projekt i syfte att konverteras vidare till Bitmap. JPEG är ett format som destruktivt komprimerar bilder, vilket gör det användbart för lagring men kvalitén av bilderna försämras [28].

2.7.4 Bitmap

Bitmap är en array bestående av binärdata som förvarar färgen för varje pixel i respektive position i array:en [29]. Detta format används bland annat i utveckling av Android-applikationer.

2.7.5 Byte buffer

Byte buffers är ett sätt att lagra och skicka data i Java. Detta format används ofta när grafikkort eller liknande komponenter skall bearbeta data eftersom strukturen gynnar dem [30]. Formatet är därför använt i samband med Tensorflow i Java.

2.7.6 Mat

Mat-klassen är från OpenCV-biblioteket och används för att lagra bild-data som exempelvis antal kanaler, tensorer och histogram. För att kunna använda funktioner för bildmanipulering från OpenCV behöver bilderna konverteras till detta format [31].

2.8 SDK och API

Software Development Kit, ofta förkortat till SDK, är en mängd verktyg som tillför funktionalitet för utvecklandet av mjukvara. En SDK kan vara ett ramverk som exempelvis Flutter där en kodbas kan användas till iOS och Android samtidigt. Biblioteket Mapbox har även SDK som tillåter implementation av kartor och funktionalitet för navigering.

Application Programming Interface (API) är ett protokoll som möjliggör kommunikation mellan kod och databaser, alternativt en kodbas och en SDK. För att utnyttja externa bibliotek med funktioner och information används därför API:er som bestämmer hur kommunikationen hanteras.

2.9 Flutter

Flutter är en SDK med öppen källkod utvecklat av Google som är till för att skapa applikationer för både Android och iOS med endast en kodbas. Flutter använder programmeringsspråket Dart som är främst designat för frontend-utveckling av applikationer [32].

2.10 Applikationsutveckling för Android

Detta avsnitt har i syfte att ge en beskrivning på designmönster, komponenter och bibliotek som användes under utveckling av Android-applikationen i Java.

2.10.1 Designmönster

Builder-pattern är ett designmönster där parametrarna för klassens konstruktor separeras genom att implementera en *builder-funktion*. Fördelen med detta är bland annat att det är möjligt att utelämna vissa parametrar när de inte behövs [33].

Observer pattern används för att observera ändringar i andra objekt eller variabel. Detta designmönster bygger på en kommunikation mellan två objekt: *observable* och *observer*. Efter att en observer registrerats till en observable, notifieras objektet när observern uppdaterats [33].

2.10.2 Aktivitet

En aktivitet (eng. *Activity*) i Android-applikationer är en typ av komponent som förser en skärm. Aktiviteter är en av huvud-byggstenarna inom apputveckling för Android och varje applikation kräver minst en aktivitet. En aktivitet har callback-metoder för olika typer av händelser, som exempelvis *OnCreate*, *OnStart*, *OnPause* och *OnDestroy* [34].

2.10.3 Fragment

Ett fragment i Android-applikationer är en återanvändbar komponent som inte kan skapas utan att kopplas till en aktivitet. Denna komponent har en egen livscykel och kan dessutom hantera egna händelser. Fragment kan även användas tillsammans med andra komponenter som exempelvis en *Bottom Navigation View* för att skapa flikar i applikationen [35].

2.10.4 Layouts

I Android-applikationer implementeras användargränssnitt genom att skapa xml-filer kallade *layouts*. Dessa filer innehåller information om alla objekt som användaren kan se och interagera med i applikationen. Varje aktivitet och fragment utnyttjar objekten i layout-filerna för att interagera med- eller uppdatera det grafiska gränssnittet [36].

2.10.5 CameraX

I detta arbete användes det senaste kamera-biblioteket för Android som kallas för CameraX. Till skillnad från tidigare versioner, kräver inte detta bibliotek specifik kod för olika enhetstyper och fungerar på de flesta Android-enheter med version 5.0 eller senare [37]. Detta bibliotek är uppdelad i 3 olika användarfall (eng. *use cases*) men endast förhandsgranskning av kamera och bildanalysering utnyttjades i detta arbete och därför utesluts förklaring av hur bildtagning fungerar. För att skapa en förhandsgranskning av kameran behövs följande 2 komponenter:

- *TextureView* - Denna klass används för att visa bildströmmen från en *Preview*.
- *Preview* - Ett användarfall som används för att få tillgång till en förhandsgranskning av kameran. För att uppdatera vyn (TextureView) används funktionen *setOnPreviewOutputUpdateListener* från Preview [38].

Bilderna som fås från kameran kan analyseras med användarfallet *ImageAnalysis*. Denna klass tillåter utvecklare att använda en egen klass för analysering av bilderna från kameran genom att implementera gränssnittet *ImageAnalysis.Analyzer* [39]. Mer om hur den egna analyserings-klassen fungerar, kan läsas i genomförande.

2.10.6 Text-till-tal

Text-till-tal funktionalitet finns inbyggt i Android-system vilket innebär att ett externt bibliotek inte är nödvändigt. Klassen *TextToSpeech* innehåller funktionen *speak* för att läsa upp meddelanden och funktionen *stop* för att avbryta uppspelningen. Funktionen *isSpeaking* kan användas för att kontrollera om ett meddelande spelas upp. Det finns även funktionalitet för att byta röst, språk samt hastighet på uppspelningen [40].

2.10.7 Inställningar

För att skapa inställningar i en Android-applikation är det fördelaktigt att använda ett biliotek kallat *preference* från AndroidX. Detta bibliotek tillåter implementation av inställningar i en xml-fil där de direkt påverkar värden i andra delar av applikationen. Biblioteket innehåller flera komponenter som kan användas för att skapa inställningarna [41]. Komponenterna som används för att implementera inställningar i applikationen i detta arbete är följande:

- *Category* - En komponent som gör det möjligt att dela upp inställningarna i kategorier.
- *List* - Denna komponent öppnar en dialogruta som innehåller en lista med knappar bredvid motsvarande etikett. Endast en knapp kan vara påslagen.
- *MultiSelectList* - Denna komponent öppnar en dialogruta som innehåller en lista med knappar bredvid motsvarande etikett. Flera knappar kan vara påslagna.

- *EditText* - Denna komponent öppnar en dialogruta som innehåller ett textfält. Användaren kan därefter skriva in ett nytt värde i textfältet.
- *Switch/CheckBox* - En komponent som kan ändra ett booleskt värde efter att användaren har interagerat med komponenten.

För att få tillgång till alla värden, kan *PreferenceManager* användas från samma bibliotek. Exempelvis för att få tillgång till värdet från en switch, används följande kod:

PreferenceManager.getDefaultSharedPreferences(context).getBoolean(key, false)

Kontexten (*context*) är ett gränssnitt för global information av en Android-applikation och används här för att få tillgång till ett värde från inställningarna. Varje komponent har en nyckel i form av en sträng som används för att kunna skilja dem olika komponenterna åt. Den sista parametern i funktionen bestämmer vilket standardvärde som ska användas om inställningen inte finns tillgänglig.

2.10.8 ReactiveX

Android-applikationen i detta arbete använder funktioner från biblioteket ReactiveX som är ett bibliotek för reaktiv-programmering [42]. Biblioteket används främst för att observera när uppdateringar sker i klassen för bildanalys. Detta för att bland annat kunna uppdatera masken från segmenteringen.

2.10.9 Mapbox

Mapbox är ett bibliotek som hjälper utvecklare att implementera kartor och funktioner för navigering i applikationer för Android och iOS [43]. I detta arbete används i huvudsak följande 3 paket från Mapbox: *Map SDK, Navigation SDK* och *Search SDK*. Map SDK tillåter skapandet av kartor där platser kan markeras ut. Paketet innehåller bland annat funktioner som kan användas för att få tillgång till användarens position. Det är även möjligt att sedan visualisera positionen på kartan [44]. Följande klasser används från Map SDK i detta arbete:

- *MapboxMap* - Denna klass innehåller information om kartan, exempelvis position, stil och zoomnivå.
- *MapView* - Denna klass använder data från MapboxMap för att skapa en karta i applikationen.
- *LocationEngine* - Denna klass genererar användarens position samt ger återkoppling varje gång den uppdateras [45].
- *LocationEngineRequest* - Denna klass innehåller parametrar för uppdatering av användarens position från LocationEngine [45].
- *LocationEngineCallback <LocationEngineResult>* - Ett gränssnitt som används för att få uppdateringar från LocationEngine [45].
- *LocationComponent* - En klass som används för att visualisera användarens position på kartan [46].

Paketet Navigation SDK gör det möjligt att skapa en färdväg mellan två specificerade positioner. Denna SDK kan även användas för att skapa instruktioner genom navigeringen och har dessutom funktionalitet som att meddela när användare har nått destinationen [47]. När en färdväg genereras, kan typen av navigation specificeras och i detta arbete används navigation för gående.

En väg mellan en startpunkt och destination i Mapbox kallas för *route*. En route består av *Legs* som motsvarar vägen mellan delmål i den fullständiga rutten. Slutligen består varje Leg av en eller flera *steps* som motsvarar varje sväng och väg byte i navigeringen [48]. I detta arbete användes följande klasser från Navigation SDK:

- *MapboxDirections* - En klass som kan generera datan för en färdväg mellan en destination och startposition. Denna funktion kräver följande värden: en destination, startplats samt åtkomstnyckel till Mapbox API. Instansen av denna klass kan användas för att hämta färdvägen från *Mapbox Directions API*.
- *DirectionsRoute* - En klass som innehåller färdvägen för en route och returneras av MapboxDirections [49].
- *NavigationOptions* - Används för att skapa en instans av Mapbox Navigation. Builder-funktionen för denna klass kräver en LocationEngine samt en API-nyckel.
- *MapboxNavigation* - En klass som hanterar navigering i Mapbox Navigation SDK. För att skapa en instans, kräver denna klass en *NavigationOptions* som parameter [50]. Klassen innehåller funktioner för att starta navigering, avsluta navigering samt lägga till en färdväg.

Funktionen *enqueueCall* används i flera delar av Mapbox när det behövs en respons från Mapbox API. Denna funktion skapar en *callback* som returnerar värden efter den får en respons får Mapbox API [50]. Mapbox innehåller även olika observers som kan utnyttjas till att exempelvis observera när en ny instruktion är tillgänglig eller när användaren har nått sin destination. I detta arbete användes följande observers från Mapbox:

- *VoiceInstructionsObserver* - Ett gränssnitt som implementerar funktionen *onNewVoiceInstructions*. Denna funktion tar in en sträng med instruktionerna för det nuvarande steget och kallas varje gång instruktionerna uppdateras [51].
- *ArrivalObserver* - Ett gränssnitt som kontrollerar när användaren når sin destination. Den innehåller funktionen *onFinalDestinationArrival* som kallas när destinationen är uppnådd [48].
- *RouteProgressObserver* - Ett gränssnitt som implementerar funktionen *OnRouteProgressChanged*. Denna funktion kallas när användaren förflyttar sig och har en *RouteProgress* som inparameter. Denna parameter innehåller information om den nuvarande färdvägen [48].

Paketet Mapbox Search SDK kan användas för att skapa en sökruta som tillåter användaren att söka och välja en specifik destination [52]. I detta arbete användes följande klasser från detta paket:

- *CarmenFeature* - En klass som håller information från *Mapbox Geocoding API*. Informationen kan exempelvis bestå av namnet på platsen och positionens koordinater [53].
- *PlaceAutocomplete* - En klass som innehåller ett förbyggt användargränssnitt för sökfunktionalitet. Genom att använda Mapbox Search API, uppdaterar den med förslag när användaren skriver in en ny destination [52].
- *Mapbox Geocoding* - Används för att generera gatuadresser från koordinater. Det är även möjligt att göra omvänd geocoding där en gatuadress istället ger koordinater [54].

2.11 Tidigare forskning

Forskning och utveckling av navigeringssystem för synskadade har pågått under en lång tid. I några av dem tidigaste elektroniska lösningarna användes ljudvågor eller laser för att avgöra avstånd och upptäcka hinder. I mer moderna navigeringssystem började användning av GPS bli vanligt. Däremot saknade tidiga implementationer funktionalitet för att upptäcka hinder. Idag finns det många applikationer som utnyttjar mobilens sensorer för att hjälpa personer med synnedsättning att exempelvis navigera [55].

Ett exempel på en navigerings-applikations för synskadade med mobilens inbyggda GPS, är applikationen BlindSquare. Denna applikation fokuserar på att ge användare instruktioner utifrån punkter av intresse (eng. *points of interest*), exempelvis butiker eller byggnader som användare tidigare besökt [56].

På senare år har metoder inom djupinlärning använts i navigeringssystem för synskadade. Ett exempel på detta är [57], där ett system för mobiltelefoner skapades för att identifiera hinder med hjälp av datorseende. I deras system utnyttjades objekt-identifiering från en mobiltelefon kopplad till en server för att identifiera hinder. Genom detta fick användaren återkoppling med information om vilket objektet som identifierats. Därefter genomfördes även en estimering på distansen mellan användaren och hindret. Målet med arbetet var att stödja navigeringen för synskadade eftersom GPS system och liknande lösningar inte kan upptäcka hinder.

3
Metod & teknikval

Detta avsnitt beskriver den valda arbetsmetoden för projektet. Metoden för arbetsprocessen gick ut på att planera, undersöka, genomföra och avsluta arbetet. Detta visualiseras i figur 11 nedan.

Figur 11. Metoden för arbetsprocessen som bestod av planering, förundersökning, genomförande och avslut.

3.1 Planering

Projektet inleddes med en planeringsfas i syfte att bland annat sätta upp tydliga, gemensamma och nåbara mål. Planeringen bestod även i att skapa en tidsplan för att kunna fördela den begränsade tiden för arbetet på ett effektivt sätt.

3.2 Förundersökning

Efter planeringsfasen, påbörjades en förundersökning kring följande områden:

- Navigeringssystem för synskadade
- Datorseende
- Plattform för utveckling

Navigeringssystem för synskadade undersöktes för att få en bättre bild av tidigare system och arbeten inom detta område. Det blev därför möjligt att identifiera viktiga delar i ett navigeringssystem för personer med synnedsättning. Efter detta söktes information om hur olika metoder inom datorseende kan implementeras. Syftet med denna informationssökning var att avgöra vilka metoder som kan implementeras i en applikation för mobiltelefoner. Främst studerades metoder för objektidentifiering och bildsegmentering. Det undersöktes sedan hur dessa metoder kunde implementeras i ett navigeringssystem och hur metoderna kan hjälpa synskadade att navigera. Slutligen undersöktes olika plattformar som stödjer exekvering av modeller på mobiltelefoner. Detta genomfördes för att utreda vilka alternativ som var tillgängliga för utveckling av applikationen.

3.3 Arbetsstruktur

Arbetet strukturerades enligt en enkel version av scrum i syfte att få struktur på arbetet utan att spendera onödig tid på protokoll som är irrelevanta i en mindre grupp. Metoder som vattenfallsmetoden passade detta arbete mindre bra eftersom mycket av projektet innebar att testa och utforska olika alternativ för implementationen. Att använda en agil-metod tillät oss istället vara mer flexibla och anpassa arbetet efter vad som visade sig fungera bäst. Applikationen utvecklades i en iterativ process vilket tillät oss också att anpassa hur långt applikationen utvecklas efter tiden som kvarstod. Varje vecka avslutades med att vi sammanfattade vad vi åstadkommit under den gångna veckan, följt av en kort planering på vad vi skulle fokusera på den kommande veckan.

3.4 Val av utvecklingsplattform och API

I början av arbetet valdes Flutter som utvecklingsplattform eftersom ett av de tidigare målen var att applikationen skulle täcka en bred publik. Ett alternativ var att välja React Native, men istället valdes Flutter eftersom utveckling av applikationer i denna utvecklingsmiljö är mer effektiv. Flutter har dessutom stöd för Google Maps som planerades att användas. Efter att en prototyp skapats, framkom det att Google har restriktioner på implementering av “turn by turn” navigering med deras API. Detta innebar att Mapbox blev alternativet som användes för att implementera GPS-navigering. När Mapbox sedan skulle implementeras i Flutter-applikationen framgick det att användning av detta bibliotek var begränsat i Flutter. Dessutom saknades fullt stöd för biblioteket OpenCV som skulle användas för vidare analys av utdatan från bildsegmenteringsmodellen. Detta innebar att utvecklingen av applikationen istället genomfördes i Java för Android-telefoner. Eftersom modellerna utvecklades i Python-kod och endast den färdiga modellen behövdes, kunde dessa delar fortfarande utnyttjas.

3.5 Val av navigeringsmetod

Det går att dela upp navigeringen i två delar: kortsiktig- och långsiktig navigering. Den kortsiktiga navigeringen avser navigering i korta sträckor. Det kan exempelvis vara att hitta vägen eller objekt i nära omgivning. Den långsiktiga navigeringen innebär längre sträckor som att ta sig från sitt hem till en affär. Målet med detta arbete var att implementera ett system som hjälper synskadade navigera utomhus vilket innebär att båda typer av navigering behövdes implementeras. För den långsiktiga navigeringen valdes GPS navigeringen eftersom denna metod är tillgänglig på nästan alla mobiltelefoner och är enkel att implementera.

Det finns många metoder för att implementera den kortsiktiga navigeringen, exempelvis genom att använda sensorer. Eftersom målet var att undersöka hur AI kan utnyttjas i ett navigeringssystem, valdes en metod baserad på djupinlärning. Exempel på metoder inom djupinlärning är objektet-identifiering och bildsegmentering. Objekt-identifiering har redan implementerats i tidigare studier inom navigering för synskadade, exempelvis i [57], men i detta arbete utvecklades istället metoder som utnyttjar bildsegmentering. Utdatan från en objekt-identifieringsmodell består av avgränsnings-rutor som omringar varje klass i en bild. För att upptäcka var en väg befinner sig i en bild är denna metod inte att föredra eftersom avgränsnings-rutan kommer innehålla delar som inte tillhör vägen, alternativt inte täcka hela vägen. Att använda bildsegmentering ger fortfarande möjligheten att avgöra var varje objekt befinner sig samtidigt som större entiteter som vägar och gräsfält kan klassificeras på pixelnivå. Med anledning av detta, valdes bildsegmentering som metod för den kortsiktiga navigeringen i detta arbete.

3.6 Datainsamling

Träning av modeller kräver en anpassad datamängd till uppgiften som modellen skall lösa. I detta arbete var denna uppgift bildsegmentering. För bildsegmentering består datamängden av bilder samt markeringar var klasserna befinner sig i bilden. Det finns redan färdiga datamängder som kan laddas ner från internet, exempelvis *cityscapes-dataset* [58]. Problemet är att detta material är skyddat av licenser och copyright. Därför var användning av denna datamängd inte ett alternativ i detta arbete. För att undvika problem med licenser, samlades istället bilder in med mobilkamera och varje objekt i bilderna markerades därefter med programmet *Labelme* [59].

3.7 Verktyg för utveckling av modeller

För träning och utveckling av modeller finns ett flertal olika verktyg som utvecklare kan utnyttja. Några av dem mest använda biblioteken inom djupinlärning är Tensorflow och PyTorch. Tensorflow innehåller Keras som gör det enkelt att konstruera och träna artificiella neuronnät. Dessutom stödjer Tensorflow exekvering av modeller på mobila enheter med Tensorflow Lite biblioteket. Dessa fördelar innebar att Tensorflow användes för utveckling av modeller i arbetet.

3.8 Användning av modellen

Estimeringarna från modellen behöver tolkas för att ge användaren relevant återkoppling. Därför implementerades en metod som kan använda datan till att avgöra var varje klass befinner sig i bilden. Att använda en lösning baserad på objektidentifiering för detta ändamål är inte att rekommendera eftersom vägar ofta hör ihop med andra vägar och kan ha olika former.

För att avgöra var användaren befinner sig på vägen, implementerades en annan metod som också utnyttjar datan från segmenteringen. Denna metod använder Canny kantdetektering tillsammans med Hough linje-transformering för att upptäcka kanterna på vägen. Det finns metoder som används för självkörande bilar som kan upptäcka körfält, men dessa metoder kräver att bilden delas upp i en så kallad region av intresse. Denna region är fast bestämd och omsluter endast vägen. Anledningen till uppdelningen är för att ta bort ointressanta linjer. Nackdelen med denna metod är att regionen är statisk vilket innebär att om kameran flyttas, medför det att resultat blir felaktigt. Detta problem uppstår inte i vår metod eftersom kantdetekteringen appliceras på masken från segmenteringen, vilket innebär att regionen av intresse blir dynamisk.

3.9 Testning av applikationen

I detta arbete testades applikationen i en iterativ process. De olika delarna i applikationen testades separat i syfte att upptäcka brister i respektive del. Därefter implementerades lösningar på problem som uppstod för att sedan testa delarna ytterligare. Efter att alla delar av applikationen gav önskvärt resultat, påbörjades testning av den fullständiga applikation där alla delar samverkade. Genomförandet av dessa tester innebar att navigera korta sträckor med ögonbindel. Testerna gav en förståelse för hur bra applikationen fungerade och vilka brister som kvarstod.

4

Genomförande

Denna del av rapport beskriver genomförande av arbetet. Områdena som behandlas är: skapande av datamängd, utveckling/implementering av modeller, GPS navigering och inställningar för applikationen.

4.1 Skapandet av en semantisk bildsegmenteringsmodell

I detta avsnitt beskrivs genomförandet för skapandet och implementeringen av en semantisk bildsegmenteringsmodell för Android. Den valda arkitekturen på nätverket var U-net eftersom den är väldokumenterad och kan användas för ändamål där både hastighet och precision i förutsägelserna är väsentliga. Genomförandet för denna del gick ut på följande 8 delar:

1. Bildinsamling
2. Markering av klasser
3. Förbehandling av bilder
4. Uppdelning av datamängden.
5. Skapande av modell
6. Träning av modell
7. Testning av modell
8. Konvertering till tflite.

4.1.1 Bildinsamling

Arbetet med bildsegmenteringen inleddes med datainsamling i form av bildtagning med mobiltelefon. Bilderna togs med en iPhone 12 pro samt en OnePlus Nord och sparades i JPEG format. Sammanlagt togs 312 bilder i bostadsområden på följande 7 klasser:

- Väg
- Trottoar
- Gräs
- Lyktstolpe
- Person
- Bil
- Sten

4.1.2 Markering av klasser

Efter datainsamlingen användes verktyget kallat Labelme [59] för att markera alla klasser i varje bild, se figur 12 på nästa sida. Programmet låter användaren välja mapp där bilderna tas ifrån samt en mapp där informationen ska sparas. För att markera en klass i bilden, klickar användaren på “create polygons”. Sedan är det möjligt att byta till nästa bild genom att klicka på “d” och informationen om varje form sparas därefter.

Figur 12. Programmet Labelme som tillåter användaren att markera objekt i bilden.

Följande riktlinjer skapades för att kunna avgöra i vilka förhållanden varje objekt ska markeras:

- En väg består enbart av vägen (utan trottoarkant)
- En trottoar består av en väg och en trottoarkant
- Gräs markerades endast då det tydligt framgick att det var gräs. Detta innebär att jord, pinnar och löv som låg på gräset undveks att markeras.
- Lyktstolpar markerades endast om de var synliga utan att förstora bilden.
- Personer markerades endast om hela personen var synlig i bilden.
- Bilar markerades endast då det tydligt framgick att det var en bil i bilden.
- Endast stora stenar markerades då det tydligt framgick att det var en sten. Stora stenar ansågs vara stenar med en diameter på cirka 0,5 meter eller större.

Markeringarna för varje bild sparas i en json-fil, se figur 13 till höger. I detta arbete användes endast *"shapes"*, *"label"* samt *"points"* för att skapa masker av träningsdatan. En json-fil kan innehålla information om flera former där varje form har ett namn och x-y koordinater för varje punkt.

Json-fil

```
version:
flags:
shapes:[
        label:
        points:[ ]
        group_id:
        shape_type:
        flags:
       ]
imagePath:
imageData:
```

Figur 13. Innehållet i json-filen som genereras av programmet Labelme.

4.1.3 Förbehandling av data

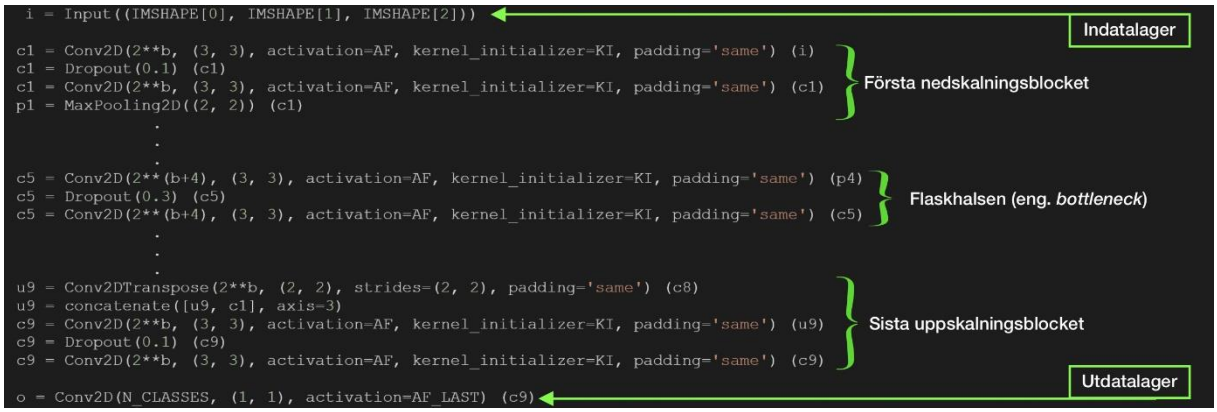
Efter att alla klasser har markerats i varje bild, förbehandlas datan innan träningen av modellen inleds. Det första steget i förbehandlingsprocessen var att producera masker utifrån varje punkt i json-filerna. För detta ändamål, implementerades metoden *generateImagesAndMasks* som skapar bilderna samt maskerna genom att itererar igenom varje bild med dess korresponderande json-fil. I denna metod används ytterligare en funktion som vi kallar för *createMasks* som läser in json-filen för en bild och tar ut koordinaterna för varje punkt med dess korresponderande klass-namn. Därefter skapas en mask genom att använda funktionen *fillPoly* från OpenCV. Efter att alla klasser har varsin mask, sparas de till en gemensam mask i formatet (original höjd, original bredd, 8) med datatypen float32, där höjd och bredd tas från bildens originalstorlek. I denna kontext innebär 8 antalet kanaler, vilket motsvarar antal klasser i bilden med bakgrunden inräknad. Efter att maskerna för varje klass har sparats, ändras deras storleken till (n, n). Värdet av n representerar indatan av modellen, exempelvis 64, 128 eller 256. Storleken ändras med funktionen *resize* från OpenCV. En mask är vid det här laget i formatet (n, n, 8). De generade maskerna sparas sedan i en lista och returneras från metoden. Listan kommer alltså innehålla 312 masker, där varje mask har formen (1, n, n, 8). Maskerna normaliseras sedan till intervallet 0-1 genom division med 255. Bildernas storlek ändras därefter till (n, n) och normaliseras på samma sätt.

4.1.4 Uppdelning av datamängd

Nästa steg var att dela in datamängden i två delar: träningsmängd och valideringsmängd. Uppdelningen utfördes med hjälp av en funktion från biblioteket *sklearn* som kallas för *train_test_split*. Denna funktion delar upp datan slumpmässigt baserat på storleken på valideringsmängden. Det innebär att det är möjligt att ange hur stor valideringsmängden ska vara och resterande del blir träningsmängden. I detta arbete bestod valideringsmängden av antingen 10 eller 20 % av hela datamängden.

4.1.5 Skapande av semantisk bildsegmenteringsmodell

Efter att datamängden delats in i två delar, skapades en segmenteringsmodell baserad på arkitekturen U-net med hjälp av funktioner från både Tensorflow och Keras. För att beskriva hur modellen implementerades visas delar av koden enligt figur 14 nedan.



Figur 14. En del av koden för implementationen av nätverket som baseras på U-net arkitekturen.

Det första lagret i nätverket är indatalagret och definieras av formen på indatan. I koden implementerades detta med en array kallade *IMSHAPE* som innehöll höjd, bredd och antalet kanaler för bilden. Utdatan från indatalagret förs sedan vidare till första nedskalningsblocket, se figur 14 på föregående sida. Denna del består av två faltningslager, ett dropout-lager och ett maxpooling-lager. Varje faltningslager i denna del har 2^b antal filter med storlek $(3, 3)$, där b sattes till ett tal mellan 3-5. Antal filter för faltningslagerna är ett tal med bas 2 genom hela nätverket. Värdet av exponenten ökar med 1 efter varje del för att öka antalet filter djupare in i nätverket. Detta värde kunde även ändras för att testa hur olika antal filter påverkar modellen. Varje faltningslager har dessutom en aktiverings-funktion som kunde väljas för att testa vad som fungerade bäst. Mellan två faltningslager, implementerades ett dropout-lager för att minska risken för överanpassning. Värdet i detta lager motsvarar hur stor del av datan som nollställs. Det sista lagret i första nedskalningsblocket består av ett maxpooling-lager med pool-storleken $(2, 2)$. Varje block i nedskalningsdelen består av samma typ av lager men har olika inparametrar. Detta gäller även för uppskalningsdelen. Nästa del i nätverket kallas för flaskhalsen (eng. *bottleneck*). Detta block består av ett dropout-lager som omges av två faltningslager. Eftersom detta block befinner sig i den djupaste delen av nätverket, används värdet $2^{(b+4)}$ för antal filter i båda faltningalagerna. Utdatan från det sista faltningslagret i flaskhalsen förs sedan vidare till första uppskalningsblocket.

Uppskalningsblocken består av ett transponerat faltningslager, ett concatenate-lager, 2 faltningslager och ett dropout-lager. Det transponerade faltningslagret tar in samma typ av parametrar som ett faltningslager i nedskalningsdelen, men skillnaden är att detta lager utnyttjar *strides* för att öka storleken på datan. Sedan används ett concatenate-lager för att skapa skip-koppling mellan ett nedskalningsblock och uppskalningsblock från samma nivå i nätverket. I figur 14 sammansluts det transponerade faltningslagret u9 med faltningslagret c1 från det första nedskalningsblocket. Slutligen har varje uppskalningsblock två faltningslager och ett dropout-lager med samma parametrar som nedskalningsblocket.

Utdatan från det sista uppskalningsblocket förs vidare till utdatalagret. Detta lager består av ett faltningslager med ett filter-antal bestämt efter antalet klasser som nätverket behandlar. Detta lager använder aktiverings-funktionen *Softmax*. För att se den fullständiga Python-koden för nätverket, se bilaga 2.

Det sista steget i denna del var att kompilera modellen med Keras funktionen *compile*. Här specificeras vilken optimeringsfunkion samt förlustfunktion som modellen skall tränas med. I arbetet användes *Adam* som optimeringsfunkion och *categorical_cross_entropy* samt *dice_loss* som förlustfunktion. Figur 15 nedan visar compile funktionen.

```
model.compile(optimizer=Adam(LEARNING_RATE),
              loss=dice_loss,
              metrics=[dice])
```

Figur 15. En del av koden där det är möjligt att bland annat specificera optimeringsfunktion och förlustfunktion.

4.1.6 Träning av semantisk bildsegmenteringsmodell

Efter att modellen har kompilerats, genomförs träning av modellen med funktionen *fit* från Keras. Denna funktion tar in många olika parametrar som kan styra träningen. Tabell 1 nedan, visar parametrarna som används i funktionen:

Tabell 1. Parametrarna som användes i Keras funktionen fit.

Parameter	**Förklaring**
x	Bilder från träningsdatan.
y	Masker från träningsdatan.
batch_size	Storleken på varje batch, sattes till mellan 8-30.
epochs	Antal epoker, sattes till mellan 400-600.
verbose	För att kunna se träningens framsteg, sattes verbose till 1.
validation_data	Datan som modellen ska evaluera förlusten med efter varje epok.
callbacks	För att spara modellen med högst precision i förutsägelserna, används *ModelCheckpoint*.

Efter träningen av modellen, skapades grafer som visar hur modellens precision och förlust har förändrats under träningen. Slutligen sparades modellen i Keras modell-format (.h5) med Keras funktionen *save*.

4.1.7 Testning av semantisk bildsegmenteringsmodell

För att beskriva processen hur en färdig tränad modell testades på en bild, delas genomförandet upp i följande 10 delar:

1. Läs in bilden som ska testas.
2. Ändra storleken på bilden.
3. Konvertera bilden till RGB-format.
4. Omvandla bilden till uint8 datatyp.
5. Normalisera (dividera med 255).
6. Expandera bildens dimensioner, från (höjd, bredd, 3) till (1, höjd, bredd, 3).
7. Utför en förutsägelse med modellen.
8. Pressa dimensionerna, från (1, höjd, bredd, antal klasser) till (höjd, bredd, antal klasser).
9. Skapa mask.
10. Visa både bilden och masken.

Det första steget var att ladda in en testbild med funktionen *imread* från OpenCV. Denna bild används för att testa modellen. Bilden konverteras därefter till RGB-format och storleken ändras till det modellen förväntar sig, exempelvis 128x128. Eftersom modellen tar in bilder med datatypen uint8, omvandlas bilden till denna datatyp. Dessutom måste bilden normaliseras och expanderas för att modellen ska kunna tolka bilden. Nästa del bestod i att använda modellen till att skapa en förutsägelse baserad på indatan. Utdatan från modellen är en array med dimensionen (1, höjd, bredd, antal klasser). Notera att antal klasser i detta arbete var 7+1 med bakgrunden inräknad. För att kunna skapa masker av denna data, pressas dimensionerna av array:en till (höjd, bredd, antal klasser) och omvandlas därefter till RGB-format. Anledningen till detta är för att OpenCV inte kan hantera bilder fler än 4 kanaler. Omvandlingen sker i en metod kallad *masks_to_RGB* som inleder med att skapa en HSV-bild utifrån datan från modellen. Därefter omvandlas bilden till RGB-format för att sedan returneras av

metoden. Den sista delen består i att visa både bilden och masken. Detta görs med vår funktion *plotMulti* som tar in en array av bilder och visar bilderna i rad. Denna metod använder funktioner från biblioteket *matplotlib*.

4.1.8 Konvertering till tflite

Innan modellen kan användas i Android-applikationen, måste modellen konverteras till en tflite-modell. För att konvertera en Tensorflow-modell till en tflite-modell, definieras en tflite-konverterare som sedan kan konvertera modellen. För att genomföra detta användes följande funktion från tflite:

TfliteConverter.from_keras_model.

Efter att konverteraren skapats, användes funktionen *convert* för att konvertera modellen till en tflite-modell.

4.2 Implementering av kamera

I applikationen implementerades kamera-funktionalitet genom att skapa en klass som vi kallar för *CameraFragment*. Denna klass ärver funktioner från klassen Fragment, vilket innebär att den har en egen livscykel. Kameran aktiveras genom att användaren klickar på kameraknappen i applikationen. Tabell 2 nedan visar de olika komponenterna som används i klassen CameraFragment.

Tabell 2. Komponenter som initieras när en instans av klassen CameraFragment skapas.

Komponent	**Typ**	**Förklaring**
texturePreview	TextureView	En förhandsgranskning av kameran.
masks	ImageView	En vy som används för att visualisera masker för alla klasser från segmenteringsmodellen.
lines	ImageView	En vy som används för att visualisera linjerna från analyseringen av segmenteringsmodellen.
segTime	TextView	En vy som visar beräkningstiden för segmenteringsmodellen.

Efter att komponenterna initierats, triggas funktionen *startCamera* som skapar följande användningsfall (eng. *use case*):

- *Preview*
- *ImageAnalysis*

Båda användningsfallen binds till kamerans livscykel. Analysering av bilder från kameran, sker i en separat klass som vi kallar för *ImageSegmentationAnalyzer*. I klassen CameraFragment, uppdateras komponenterna *masks*, *lines* och *segTime* med en observer från biblioteket ReactiveX. Denna observer prenumererar på en observable i klassen ImageSegmentationAnalyzer. Komponenten texturePreview uppdateras i klassen CameraFragment och ger en förhandsvisning av kameran.

4.3 Implementering av modellen i applikationen

För att analysera bilderna som fås från kameran, skapades klassen *ImageSegmentationAnalyzer* som implementerar gränssnittet *Analyzer*. Syftet med denna klass är att ladda in en tflite-modell, exekvera modellen med en interpretator och sedan skapa masker utifrån datan som fås från modellen. Klassen *ImageUtils* används av ImageSegmentationAnalyzer för att konvertera bilderna från kameran till rätt format för interpretatorn. Dessutom implementerades klassen *MaskAnalysis* som används för analysering och manipulering av maskerna. Implementeringen av modellen kan delas in i följande delar:

- Konvertera bild
- Exekvera modell
- Skapa masker
- Analysera masker

4.3.1 Konvertera bild

CameraX producerar bilder i YUV 420 888 format men interpretatorn från tflite tar in data i form av bytebuffer. Med anledning av detta krävdes några steg för att konvertera bilderna som fås från kameran från YUV till bytebuffer. Följande steg krävdes:

- YUV → NV21
- NV21 → JPG
- JPG → Bitmap
- Bitmap → Bytebuffer

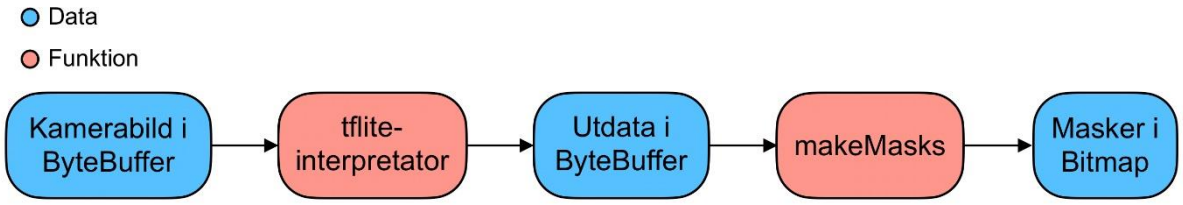
Det första steget var att konvertera bilderna som fås från kameran i formatet YUV, till formatet NV21. Detta görs med funktionen *YUV420toNV21* i klassen *ImageConverters*. Efter att bilden har konverterats till NV21, konverteras bilden till JPEG format. Detta görs med funktionen *NV21toJPG*. Därefter avkodas array:en till Bitmap med *Bitmapfactory.decodeByteArray* för att sedan matas in i en funktion som kallas för *resizebilinear*. Denna funktion ändrar storleken på en Bitmap till samma storlek som tflite-modellen förväntar sig, exempelvis 128x128. Sista steget i konverteringsprocessen är att konvertera bilden i Bitmap format till Bytebuffer. Detta görs med funktionen *Bitmap2Bytebuffer*. Bilden i ByteBuffer formatet kan sedan användas av interpretatorn från tflite för att skapa förutsägelser utifrån indatan.

4.3.2 Exekvera modell

Efter att bilderna från kameran har konverterats från YUV till bytebuffer, används interpretatorn från tflite som exekverar modellen och ger utdata i form av en Bytebuffer som kallas för *outputData*. Denna Bytebuffer innehåller värden mellan 0-1 som representerar precisionen i förutsägelserna för varje klass och varje pixel i bilden.

4.3.3 Skapa masker

För att omvandla datan som fås från interpretatorn till masker i Bitmap format, skapades metoden *makeMasks*. Viktigt att notera är att denna del av applikationen arbetar med modellens bildstorlek, vilket exempelvis kan vara 128x128. Denna funktion returnerar två bitmaps. Den ena innehåller masker för alla klasser och den andra innehåller masken endast för vägen. För en pixel, avgörs vilken klass som har högst värde från modellen, det vill säga ett tal mellan 0-1. Den klassen som har högst värde från modellen för pixeln, sparas i form av ett tal mellan 0-7 (antal klasser + bakgrund) i en tvådimensionell array med exempelvis 128x128 som storlek. Denna array:en kallas för *segmentBits* och kommer vara fylld med tal mellan 0-7 som motsvarar dem olika klasserna. Därefter färgas maskerna genom att ange färgen för den korresponderande klassen i funktionen *setPixel*. Slutligen returneras båda maskerna i en bitmap-array. Figur 16 nedan visar denna process.



Figur 16. Processen från att en konverterad bild används som indata för tflite-interpretatorn till att en mask skapas utifrån utdatan från interpretatorn.

4.3.4 Analysera maskerna från segmenteringen

I detta arbete har följande metoder implementerats för analysering av maskerna: *objectLocations* och *positionOnRoad*. Den förstnämnda metoden går ut på att beräkna var varje klass befinner sig på skärmen genom att dela upp skärmen i tre regioner: vänster, mitten och höger. För att avgöra var varje objekt befinner sig, jämförs antalet pixlar i varje region för varje klass. Den region som får flest antal pixlar, motsvarar objektets placering. Eftersom vägen ofta är bredare än andra objekt kan storleken på den mittersta regionen justeras efter behov med ett tröskelvärde. Metoden sparar placering av varje objekt i form av en sträng i följande format: *klassnamn + placering.* Därefter returneras en bitmap som består av två linjer för att visa storleken på varje region (se resultat för bilder).

Den andra metoden kallar vi för positionOnRoad. Syftet med denna metod är att beräkna var användaren befinner sig på vägen och ge instruktion om att gå till vänster, höger eller hålla den nuvarande positionen. För att åstadkomma detta utnyttjades Canny kantdetektering samt Houg linjetransformering (mer om dessa metoder kan läsas i teknisk bakgrund). Metoden positionOnRoad går ut på följande 7 steg:

1. Konvertera väg-masken till Mat format
2. Konvertera till gråskala
3. Applicera Canny kantdetektering
4. Utnyttja Hough linjetransformering
5. Filtrera ut linjer
6. Beräkna skärningspunkt
7. Avgör var användaren befinner sig på vägen

Det första steget var att konvertera masken i Bitmap-format till Mat-format för att kunna använda metoderna i OpenCV. Bilden konverteras sedan till gråskala och Canny kantdetektering appliceras därefter med funktionen Canny från OpenCV. Nästa del i funktionen PositionOnRoad var att utnyttja Hough linjetransformering till att hitta vägens yttre kanter. Från OpenCV användes funktionen HougLines för detta ändamål. Denna funktion returnerar många linjer som inte tillhör vägens kanter. Att höja tröskelvärdet medför att färre linjer detekteras men det kan även leda till att linjer som tillhör vägens kanter tas bort. Av denna anledning krävdes en annan metod för att filtrera ut linjerna. Genom att beräkna lutningen samt längden för varje linje var det möjligt att avgöra om linjen tillhörde vägens kant. Om lutningen är större än 0,5 tillhör linjen den högra sidan av vägen. Om lutningen däremot är mindre än 0,5 är linjen på den vänstra sidan av vägen. Detta reducerade antal linjer avsevärt, men ytterligare filtrering krävdes för att endast få en linje för den vänstra- och högra kanten av vägen. För att åstadkomma detta, används endast de längsta linjerna i nästa steg.

Efter att två linjer för vägens kanter har tagits fram, beräknas linjernas skärningspunkt med hjälp av determinanten. Formeln för två linjer i Hough rummet kan beskrivas med matriser enligt nedan:

$$\overset{A}{\begin{bmatrix} x \\ y \end{bmatrix}}\overset{B}{\begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ \cos(\theta_2) & \sin(\theta_2) \end{bmatrix}} = \overset{C}{\begin{bmatrix} r_1 \\ r_2 \end{bmatrix}}.$$

Skärningspunkten är (x, y) och fås genom att multiplicera inversen av matris B med matris C. Inversen av matris B beräknas enligt:

$$B^{-1} = \det(B)\begin{bmatrix} \sin(\theta_2) & -\sin(\theta_1) \\ -\cos(\theta_2) & \cos(\theta_1) \end{bmatrix}$$

där

$$det(B) = \frac{1}{\cos(\theta_1) * \sin(\theta_2) - \sin(\theta_1) * \cos(\theta_2)}.$$

Därefter beräknas skärningspunkten enligt följande:

$$\begin{bmatrix} x \\ y \end{bmatrix} = B^{-1}\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \det(B)\begin{bmatrix} \sin(\theta_2) & -\sin(\theta_1) \\ -\cos(\theta_2) & \cos(\theta_1) \end{bmatrix}\begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$$
$$= \frac{1}{\cos(\theta_1) * \sin(\theta_2) - \sin(\theta_1) * \cos(\theta_2)}\begin{bmatrix} \sin(\theta_2)\, r_1 - \sin(\theta_1) r_2 \\ \cos(\theta_1) r_2 - \cos(\theta_2)\, r_1 \end{bmatrix}.$$

Den slutgiltiga formeln används i funktionen *getIntersectionPoint* för att beräkna skärningspunkten för linjerna. Efter att skärningspunkten har beräknats, dras en linje från punkten till en punkt längst ner i mitten av skärmen. Denna linje används för att avgöra var användaren befinner sig på vägen. Om lutningen på linjen är mindre än ett tröskelvärde, ska användaren rotera till vänster. Om lutningen är större än ett tröskelvärde, ska användaren rotera till höger. Vid fallen då lutningen varken är större än eller mindre än tröskelvärdet, är användaren i mitten av vägen. Instruktionerna sparas i en sträng och metoden returnerar tre linjer, den vänstra kant-linjen, den högra kant-linjen och linjen som går från punkten längst ner i mitten av skärmen till skärningspunkten. Användaren får sedan instruktioner via text-till-tal som exempelvis *”rotate left”*, *”rotate right”* eller *”middle”*.

4.4 Navigering med Mapbox

Denna del beskriver hur biblioteket Mapbox användes i Android-applikationen. Följande funktionalitet implementerades: karta, navigering, instruktioner och sökfunktionalitet.

4.4.1 Installation av Mapbox

För att implementera GPS-navigering i Android-applikationen, användes Mapbox Android SDK. Det första steget för att kunna använda Mapbox var att skapa ett Mapbox-konto för att generera en tillgångs-nyckel (eng. *access token*). Utan denna nyckel var det inte möjligt att få tillgång till data från Mapbox API.

4.4.2 Implementation av kartan

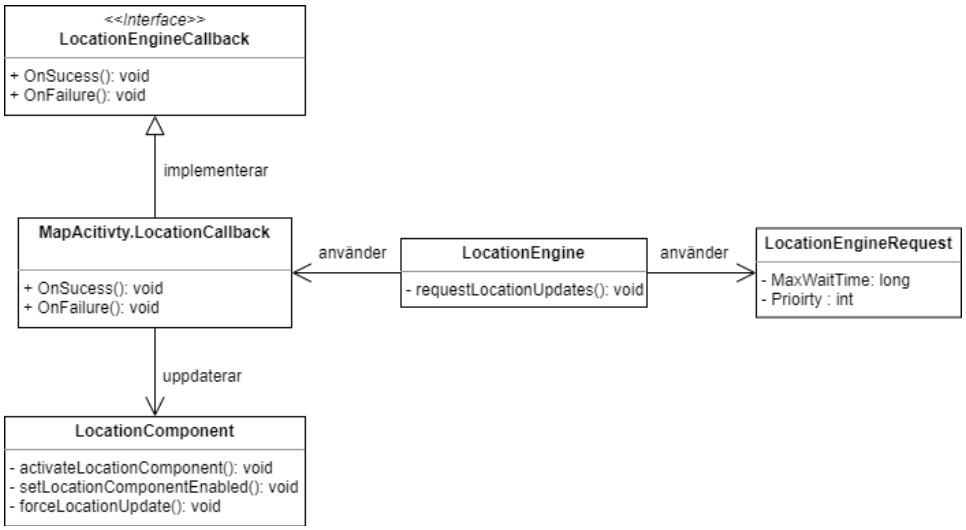
För att skapa en karta med Mapbox, krävs en *MapView* som används för att visualisera kartan och en *MapboxMap* som innehåller datan för kartan. I applikationens layout-xml skapas en MapView som sedan refereras i Java-kod genom funktionen *findViewbyID(R.id.mapview)*. Genom funktionen *getMapAsync* är det sedan möjligt att starta kartan. Efter att Mapbox kartan laddats in, kallas funktionen *OnMapReady* där stilen på kartan ställs in med funktionen *setStyle*. Efter att kartans stil har initierats, kallas funktionen *OnStyleLoaded*. Denna funktion utnyttjar funktionerna *EnableLocationComponent*, *InitSource*, *InitLayers* och *InitSearchFab* som hanterar visuella delar av kartan och sökrutan.

4.4.3 Uppdatering av användares position

För att kunna uppdatera användarens position med Mapbox SDK, skapas en instans av klassen *LocationEngine*. Sedan byggs en *LocationEngineRequest* med följande parametrar:

- Noggrannhet i uppdateringarna av användarens position
- Maximala vänte-tiden för uppdateringarna

Därefter skapades klassen *LocationCallback* som implementerar gränssnittet *LocationEngineCallback*. Denna klass hanterar uppdateringar av användarens position på kartan genom att uppdatera *LocationComponent* (pucken på kartan) med funktionen *ForceLocationUpdate.* Samtidigt sparas koordinaterna av denna position för att användas i navigeringen. Efter att alla klasser skapats, initieras uppdateringen av användarens position med funktionen *requestLocationUpdates* från LocationEngine med LocationCallback och LocationEngineRequest som parametrar. Figur 17 på nästa sida, visar relationen mellan dessa klasser.

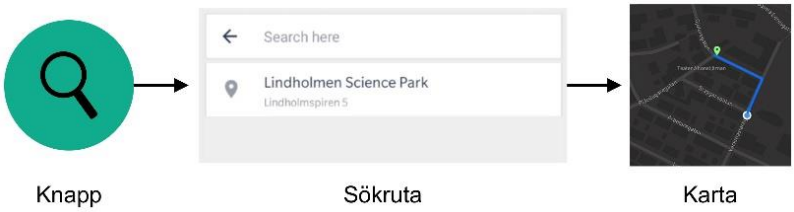


Figur 17. UML-diagram som visar relationen mellan Mapbox-klasser som används för platsuppdatering.

4.4.4 Interaktion med kartan och sökfunktion

Efter att användarens platsinformation framtagits, krävs en destination för att påbörja en navigerings-session i Mapbox. Användaren kan ange en destination genom att antingen klicka på kartan eller genom att ange destinationen i ett textfält. För att tillåta användaren att interagera med kartan användes en *MapClickListener*. Denna lyssnare kallar på funktionen *OnMapClick* varje gång ett tryck registreras på kartan. Positionen av trycket tas in som en parameter i funktionen och destinationen uppdateras därefter med denna position. Därefter skapas en färdväg mellan användarens position och destinationen med funktionen *generateRoute*.

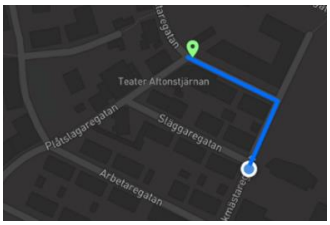
Den alternativa metoden för att generera en destination var genom att använda en sökfunktionalitet från Mapbox. Det första steget för att implementera denna funktionalitet var att skapa en knapp som öppnar en sökruta när den registrerar ett tryck från användaren. Sökrutan implementerades genom att skapa en ny instans av klassen *PlaceAutocomplete* från Mapbox. När användaren klickar på knappen, skapas en ny aktivitet med en textruta där användaren kan ange sin destination. Dessutom visas tidigare destinationer i en lista. Efter att användaren har valt en destination, aktiveras funktionen *onActivityResult*. Denna funktion får information om destinationen genom en *CarmenFeature*. Det är möjligt att få ut både namnet på platsen samt dess position från denna klass. På samma sätt som i OnMapClick, uppdateras kartan och en ny färdväg genereras med funktionen generateRoute. För en person med synnedsättning är det möjligt att använda tal istället för skrift för att ange en destination i textfältet. Denna funktionalitet finns redan tillgänglig i Android-mobiltelefoner och är kompatibel i den framtagna applikationen. Figur 18 nedan visar hur användaren kan söka och välja en destination i en sökruta.



Figur 18. Användaren kan skapa en färdväg genom att klicka på knappen och ange en destination i textfältet.

4.4.5 Skapandet av en färdväg

För att generera en färdväg mellan användarens position och destination skapades funktionen *generateRoute*. Funktionen skapar en instans av *MapboxDirections* och använder dess builder-funktion för att välja vilken typ av väg som skall genereras. Därefter används funktionen *enqueueCall* för att hämta färdvägen från Mapbox API. Vid fallen då en giltig respons mottagits, tas en färdväg fram med funktionen *response.body().routes().get(0)*. Färdvägen kan därefter visualiseras genom att skapa en linje på kartan som visas i figur 19 nedan.



Figur 19. Färdvägen visualiseras på kartan i form av en blå linje mellan användarens position (blå puck) och destination (grön vägpunkt)

4.4.6 Navigering till destination

Efter att en färdväg har hämtats från Mapbox-API, kan den användas för navigering. För att hantera navigeringen skapades följande funktioner: *initNavigation*, *startNavigation* och *stopNavigation*. Första gången navigeringen startas, aktiveras funktionen InitNavigation som initierar navigeringen genom att skapa en instans av klassen *MapboxNavigation*. Denna klass tar in en parameter kallad *NavigationOptions* som använder API-nyckeln och referens till en *LocationEngine*. För att få återkoppling från navigeringen, kan flera observers användas. I applikation används följande observers:

- *ArrivalObserver*
- *VoiceInstructionsObserver*
- *RouteProgressObserver*

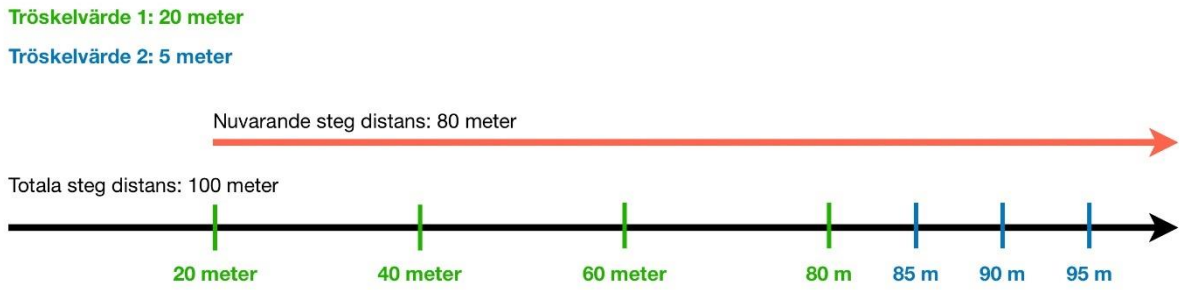
Mer om hur dessa observers fungerar, kan läsas i teknisk bakgrund. För att starta navigeringen, används funktion startNavigation. Denna funktion initierar klassen MapboxNavigation med en färdväg. Därefter startas en navigerings-session med funktionen *startTripSession* från Mapbox. En pågående navigerings-sessionen stoppas när användaren nått sin destination genom att använda ArrivalObserver. För att informera användaren om att destinationen är uppnådd, implementerades TTS-funktionalitet. Ett exempel på detta kan vara: *”Arrived at destination”*.

För att ge användaren instruktioner under navigeringen, användes VoiceInstructionsObserver. När användaren når ett nytt steg i navigerings-sessionen, kallas funktionen *newVoiceInstructions* som returnerar instruktioner. Under de tester som utfördes, upptäcktes att instruktionerna gavs för tidigt i många fall. Därför implementerades en lösning på detta problem genom att skapa en funktion som kontrollerar den kvarstående distansen och totala avståndet till nästa steg i navigeringen. Detta implementerades för att ge nya instruktioner när användaren har förflyttat sig en viss distans. Genom *routeprogress*, som fås av *onRouteProgressChanged*, kunde nuvarande steget hämtas genom följande funktionsanrop:

routeProgress.getCurrentLegProgress().getCurrentStepProgress().getStep().

Steget som returneras från ovanstående funktionsanrop, kunde sedan användas för att spara totala- och nuvarande distansen för steget. Därefter kan instruktioner ges till användaren när det exempelvis var 2 meter kvar till nästa steg. Instruktionen för nästa steg läses sedan upp när *onNewInstructionsObserver* anropas.

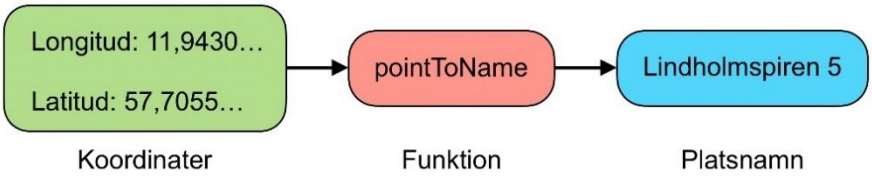
För att uppdatera instruktioner under navigeringen, implementerades en funktion som kontrollerar när användaren förflyttat sig en bestämd distans. Denna funktion subtraherar den totala distansen av ett steg med den nuvarande distansen. Värdet som fås från denna subtraktion kunde sedan användas till att ge användaren instruktioner varje gång användaren förflyttat sig en specifik distans. Figur 20 nedan visar när instruktioner ges till användaren. Den totala distansen för hela steget är 100 meter i detta exempel. Användaren får instruktioner på ett intervall av 20 meter fram tills det är 20 meter kvar till att den totala steg-distansen är uppnådd. Därefter ges instruktionerna på ett intervall av 5 meter.



Figur 20. Exempel på hur avstånden fördelas.

4.4.7 Omvandling från position till platsnamn

För att omvandla en punkt till ett platsnamn, skapades en funktion kallad *pointToName*. Denna funktion tar in en koordinatpunkt (longitud och latitud) och returnerar namnet på platsen som motsvarar punkten. Detta genomförs genom att skapa en *MapboxGeocoding* som tar emot punkten i sin builder-metod. Funktionen *enqueueCall* kan sedan användas för att få en *GeocodingResponse* från Mapbox-API. Först kontrolleras det att responsen är fullständig och sedan kan en *CarmenFeature* tas fram som innehåller namnet på platsen. Namnet returneras därefter av funktionen. Figur 21 visualiserar omvandlingsprocessen med ett exempel.



Figur 21. Ett exempel på hur koordinater omvandlas till namnet på platsen.

4.5 Text-till-tal

För att implementera text-till-tal (TTS) funktionalitet i applikationen, skapades klassen *VoiceFeedback* som skapar en instans av klassen *TextToSpeech* från Android. Språket för TTS-motorn kan ställas in med funktionen *setLanguage*. I klassen *VoiceFeedback* implementerades funktionerna *startSpeech* och *stopSpeech* för att enklare hantera TTS-meddelanden. Båda funktionerna använder metoder från TextToSpeech. Funktionen startSpeech tar in en sträng samt ett heltal som avgör prioritet för meddelandet. Med denna implementation kommer meddelanden med hög prioritet avbryta meddelanden med låg prioritet. För att kontrollera om ett meddelande fortfarande spelades upp, användes funktionen *isSpeaking* från TextToSpeech. När denna funktion returnerar *False*, spelas ett nytt TTS-meddelande upp med funktionen *speak*.

4.6 Inställningar

Inställningarna i applikationen implementerades med biblioteket *preference* från AndroidX. Det första steget var att skapa en xml-fil för de grafiska gränssnitten (settings.xml) och en xml-fil för datan som inställningarna ska byta mellan (arrays.xml). Tabell 3 nedan visar inställningarna som implementerades i applikationen:

Tabell 3. Alla inställningar i applikationen.

	Inställning	**Komponent**	**Typ**	**Förklaring**
1	Stil på kartan	List	String	Ändrar stil på kartan.
2	Visa beräkningstid	Switch	Boolean	Aktiverar/inaktiverar text som visar beräkningstid för modellen.
3	Aktivera segmenteringen	CheckBox	Boolean	Aktiverar/inaktiverar segmenteringen.
4	Välj segmentering	List	String	Ändrar segmenteringsmodell.
5	Visa segmentering	Switch	Boolean	Aktiverar/inaktiverar visning av maskerna från segmenteringen.
6	Val av klasser	MuliSelectList	String	Väljer vilka masker som ska skapas.
7	Metod	List	String	Ändrar metod för analys av maskerna: ingen metod, objectLocations eller postionOnRoad.
8	Tröskelvärde	EditText	String	Anger tröskelvärdet för funktionen objectLocations.
9	Visa linjer	Switch	Boolean	Aktiverar/inaktiverar visning av linjerna från funktionerna objectLocations eller postionOnRoad.
10	TTS hastighet	List	String	Ändrar hur ofta TTS-återkoppling ges till användaren.

En klass som vi kallar för *SettingsFragment* implementerades främst för två ändamål. Klassen är en typ av fragment som skapas i klassen *MainApp*. I SettingsFragment appliceras en lyssnare på komponent 8, se tabell 3 ovan, för att enbart tillåta heltalsvärden. Klassen har även två funktioner som inaktiverar komponenter som endast ska vara aktiva då en annan komponent är påslagen. När inställning 3 är inaktiv, kommer inställning 4, 5, 6, 7, 8, 9, 10 inaktiveras, vilket innebär att användaren inte kommer kunna interagera med dessa inställningar. När inställning 7 är inaktiv kommer även inställning 8 vara inaktiv.

5
Resultat

I detta avsnitt, redovisas resultatet av arbetet. Dessa resultat var från träning/testning av modeller och resultatet av alla delar i applikationen.

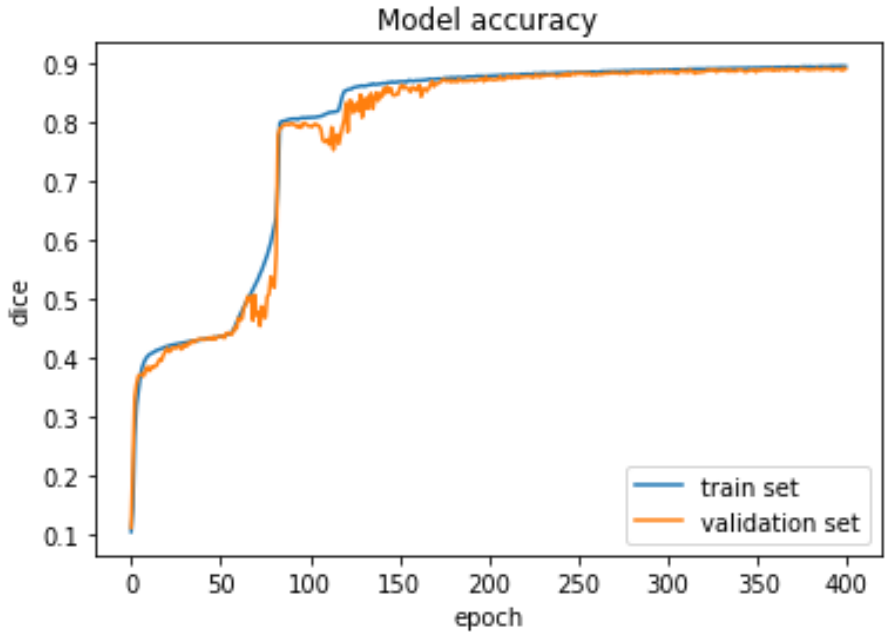
5.1 Resultat från träning av segmenteringsmodeller

I tabell 4 nedan, visas resultatet av träningen för 6 olika segmenteringsmodeller. Resultatet från träningen av modellerna visar på att precisionen uppnåddes till mellan 82-89 % för valideringsmängden i alla tester. Modellen som gav högst precision (89 %) var test 6. I tabellen nedan framgår det att små förändringar i både antal epoker och batch-storlek, har en minimal påverkan på precisionen, vid jämförelse mellan test 2 och 3. Överlag visar resultatet att dice-loss gav upphov till bättre presterande modeller, med avseende på precision. Det framgår även att högre upplösning på indatan inte nödvändigtvis leder till bättre resultat, se test 5 och 6. Precisionen försämrades när inlärningshastigheten sänktes från 10^{-4} till 10^{-5}.

Tabell 4. Resultat från träningen av 6 olika modeller.

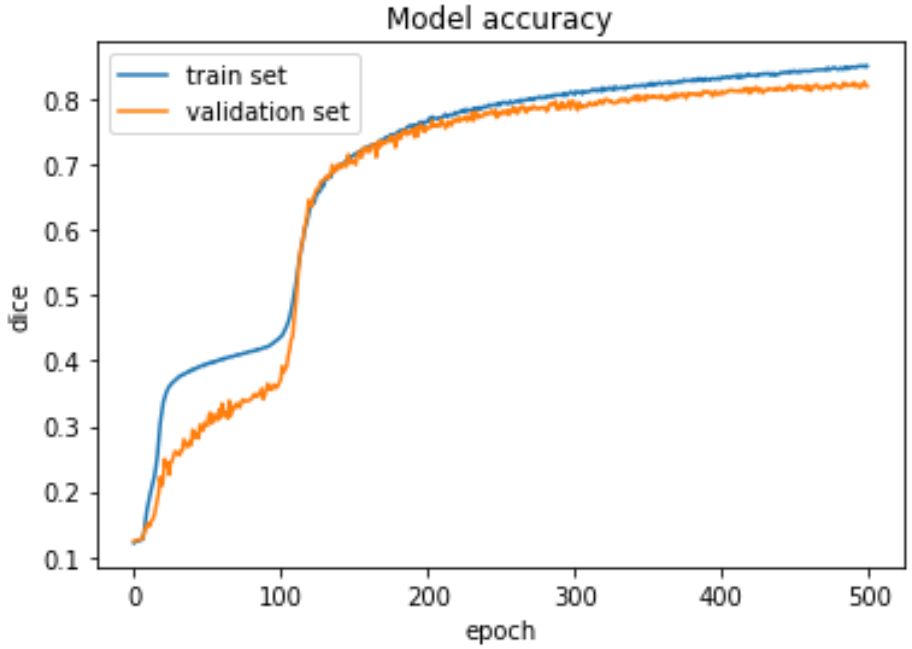
Test	Upplösning	Epoker	Batch-storlek	Förlustfunktion	Inlärningshastighet	Precision (%)
1	128x128	500	30	categorical crossentropy	10^{-4}	88,73
2	128x128	600	30	categorical crossentropy	10^{-5}	82,67
3	128x128	500	10	categorical crossentropy	$5 * 10^{-5}$	82,67
4	64x64	600	30	dice loss	$5 * 10^{-5}$	85,52
5	256x256	400	10	dice loss	$5 * 10^{-5}$	88,49
6	128x128	400	10	dice loss	$5 * 10^{-5}$	89,20

Figur 22 på nästa sida visar en graf på hur precisionen förändras under träningen för test 6. Grafen visar att precisionen ökar drastiskt på bara några få epoker i början av träningen, kort därefter planar den ut. Efter cirka 60 epoker sker ytterligare en stark förändring fram till omkring 90 epoker då precisionen inte förändras förens efter 120 epoker. Efter cirka 175 epoker började förändring av precision avta och modellen närmade sig ett stabilt läge. Modellen uppnådde en precision på 89% för validerings-mängden . Detta innebär att förutsägelserna från modellen endast skiljer sig med 11% från dem sanna maskerna.



Figur 22. Precision för varje epok under träningen för test 6. Detta test använde dice-loss som förlustfunktion.

Modellerna tränades med 2 olika förlustfunktioner: *categorical cross entropy* och *dice-loss*. När categorical cross entropy användes, krävdes 400-500 epoker för att uppnå resultat med en precision över 80%. Detta visualiseras i figur 23 på nästa sida. Dessutom framkom överanpassning i dessa modeller vilket resulterade i att värdet på förlusten ökade mot slutet av träningen. I de modeller där dice-loss användes som förlustfunktion, blev träningen mer stabil och en precision på över 80 % kunde uppnås efter cirka 100 epoker, se figur 22.

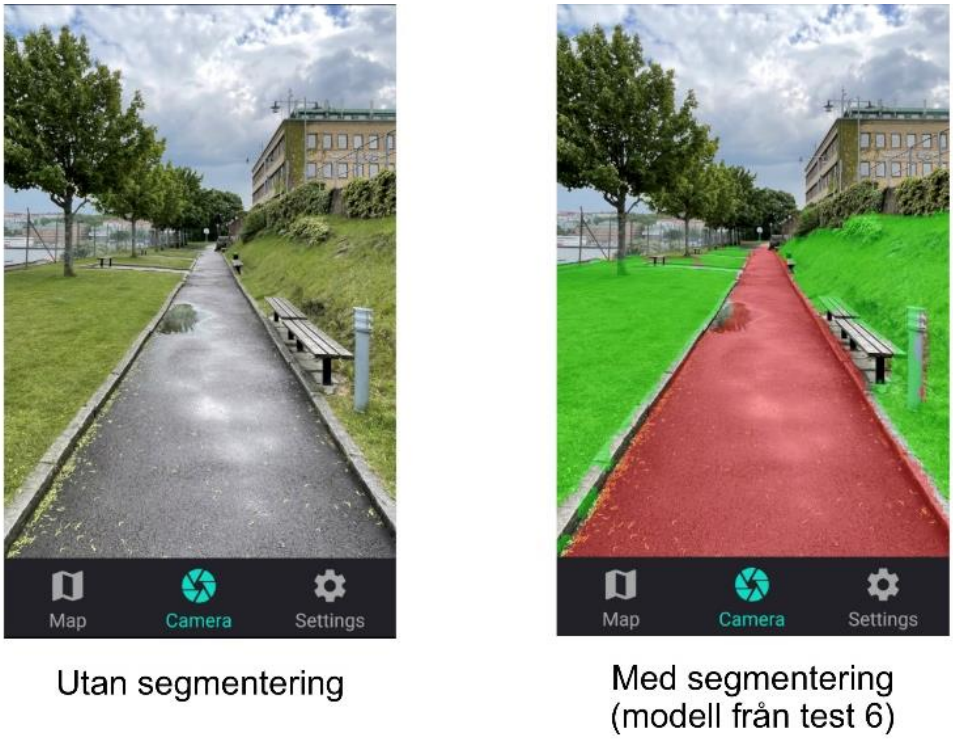


Figur 23. Precision för varje epok under träning för test 3. Detta test använde categorical cross-entropy som förlustfunktion

Modeller med inlärningshastighet på över 10^{-4} resulterade i att precisionen kunde sjunka under träningen och att modellen överanpassas efter träningsdatan. Ett för lågt värde på inlärnings-hastigheten, exempelvis 10^{-6}, innebar att det tog fler epoker för att uppnå en precision på över 80 %.

5.2 Resultat från tester av modellen

Alla modeller som framställdes i detta arbete, kunde identifiera både vägar och ytor av gräs utan problem. Både bilar och trottoarer kunde upptäckas vid ett fåtal tillfällen med modeller som använde *categorical cross entropy* som förlustfunktion. Klasserna som modellerna hade svårt att identifiera, klassificerades felaktigt vid några tillfällen under testning. Följande klasser kunde inte identifieras av någon modell: lyktstolpe, person och sten. När dice-loss introducerades, blev maskerna för väg och gräs något mer lik den sanna masken. Däremot visade sig att både bilar och trottoarer inte längre kunde identifieras. Segmenteringsmodellen från test 6 visas i figur 24 nedan. I figuren framgår det att modellen kan identifiera vägen och gräs med hög precision. Viktigt att notera är att testerna utfördes på vägar som inte framkommer i datamängden som användes för träningen av modellen.



Figur 24. Till vänster i figuren visas en skärmavbild på kamera-delen i applikationen utan segmentering när kameran var riktad mot en väg. Till höger i figuren har segmenteringsmodellen från test 6 aktiverats. Röd mask motsvarar klassen väg och grön mask motsvarar klassen gräs.

Figur 25 nedan visar resultatet från segmenteringen på en väg som täcks av skuggor. Notera även att gräs inte är lika framträdande som i figur 24. Det framgår nedan att skuggor bidrar till att modellen får svårt att avgöra vad som tillhör vägen, men att majoriteten av vägen samt gräs ändå kan identifieras av modellen.

Figur 25. Till vänster i figuren visas en skärmavbild på kamera-delen i applikationen utan segmentering när kameran var riktad mot en väg täckt av skuggor. Till höger i figuren har segmenteringsmodellen från test 6 aktiverats. Röd mask motsvarar klassen väg och grön mask motsvarar klassen gräs.

5.3 Resultat från tester med olika upplösning på modellen

Storleken på indatan för modellerna som tränades i detta arbete var främst: 128x128 och 256x256. Mindre eller större bildstorlek resulterade i låg precision i förutsägelserna eller att modellen blev för krävande för mobiltelefonen. En modell av storleken 64x64 skapades i syfte att undersöka hur snabbt den kunde exekveras på den använda hårdvaran. Denna modell visade sig vara betydligt snabbare jämfört med modellerna som hanterar större upplösning på indatan. Nackdelen med denna modell var att segmenteringen inte blev lika detaljerad. Modellerna som använder bilder med storlek 256x256 hade fördelen att de kunde urskilja klasser med högre precision jämfört med modeller som behandlar bilder med lägre upplösning. Dessa modeller krävde mycket processorkraft från mobiltelefonen vilket bidrog till en lägre uppdateringsfrekvens. Modellerna som hanterade bilder med storlek 128x128 kunde urskilja klasserna med tillräckligt hög precision för att användas för vidare analys av maskerna. Dessutom var uppdateringsfrekvensen för dessa modeller relativt hög. I tabell 5 nedan visas uppdaterings-frekvensen för modellerna med olika bildstorlek i applikationen när dem exekveras på en Oneplus Nord 100.

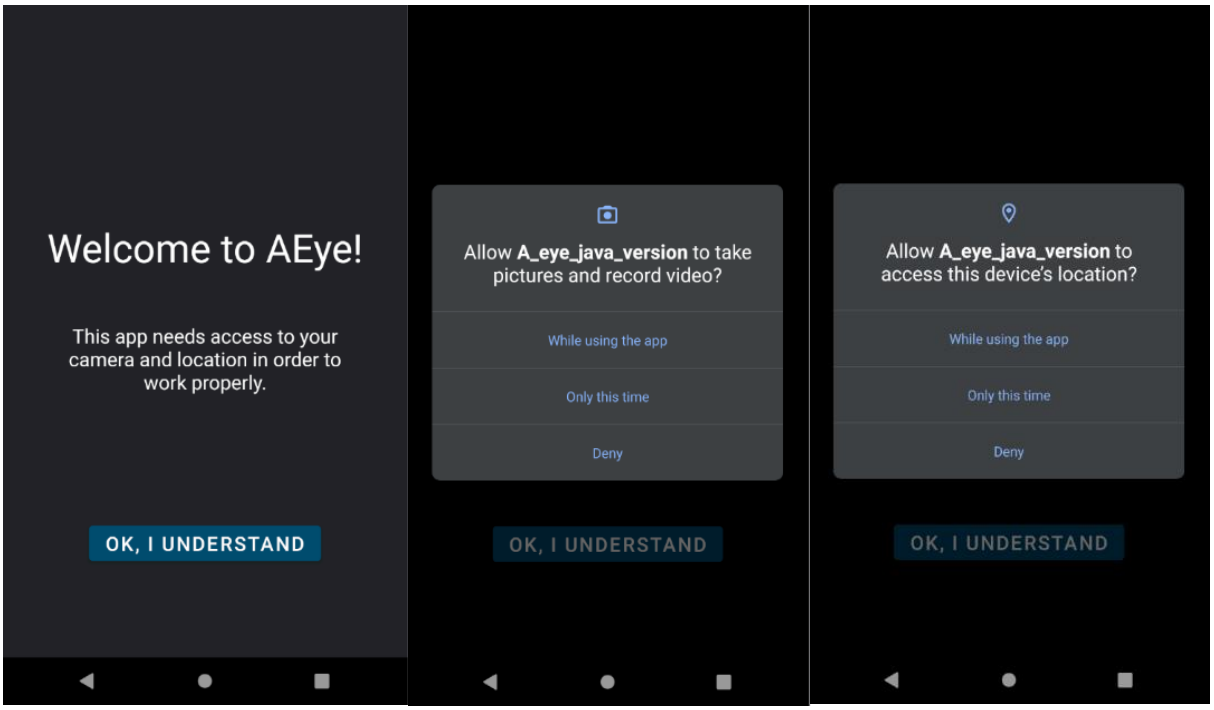
Tabell 5. Beräkningstid för de olika modell-storlekarna

Upplösning	Uppdateringsfrekvens (millisekunder)
64x64	~ 140
128x128	~ 300 - 400
256x256	~ 1000 - 1400

5.4 Resultat av den slutgiltiga applikationen

Applikationen har ett användargränssnitt som är anpassat för personer med synnedsättning genom att implementera extra stora knappar och enkel navigation inom applikationen. Dessutom är applikationen kompatibel med Androids stödfunktionalitet *”talkback”*. Se bilaga 3 för information om programstrukturen.

Startsidan upplyser användaren att applikationen kräver behörighet att använda kameran och plats-information. Efter att knappen “OK, I UNDERSTAND” har tryckts, visas en dialogruta där användaren kan välja om applikationen ska få tillgång till det som efterfrågas. Applikationen går vidare till huvuddelen först efter att både användning av kameran och plats-information har accepterats. Figur 26 nedan visar utseendet av denna sida.

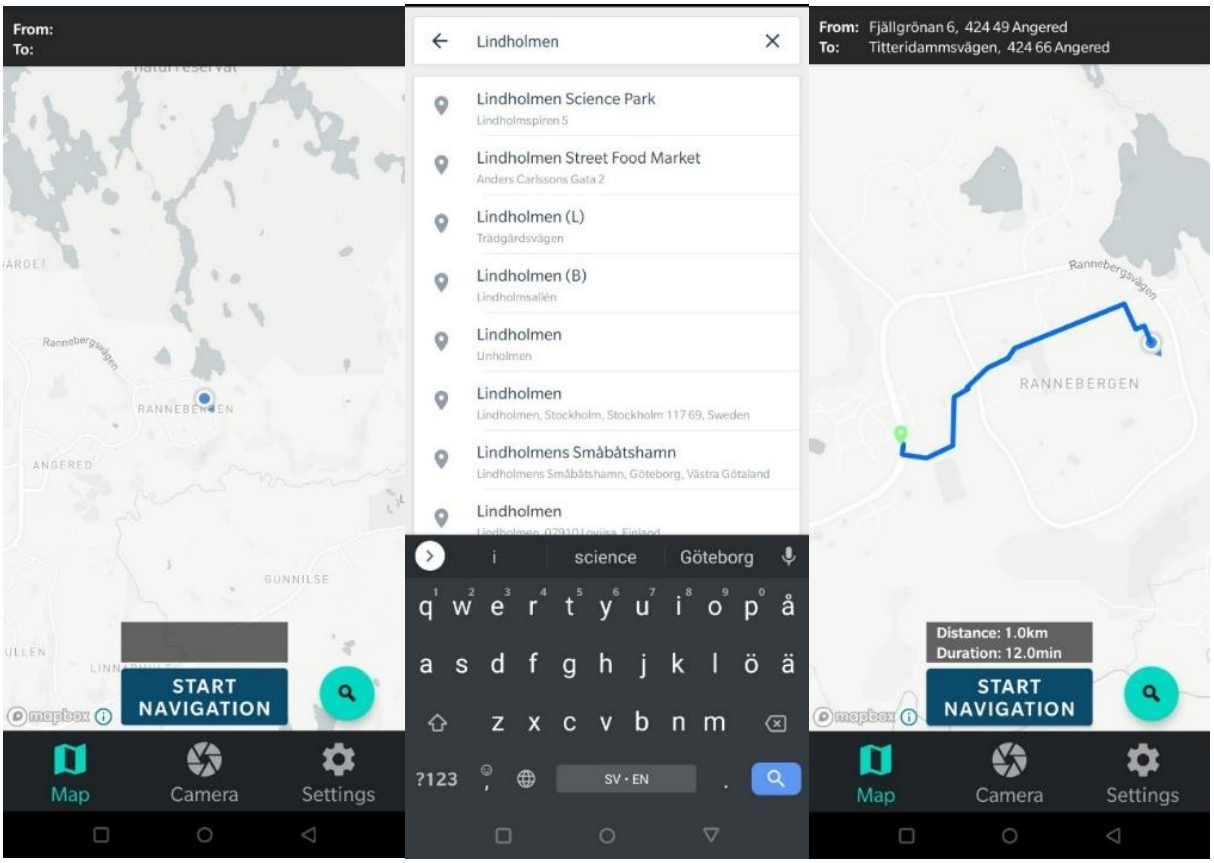


Figur 26. Startsidan i Android-applikationen frågar efter behörighet att använda kameran och användarens position.

Efter att användaren har godkänt att applikationen får använda enhetens platsinformation och kamera, kommer huvud-applikationen starta som består av en sida för kartan, en sida för kameran och en sida för inställningar. Sidan för kartan i applikationen består av en Mapbox-karta där användarens position uppdateras och visualiseras.

Användaren kan antingen klicka på kartan för att välja en destination eller söka efter en plats. När en destination valts visualiseras vägen genom en linje på kartan och knappen för att starta navigeringen blir aktiv. Navigeringen kan sedan startas genom att trycka på knappen “START NAVIGATION”. Under en navigerings-session är det inte möjligt att välja en ny destination. Först när användaren har avslutat en pågående session, kan en ny destination anges. När navigeringen startas, får användaren instruktioner i form av text-till-tal meddelanden för att navigera till sin destination. Instruktionerna anges först med ett meddelande följt av avståndet till nästan steg. Ett exempel på en instruktion kan vara följande: *“continue forward for 50 meters”*. Efter att användaren nått en sväng, ges en instruktion

om att svänga exempelvis: *"turn left"*. Under navigeringen fås nya instruktioner mer frekvent när användaren befinner sig nära nästa steg. När det är över 20 meter kvar till nästa steg, ges instruktioner på ett intervall av 20 meter. När användaren befinner sig mindre än 20 meter till nästa steg, ges instruktioner på ett intervall av 5 meter. Figur 27 nedan visar kart-sidan av applikationen.



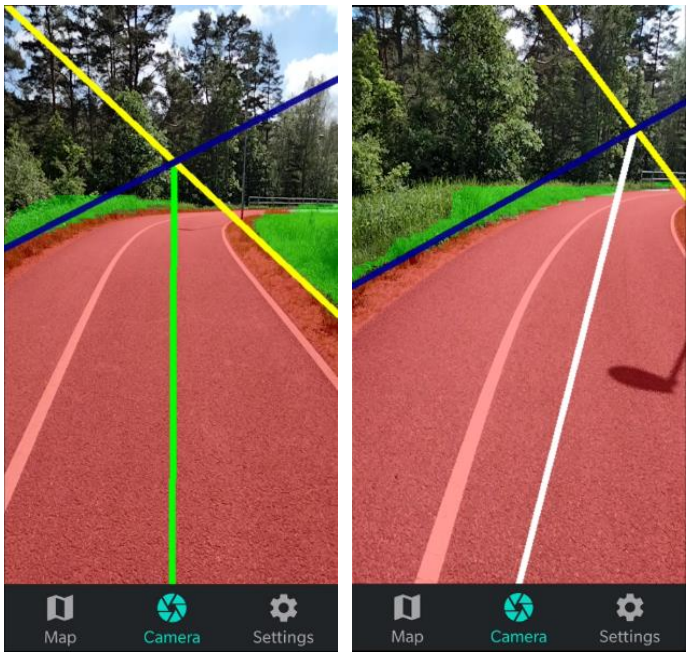
Figur 27. Kart-sidan av applikationen.

Vid enstaka fall gavs instruktioner som kan vara svåra att tolka av personer med synnedsättning. Exempelvis instruktioner som: *"Walk east, then turn right"*. Det framkom även emellanåt att instruktionerna som fås när användaren förflyttat sig, gavs för sent eller ignorerades helt.

5.5 Sida för kameran

Sidan för kameran i applikationen består av en förhandsvisning av kameran, masker från segmenteringen samt linjerna från de olika implementationerna av bildanalys. I applikationen kan användaren välja vilken metod som ska användas. Den ena metoden *positionOnRoad* och den andra var *objectLocations*. Båda metoderna ger återkoppling till användaren via text-till-tal. Viktigt att notera är att metodernas prestation är direkt kopplad till segmenteringsmodellernas precision i förutsägelserna.

Metoden positionOnRoad använder Canny kantdetektering tillsammans med Hough linjedetektering på utdatan från segmenteringsmodellen för att beräkna mittpunkten av gångvägar. Genom att kontrollera mittpunkten av vägen kunde användarens relation till vägens mittpunkt kontrolleras. Denna metod förutsätter att kameran är riktad i samma riktning som användaren. När användaren befinner sig för långt åt höger/vänster eller håller på att gå av vägen, kan applikation meddela användaren om detta. Återkopplingen sker i form av enkla meddelanden via text-till-tal. När användaren inte är riktad i mot vägen, kommer applikationen ge kommando till användaren att rotera. När användaren befinner sig centrerat på vägen kommer applikationen meddela användaren att hålla denna riktning. Figur 28 nedan visar två bilder på applikationen när kameran är riktad mot en väg som svänger av till höger. Det gröna strecket indikerar att användaren ska hålla sin riktning. När strecket är i färgen vit, bör användaren rotera till antingen höger eller vänster, i figur 28 kommer ett text-till-tal meddelande ges till användaren om att rotera till höger med text-till-tal meddelandet: *”rotate right”*.



Figur 28. Metod för att identifiera vägens mittpunkt.

En gräns sattes på hur ofta information förmedlas till användaren. Denna uppdateringsfrekvens kan ställas in av användaren till följande värden, 1 2, eller 5 sekunder.

Att identifiera vägens mittpunkt blir problematiskt när vägen täcker hela skärmen. I dessa situationer identifierades inte alltid mittpunkten, alternativt returnerades ett felaktigt resultat.

Resultatet från metoden *objectLocations* visas i figuren nedan. Denna metod kan ge användaren återkoppling om var majoriteten av klasserna befinner sig på skärmen. Återkopplingen till användaren sker via text-till-tal. Exempelvis i figur 29 nedan, ges följande information till användaren via text-till-tal: *"Grass left, Road middle"*.

Figur 29. Metod för att hitta klassernas position relativt till användaren.

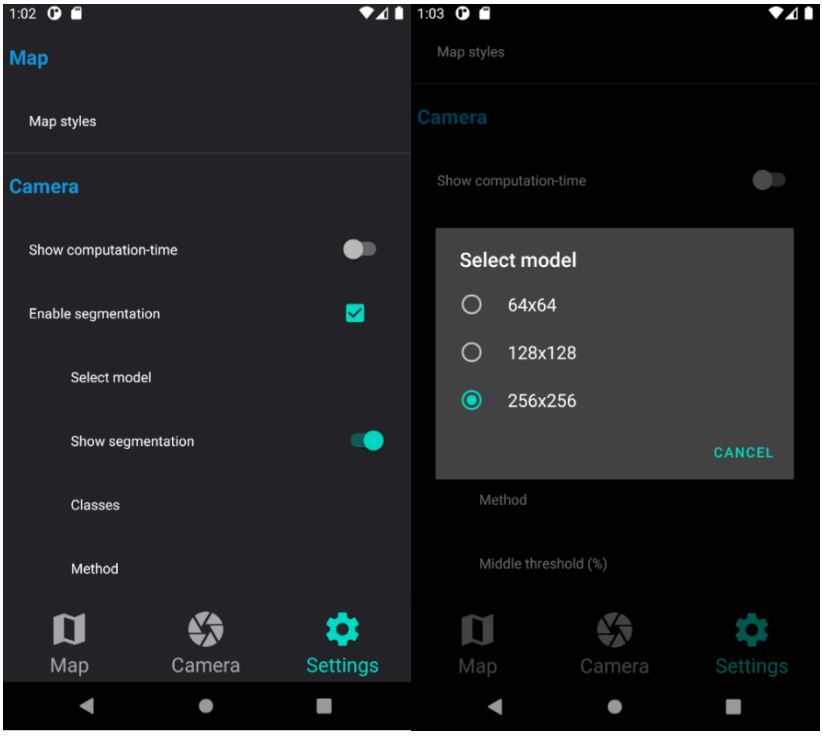
Dessa metoder fungerade bra när de används på vägar i bostadsområden. I stadsmiljöer där gräs inte framkommer lika ofta och på vägar bestående av stenblock, kunde resultatet av segmenteringen variera. Detta påverkade prestationen av båda metoder.

5.6 Inställningssidan

För att hantera inställningar i Android-applikationen används inställningssidan. Sidan innehåller följande inställningar:

- Still på Mapbox karta
- Visa beräkningstid för segmenteringen
- Aktivera segmentering
    - Val av modell
    - Visualisering segmenterings resultat
    - Klasser som är aktiverade för segmentering
    - Val av tolkning metod av segmenteringen
    - Storlek av mittensektion
    - Visa linjer
    - Hastighet av Segmentering feedback

Figur 30 nedan visar utseendet av inställningssidan i applikationen.



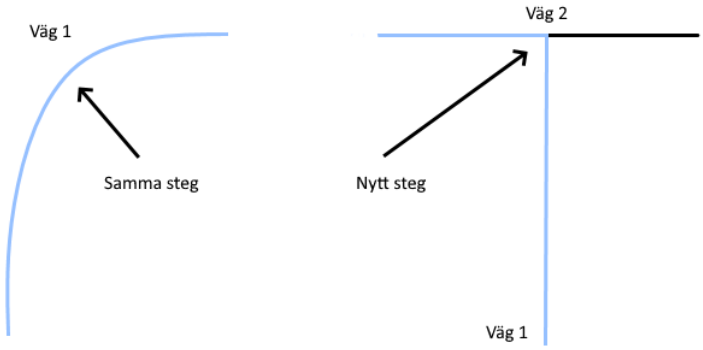
Figur 30. Sidan för inställningar i applikationen.

5.7 Resultatet från tester av Android-applikationen

Efter att alla system i applikationen var implementerade, var nästa steg att testa applikationen i praktisk användning. Resultaten från dessa tester är indelade i olika delar.

Under första testet av applikationen framstod ett flertal problem. Ett problem var att instruktionerna om att svänga gavs tidigt. Till följd av detta missades svängar under testerna. Metoderna som utnyttjar bildsegmenteringen kan motarbeta detta något men det var fortfarande ett stort problem.

Efter att ytterligare arbete lagts på Mapbox-navigeringen, implementerades en lösning som innebar att användaren får instruktioner om att svänga vid rätt tidpunkt. Ett problem som kvarstod var att GPS-navigeringen hade svårt att urskilja om en väg är rak eller svänger vid ett flertal tillfällen. Detta resulterar i att användaren inte får tillräckligt med instruktioner i alla situationer. Figur 31 nedan visualiserar detta problem.



Figur 31. Väg 1 till vänster visar en sväng som tolkas som samma steg i Mapbox. Till höger visas ett exempel där två vägar identifieras och därför tolkar Mapbox detta som ett nytt steg.

När både modellen och GPS-navigeringen testades samtidigt, framkom det att segmenteringen kunde användas för att upptäcka när vägen svänger av, vilket GPS-navigeringen hade problem med.

6
Diskussion

I denna del diskuteras och evalueras resultatet av arbetet. Här värderas även arbetets relation till etik och miljö.

6.1 Begränsad data och resurser

I detta arbete skapades en egen datamängd utifrån egna framtagna bilder. Detta innebär att storleken på datamängden blev begränsad eftersom detta arbete genomfördes av endast två personer. Denna begränsning leder till att modellen blir sämre anpassad till olika typer av miljöer. Även datorerna som användes för träningen hade begränsningar i prestanda. Att träna en modell kunde ta 5-10 timmar på våra datorer även om träningen kördes på GPU:n. Senare i arbetet användes därför Googles GPU:er i Google colab. Det fanns även en tidsbegränsning på användandet av GPU:n från Google. Vid ett flertal tillfällen fick vi vänta mellan 1-4 timmar innan träningen kunde fortsätta. Även med Googles GPU:er kunde träningen med vår datamängd på endast 312 bilder ca 3 timmar.

6.2 Resultat av bildsegmentering

Av resultatet från bildsegmenterings-modellerna framgår det att precisionen i förutsägelserna låg överlag på en hög nivå (82-89 %). Testerna visade även på att modellerna gav ett tillförlitligt resultat eftersom maskerna passar in väl med objekten i test-bilderna.

Problemet med att alla klasser inte kunde identifieras kan ha flera förklaringar. En orsak till problemet kan vara hur de “korrekta” maskerna skapades. Bilderna i annoteringsfasen var obehandlade vilket innebar att de hade hög upplösning. När bilderna sedan minskas i storlek, kan information gå förlorad. Detta gäller exempelvis för lyktstolpar, som tar upp en liten del av bilden. Upplösningen på bilderna efter förbehandlingen försämras vilket kunde leda till att det blev svårt för modellen att urskilja speciella särdrag för denna klass.

Ytterligare en förklaring på problemet kan vara att storleken av datamängden var för liten samt att klasserna person, lyktstolpe och sten framkom för sällan i bilderna. Dessutom är klasser som täcker en större del av bilderna enklare att identifiera. En mer balanserad fördelning över hur ofta varje klass framkommer i bilderna samt en större datamängd hade lett ett bättre resultat.

6.3 Övergång från Flutter till Android

Som tidigare nämnts i rapporten, påbörjades utveckling av applikationen först i Flutter och övergick senare i till ett Android-projekt med Java. Denna övergång genomfördes eftersom användningen av OpenCV och Mapbox var begränsat i Flutter. Applikationen i Flutter blev av denna anledning en prototyp över hur olika modeller kunde exekveras på mobiltelefoner. Om arbetet istället hade börjat med att utveckla Android applikationen hade vi haft betydligt mer tid att utveckla en mer fullständig applikation. Detta innebar att vi var tvungna att bortse från viss funktionalitet som exempelvis inomhus navigering.

6.4 Testning av applikation

Det tog längre tid än planerat att utveckla en fullständig version av applikationen. Delvis på grund av övergången till Android. Detta resulterade i att praktisk testning av den fullständiga applikationen genomfördes i slutet av projektet. Detta innebar att många delar av applikationen som fungerade sämre blev tydliga men det fanns begränsad tid att förbättra dessa brister. Om testerna istället hade genomfört tidigare i projektet, är det sannolikt att applikationen hade kunnat utvecklats till ett bättre tillstånd. Applikationen testades även bara av personer utan synnedsättning. För att få mer relevant återkoppling om applikationens brister, hade det varit fördelaktigt om en person med synnedsättning kunde testa applikationen istället. Detta hade även bättre visat hur applikationen fungerar i praktisk användning.

6.5 Begränsningar i GPS-navigeringen

Ett fundamentalt problem med GPS-navigeringen är att endast skarpa svängar eller svängar som resulterar i ett väg-byte ger upphov till att användaren får instruktioner. I resultatet framgår det att GPS-navigeringen inte kan urskilja om en väg är rak eller om det är en kurva. Detta innebär att kurvor som inte korsar med en annan väg, inte tolkas som svängar vilket medför att inga instruktioner ges till användaren. För användare med synnedsättning, innebär detta att navigeringen inte blir lika pålitlig. För att lösa detta måste antingen GPS-navigeringen ge mer exakta instruktioner som exempelvis att en väg är rak eller svänger. Alternativt kan andra metoder som utnyttjar djupinlärning implementeras för att upptäcka svängar och på så sätt göra navigeringen mer pålitlig.

6.6 Applikationens funktionalitet i praktiken

Den slutgiltiga applikationens prestanda varierar beroende på vilken omgivning användaren befinner sig i. Eftersom bilderna för träningsdatan endast tagits i bostadsområden, fungerar modellen sämre exempelvis i stadscentrum. Detta innebär att modellen kan ge bra resultat på specifika platser men sämre på andra platser. Om en synskadad person skulle använda applikationen, kan de därför inte vara säkra på hur bra den fungerar i den valda sträckan. Applikationen är därför tillräckligt bra för att användas som ett “proof of concept” på förbestämda sträckor. Men för användning av personer med synnedsättning skulle ytterligare utveckling krävas för att kunna användas på ett säkert sätt.

6.7 Hårdvarans påverkan på tester

Testerna genomfördes på en mobil med låg prestanda vilket har både för- och nackdelar. En fördel är exempelvis att om applikationen fungerar bra under testningen bör den även fungera på mobiler med bättre prestanda. Nackdelen är att hårdvaran kan begränsa hur väl segmenterings-modellen presterar och därmed leda till att optimeringar behöver implementeras. Dessa optimeringar innebär oftast att modellens hastighet ökar till kostnad av modellens precision i förutsägelserna.

6.8 Storlek av indata

Under träningen av segmenteringsmodellen blev det tydligt att modeller som hanterade bilder av högre upplösning var betydligt mer krävande på hårdvaran. Modeller baserade på bilder med lägre upplösning, kan förlora precision i förutsägelserna. Det är därför upp till utvecklaren att avgöra vad som prioriteras.

6.9 Miljö

Djupinlärning har primärt två aspekter som påverkar miljön: träning av modeller och lagring av data. Modeller som har i syfte att prestera bra i olika miljöer, kräver mer data vilket innebär en större miljöpåverkan. Främst blir det länge träningstider vilket direkt leder till mer elkonsumtion. Större datamängder leder även till mer data som behöver lagras, vilket i sin tur leder till att fler serverdatorer behövs. Exempelvis Google colab som används i detta arbete, tillåter användare att exekvera Python-kod på Googles servrar.

6.10 Etik

Att utveckla ett navigeringssystem för synskadade kan direkt anses vara etiskt. Projektets huvudsakliga syfte var att skapa ett alternativt hjälpmedel till navigering för personer med synnedsättning. Projektet var inte menat för att utvinna vinst utan för att utforska och testa olika metoder för ett navigeringssystem. Eftersom applikationen utnyttjar information om användarens plats (som är en del av den personliga integriteten), är det viktigt att hantera datan på ett korrekt sätt. Den slutgiltiga applikationen har en funktion som innebär att användaren måste godkänna att platsinformation används. Det är även viktigt att meddela användaren vad datan används till och om den delas med tredje part. I detta arbete används plats-information av Mapbox API:n som har en egen integritetspolicy. Eftersom applikationen som utvecklades i detta projekt främst var för testning, behandlades inte integritetsfrågor. Däremot om applikationen skulle utvecklats till en färdig produkt, blir dessa frågor relevanta. En annan viktig aspekt gällande etik är säkerheten av applikationen. För att denna applikation skall kunna användas på ett säkert sätt av synnedsatta användare, måste återkopplingen från applikation vara exakt och fungera i de flesta alla förhållanden och miljöer.

7
Slutsats

Applikationen som skapats under detta projekt är ett *"proof of concept"*. I sitt nuvarande tillstånd saknar den funktionalitet som skulle vara nödvändig för självständig användning av personer med synnedsättning. Exempelvis måste användaren ange en destination via en textruta eller genom att klicka på kartan, vilket kan vara svårt för synskadade. Applikationen saknar även stöd för röstinmatning av instruktioner. Röstigenkänning hade även kunnat användas istället för, eller som ett komplement till knappar så att användaren kan kontrollera olika delar i applikationen. Android innehåller däremot redan stödfunktionaliteten kallad *"talkback"* som är kompatibel med applikationen. Eftersom detta stöd redan fanns tillgänglig, behövdes inte denna funktionalitet implementeras.

Som tidigare nämnts fungerar GPS navigeringen och användaren får instruktioner om att svänga vid korsningar. Ett problem som kvarstod var att applikationen hade svårt att avgöra om en väg är rak eller svänger av. En lösning på detta problem var att använda metoderna som tolkar utdatan från segmenteringsmodellen. En alternativ lösning hade kunnat vara att implementera ett mer utvecklat GPS-system som ger användaren tydligare instruktioner om användarens riktning respektive vägens riktning.

Modellerna som utvecklats i detta projekt har varit begränsade av storleken på den framtagna datamängden samt antalet klasser som behandlats. Bortsett från detta visar resultaten att modellerna ger tillräcklig hög precision för att evaluera och testa den praktiska implementationen i ett navigeringssystem. Det är däremot viktigt att ha i åtanke att en modell som tränats med en större datamängd har större potential i att upptäcka fler klasser med hög precision. Enligt [60] behöver datamängden bestå av 75 - 100 bilder per klass för att uppnå goda resultat i klassificeringen. I detta arbete bestod datamängden av 312 bilder och de flesta klasserna framkom endast ett fåtal gånger i bilderna. Eftersom en stor andel av bilderna innehåller vägar och gräs blir det tydligt varför dessa klasser kan identifieras med hög precision. Applikationen som utvecklades under detta arbete var en prototyp vilket innebär att modellen inte behöver fungera i alla miljöer. Skulle applikationen användas kommersiellt bruk, behöver den däremot fungera i mer generella situationer för att synnedsatta användaren skall kunna använda den på ett säkert sätt. Detta innebär att ett en större och mer varierande datamängd hade varit väsentlig.

Metoderna som utvecklades i detta arbete för att tolka utdatan från segmenteringsmodellen har båda visat sin potential för vidareutveckling. Däremot har metoderna brister i sin nuvarande implementering. Eftersom metoderna analyserar utdatan från segmenteringsmodellen, hade en modell med högre precision och hastighet inneburit att metoderna hade givit ett mer tillförlitligt resultat. Metoden *positionOnRoad* hjälper användaren att hålla sig på vägen och upptäcka kurvor. Denna metod har däremot en fundamental brist och detta är att den inte fungerar när kanterna av en väg inte framkommer i kamera-bilden. En möjlig lösning på detta problem är att byta till alternativa metoder om vägens mittpunkt inte kan identifieras. När vägens kanter är synliga i kamera-bilden, kan denna metoden användas som ett komplement till GPS navigeringen eftersom den upptäcker svängar och ger användaren instruktioner för att hålla sig på vägen.

Den andra metoden, *objectLocations*, som identifierar klasser och beräknar var de befinner sig i relation till användaren fungerade för testning. Metoden blev begränsad av antalet klasser som modellen kunde identifiera. För att denna metod skulle fungera bättre i praktisk tillämpning, skulle modellen behöva tränas på fler klasser. Ett problem som kan uppstå om modellen behandlar fler klasser är att metoden kan ge för långa meddelanden till användaren. En lösning på detta hade varit att implementera återkoppling genom exempelvis haptik eller använda kodade meddelanden för att göra dem kortare.

Slutligen löser dessa metoder två olika problem, den ena hjälper användaren hålla sig på vägen och den andra ger användaren information om var objekt befinner sig. I fortsatt utveckling skulle det därför vara fördelaktigt om båda av dessa metoder kunde kombineras och användas samtidigt i applikationen.

Sammanfattningsvis har detta arbete lett till en vidare förståelse för hur en bildsegmenteringsmodell kan implementeras och kombineras med ett GPS-system i en Android-applikation. Detta projekt har visat på att det är möjligt att skapa en applikation som har potential att hjälpa personer med synnedsättning att navigera i utomhusmiljöer. Med en större datamängd och vidare utveckling av applikationen, kan en liknande produkt användas som ett komplement till befintliga hjälpmedel inom navigering för personer med synnedsättning.

Referenser:

[1] Synskadades Riksförbund, “Vem är synskadad?” 2016. [Online]. Tillgänglig: https://www.srf.nu/leva-med-synnedsattning/om-synskador/vem-ar-synskadad/, (Hämtad 2021-03-01).

[2] TT, “Lång väntetid för att få ledarhund”, Göteborgs-Posten, Nov, 22, 2015. [Online]. Tillgänglig: https://www.gp.se/nyheter/sverige/l%C3%A5ng-v%C3%A4ntetid-f%C3%B6r-att-f%C3%A5-ledarhund-1.167353, (Hämtad: 2021-03-01).

[3] Google, “Flytta mellan objekt på startskärmen med TalkBack”, 2021. [Online]. Tillgänglig: https://support.google.com/accessibility/android/answer/6283678?hl=sv&ref_topic=10601570, (Hämtad 2021-06-15).

[4] N Ketkar, “Deep Learning with Python” Berkeley, CA, USA: Apress, 2017.

[5] IBM Cloud Education, “Supervised Learning”, 2020. [Online]. Tillgänglig: https://www.ibm.com/cloud/learn/supervised-learning, (Hämtad 2021-05-25).

[6] M. Arif Wani, F Ahmad Bhat, S Afzal , A Iqbal Khan, “Advances in Deep Learning”, Singapore: Springer, 2020. ss. 1-11

[7] R Shanmugamani, A Ghani A Rahman, S Maurice Moore, N Koganti, “Deep Learning for Computer Vision”, 1 uppl, Packt Publishing, 2018. [Online]. Tillgänglig: https://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=5254596#goto_toc, (Hämtad 2021-05-20).

[8] A Ghatak, “Deep Learning with R”, Singapore: Springer, 2019.

[9] A Jain, A Fandango, A Kapoo, “TensorFlow Machine Learning Projects”, Packt Publishing, 2018. [Online].
Tillgänglig: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789132212, (Hämtad 2021-05-25).

[10] X Li, X Sun, Y Meng, J Liang, F Wu, Ji Li , ”Dice Loss for Data-imbalanced NLP Tasks”, Department of Computer Science and Technology, Zhejiang University, Kina, 2019. [Online]. Tillgänglig: https://arxiv.org/abs/1911.02855

[11] D P. Kingma,J Lei Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION”, “*3rd International Conference for Learning Representations”*, San Diego, 2015

[12] V Dumoulin , F Visin , “A guide to convolution arithmetic for deep learning”, Université de Montréal, Kanda, 2016. [Online]. Tillgänglig: https://arxiv.org/pdf/1603.07285v1.pdf, (Hämtad 2021-05-21).

[13] Python Software foundation, “What is Python? Executive Summary” .[Online]. Tillgänglig: https://www.python.org/doc/essays/blurb/, (Hämtad 2021-06-01).

[14] Tensorflow, “TensorFlow Lite guide”, 2021. [Online].
Tillgänglig: https://www.tensorflow.org/lite/guide, (Hämtad 2021-05-20).

[15] Keras, “Keras layers API”, 2021 . [Online]. Tillgänglig: https://keras.io/api/layers/, (Hämtad 2021-05-20).

[16] Keras, “Model Training APIs”, 2021. [Online].
Tillgänglig: https://keras.io/api/models/model_training_apis/, (Hämtad 2021-05-20).

[17] Keras,” callbacks API”, 2021. [Online]. Tillgänglig: https://keras.io/api/callbacks/, (Hämtad 2021-05-20).

[18] Google, “Colaboratory: Frequently Asked Questions”, 2021. [Online]. Tillgänglig: https://research.google.com/colaboratory/faq.html, (Hämtad 2021-05-20).

[19] O Ronneberger, P Fischer, T Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany, 2015. [Online]. Tillgänglig: https://arxiv.org/abs/1505.04597, (Hämtad 2021-03-30).

[20] M Elgendy ,” Image preprocessing” i *Deep Learning for Vision Systems*, Manning Publications 2020, kap. 1.5.

[21] L Huang, J Qin, Y Zhou, F Zhu, L Liu, L Shao, ”Normalization Techniques in Training DNNs: Methodology, Analysis and Application”, 2020. [Online]. Tillgänglig: https://arxiv.org/pdf/2009.12836.pdf, (Hämtad: 2021-06-17).

[22] B Zoph , E D. Cubuk , G Ghiasi, T Lin, J Shlens, Q V. Le, “Learning Data Augmentation Strategies for Object Detection”[Online], https://arxiv.org/pdf/1906.11172v1.pdf, (Hämtad 2021-5-27)

[23] imgaug, Overview of Augmenters. [Online]. Tillgänglig: https://imgaug.readthedocs.io/en/latest/source/overview_of_augmenters.html, (Hämtad 2021-05-27).

[24] OpenCV, “About”. [Online]. Tillgänglig: https://opencv.org/about/, (Hämtad 2021-06-01).

[25] OpenCV , “Canny Edge Detection”. [Online]. Tillgänglig: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html, (Hämtad 2021-05-27).

[26] OpenCV, “Hough Line Transform”. [Online]. Tillgänglig: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html, (Hämtad 2021-05-27).

[27] Android Developers , “ImageFormat”.[Online]. Tillgänglig: https://developer.android.com/reference/android/graphics/ImageFormat, (Hämtad 2021-05-20).

[28] JPEG, “Overview of JPEG 2000” .[Online]. Tillgänglig: https://jpeg.org/jpeg2000/index.html, (Hämtad 2021-05-20).

[29] T Patrick, “Programming Visual Basic 2008”, USA: O'Reilly Media, Inc, 2008, s 498

[30] Oracle, “ByteBuffer”. [Online]. Tillgänglig: https://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html, (Hämtad 2021-05-20).

[31] OpenCV, “Mat - The Basic Image Container” .[Online]. Tillgänglig: https://docs.opencv.org/3.4/d6/d6d/tutorial_mat_the_basic_image_container.html, (Hämtad 2021-05-20).

[32] Flutter, “FAQ”. [Online]. Tillgänglig: https://flutter.dev/docs/resources/faq, (Hämtad 2021-02-25).

[33] V Sarcar, “Java Design Patterns”, Berkeley, CA, USA: Apress, 2019. https://link.springer.com/book/10.1007/978-1-4842-4078-6

[34] Android Developers, “Introduction to Activities”.[Online]. Tillgänglig: https://developer.android.com/guide/components/activities/intro-activities, (Hämtad 2021-05-15).

[35] Android Developers, “Fragments”. [Online]. Tillgänglig: https://developer.android.com/guide/fragments, (Hämtad 2021-05-15).

[36] Android Developers , “Declaring-layout”. [Online]. Tillgänglig: https://developer.android.com/guide/topics/ui/declaring-layout, (Hämtad 2021-05-15).

[37] Android Developers , “CameraX overview”. [Online]. Tillgänglig: https://developer.android.com/training/camerax, (Hämtad 2021-05-15).

[38] Android Developers, “Implement a preview”. [Online]. Tillgänglig: https://developer.android.com/training/camerax/preview, (Hämtad 2021-05-15).

[39] Android Developers, “Analyze images”. [Online]. Tillgänglig: https://developer.android.com/training/camerax/analyze, (Hämtad 2021-05-15).

[40] Android Developers, “TextToSpeech”. [Online]. Tillgänglig: https://developer.android.com/reference/android/speech/tts/TextToSpeech, (Hämtad 2021-05-15).

[41] Android Developers , “androidx.preference”. [Online]. Tillgänglig: https://developer.android.com/reference/androidx/preference/package-summary, (Hämtad 2021-05-15).

[42] ReactiveX, “ReactiveX”. [Online]. Tillgänglig: http://reactivex.io/intro.html, (Hämtad 2021-05-15).

[43] Mapbox, “company” .[Online]. Tillgänglig: https://www.mapbox.com/about/company/, (Hämtad 2021-04-21).

[44] Mapbox, “Maps SDK for Android”. [Online]. Tillgänglig: https://docs.mapbox.com/android/maps/guides/, (Hämtad 2021-04-21).

[45] Mapbox, “Package com.mapbox.android.core.location”. [Online]. Tillgänglig: https://docs.mapbox.com/android/telemetry/api/libcore/3.1.0/com/mapbox/android/core/location/package-summary.html, (Hämtad 2021-04-21).

[46] Mapbox , “Showing device location”. [Online]. Tillgänglig: https://docs.mapbox.com/android/maps/guides/location-component/, (Hämtad 2021-04-21).

[47] Mapbox, “Navigation SDK for Android”. [Online]. Tillgänglig: https://docs.mapbox.com/android/navigation/guides/, (Hämtad 2021-04-21).

[48] Mapbox, “Route progress”. [Online]. Tillgänglig: https://docs.mapbox.com/android/navigation/guides/route-progress/, (Hämtad 2021-04-21).

[49] Mapbox, “Directions”. [Online]. Tillgänglig: https://docs.mapbox.com/android/java/guides/directions/, (Hämtad 2021-04-21).

[50] Mapbox, “Package com.mapbox.navigation.core”. [Online]. Tillgänglig: https://docs.mapbox.com/android/navigation/api/1.5.1/libnavigation-core/-core/com.mapbox.navigation.core/, (Hämtad 2021-04-21).

[51] Mapbox, “Maneuver instructions”. [Online]. Tillgänglig: https://docs.mapbox.com/android/navigation/guides/manuever-instructions/, (Hämtad 2021-05-20).

[52] Mapbox, “Search SDK for Android”. [Online]. Tillgänglig: https://docs.mapbox.com/android/search/guides/, (Hämtad 2021-04-20).

[53] Mapbox , “Class CarmenFeature”. [Online]. Tillgänglig: https://docs.mapbox.com/archive/android/java/api/libjava-services/2.2.6/com/mapbox/services/api/geocoding/v5/models/CarmenFeature.html, (Hämtad 2021-05-20).

[54] Mapbox, “Geocoder”.[Online]. Tillgänglig: https://docs.mapbox.com/android/java/guides/geocoder/, (Hämtad 2021-05-20).

[55] S Rea, A Araujo, “Navigation Systems for the Blind and Visually Impaired: Past Work, Challenges, and Open Problems”, *"Sensors, Sensor Technologies for Caring People with Disabilities"*, vol. 19, August 2019, doi: https://doi.org/10.3390/s19153404

[56] blindsquare, “What is BlindSquare?” 2021. [Online]. Tillgänglig: https://www.blindsquare.com/about/, (Hämtad 2021-05-20).

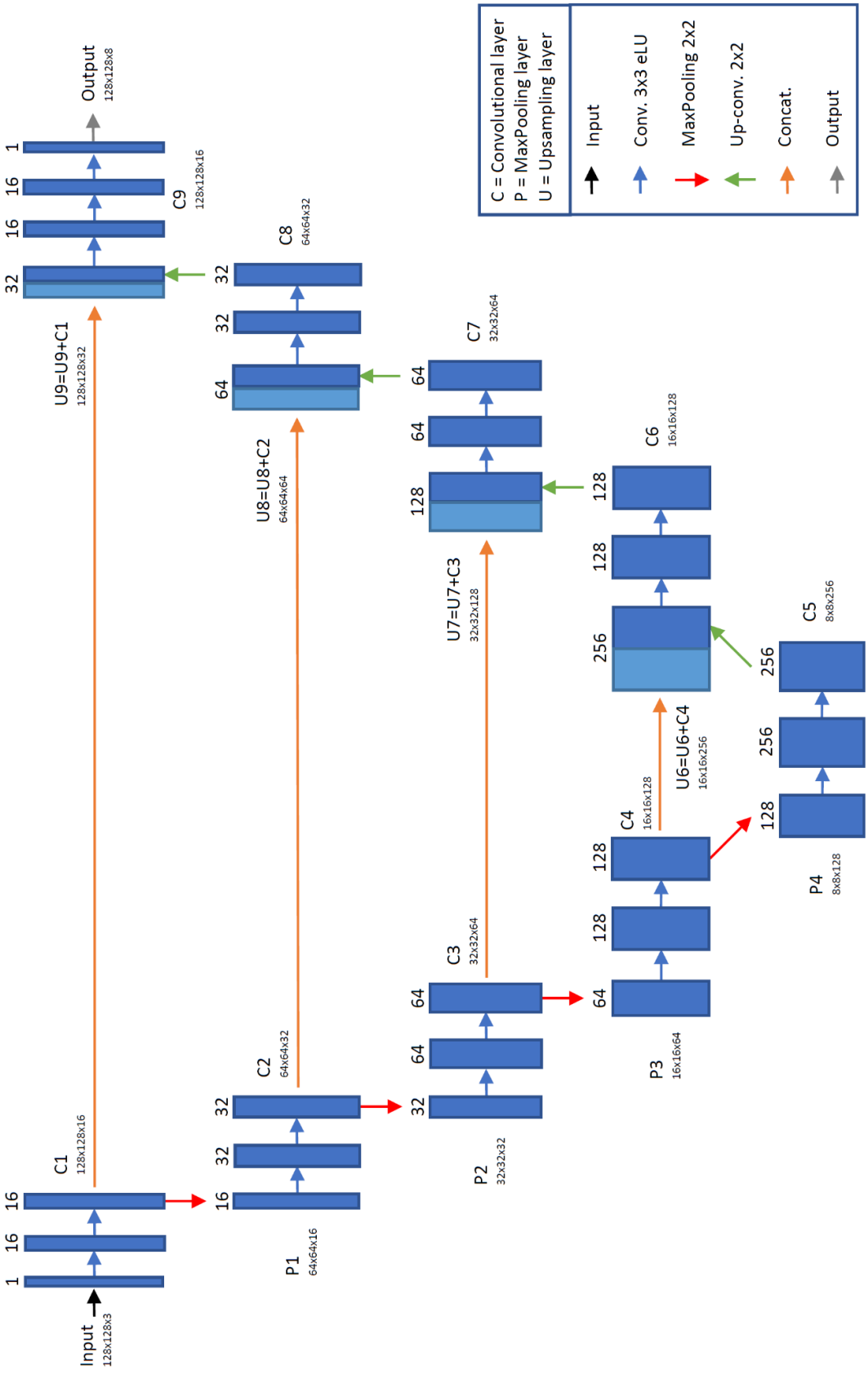
[57] B Lin,C Lee, P Chiang, “Simple Smartphone-Based Guiding System for Visually Impaired People”, “*Sensors*, Special Issue "Smartphone-based Pedestrian Localization and Navigation", vol. 17, June. 2017, doi: https://doi.org/10.3390/s17061371

[58] Cityscapes Dataset, “Cityscapes Dataset”. [Online]. Tillgänglig: https://www.cityscapes-dataset.com/, (Hämtad 2021-05-20).

[59] K Wada, labelme: Image Polygonal Annotation with Python, [Online]. Tillgänglig: https://github.com/wkentaro/labelme, (Hämtad 2021-02-25).

[60] C Beleites, U Neugebauer, T Bocklitz, C Krafft, J Popp, “Sample Size Planning for Classification Models”, Analytica Chimica Acta, 760 (2013) 25 - 33, doi: https://doi.org/10.1016/j.aca.2012.11.007

Bilaga 1 – U-net arkitekturen



Figuren på föregående sida visar ett exempel på hur en U-net modell kan vara uppbyggd. Nedskalnings-delen visas till vänster i figuren medan uppskalnings-delen syns till höger. Längst ner i nätverket finns den så kallade flaskhalsen (eng. *bottleneck*) som är övergången mellan dem två olika delarna. Pilarna i figuren beskriver vilken process som utförs och varje block motsvarar ett lager. För att förklara hur arkitekturen är uppbyggd och hur den fungerar, beskrivs ett konkret exempel utifrån figuren ovan.

Förklaring av varje del i U-net arkitekturen:

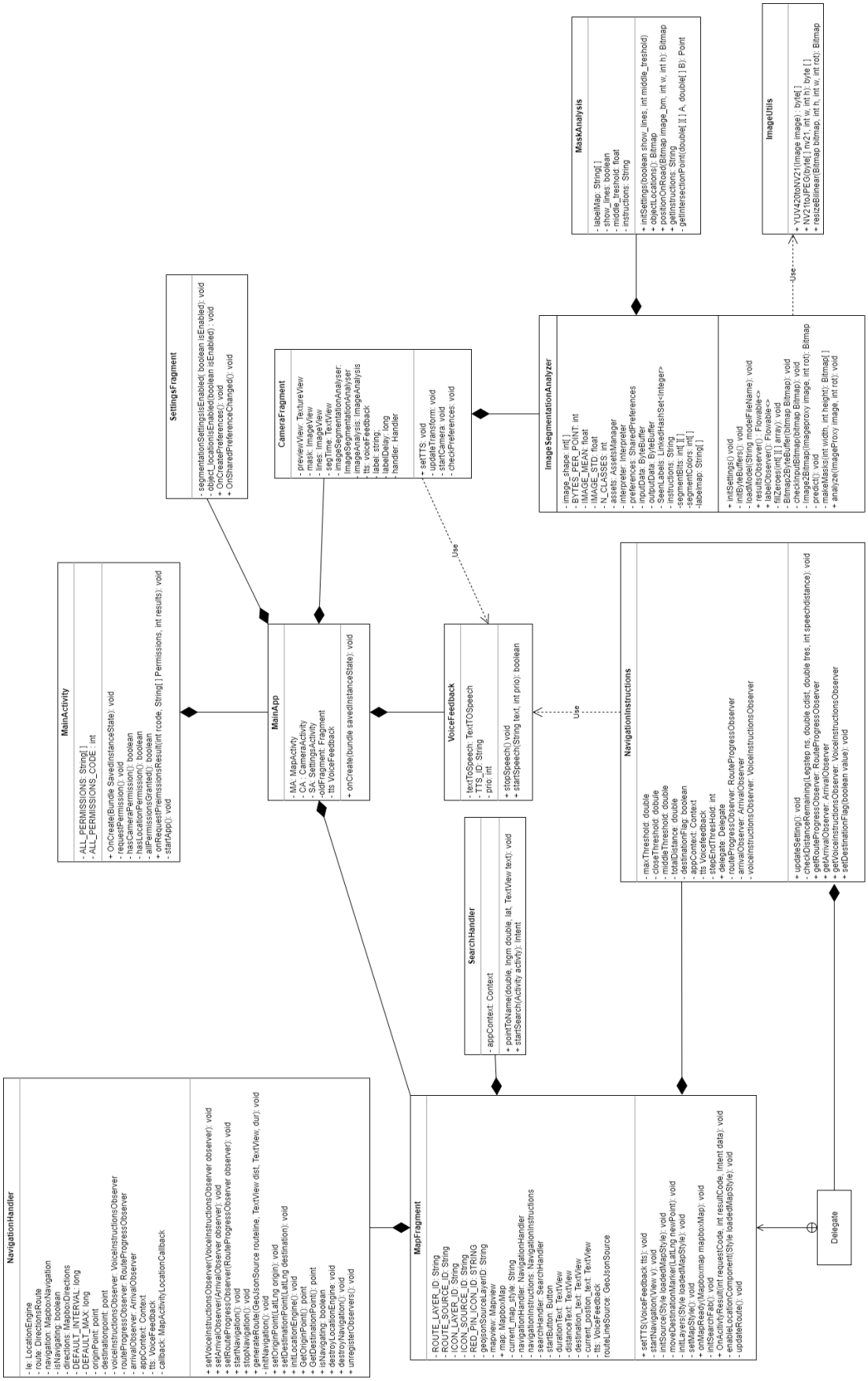
1. Nätverket tar in bilder med formen (128, 128, 3) i indata-lagret (Input).
2. Datan förs vidare till två faltningslager (C1) med formen (3, 3) och med 16 filter vardera. Datan har formen (128, 128, 16).
3. Därefter går datan vidare till ett max-pooling lager (P1) med formen (2, 2) vilket innebär att storleken på bilden halveras. Datan har formen (64, 64, 16).
4. Datan förs vidare till två faltningslager (C2) med formen (3, 3) och med 32 filter vardera. Datan har formen (64, 64, 32).
5. Därefter går datan vidare till ett max-pooling lager (P2) med formen (2, 2) vilket innebär att storleken på bilden halveras igen. Datan har formen (32, 32, 32).
6. Datan förs vidare till två faltningslager (C3) med formen (3, 3) och med 64 filter vardera. Datan har formen (32, 32, 64).
7. Därefter går datan vidare till ett max-pooling lager (P3) med formen (2, 2) vilket innebär att storleken på bilden halveras igen. Datan har formen (16, 16, 64).
8. Datan förs vidare till två faltningslager (C4) med formen (3, 3) och med 128 filter vardera. Datan har formen (16, 16, 128).
9. Därefter går datan vidare till ett max-pooling lager (P4) med formen (2, 2) vilket innebär att storleken på bilden halveras igen. Datan har formen (8, 8, 128).
10. Datan förs vidare till två faltningslager (C5) med formen (3, 3) och med 256 filter vardera. Datan har formen (8, 8, 256) och befinner sig i bottleneck.
11. Uppskalnings-delen inleds. Datan förs vidare till ett transponerat faltningslager (U6) som sammanfogas med C4 från nedskalnings-delen. Datan har formen (16, 16, 256).
12. Datan förs vidare till två faltningslager (C6) med formen (3, 3) och med 128 filter vardera. Datan har formen (16, 16, 128).
13. Därefter går datan vidare till ett transponerat faltningslager (U7) som sammanfogas med C3 från nedskalnings-delen. Datan har formen (32, 32, 128).
14. Datan förs vidare till två faltningslager (C7) med formen (3, 3) och med 64 filter vardera. Datan har formen (32, 32, 64).
15. Därefter går datan vidare till ett transponerat faltningslager (U8) som sammanfogas med C2 från nedskalnings-delen. Datan har formen (64, 64, 64).
16. Datan förs vidare till två faltningslager (C8) med formen (3, 3) och med 32 filter vardera. Datan har formen (64, 64, 32).
17. Därefter går datan vidare till ett transponerat faltningslager (U9) som sammanfogas med C1 från nedskalnings-delen. Datan har formen (128, 128, 32).
18. Datan förs vidare till två faltningslager (C8) med formen (3, 3) och med 16 filter vardera. Datan har formen (128, 128, 16).
19. Datan når utdata-lagret som består av ett faltningslager med formen (1, 1) och 8 filter.

Antalet filter i det sista lagret motsvarar antalet klasser, vilket var 8 i detta exempel. Utdatan innehåller i detta fall 8 värden mellan 0-1 för varje pixel. Datan kan sedan användas för att skapa masker genom att välja klassen med högst värde för varje pixel.

Bilaga 2 – Python-kod för segmenteringsnätverket

```
 i = Input((IMSHAPE[0], IMSHAPE[1], IMSHAPE[2]))

c1 = Conv2D(2**b, (3, 3), activation=AF, kernel_initializer=KI, padding='same') (i)
c1 = Dropout(0.1) (c1)
c1 = Conv2D(2**b, (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(2**(b+1), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (p1)
c2 = Dropout(0.1) (c2)
c2 = Conv2D(2**(b+1), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(2**(b+2), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (p2)
c3 = Dropout(0.2) (c3)
c3 = Conv2D(2**(b+2), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(2**(b+3), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (p3)
c4 = Dropout(0.2) (c4)
c4 = Conv2D(2**(b+3), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(2**(b+4), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (p4)
c5 = Dropout(0.3) (c5)
c5 = Conv2D(2**(b+4), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c5)

u6 = Conv2DTranspose(2**(b+3), (2, 2), strides=(2, 2), padding='same') (c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(2**(b+3), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (u6)
c6 = Dropout(0.2) (c6)
c6 = Conv2D(2**(b+3), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c6)

u7 = Conv2DTranspose(2**(b+2), (2, 2), strides=(2, 2), padding='same') (c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(2**(b+2), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (u7)
c7 = Dropout(0.2) (c7)
c7 = Conv2D(2**(b+2), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c7)

u8 = Conv2DTranspose(2**(b+1), (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(2**(b+1), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (u8)
c8 = Dropout(0.1) (c8)
c8 = Conv2D(2**(b+1), (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c8)

u9 = Conv2DTranspose(2**b, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(2**b, (3, 3), activation=AF, kernel_initializer=KI, padding='same') (u9)
c9 = Dropout(0.1) (c9)
c9 = Conv2D(2**b, (3, 3), activation=AF, kernel_initializer=KI, padding='same') (c9)

o = Conv2D(N_CLASSES, (1, 1), activation=AF_LAST) (c9)
```

Bilaga 3 – Programstruktur för Android-applikationen



Java-applikation som utvecklades i detta arbete består huvudsakligen av 3 delar.

- Main-paketet som hantera huvudtråden för applikation och kopplar ihop programmet.
- Navigerings-paketet hanterar det visuella samt koden bakom kart och navigeringssystemet.
- Segmenterings-paketet tar in bild från kamera och sedan använder bildsegmentering och analyserar ut-datan.

Applikationens struktur är baserad på arkitektur-mönstret MVC. Varje Activity och Fragment användes som "controllers", layout-filer antogs tillhöra "views" och resterande klasser blev "models". Main-paketet består av följande klasser och layout-filer:

- MainActivity
  - activity_main.xml
- MainApp
  - activity_mainapp.xml
- SettingsFragment
  - settings.xml
- VoiceFeedback

Applikation startas först i MainActivity, och MainApp blir sedan huvudtråden som applikationen kör i. SettingsFragment och VoiceFeedback samt andra aktiviteter skapas och initieras i klassen MainApp.

Navigerings-paketet består av följande klasser och layout filer:

- MapFragment
  - activity_map.xml
- NavigationHandler
- NavigationInstructions
- SearchHandler

En instans av MapFragment skapas i MainApp. Därefter skapar MapFragment instanser av NavigationHandler, NavigationInstructions och SearchHandler som hanterar olika delar av Mapbox-kartan och navigeringen.

Segmenterings-paketet består av följande klasser och layout filer:

- CameraFragment
- activity_camera.xml
- ImageSegmentationAnalyzer
- ImageUtils
- MaskAnalysis

Precis som för MapFragment skapas CameraFragment i MainApp. I CameraFragment skapas sedan en instans av ImageSegmentationAnalyzer som hanterar bild-analysering. I denna klass används statiska funktioner från klassen ImageUtils. ImageSegmentationAnalyzer skapar dessutom en instans av MaskAnalysis för att analysera maskerna från segmenteringen.