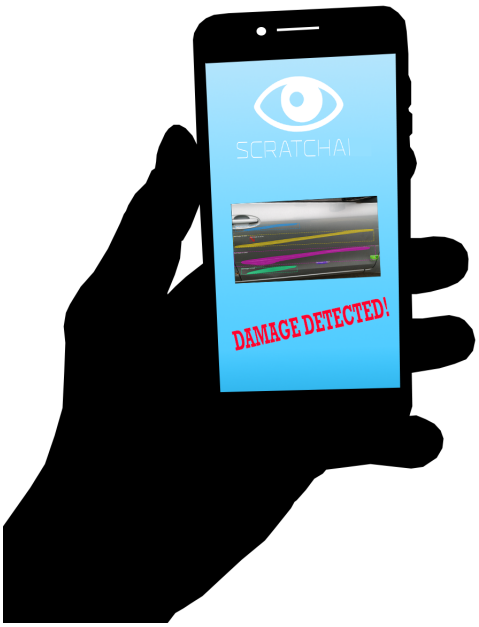


ScratchAI: En mobilapplikation för identifiering av fordonsskador med maskininlärning

Examensarbete inom Data- och informationsteknik

Anton Lutteman
Kayed Mahra

Institutionen för Data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2020

Examensarbete 2020

ScratchAI: En mobilapplikation för identifiering av fordonsskador med maskininlärning.

Anton Lutteman
Kayed Mahra

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2020

ScratchAI: En mobilapplikation för identifiering av fordonsskador med maskininlärning
ANTON LUTTEMAN
KAYED MAHRA

Examinator: Peter Lundin, Institutionen för Data- och informationsteknik
Handledare:
Sakib Sistek, Institutionen för Data- och informationsteknik
Henrik Fagrell, Diadrom Systems AB

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

Institutionen för Data- och Informationsteknik
Göteborg 2020

Sammanfattning

Inspektioner av fordon för identifiering av skador utförs många gånger manuellt och skrivs ner på ett pappersformulär som sedan överförs till en dator för att sparas. Denna process är tidsödande och innehåller många manuella steg, men det finns något som de flesta människor har idag som skulle kunna underlätta och effektivisera arbetet. En mobiltelefon kan genom en applikation identifiera och dokumentera skador på ett effektivt sätt. Mobiltelefonen kan utföra detta arbete genom bilder tagna med dess kamera och ett neuralt nätverk för identifiering av skadorna. Syftet med detta arbete är att undersöka hur neurala nätverk kan användas för att underlätta arbetet inom detta område.

Frågeställningen blir således:

Hur kan vi genomföra identifiering av fordonsskador med en smartphone som använder maskininlärning för bildanalys?

Tillvägagångssättet i detta arbete var att välja två modeller för identifiering av skador. En klientbaserad modell, lätt nog att köras direkt på en mobiltelefon, samt en tyngre serverbaserad modell. En mängd data på skadade och intakta fordonsdörrar samlades in och användes i en rad experiment för träning av modellerna.

Det teoretiska bidraget i arbetet är att man med relativt lite träningstid och datamängd kan uppnå goda resultat vid identifiering av skador. Det praktiska bidraget i arbetet är en mobilapplikation som klarar att identifiera skador med både sin klient- och serverbaserade modell.

Keywords: Maskininlärning, Neurala nätverk, Fordonsdiagnostik, Mask R-CNN, MobileNet, Android

Abstract

The inspection of vehicles is often carried out using physical forms that are later transferred to a digital system. This process is often time consuming and involves many manual steps. However, there is something that most people carry with them today that could simplify this chain of steps - the smartphone. With a mobile application vehicle body damange can be identified and saved in an effective manner. This is possible by taking an image of the damage using the smartphone's camera and using a neural network to identify the damage. The purpose of this thesis is to investigate how neural networks can be used to simplify the involved tasks in this problem.

The research question is thus:

How can we conduct identification of vehicle damage using a smartphone and machine learning for image analysis?

The approach in this work was to choose two models for identification of damage, one client based and one server based. The client based is light weight enough to run inference directly on the smartphone device and the server based uses segmentation to highlight the areas of damage. Image data of damaged and intact vehicle doors was collected and used in a series of experiments to train the models.

The theoretical contribution of this work is that with relatively little training time and a small data set one can achieve good results when identifying damage. The practical contribution of this work is a mobile application that manages to identify vehicle door damage with both the client based and server based models.

Keywords: Machine learning, Neural networks, Vehicle diagnostics, Mask R-CNN, MobileNet, Android

Förord

Examensarbetet utfördes som kandidatarbete hos Institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola och behandlar utvecklingen av en mobilapplikation för identifiering och rapportering av ytliga fordonsskador med hjälp av maskininlärning

Vi vill tacka vår handledare hos *Diadrom Systems AB, Henrik Fagrell* för inspiration till projektet och möjlighet att forma det efter vår egen vision.
Vi vill också tacka vår handledare på *Chalmers Tekniska Högskola, Sakib Sistek* för hans engagemang och energi genom hela arbetet.

Ordlista

API Applikationsprogrammeringsgränssnitt, av engelskans Application programming interface

CNN Convolutional neural networks.

Egenskapskartor Engelskans Feature maps

Faltningsnätverk En klass djupa neurala nätverk där faltning är en central matematisk operation i beräkningarna. Kallas på engelska för ”Convolutional neural networks”.

Främmande nyckel Engelskans foreign key

Hyperparameter En grupp parametrar som kan ställas in *innan* träning av en modell påbörjats.

Momentan modell En modell med tillfällig viktkonfiguration mellan epoker vid träning.

RoI Region of Interest

RPN Region Proposal Network

Primär nyckel Engelskans primary key

Stokastisk lutningsnedstigning Engelskans Stochastic Gradient Descent. Är en optimeringsalgoritm.

Framåtpropagerande neurala nätverk Engelskans Feed Forward Neural Networks”. Är en typ av acykliska neurala nätverk där informationsflödet sker åt ett håll.

Innehåll

Kapitel 1

Introduktion

När ett fordon skadats innebär det per automatik en rad manuella steg som exempelvis pappersarbete, telefonssamtal, försäkringsärenden m.m. Detta kan i många fall upplevas tidsödande för fordonsägaren. Med dagens datorseendeteknik bör denna processkedja kunna effektiviseras. Artificiella neurala nätverk har visat stor potential för datorseende med en särskild typ av arkitektur som kallas för CNN (Convolutional Neural Networks) [1].

Syfte
Detta arbete handlar om CNN och undersöker ett möjligt användningsområde inom ytlig skadediagnostik av fordon.

Mål
Målet är att konfigurera ett befintligt nätverk med ett koncept som kallas överföringsinlärning [2] för att finjustera en modell till att identifiera ytliga fordonsskador. Vidare avses denna funktionalitet användas i en mobilapplikation som kan lagra registreringsnummer för en fordonspool och generera fordonsspecifika skaderapporter som också sparas i en lokal databas.

Frågeställning
Frågeställningen att besvara är:

Hur kan vi genomföra identifiering av fordonsskador med en smartphone som använder maskininlärning för bildanalys?

För att besvara frågeställningen kommer två CNN-nätverk att konfigureras och tränas. Det som skiljer dessa åt är att det ena är ett betydligt större nätverk som kommer att köras på en server, och det andra är en lättviktsversion som intregreras i applikationen. Vidare använder det serverbaserade nätverket sig av segmentering för att ringa in området för skadan som detekterats. Detta är en funktionalitet som skulle kunna dras nytta av i vidareutveckling av systemet (kapitel 7.5).

Avgränsningar

För att utforma arbetet med hänsyn till kursens tidsram har ett antal avgränsningar definierats:

- Endast ytliga skador kommer att beaktas i arbetet.
- Med ytliga skador avses repor och bucklor som inte utgör ett uppenbart hinder för fordonets förmåga att framföras.
- Maskininlärningen kommer att ske med hjälp av konfiguration/modifiering av erkända och befintliga modeller. Ingen nätverksarkitektur kommer alltså att skapas från grunden.
- Mobilapplikationen kommer att delvis implementeras med stöd för serverbaserad kommunikation utanför ett lokalt nätverk, men kommer att demonstreras inom ett lokalt nätverk.
- Databasen kommer att vara en lokal databas på den mobilenhet applikationen är installerad.

Rapportens struktur

Denna introduktion följs av ett kapitel som behandlar teknisk bakgrund inom ämnet och de ramverk som använts i arbetet. Därefter presenteras metoden och systemkonstruktionen. Slutligen presenteras resultaten som som följs av en diskussion och slutsats.

Kapitel 2

Angränsande tidigare projekt

Stora framgångar för datorseende uppnåddes år 2012 när Krizhevsky et al. [1] släppte rapporten ”ImageNet Classification with Deep Convolutional Neural Networks”. Detta genombrott påvisade potentialen av så kallade faltningsnätverk för att utföra klassificering i bildanalys. Sedan dess har djupa neurala nätverk av klassen CNN (Convolutional Neural Networks) använts flitigt för olika bildanalytiska uppgifter.

Exempelvis används tekniken inom medicinsk bildanalys för att underlätta identifieringen av malignartade celler vid bröstcancer. I ett forskningsarbete utfört av Dabeer et al. [3] lyckades man uppnå en träffsäkerhet om 99.86% i modellens utlåtanden av testdata.

Bildanalys består inte enbart av att klassificera objekt, utan innefattar en rad olika uppgifter som exempelvis detektion och segmentering. För att utföra dessa uppgifter krävs mer sofistiskerade modeller som vidgar potentialen som ges av faltningsnätverk. En sådan modell är Mask R-CNN som föreslås av Zhang et al. [4] för att underlätta trafikstockning som uppstår till följd av trafikolyckor. Problemet som avses lösas med tekniken är det problem som uppstår efter en olycka då bilder skall tas och skador evalueras, vilket i många fall är tidsödande. Arbetets resultat visar på hög träffsäkerhet, men framtida förbättringar för segementering och utökat dataset föreslås.

Likt de arbeten som redogjorts för ovan avser detta arbete undersöka ett specifikt användningsområde som kan dra nytta av denna teknik. I vårt arbete använder vi både konventionella CNN-nätverk och den mer softistikerade varianten som utför segmentering. Likt det tillvägagångssätt som använts i artikeln av Zhang et al. [4] bygger finjusteringen av modellerna på såkallad överföringsträning. En gemensam faktor i vårt arbete och de ovan nämnda är den centrala uppgift som innefattar att anpassa modeller för ett särskilt ändamål. Denna uppgift kräver god kännedom av modellernas dynamik för att kunna dra sunda slutsatser från experiment som genomförs.

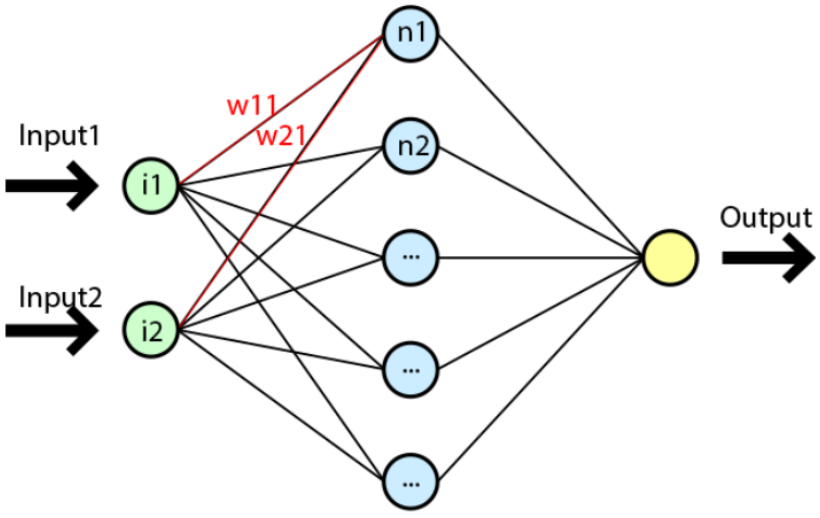
Kapitel 3

Bakgrund

Följande kapitel ämnar att ge en teoretisk bakgrund gällande de generella koncept som berörs av artificiella neurala nätverk och arkitekturen för faltningsnätverk (CNN). Mycket av det som tas upp gällande neurala nätverk görs i kontext av faltningsnätverk. Trots att CNN skiljer sig från andra neurala nätverk så delar de många fundamentala egenskaper och principer. Utöver detta ämnar kapitlet ge en teknisk bakgrund till den mjukvara som använts i arbetet.

3.1 Framåtpropagerande neurala nätverk

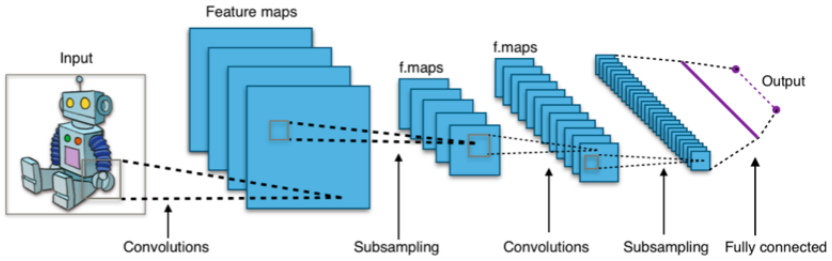
Framåtpropagerande neurala nätverk (Feedforward neural networks) är en samling neurala nätverk som utmärks av att dess grafer är acykliska. Detta innebär att informationsflödet sker åt ett håll (*propagerar*). Figur 3.1 nedan illustrerar detta. En mer genomgående förklaring för hur denna dynamik fungerar kommer senare i detta kapitel.



Figur 3.1: Exempel på ett framåtpropagerande neuralt nätverk, redigerad, Källa se: [5]
Indata lagras i de *gröna* noderna varpå tillståndet för det dolda lagret med *blå* noder beräknas med hjälp av styrkan (*vikten*) mellan de inkommande signalerna till en blå nod och tillstånden för de neuroner som skickar signalerna från föregående lager.

3.2 CNNs

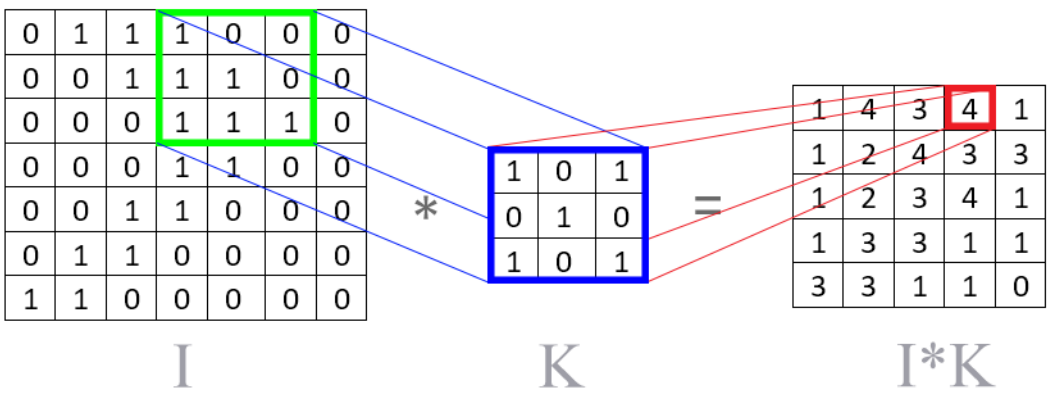
Faltningsnätverk (Convolutional neural networks, CNN) är inspirerade från hur hjärnans syncentrum behandlar visuell information [6]. Följaktligen har den artificiella konstruktionen många likheter med dess biologiska motsvarighet. En sådan likhet är hur specifika neuroner är kopplade till särskilda regioner i näthinnan och svarar starkt av ljus från en specifik rumslig region i synfältet. För att en dator i praktiken ska kunna härma förståelsen för vad den ”ser” måste den ta hänsyn till rumsliga och tidsmässiga förändringar och beroenden mellan pixlar i en bild. CNN uppnår detta genom att konstruera och finjustera olika filter dynamiskt under inlärning. Detta eliminerar behovet av manuellt framtagna filter. En annan fördel med CNN är att de kräver näst intill ingen förbehandling av indatan utöver bildstorlek. Nätverket kan sedan med slutledning hitta mönster från obehandlade bilder [7]. Detta avsnitt kommer att ta upp tekniken bakom prestandan och dynamiken av faltningsnätverk.



Figur 3.2: Grundläggande arkitektur bakom ett faltningsnätverk, Källa: se [8]

3.2.1 Faltningslager

Syftet med faltningslagret är att låta nätverket härleda och lära sig egenskaper från en bild som indata. Ett nätverk kan innehålla flera faltningslager, där de första oftast är de som härleder enkla egenskaper som linjer och kurvor. På detta sätt kan nätverket lära sig mer abstrakta egenskaper. För att erhålla en karta över de lärda egenskaperna måste ett filter flytta sig över indatan och utföra matrismultiplikationer. Detta filter kallas för *kärnan* och dess dimensioner betraktas som mottagningsfältet. Processen genererar det som kallas för en egenskapskarta. Kärnan kartlägger i praktiken neuroner i egenskapskartan till olika regioner i bilden. Kärnan har normalt sett samma djup som bilden. I fallet med RGB (red, green, blue) är djupet tre och förhåller sig till grundfärgerna. Ett faltningslager består normalt sett av flera kärnor där varje kärna ansvarar för att filtrera fram en viss egenskap. Förhållandet mellan antalet kärnor och egenskapskartor är 1:1.

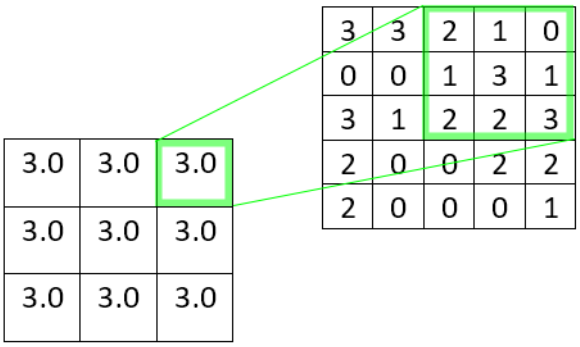


Figur 3.3: Faltning över en 7x7 bild som insignal med en 3x3 kärna.

Som man kan se i figur 3.3 motsvarar matrisen längst åt höger utvärdet efter faltning. Många gånger resulterar faltning i en matris som är mindre än initialt, vilket i praktiken innebär ett minskat antal neuroner. Detta har två implikationer: Det reducerar risken för överanpassning (3.2.5) och det gör träning mindre kostsamt till följd av färre antal parametrar [9].

3.2.2 Pooling

Poolinglager är kopplade till utdatan från en eller fler faltningslager. Syftet med detta lager är att reducera storleken av den härledda egenskapskartan utan att förlora information om de egenskaper som härleddes under faltning. Principen är mycket lik den vid faltning.



Figur 3.4: 3x3 maxpooling över en 5x5 egenskapskarta. Den största utsignalen från en neuron inom poolingområdet är 3, och är därför den utsignal som blir bibehållen.

Poolingmetoden som används i figur 3.4 kallas för maxpooling. Den neuron med högst utvärde i poolingregionen, här 3x3, behålls. Poolingregionen förflyttas med ett förbestämt antal steg likt faltning. Det finns olika varianter av pooling som baseras på exempelvis medelvärdet av regionen eller det kvadratiska medelvärdet. Tanken är att dra nytta av att egenskaperna inte påverkas av förändringar i rotation och skala och därmed kan man minska antalet parametrar utan att förlora information. Återigen är detta önskvärt beräkningsmässigt.

3.2.3 Aktivering

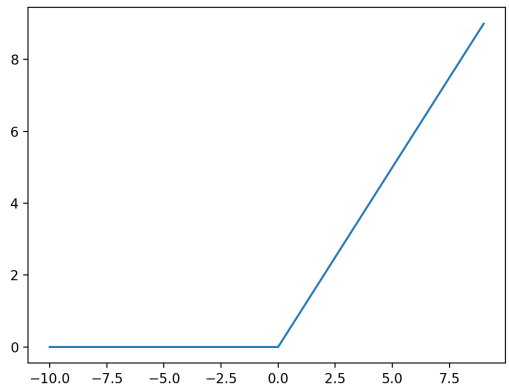
Tillskillnad från hjärnans neuroner, är inte artificiella neurons aktivitet kontinuerliga. Aktivering sker istället i diskreta tidssteg med binära tillstånd som representerar aktiv eller inaktiv. Inom artificiella neurala nätverk beror aktiveringen av en neuron på de inkommande signalerna från det föregående lagrets neurala utsignaler ihop med styrkan *(vikten)* mellan dem. För varje koppling mellan neuroner summeras produkten av de föregående neuronernas tillstånd och vikt. Summan jämförs därefter med ett gränsvärde. Detta kallas för det *lokala fältet*.

$$n_i(t+1) = g(\sum_{j=1}^{N} w_{ij} n_j(t) - \mu_i) \tag{3.1}$$

I ekvation 3.1 är $n_i(t+1)$ det neurala tillståndet för neuron n_i i det följande tidssteget. Inom parantesen i högerledet är det som tidigare kallades för det lokala fältet, låt det betecknas med b_i. Det används sedan som en parameter i aktiveringsfunktionen $g(b_i)$, i detta fall Rectifying Linear Activation Function (ReL), som kan ses i ekvation 3.2 nedan.

$$g(x) = max(0, x) \tag{3.2}$$

Det finns många olika sorters aktiveringsfunktioner, men den de facto standarden för dolda lager i djupa neurala nätverk idag är ReL. Anledningen till varför ReL är att föredra som aktiveringsfunktion är för att den förenklar lutningsbaserad optimering (3.2.6) av modellen, medan den bibehåller egenskaper som bidrar till generalisering (3.2.5) [10]. I praktiken innebär förenklingen att djupare lager i nätverket bättre kan dra nytta av viktjusteringarna som görs vid träning. Fördelarna är tack vare att ReL fungerar som en *delvis linjär funktion* då hälften av definitionsmängden $x > 0$ representeras linjärt och den andra hälften $x \leq 0$ representeras icke-linjärt.



Figur 3.5: Grafisk representation av ReL

Syftet med det neurala nätverket är att göra utlåtanden som är baserade på en viss nivå av konfidens. För detta ändamål är en populär aktiverinsfunktion i ett nätverks utgångslager *Softmax*. Softmax normaliserar summan av konfidenserna till ett. Därför är Softmax bra för multi-klassproblem där modellens uppgift är att uttala sig om vilken klass som förekommer i en bild och det råder ömsesidig uteslutning. I ett multi-klassproblem representeras de tre klasserna A, B, C som en vektor där endast ett element är 1 och övriga är 0, exempelvis $A = [1, 0, 0], B = [0, 1, 0], C = [0, 0, 1]$. Ger man ett väl presterande nätverk en bild där klass A förekommer så kan ett svar se ut som följande: $[0.97, 0.02, 0.01]$. Här motsvarar första index i svarsvektorn konfidensen för att det är A som förekommer i bilden och är en hög träffsäkerhet.

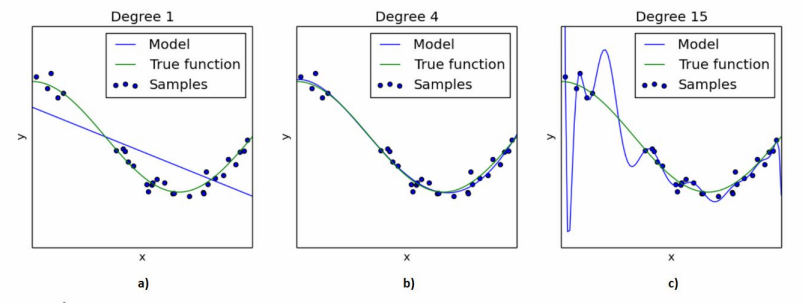
3.2.4 Träning av modell

Att träna ett nätverk handlar i grund och botten om att finjustera de vikter mellan neuroner tills att nätverket får en acceptabel träffsäkerhet när det matas med ny data. För att uppnå god träffsäkerhet krävs mycket data. I praktiken kan det för gemene person vara väldigt svårt att få tag på den mängd som krävs. Detta problem kan lösas med *överföringsträning* som är ett koncept där man initierar en modells vikter med en förtränad profil. Ett exempel är ImageNet som tillhandahåller vikter baserade på träning av över 14 miljoner bilder och 20000 olika klasser [11]. Anpassningen av modellen går sedan till genom att träna vidare ett visst antal lager i nätverket på egen data och ändra det sista lagret för att få modellen att klassificera det antal klasser ens specifika problem har. På så sätt drar man nytta av den enorma mängd data en modell tränats på av en större organisation, och riktar sedan in modellen för ett specifikt problem den inte exponerats för tidigare.

En modell tränas genom att datan delas upp i tre kategorier: *träningsdata, valideringsdata* och *testdata*. Proportionerna är omdiskuterade, men generellt utgör träningsdatan en stor andel av den totala mängden data. Datan för alla kategorier är märkt med den klass som representeras, exempelvis en bild på en katt eller hund. En *epok* är en iteration där modellen tränats på all data i träningskategorin och sedan testats mot valideringsdatan som utgör en referens för hur modellen presterar på data den inte tränats på. En *förlustfunktion* [12] fungerar som ett mått för estimering av hur mycket fel i utlåtanden modellen haft under varje iteration. Detta mått används sedan av en optimeringsalgoritm som exempelvis stokastisk lutningsnedstigning [13] för att justera nätverkets vikter mellan iterationerna. Optimeringsalgoritmen (se 3.2.6) strävar efter att minimera totala felvärdet för utlåtanden av valideringsdata mellan iterationerna. När det förbestämda antalet epoker fortlöpt testas modellen slutligen på data ur testkategorin. Hur den presterar på denna data är det slutgiltiga måttet för hur bra modellen tränats för ny data. Anledningen till varför inte validersdatan används för detta ändamål är på grund av en indirekt bias. Visserligen har inte modellen tränats direkt på valideringsdatan, men eftersom att den fungerat som referens för att justera vikterna har den en indirekt påverkan.

3.2.5 Generalisering

Ett vanligt och potentiellt problem för djupa neurala nätverk är ett fenomen som kallas *överanpassning*. Det innebär att nätverket presterar bra på träningsdata men dåligt på ny data som exempelvis testdatan. Motsatsen till överanpassning kallas för *underanpassning* och innebär att modellen har svårt att lära sig från träningsdatan och presterar således dåligt för både träningsdata och testdata.



Figur 3.6: a) underanpassning, b) ideal, c) överanpassning

I figur 3.6 kan man se hur en linjär funktion i a) är otillräcklig för att uttrycka träningsdatan. Resultatet av detta är en modell som har svårt att lära sig av datan och kommer att prestera dåligt. Överanpassning manifesteras i c) som en funktion med för många parametrar, här överanpassar sig modellen till träningsdatan och lär sig även bruset, vilket ger en modell som presterar väl på träningsdatan men inte på ny data. Det bästa scenariot är b) där modellen har optimalt antal parametrar. Detta kan återkopplas till varför djupa neurala nätverk i synnerhet är känsliga för överanpassning och, att det existerar en relation mellan mängden parametrar och risk för överanpassning [14].

3.2.6 Optimering

En *optimeringsalgoritms* uppgift är att räkna ut hur vikterna mellan neuroner skall uppdateras. Detta sker vanligvis mellan varje epok. För att kunna utföra beräkningen behöver algoritmen ha koll på hur modellen presterat under den senaste epoken med den nuvarande konfigurationen av vikter inom nätverket. Optimeringsalgoritmen får ett värde av en så kallad *förlustfunktion* som under epokens gång håller koll på differensen mellan det faktiska resultatet och det önskade resultatet av träningen. Själva optimeringen sker sedan genom att algoritmen försöker gradvis minimera differensen som återfås av förlustfunktionen genom att mellan varje epok uppdatera vikterna i nätverket.

Uppdateringen av vikterna sker med en metod som kallas för *bakåtpropagering*. Principen bygger på att en viss neurons fel i utsignalen beror på dess insignaler som härstammar från föregående lagers neuroner. Processen påbörjas i nätverkets ytterlager och propagerar därefter bakåt till dess förstalager som tar emot indatan.

3.2.7 Bildaugmentation

Bildaugmentation är ett sätt att utöka den mängd data man har för att träna ett nätverk. Det finns två tillvägagångssätt: Antingen augmenteras bilderna och sparas (vilket kan vara mycket minneskrävande), eller så görs augmenteringen *online* vilket sker under träning och de augmenterade bilderna förekommer endast temporärt. Keras (3.3.7) erbjuder online-augmentation och exempel på operationer är:

- Vända data horisontellt/vertikalt
- Omskalning
- Rotering
- Olika former av färgmanipulation

3.2.8 Objektssegmentering

Objektsegmentering är när man för varje klassificerat objekt i en bild försöker identifiera exakt vilka pixlar i bilden som hör till det identifierade objektet och inte. Det är då möjligt att skapa en så exakt representation av objektet som identifierats som möjligt.

3.2.9 Region Proposal Network (RPN)

Ett region proposal network (RPN) genererar med hjälp av en bild en mängd av rektangulära regioner i bilden där intressanta föremål kan befinna sig samt en poäng med hur troligt det är att den rektangulära regionen tillhör ett föremål eller bakgrunden i bilden [15].

3.2.10 Fast R-CNN

Fast R-CNN-nätverk tar en bild och en mängd av förslag på föremål. Nätverket behandlar bilden och genererar förslag på intressanta regioner (RoI) där det kan finnas ett av de aktuella föremålen. För varje RoI produceras en sannolikhet genom aktiveringsfunktionen softmax för den mängd av föremål som nätverket matades med samt en bakgrundsklass. Nätverket klassificerar RoI och avgränsar regionen i ett rektangulärt område [16].

3.2.11 Faster R-CNN

Faster R-CNN består av två steg: Det första steget är ett RPN som genererar förslag på intressanta rektangulära regioner där föremål kan befinna sig. Det andra steget är ett Fast R-CNN som använder RoI pooling för att klassificera dom intressanta regionerna och minimera den rektangulära regionen anpassad till ett klassificerat föremål [15].

3.2.12 Mask R-CNN

Mask R-CNN är ett ramverk för objektsegmentering utvecklat av Facebook AI Research. Ramverket bygger på Faster R-CNN, men utökar den genom att maskera föremålet som identifierats. Ramverket är identisk för det första steget som är RPN. I det andra steget körs klassificering och RoI pooling parallellt med segmentering för att generera en mask för varje RoI [17].

3.3 Mjukvara

Detta avsnitt tar upp de ramverk och programmeringsspråk som använts i arbetet.

3.3.1 Java

Java är ett objektorienterat programmeringsspråk som används för allt ifrån diskmaskiner till spel. Språket möjliggör användning av objektorienterade principer som arv, inkapsling, abstraktion och polymorfism. Skälet till att Java används i projektet är då applikationen skrivs för plattformen Android och detta görs i Android studio med Java [18].

3.3.2 Android studio

Android studio är den officiella integrerade utvecklingsmiljön för Googles operativsystem Android. Miljön bygger på JetBrains IntelliJ IDEA för Java [19]. I Android Studio finns möjlighet att utveckla, testa och exportera mobilapplikationer på ett enkelt sätt [20].

3.3.3 Room

Room är ett bibliotek för Android Studio som ger ett lättanvänt abstraktionslager för skapande och användning av lokala databaser. Room avgränsar utvecklaren från det underliggande databassystemet SQLite. Fördelarna med Room är att det är effektivt, lättanvänt och utför felkontroll redan vid kompilering [21].

3.3.4 TFlite

TFlite (TensorFlow Lite) är Googles lättviktsversion av TensorFlow för klientapplikationer som exempelvis mobila applikationer. Det är ett open-source ramverk som möjliggör konvertering av befintliga modeller till .tflite format och användning av modellerna för slutledning. TFlite erbjuder också möjligheten att konvertera modeller från att använda 32-bitars float till 8-bitars int för att optimera slutledningstiden. Generellt byggs och tränas modeller på datorer med god prestanda, för att sedan kunna konverteras och användas på enheter likt mobiltelefoner. TFlite går att använda i Android Studio. Fördelen med att kunna använda modeller lokalt istället för att skicka begäran till en server är att svarstiden blir betydligt kortare [22].

3.3.5 Python

Python är ett programmeringsspråk som är känt för sin lättanvändhet, enorma standardbibliotek och stöd för olika programmeringsparadigmer som exempelvis objektorienterad programmering och funktionell programmering. Språket har blivit populärt inom maskininlärning då det finns ett stort utvecklarstöd för att förbättra och underhålla bibliotek som skapar abstraktionslager över den underliggande tekniken. Detta gör det enkelt att skapa, träna och exportera artificiella neurala nätverk [23].

3.3.6 Tensorflow

Tensorflow är en plattform för maskininlärning med öppen källkod utvecklad av Google. Plattformen har ett brett utbud av verktyg, bibliotek och resurser för maskininlärning som kan användas av både forskare och utvecklare vid framtagning av applikationer i området [24].

3.3.7 Keras

Keras är ett högnivå-API för neurala nätverk skrivet i Python. Keras kan köras ovanpå befintliga plattformar för maskininlärning såsom Tensorflow. API:et utvecklades för att möjliggöra snabb utveckling av experiment med neurala nätverk genom sin användarvänlighet, modularitet och skalbarhet. Keras kan användas utan att användaren behöver interagera själv med komplexa plattformar som exempelvis Tensorflow [25].

3.3.8 CUDA

CUDA är en beräkningsplattform utvecklad av NVIDIA som möjliggör beräkningar på en dators grafikkort. Genom att utnyttja grafikkortets förmåga att utföra beräkningar ökar beräkningshastigheten kraftigt jämfört med beräkningar på en processor. I applikationer som använder grafikkort körs den sekventiella delen av programmet på processorn medans den tunga beräkningsdelen av programmet körs parallellt på grafikkortets kärnor [26].

3.3.9 cuDNN

cuDNN är ett bibliotek för grafikkortsacceleration för djupa neurala nätverk utvecklat av NVIDIA. cuDNN har optimerade implementationer för standardrutiner anpassade för djupa neurala nätverk. Biblioteket ökar beräkningsgraden hos plattformar för maskininlärning som exempelvis Tensorflow [27].

3.3.10 Anaconda Individual Edition

Anaconda IE är ett program med öppen källkod utvecklat för att förenkla utförandet av maskininlärning och vetenskapliga beräkningar i Python på en enskild dator. Programmet förenklar hanteringen av paket och bibliotek i virtuella miljöer [28].

3.3.11 Matterport Mask R-CNN

Matterport Mask R-CNN är en implementering av Mask R-CNN i Python, Keras och Tensorflow. Modellen genererar avgränsade rutor och segmenteringsmasker för varje instans av ett objekt i en bild. Implementeringen följer mestadels Mask R-CNN men avviker i ett fåtal fall till fördel för förenkling och generalisering. Exempelvis skalar man om alla bilder inför träning till samma storlek utan att påverka sidförhållanden, man skapar avgränsningsrutor under träning istället för att läsa in dem och man ändrar inlärningshastigheten [29] [17].

3.3.12 MobileNet

Som svar på ett allt större intresse för mindre och effektiva neurala nätverk lanserades år 2017 MobileNet [30]. CNN-arkitekturen för MobileNet har strömlinjeformats och har väldigt få parametrar i jämförelse med andra typer av modeller inom samma klass av nätverk. Tanken är att denna lättviktsmodell skall kunna användas i inbyggda system och mobila enheter med god prestanda och träffsäkerhet. Detta uppnås med en särskild typ av faltning som kallas för breddvis separerbar faltning [31].
Modellen erbjuder möjligheten att anpassa hyperparametrar för att konfigurera nätverket till särskilda ändamål som exempelvis geo-lokalisering och objektdetektion.

Kapitel 4

Metod och utförande

I detta kapitel redogörs det tillvägagångssätt och resonemang som ligger bakom många av de val som gjorts under arbetets gång. Valen innefattar identifiering av lämpliga modeller för konfiguration och överföringsträning, val av operativsystem för mobilapplikationen samt metodiken bakom datainsamlingen och datans förbehandling. Experiment definierades därefter för att undersöka olika parametrars bidrag till förbättrandet av modellerna för just detta ändamål.

4.1 Modeller

I detta avsnitt presenteras resonemanget bakom valet av just de modeller som användes i arbetet.

4.1.1 Klientmodell

Eftersom Keras används för maskininlärning i detta projekt valdes en klientmodell ut ur de som finns tillgängliga i Keras API [32]. Som tidigare nämnt är ett kriterie för en klientbaserad klassificeringsmodell på en mobilenhet att modellen skall vara liten men effektiv. För att hitta en modell som kan uppnå god precision med ett relativt sett litet antal parametrar användes Keras tabell för de olika tillgängliga modellerna som underlag.

Som man kan se i figur 4.1 är MobileNet-arkitekturerna små storleksmässigt men uppnår ändå god träffsäkerhet jämfört med andra modeller trots betydligt lägre antal parametrar. Träffsäkerheten representeras här av två kategorier *Top-1 Accuracy* och *Top-5 Accuracy*.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

Figur 4.1: Lista med tillgängliga modeller i Keras API

Kategorierna representerar modellens träffsäkerhet i klassificering av ImageNet-tävlingens testbilder. Den förstnämnda kategorin avser andelen av testbilderna där modellens högsta konfidensvärde representerar samma klass som testbilden representerar, och därmed rätt klassificering. Den andra kategorin avser en andel där klassen som representeras av testbilden ingår i de fem högsta konfidensvärden som ges av modellen vid klassificering. Eftersom detta arbete behandlar två klasser *skada* och $icke - skada$ så är den första kategorin av mest intresse, då utlåtande ges om två klasser snarare än fem eller fler.
De två kandidaterna är alltså MobileNet och MobileNetV2. Det förstnämnda valdes för detta projekt på grund av det förenklar modifiering av arkitekturen vilket beskrivs närmare i 4.6.1.

4.1.2 Servermodell

Till den serverbaserade modellen fanns möjlighet att använda en tyngre modell än den som var tänkt att köras på mobilapplikationen. Här önskades en modell som utöver att klassificera en bild som skadad eller icke-skadad också kunde identifiera flera instanser av skador i en bild. Efter en tids efterforskning valdes instanssegmentering genom Mask R-CNN som modell för servern. Anledningen till att instanssegmentering valdes var för att den utöver att generera ett avgränsningsområde till en identifierad instans också genererar en maskering av exakt det område som identifierats, se figur 4.2. Maskeringen bidrar till att man kan försäkra sig om att det område som modellen identifierade som skada faktiskt är det skadade området och inte något annat i bilden. Mask R-CNN valdes eftersom den är den populäraste metoden för instanssegmentering [33] och har lättillgänglig resurser för implementation.

Figur 4.2: Exempel på segmentering med Mask R-CNN

4.2 Mobilapplikation

Systemet utvecklades som en mobilapplikation då det ansågs lämpligt för en rad intressanta användningsområden. Skadeidentifiering i ett kompakt format som kan tas med var som helst skulle kunna användas för att exempelvis få kostnadsförslag på reparationer utan att besöka verkstaden, skicka med utökad information vid skadeanmälan till försäkringsbolag, direkt skaderapportering av fordon som tillhör en bilpool vid inträffandet av skadan, eller möjligtvis besiktning av hyrbilar vid avhämtning och återlämning. Vidare skulle lätta klientbaserade neurala nätverk lämpa sig i inbyggda system och IoT-komponenter i allmänhet. Då en mobilapplikation avsågs utvecklas uteslöts inte heller möjligheten för serverbaserad klassificering med mer sofistiskerade modeller, vilket också är fallet i detta arbete. Dessa argument ligger till grund för valet att utveckla en mobilapplikation.

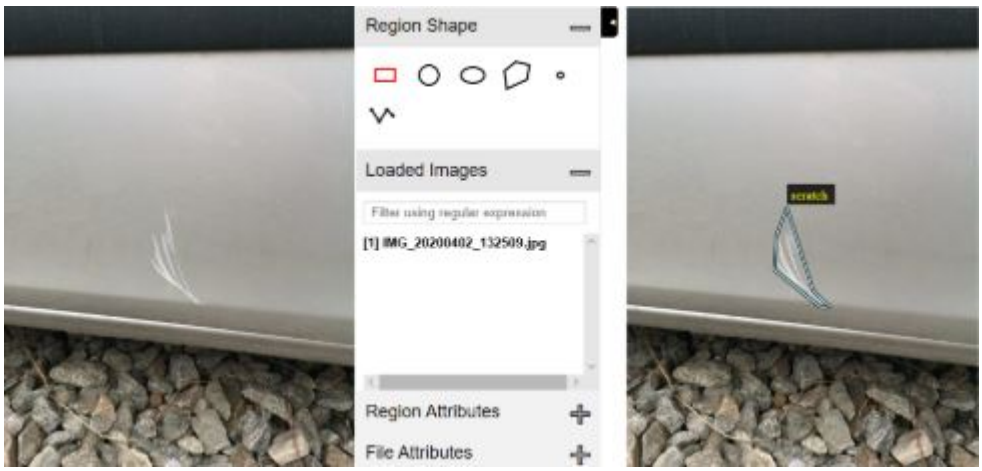
Android valdes som det operativsystem applikationen utvecklades för. Det fanns ingen direkt teknisk orsak till detta beslut mer än att det är den typen av mobila enheter som fanns tillgängliga för arbetet.

4.3 Datainsamling

För att träna ett neuralt nätverk för bildklassificering krävs generellt ett stort dataset av bilder. Eftersom applikationens mål är att identifiera repor och bucklor på en fordonsdörr behövde datasetet byggas med bilder anpassade för de ändamålet. Datasetet byggdes med över 1000 egentagna bilder på skadade dörrar samt närbilder på fordonsskador. Bilderna samlades in hos aktörer i Göteborgsregionen som hanterar skadade eller förverkade fordon. Bilderna samlades in med kamera hos mobiltelefoner som är det tänkta användningsområdet för applikationen. Bilderna används som tränings- och valideringsdata för maskininlärning i applikationen. Innan bilderna användes för träning av modeller behövde de beskäras så att bildörren var i fokus snarare än omgivningen. Därefter ändrades storleken på bilderna för att inläsningen innan träning inte skulle ta för lång tid.

4.3.1 Annotering av data för Mask R-CNN

Bilderna maskerades för användning i Mask R-CNN med hjälp av VVG Image Annotator och delades upp i tränings och valideringsdata. Exempel på maskering av en skada kan ses i figur 4.3 nedan [34]. Till tränings- och valideringsdata skapades en JSON-fil med information om varje maskerings plats på bilder i form av x- och y-koordinater, samt vilken typ av skada som förekom på bilden.



Figur 4.3: Exempel på annotering av en repad bildörr för användning av Mask R-CNN.

4.4 Användning av Modeller

Användning av Matterport Mask R-CNN

För att kunna använda Matterport Mask R-CNN krävdes en del förarbete. Dels behöves ett grafikkort anpassas för att kunna utföra beräkningar som maskininlärning drar nytta av, detta för att effektivt kunna köra den tunga modellen. Eftersom utvecklingen inom maskininlärning går snabbt fram och alla nya versioner av bibliotek och program inte är bakåtkompatibla behövde systemet anpassas med rätt versioner av de bibliotek och program modellen är utvecklad för att använda. Det kan vara värt att notera att det är möjligt att använda modellen med andra system än Windows som användes i detta projekt och andra versioner av bibliotek och program än de som använts, se installationsguide i Bilaga C.

Användning av MobileNet

För att installera den nödvändiga mjukvaran för att använda MobileNet och övriga modeller som erbjuds av Keras API [32], skapades en virtuell miljö med Anaconda likt följande:

```
1       conda create -n tf-gpu tensorflow-gpu
2       conda activate tf-gpu
```

Följande versioner installerades då:

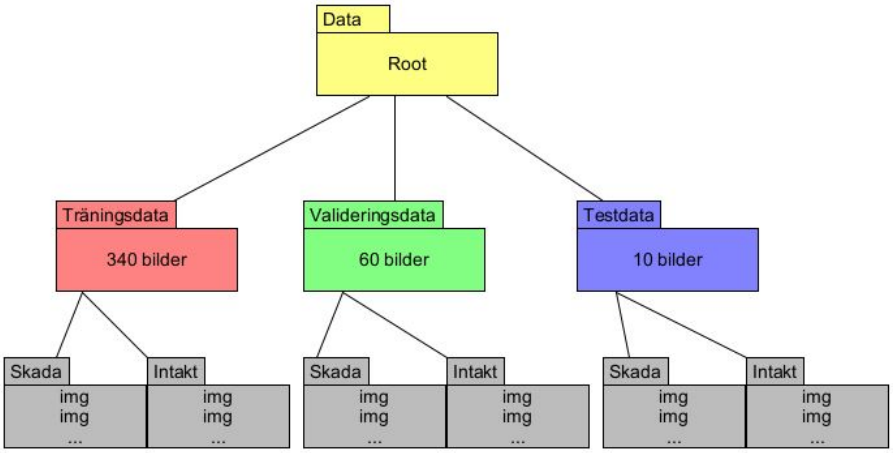
```
1       Tensorflow-gpu  2.1.0
2       cudnn           7.6.5
3       cudatoolkit     10.1.243
```

4.5 Träning av modeller

Eftersom det dataset som använts är begränsat och det ofta krävs enormt stora dataset för att nå bra resultat vid maskininlärning användes överföringsträning för både klient- och servermodellen.

4.5.1 Klientmodell

Innan träning organiserades datan enligt följande:



Figur 4.4: Filstruktur för organisering av data inför träning

Vid inläsning av data med denna filstruktur blir det enkelt att associera en viss bild med dess klass baserat på mapparnas namngivning. För att modellen inte skall tränas på datan som den ordning den lästes in, så blandas de inlästa bilderna mellan varje epok. Inför träning skapades en *callback* [35] som har i uppgift att bevaka träningsförloppet och konfigureras till att göra olika saker beroende på olika värden man önskar bevaka. Callbacks är en del av Keras standardbibliotek.

```
1  # Configure ModelCheckpoint callback
2  cb_list = [
3          ModelCheckpoint(MODEL_NAME,
4                  monitor='val_accuracy',
5                  mode='max',
6                  save_best_only=True,
7                  verbose=1)]
```

I listan av callbacks som skapas på rad 2 skapas en *ModelCheckpoint* som sparar en version av den tränade modellen vid specifika tillfällen. Rad 4 och 5 innebär att träffsäkerhet för validering är det värde som skall bevakas med avseende på maximalt uppnått värde mellan epoker. Rad 6 innebär att en version av modellen sparas varje gång en förbättring skett. Rad 7 möjliggör återkoppling till användaren under träningens gång. Överföringsträning påbörjades därefter med ImageNets förtränade vikter.

4.5.2 Servermodell

Vid träning av Mask R-CNN-modellen för klassificering och maskering av skador användes överföringsträning med redan tränade vikter på COCO-datasetet [36]. För att anpassa inlärningen till grafikkortets resurser kunde inte alla lager i nätverket tränas vidare på det nya dataset som tagits fram. Trots dessa begränsningar resulterade det ändå i möjligheten att uppnå resultat med ett begränsat dataset och begränsade minnesresurser på grafikkortet. Under experimentförloppet tränades modellen med endast en klass att identifiera. Initialt tränades modellen för endast repor och som sista experiment för skador där både repor och bucklor är inkluderade.

4.6 Experiment

I detta avsnitt presenteras de experiment som avses utföras i syfte att optimera och finjustera modellerna.

4.6.1 Klientmodell

Detta avsnitt består av ett antal delexperiment som avser undersöka olika hyperparametrars bidrag till den slutgiltiga modellen. Detta görs i syfte att finjustera modellen. Det bästa bidraget från varje del följer med till nästa delexperiment. I experimenten användes samma filstruktur och data som beskrivs i 4.5.1. Vidare användes en ModelCheckpoint som sparade modellen varje gång träffsäkerheten förbättrades under träningen. Vikterna är i alla experiment initierade med ImageNets vikter. Testerna utfördes med följande specifikationer: Windows 10, Nvidia RTX 2060 SUPER, Intel i5 4690k, 16gb RAM.

Epoker	32
Inlärningskoefficient	0.0001
Alpha	1
Frozen index	0
Cutoff	Nej
Weight init	ImageNet
Model	MobileNet

Tabell 4.1: Startkonfiguration

Experiment 1: Antal tränade lager

Modellens vikter är initierade med ImageNets vikter. Detta experiment avser undersöka hur resutlatet påverkas av att träna vidare på ett visst antal lager snarare än samtliga lager i nätverket. För att uppnå detta används en metod som kallas *frysning* och man kan då definiera inom vilket intervall ett antal lager förblir oförändrade med de initierade vikterna.

Frozen index	Beskrivning
0	Alla 94 lager tränas
-5	Sista 5 lager tränas
-10	Sista 10 lager tränas
-20	Sista 20 lager tränas
-40	Sista 40 lager tränas

Tabell 4.2: Delexperiment

Experiment 2: Antal filter

MobileNet har en hyperparameter som kallas *alpha*. Parametern är proportionell mot antal filter i nätverket. På så vis kan man styra i vilken grad modellen skall lära sig egenskaper. Parametern kan anta värden: 0.25, 0.5, 0.75 och 1 som är standard. Samtliga inställningar kommer att undersökas i experimentet.

Alpha	Beskrivning
1	Standardkonfiguration
0.75	25% minskning av filter
0.5	50% minskning av filter
0.25	75% minskning av filter

Tabell 4.3: Delexperiment

Experiment 3: Minska antal lager

En bidragande faktor till överanpassning är antalet parametrar i ett nätverk. Detta antal måste sättas i relation till komplexiteten av problemet. En möjlig hypotes är således att färre antal parametrar kan minska förlusten genom att hindra modellen från att lära sig brus.
Innan borttagning av rad 4-12:

```
...
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)              0
_________________________________________________________________
reshape_1 (Reshape)          (None, 1, 1, 512)        0
_________________________________________________________________
dropout (Dropout)            (None, 1, 1, 512)        0
_________________________________________________________________
conv_preds (Conv2D)          (None, 1, 1, 1000)       513000
_________________________________________________________________
reshape_2 (Reshape)          (None, 1000)             0
_________________________________________________________________
act_softmax (Activation)     (None, 1000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                2002
=================================================================
Total params: 1,344,538
Trainable params: 1,333,594
Non-trainable params: 10,944
_________________________________________________________________
```

Efter borttagning av rad 4-12

```
...
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                1026
=================================================================
Total params: 830,562
Trainable params: 819,618
Non-trainable params: 10,944
_________________________________________________________________
```

4.6.2 Servermodell

Modellerna som togs fram genom experimenten på servermodellen tränades på en dator med följande specification: Operativsystemet Windows 10, grafikkortet Nvidia GTX 1060, processor Intel i5-7600 och 32 GB ram. De program, paket och bibliotek som användes för att träna modellerna finns att läsa om i avsnitt 4.4. Alla bilder som använts i experimenten är annoterade enligt avsnitt 4.3.1. Bilderna är för varje experiment uppdelade i en träning- och en valideringsdel, där träningsdelen består av ca 85% av bilderna och resterande 15% hör till valideringsmängde. Träningen av alla experiment utfördes med hundra steg per epok, tio epoker och femtio valideringssteg. Bilderna skalades av modellen om till 1024*1024 utan att ändra på sidoförhållanden. Modellen tränades för endast två klasser i alla experiment. I experiment 1 och 2 tränades modellen för klasserna repa och bakgrund och i experiment 3 tränades modellen för skada och bakgrund eftersom experiment 3 innehåller bilder på både repor och bucklor. I alla experiment tränas nätveket med en bild åt gången på grafikkortet, inlärningsgraden är 0.001 och kravet på säkerhet för identifiering av skada är 90%. För att kunna utvärdera de olika experimenten valdes 32 bilder ut som vardera innehöll minst en repa eller buckla som modellerna kunde testas på. Dessa bilder kommer modellerna inte tränas på utan kommer endast användas för att presentera resultatet i kapitel 6.3. Vid alla experiment användes överföringsträning med utgångspunkt från tränade vikter på COCO-datasetet.

När huvuden i nätverket nämns i experimenten nedan menas följande delar av nätverket: RPN, segmentering samt klassificering. Hur modellerna som tagits fram i de olika modellerna presterar presenteras i kapitel 6.3.

Experiment 1
Till experiment 1 annoterades ett mindre antal bilder på repade fordonsdörrar. De annoterade bilderna var 66 stycken och användes för träning och validering. I det här experimentet tränades endast huvuden i nätverket. Poängen med experimentet var dels att snabbt testa modellen och se om det gick att nå resultat med ett extremt litet dataset samt få en utgångspunkt att jämföra med i senare experiment där datasetet var tänkt att utökas.

Experiment 2
Till experiment två utökades antalet annoterade bilder till 308. Utöver det utökade datasetet utfördes experiment 2 i tre steg: I första steget tränades nätverket på samma sätt som i experiment 1 där endast huvuden tränandes. Detta för att kunna se hur det utökade datasetet påverkade modellens resultatet. Därefter ökades antalet lager i nätverket som tränades. I steg två tränades även en liten del av de lager i nätverket som genererar egenskapskartor för bilder. Eftersom överföringträning används där vikterna kommer från en modell som tränats för ett stort antal klasser som skiljer sig ifrån en repa på en bildörr, var förhoppningen att bättre anpassade egenskapskartor skulle bidra till ett bättre resultat. För steg tre i experiment 2 utökades de lager som generar egenskapskartor för att ytterligare se om modellen gynnades av att kunna ändra fler vikter i nätverket. Vidare gjordes ett försök att träna alla lager i nätverket, men med ökat antal lager som tränas ökar minnesanvändnigen hos grafikkortet och i det här fallet räckte resurserna inte till. Här fanns en möjlighet att minska minnesanvändningen på andra områden, till exempel

antal RoIs per bild, bildens storlek eller en mängd andra parametrar vid träning av nätverket som kanske skulle gjort det möjligt att träna alla lager. Här gjordes avvägningen att inte anpassa andra delar i nätverket eftersom det skulle bidra till påverkan på resultatet på ett oförutsägbart sätt.

Experiment 3
För experiment 3 skapades ett nytt dataset med 264 annoterade bilder på fordonsdörrar, denna gång med repor och bucklor. Nätverket tränades på samma sätt som i experiemnt 2 steg tre. Här var målet att skapa en modell som skulle kunna identifiera både bucklor och repor på en skadad fordonsdörr. Denna modellen är den som var tänkt att användas av mobilapplikationen.

Kapitel 5

Systemutveckling

I det här kapitel beskrivs mobilapplikationens olika delar och hur interaktionen med användaren är tänkt. Eftersom målet med projektet är att belysa möjligheten att identifiera fordonsskador med en mobiltelefon har grundläggande funktionalitet utvecklats som proof of concept. För en övergripande bild av användarinteraktionen, se bilaga A, och för en övergripande bild av användargränssnittet se F.

5.1 Mobilapplikationen

Mobilapplikationen som är döpt till ScratchAI är skapad för mobiltelefoner som använder operativsystemet Android och är utvecklad i Android studios. Applikationens design och utveckling beskrivs i följande delavsnitt som bland annat tar upp kravställning, användargränssnitt och arkitekturmönster.

5.1.1 Krav

För de grundläggande funktioner som applikationen skall tillhandahålla definierades följande kravställning:

- Användaren skall mötas av en inloggningsskärm.
- Användaren skall kunna lägga till och ta bort fordon i sin fordonspool.
- Användaren skall kunna se skadehistorik för sina fordon.
- Användaren skall kunna göra en inspektion av ett fordon.
- Användaren skall kunna ta en bild på ett fordon.
- Användaren skall kunna välja modell för analys av bilden.
- Applikationen skall generera en skaderapport vid identifierad skada.
- Användaren skall kunna spara skaderapporten som genererats.
- Användaren skall vid identifierad skada få någon form av respons som representerar modellens konfidens av skada.
- Användaren skall ha tillgång till instruktioner för hur en bild bäst tas vid klassificering.

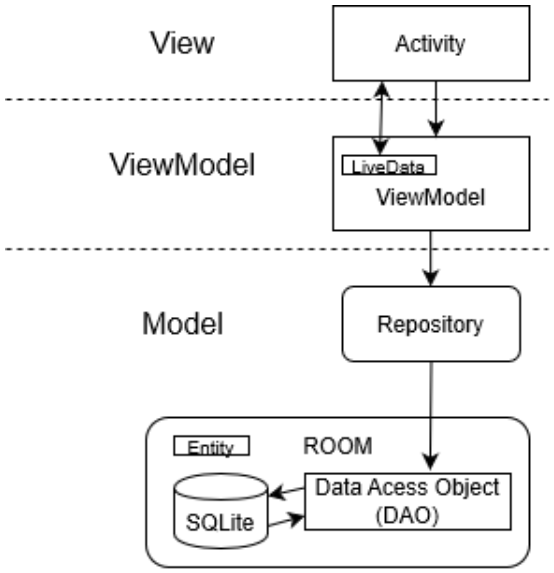
5.1.2 Användargränssnitt

Användargränssnittets är minimalistiskt framtaget och innehåller endast den funktionalitet som efterfrågats i kraven. Gränsnittets olika delar är byggda som aktiviteter i [37] Android och användaren flyttas mellan aktiviteter utifrån de val den gör i applikationen. En kort förklaring av aktiviteterna följer nedan.

Inloggningen kontrollerar användarens inloggningsuppgifter. *Panelen* ger användaren översikt över applikationens huvudfunktioner och möjlighet att navigera mellan dem. *Fordon*, användaren kan se registrerade fordon och lägga till eller ta bort fordon. Vid borttagning av ett fordon försvinner också fordonets skadehistorik. *Historik*, användaren kan se skaderapporter och filtrera på fordon. *Inspektion*, användaren kan inspektera ett fordon genom att ta en bild och välja modell för att generera en skaderapport. Vid identifierad skada får användaren möjlighet att komplettera skaderapporten med en kommentar. Det finns också möjlighet att visa utökad data som återfås vid klassificering, samt se instruktioner för hur en bild bäst tas.

5.1.3 Arkitekturmönster

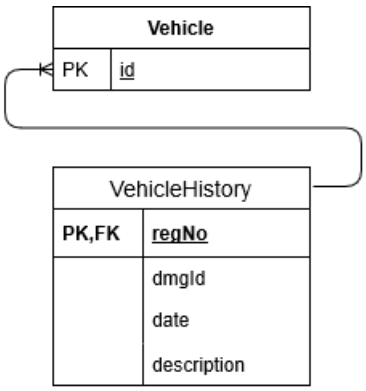
För att ge användaren kontroll över fordon i poolen och möjlighet att spara och koppla skaderapporter till fordon utvecklades en lokal sql-databas i Room. MVVM (Model-View-ViewModel) tillämpades för att dela upp kommunikationen med databasen i olika abstraktionslager.



Figur 5.1: Illustration av hur MVVM använts

Figuren ovan beskriver MVVM-mönstret som använts. *View* informerar ViewModel om användarens interaktion med en aktivitet. *ViewModel* har i uppgift att exponera view för strömmar av data från lägre abstraktionslager. ViewModel är en klass som också används också för retention av data då dess livscykel överstiger en aktivitet. Därför kan en aktivitet kopplad till en ViewModel förstöras och återskapas utan dataförlust. Exempelvis sker detta vid rotation av gränssnittet mellan porträtt- och landskapsläge. *Model* utgör ett abstraktionslager över datakällan och arbetar ihop med ViewModel för att spara och hämta data från databasen [38].

Arkitekturens lägsta nivå består av en SQLite-databas. Ramverket Room ger ett lättanvänt grässnitt för att implementera databasen med felkontroll redan vid kompilering. De minsta beståndsdelarna i databasen består av två tabeller (se figur 5.2). I fordonstabellen är registreringsnummer en primär nyckel, vilket innebär att endast unika nummer får förekomma i databasen. Dessa nycklar är sedan kartlagda som en främmande nyckel i historiktabellen vilket innebär att om ett fordon tas bort från databasen så försvinner också dess skadehistorik.



Figur 5.2: Databasen består av två tabeller. Vehicle sparar registreringsnummer för poolens fordon och VehicleHistory ett fordons associerade skaderapporter.

För att kunna utföra databasoperationer på tabellerna skapas gränssnitt som kallas för DAO (Data access object). Dessa gränssnitt definierar de operationer som skall kunna utföras på databasens tabeller och Room genererar vid kompilering faktiska objekt med tillhörande metoder som möjliggör detta. Exempel på ett DAO för tabellen *Vehicle* följer nedan.

```
1  @Dao
2  public interface VehicleDao {
3
4      @Insert
5      void insert(Vehicle vehicle);
6
7      @Delete
8      void delete(Vehicle vehicle);
9
10     @Query("DELETE FROM vehicle_table")
11     void deleteAllVehicles();
12
13     @Query("SELECT * FROM vehicle_table ORDER BY id ASC")
14     LiveData<List<Vehicle>> getAllVehicles();
15 }
```

Att lägga till eller ta bort ett registreringsnummer tillhör standardoperationerna och konversionen till faktiska metoder görs automatiskt. För särskilda operationer som exempelvis att ta bort alla fordon eller hämta alla fordon sorterade behöver metodprototypen kompletteras. Detta uppnås med en annotering innehållande SQL-anropet som utför operationen. LiveData som kan ses i rad 14 är en datatyp som

omsluter returlistan med fordon och håller listan uppdaterad på en bakgrundstråd asynkront med hjälp av observermönstret [39].

Databasen byggs med en abstrakt klass innehållandes tabellerna och instansieras som en singletonklass [40] för att undvika mer än en instansiering av databasen. Görs ett försök att skapa mer än en databas så returneras den befintliga.
Repository är en javaklass som implementerar metoder för de olika databasoperationerna som ViewModelklassen kan använda sig av. När metoderna anropas och en operation skall utföras görs detta asynkront med AsyncTask för att inte belasta den tråd som gränssnittet körs på [41]. I detta avsnitt beskrivs implementationen av bildanalysen.

5.1.4 Klient

För att klientmodellen skall kunna köras på mobilapplikationen måste den konverteras från det befintliga formatet efter träning till ett TFlite-kompatibelt format. Tensorflows bibliotek tillhandahåller olika klasser och metoder för att uppnå detta. Filen som konverterats består av modellens arkitektur samt de vikterna som justerades under träning. Väl konverterad, importerades TFlite-filen och en textfil med kategorierna *skada* och *icke-skada* till Androidprojektet.

5.1.5 Server

Pythonservern kommunicerar med androidapplikationen över sockets. När applikationen kopplat sig mot servern tar servern emot en bildfil från applikationen och låter Mask R-CNN-modellen göra en analys av bilden. Om analysen resulterar i en upptäckt skada sparas bilden på systemet med de skadade områdena maskerade. Den nya bilden med maskerade skador skickas sedan tillbaka till applikationen. Om analysen av bilden inte resulterar i en upptäckt skada förmedlar servern detta till applikationen och ingen bild skickas tillbaka.

Kapitel 6

Resultat

I detta kapitel presenteras i huvudsak resultaten av de experiment som utförden på MobileNet och Mask R-CNN.

6.1 Applikation

Med avseende på kravställningen i 5.1.1 så uppfylldes samtliga krav med implementationen av applikationen. Utöver dessa krav för funktionalitet blev det slutgiltiga användargränssnittet lätt att förstå och minimalistiskt. För bilder på gränssnittet se bilaga F.

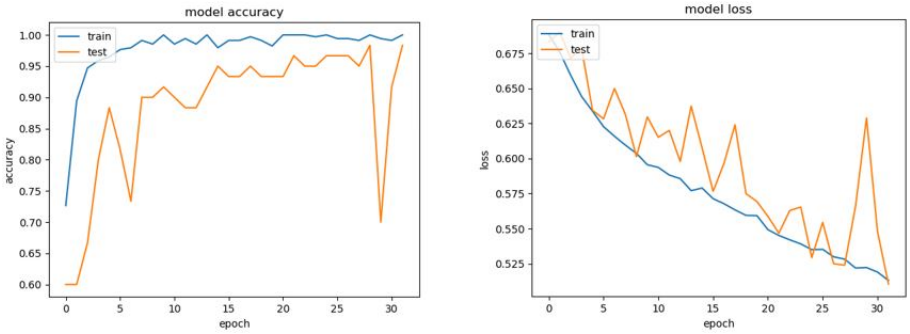
6.2 MobileNet

De parameterinställningar som visade sig bidra med högst träffsäkerhet och lägst förlust till modellen var: alpha = 0.5, frozen index = -20 samt borttagning av lager i slutet av nätverket. Experiment 3 visade också att högre träffsäkerhet och lägre förlust uppnåddes genom att öka antal epoker till 64. För att se hur de bästa momentana modellerna från varje experiment presterade på testbilderna se bilaga B.

6.2.1 Experiment 1

FROZEN_INDEX	Förlust (trä.)	Träffsäkerhet (trä.)	Förlust (val.)	Träffsäkerhet (val.)
0	0.5539	1.0000	0.5882	0.9333
-5	0.6281	0.8824	0.6486	0.7833
-10	0.5354	0.9971	0.5640	0.9667
-20	0.5219	1.0000	0.5661	0.9833
-40	0.5901	1.0000	0.6171	0.9833

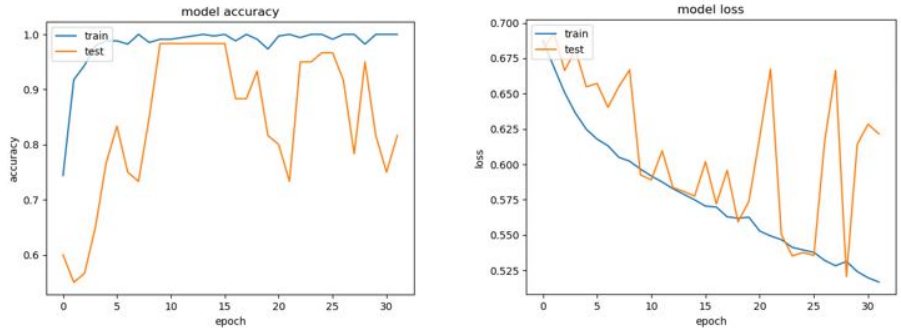
Tabell 6.1: Resultat från de bästa epokerna med avseende på valideringsträffsäkerhet i experiment 1



Figur 6.1: Träffsäkerhet och förlust vid träning av sista 20 lager

6.2.2 Experiment 2

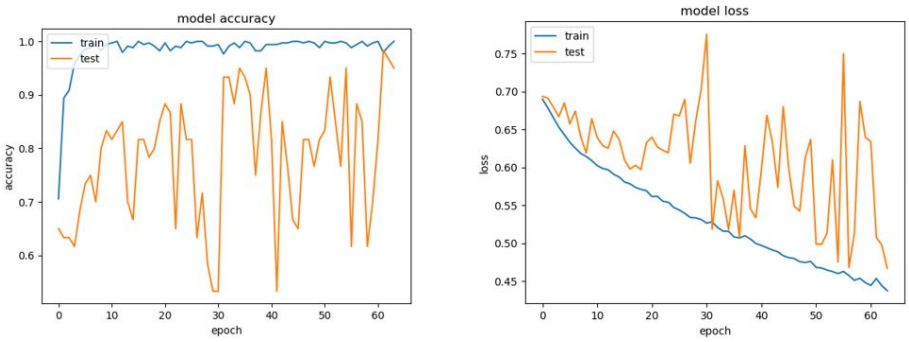
Att sätta alpha till 0.5 gav periodvis högre träffsäkerhet medan förlusten förblev hög i samtliga fall. I graferna framgår det också en varierad grad av instabilitet. Graferna för samtliga värden av alpha återfinns i bilaga D.



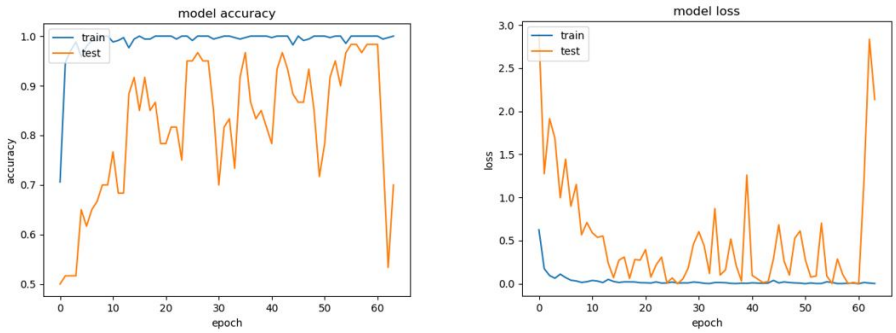
Figur 6.2: Träffsäkerhet och förlust med alpha 0.5

6.2.3 Experiment 3

Att ta bort ett antal lager enligt beskrivningen i metoddelen resulterade i en markant sänkning av förlusten som man kan se i figur 6.4. Även träffsäkerheten blev mindre instabil till följd av ändringen. De bästa epokerna med avseende på träffsäkerhet uppnådde samma högsta träffsäkerhet om 0.9833. Däremot var förlusten i samma epok 0.0027 för modellen med borttagna lager och 0.5078 i modellen utan borttagna lager.



Figur 6.3: Träffsäkerhet och förlust utan borttagna lager



Figur 6.4: Träffsäkerhet och förlust med borttagna lager

6.3 Mask R-CNN

Modellerna som tagits fram i experimenten testades mot de 32 bilder som valts ut som testbilder. Notera att endast modellen från experiment 3 är tränad med bilder på bucklor och att det finns en del bilder bland de 32 som inte innehåller några repor utan endast bucklor. Bedömningen av resultatet i tabell 6.2 nedan är av mänsklig observation där varje bild hamnar i en av de fyra kategorierna. Alla testbilder innehåller minst en repa eller buckla av en mängd olika karaktärer. Resultatet av hur modellerna presterade under träningsfasen följer i figur E.1 till E.10 i bilaga E, notera att förlusten är den summerade förlusten av alla nätverkets delar (RPN, Klassificering, Avgränsningsområde och Maskering).

	Korrekt/Mestadels Korrekt Maskering	Delvis Korrekt Maskering	Felaktig Maskering	Ingen Maskering
Experiment 1	0	10	2	20
Experiment 2.1	3	13	11	5
Experiment 2.2	5	9	3	15
Experiment 2.3	15	7	1	9
Experiment 3	18	8	6	0

Tabell 6.2: Resultat av test på de 32 bilderna

Kapitel 7

Diskussion och slutsats

I detta kapitel kommer resultaten från samtliga experiment, implementation av applikationen och eventuella felkällor att diskuteras. Därefter presenteras slutsatsen, som följs av ett övervägande gällande hållbar utveckling och etik.

7.1 Utvärdering av mobilapplikation

Som nämnt i resultat så uppnåddes samtliga mål för applikationen. Manuellt testande har visat att applikationen beter sig som den ska vid förväntad användning. Fler automatiserade instrumenttester och modultester hade dock varit önskvärt.

7.2 Utvärdering av modeller

MobileNet
I samtliga experiment som utfördes för klientmodellen kunde man observera att modellen lärde sig den data som den tränades på enkelt och snabbt. Denna effekt kan tillskrivas till det faktum att ImageNet-vikterna redan medför en hel del information gällande olika objekts former vilket kan ha påskyndat inlärningen. Detta kan vara en möjlig orsak till varför valideringsträffsäkerheten är generellt instabil genom samtliga experiment. Modellen kan vid träning då gå över till att lära sig brus (överanpassning) som blir svårt att validera då det kan skilja sig mycket i detaljer mellan bilderna.

I experiment 1 (6.2.1) visade det sig att överföringsträning av de 20 sista lager i modellen gav bäst resultat med avseende på valideringsträffsäkerhet och valideringsförlust sammantaget (tabell 6.1). Det resultat som avviker mest från övriga delexperiment är då endast de sista fem lager tränades. En möjlig anledning är att dessa lager inte har lika stor betydelse för nätverkets förmåga att lära sig.

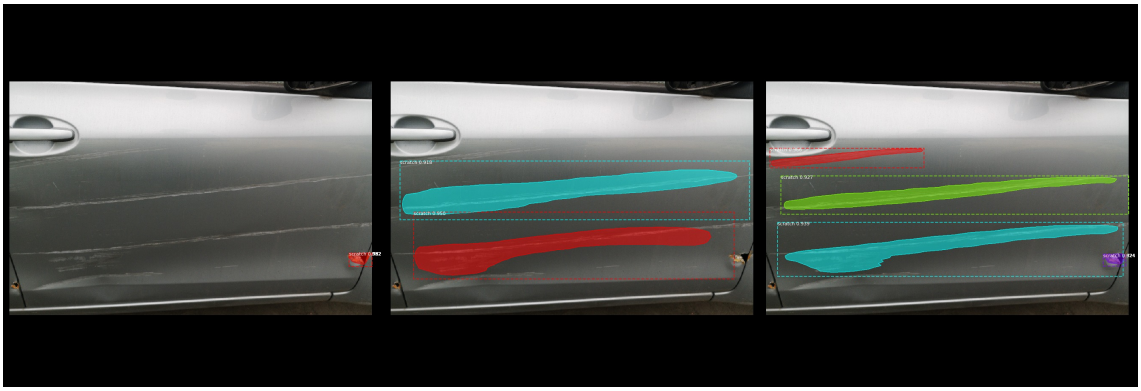
I experiment 2 (6.2.2) kunde man i figur D.2 se flera höga värden för valideringsträffsäkerheten och mindre dramatiska fall i grafen jämfört med andra värden på alpha. Ett lågt alpha minskar antal filter och således egenskaper modellen kan lära sig. Har man ett lågt alpha kan det manifesteras i en generellt lägre träffsäkerhet som man ser i figur D.1. Ett högre värde på alpha (0.75, 1.0) visade sig orsaka större instabilitet och lägre generell valideringsträffsäkerhet se figurerna D.3 och D.4.

I experiment 3 (6.2.3) kunde man observera genrellt mycket lägre förlust både vid validering och träning. Detta är förväntat då för många parametrar resulterar i överanpassning vilket gör det svårare för modellen att göra utlåtanden med hög konfidens. Trots detta kan man observera fortsatt instabilitet under träning och validering. Inte helt oförväntat var modellen från experiment 3 säkrare och hade fler rätt i sina utlåtanden av testbilderna se bilaga B.

Mask R-CNN
Resultaten för de experiment som utfördes med Mask R-CNN presenteras i bilaga E.

I experiment 1 visar modellen potential att identifiera repor trots det minimala datasetet på 66 bilder. Modellen markerade en del repor på testbilderna delvis korrekt, se tabell 6.2, men lyckades inte markera några skador helt korrekt. Detta är troligtvis en följd av problematiken med att ingen av skadorna är den andra lik och att modellen inte tränats med tillräckligt stort dataset för lära sig skadornas olika karaktär.

I experiment 2 ökades datasetet vilket bidrog till att modellen i experiment 2.1 lyckades identifiera fler repor men också var mer benägen att göra felaktiga maskeringar, se tabell 6.2. När de lager som genererade egenskapskartor inkluderades i träningen under experiment 2.2 och 2.3 ökade träffsäkerheten samtidigt som de felaktiga maskeringarna minskade markant. Egenskapskartorna har alltså stor betydelse för hur modellen presterar när man använder överföringsträning och tränar för en ny typ av problem. Utvecklingen av hur träffsäkerheten förbättras under experiment 2 kan ses i figur 7.1.



Figur 7.1: Utveckling under Experiment 2

I experiment 3, som gav den slutgiltiga modellen för Mask R-CNN och var den som kom att användas i mobilapplikationen tränades modellen för både repor och bucklor under klassen skada. Modellen är den som presterar bäst på testbilderna se tabell 6.2. Modellen lyckades utöver att identifiera de flesta repor i testbilderna också identifiera ett antal bucklor, men hade också en större andel felaktiga maskeringar jämfört med den slutliga modellen i experiment 2. Anledningen till detta är att bucklorna bidrar till en ökad komplexitet där bland annat reflektioner och dörrarnas utformning kan tolkas av modellen som skada, detta går att läsa mer om under avsnittet *Felkällor* nedan.

7.2.1 Data

Av resultaten att döma finna starka indikationer på att ett utökat dataset skulle kunna bidra på ett positivt sätt i hur de slutgiltiga modellerna presterar. Man kan exempelvis se i experimenten på Mask R-CNN skillnaden i valideringsförlust mellan experiment 1 figur E.2 och experiment 2 figur E.4 där förlusten minskar med det utökade datasetet trots att träningsförlusten för experiment 1 figur E.1 är lägre än förlusten i experiment 2 figur E.3. I MobileNet kan man se att modellerna når hög träffsäkerhet på träningsdata bara efter ett fåtal epoker, se exempelvis figur 6.1 samtidigt som träffsäkerheten vid validering i experimenten ökar instabilt och når sin topp långt senare. Med ett utökat dataset vid träning är ett antagande att modellen når toppen av träffsäkerhet vid träning senare men samtidigt har en stabilare och högre träffsäkerhet på valideringen.

7.2.2 Felkällor

Mask R-CNN
Eftersom datasetet i experiment 1 är extremt litet kan de få bilderna i datasetet haft påverkan på förlusterna som inte är direktrelaterat till datasetets storlek utan istället är relaterat till de få bildernas komplexitet.

MobileNet
Man kan observera en stor skillnad mellan figurerna 6.1 och D.4 trots att modellen i dessa fall tränas med exakt samma konfiguration då alpha = 1.0 är med i startkonfigurationen av experimenten. Anledningen till såpass stor variation är som nämnt i 4.5.1 att bilderna blandas. Detta resulterar i högre träffsäkerhet men väldigt olika träningsförlopp från gång till gång. *Ordningen* av datan har alltså större betydelse än väntat.

En annan möjlig felkälla är att samtliga bilder på intakta dörrar togs under helt andra väderförhållanden än bilderna för skadade dörrar. I fallet med de intakta var det en solig dag vilket introducerade mycket reflektion och kraftigare skuggor i vissa av bilderna. I fallet med de skadade var det mulet och mindre störfaktorer i bilderna som togs. Man kunde i vissa bilder på intakta dörrar observera skuggor och reflektioner som kunde misstolkas som skador.

Gemensamma felkällor
Efter experimenten råder det inget tvivel om att enskilda bilder kan ha stor effekt på träningen av modellen. Detta kan vara på grund av att exempelvis få bilder har en synlig backspegel, vissa bilder har grova dörrlister eller kromade detaljer som reflekterar ljus. Alla dessa variationer kan i ett litet dataset betraktas som felkällor. Hade man i stället haft ett betydligt större dataset skulle dess effekt på modellerna antagligen minskat. För att se exempel på sådant gränsfall se bilaga G.

En annan felkälla är att i många fall var bildörrarna mer eller mindre smutsiga. Stänk som hade liknande karaktärsdrag som repor kan ha påverkat resultatet negativt.

7.3 Slutsats

Frågeställningen som formulerades initialt i arbetet har besvarats genom de resultat som uppnåtts. Med hjälp av de två modellerna som undersöktes och finjusterades enligt experimenten, kunde identifiering av skador ske. Applikationen som implementerades lyckades tillhandahålla denna teknik som önskat, och erbjuder de praktiska användarfunktioner som definierades enligt kraven i 5.1.1. Användandet av CNN nätverk i detta sammanhang gör att arbetets syfte uppnåtts. Definitiva slutsatser gällande förbättringar av modellerna under experimenten kan inte dras då mer utförliga och utökade tester hade behövts.

7.4 Hållbar utveckling och etik

Syftet med arbetet var att belysa ett möjligt användningsområde för bildanalys mha. maskininlärning och fordonsskador. Inom ramen för det användningsområde vi valde att undersöka visar det sig att ett system likt det som utvecklas kan effektivisera olika arbetsuppgifter utan att nödvändigtvis ersätta dem. Ur etisk synvinkel är detta fördelaktigt då det istället för att ersätta, underlättar och förbättrar en arbetsuppgift. Exempelvis behöver inte inspektioner av fordon göras med papper och penna då ett databassystem införts. Inte heller behöver fysiska kvitton scannas in elektroniskt eller bokföras fysiskt. Bildanalysen kan underlätta identifiering av skador som personen som utför inspektionen inte har lagt märke till. Med det sagt skulle tekniken om den vidareutvecklas och tillämpades i annan utformning kunna ersätta arbetsuppgifter för kontroll av fordon, vilket inte ligger i linje med projektets målsättning.

7.5 Fortsatt arbete

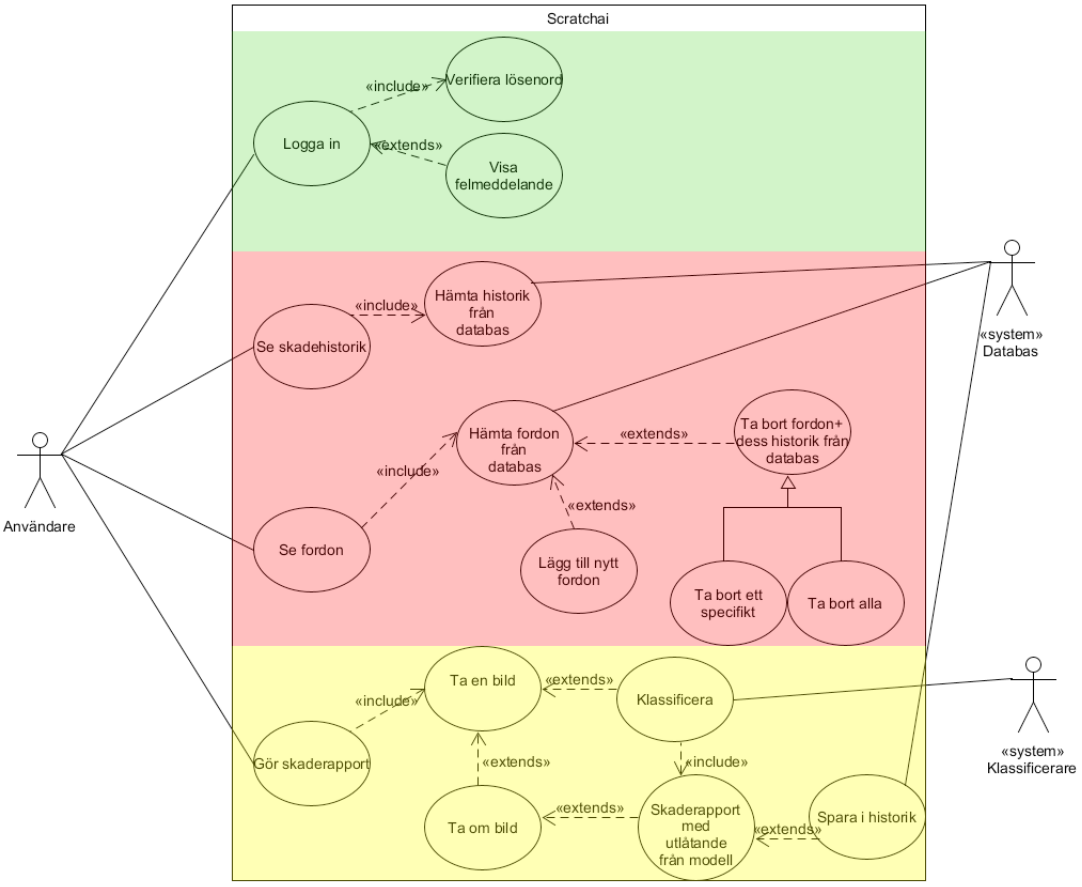
Ett fortsatt arbete skulle innebära en djupare undersökning av en specifik modell. Fördelen med att endast jobba med en modell tillskillnad från två som var fallet i detta arbete, är då att man kan bilda fler hypoteser och göra fler experiment inom den tidsram som finns.

Mask R-CNN erbjuder väldigt mycket användbar data med hjälp av segmentering som skulle kunna dras nytta av i ett fortsatt arbete, exempelvis för positionering av skada, omfattning och om det är en ny eller gammal skada. Vidare skulle ett utökat dataset varit önskvärt då det kan normalisera effekten av bilder med avvikande egenskaper som nämnt i felkällor. Utöver detta hade det varit idealiskt att samla all data och genomföra alla tester i en kontrollerad miljö. Med detta menas en miljö där ljus, avstånd från dörr när bilder tas och vinkel är konstanta. Utöver detta skulle alla bilar vara rena och därmed utesluts smuts som en möjlig felkälla.

Gällande mobilapplikationen hade det varit användbart att spara bilder på skador i samband med generering av skaderapport, så att användaren kan se skadan när en skaderapport visas.

Bilaga A

Use case diagram



Bilaga B

Testbilder och resultat för MobileNet

Img_id	Model (exp1)	Model (exp2)	Model (exp3)
Intact0	[0.4351841 0.5648159]	[0.4833446 0.51665545]	[0.07577761 0.92422235]
Intact1	[0.43518424 0.5648158]	[0.4463358 0.55366427]	[0.01390334 0.9860967]
Intact2	[0.4351841 0.56481594]	[0.4813355 0.5186646]	[0.01966875 0.98033124]
Intact3	[0.43519476 0.56480527]	[0.53702897 0.46297112]	[0.0669982 0.9330018]
Intact4	[0.43518463 0.5648154]	[0.4873854 0.5126146]	[0.00466431 0.99533576]
Damage0	[0.53900295 0.46099696]	[0.5440776 0.45592245]	[0.98456764 0.01543239]
Damage1	[0.57025373 0.4297463]	[0.5414897 0.4585103]	[0.9946648 0.00533521]
Damage2	[0.4352856 0.56471443]	[0.53153366 0.4684664]	[0.9717048 0.02829517]
Damage3	[0.5702541 0.42974594]	[0.5410064 0.4589936]	[0.998988 0.00101196]
Damage4	[0.44846964 0.5515304]	[0.47793475 0.5220652]	[0.04343923 0.95656085]

Bilaga C

Installationsguide Matterport Mask R-CNN

Anaconda användes som python miljö för att enkelt kunna installera och hantera paket och bibliotek. Kommandon som följer nedan är skrivna i Anacondas kommandoprompt

```
conda install python==3.6.2
```

```
conda install keras==2.0.8
```

```
pip install tensorflow-gpu==1.6.0
```

CUDA 9.0 laddes ner [42] och installerades för att möjligöra beräkningar på grafikkortet. Eftersom CUDA 9.0 är en äldre version så krävs även en äldre version av Microsoft Visual Studio [43] för att få en fullständig installation. Ytterligare laddades cudNN 7.0.5 ner [44] och installerades [43] så att Tensorflow kan utnyttja grafikkortets beräkningskraft.
För att använda implementeringen av Mask R-CNN klonades datakatalogen [29] från github och installerades i två steg. Först kördes requirements filen och därefter installationsfilen. Notera dock att requirements filen innehåller instruktioner för installation av Tensorflow och Keras som plockats bort manuellt för att undvika att nya versioner installeras som inte har bakåtkompabilitet.

```
python install -r requirements.txt
```

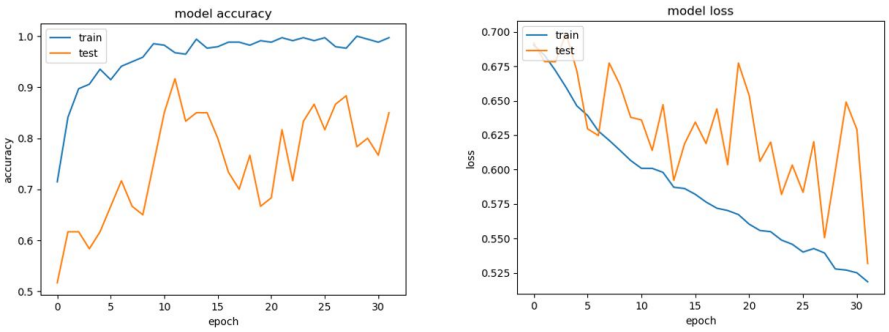
```
python setup.py install
```

Till sist uppdaterades filen model.py i datakatalogen för att komma åt ett programfel som uppstår i Keras på Windows. Två rader kod i funktionen *train* uppdaterades till följande:

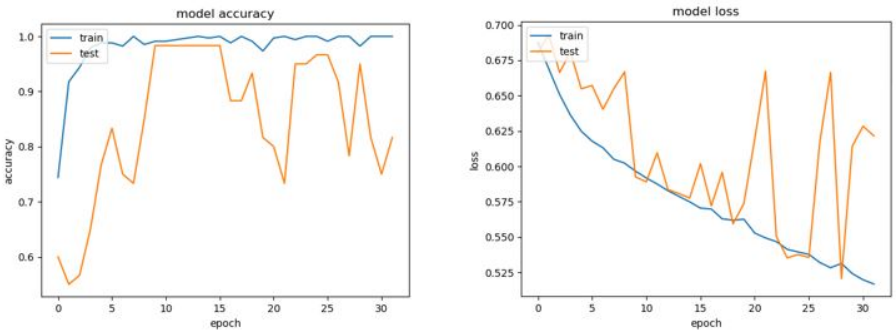
```
workers = 1
use_multiprocessing=False
```

Bilaga D

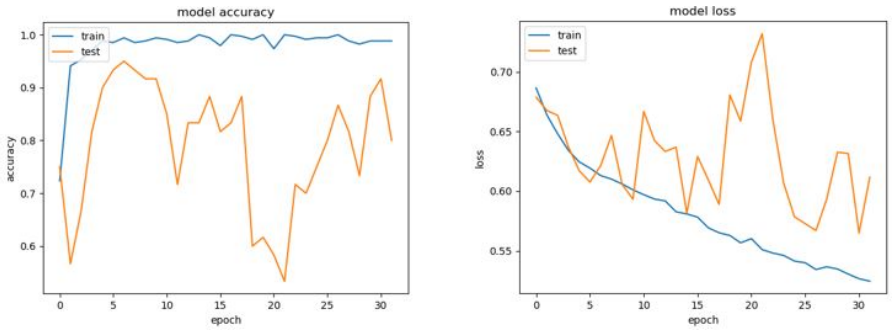
Resultat från experiment med MobileNet



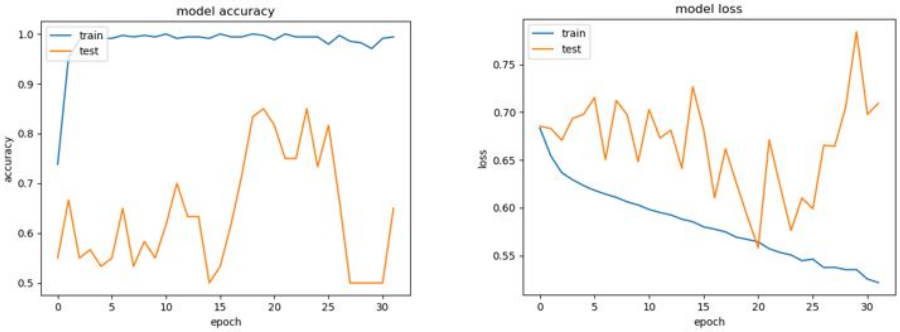
Figur D.1: Träffsäkerhet och förlust med alpha 0.25



Figur D.2: Träffsäkerhet och förlust med alpha 0.5



Figur D.3: Träffsäkerhet och förlust med alpha 0.75

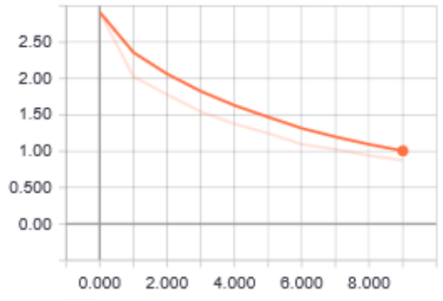


Figur D.4: Träffsäkerhet och förlust med alpha 1.0

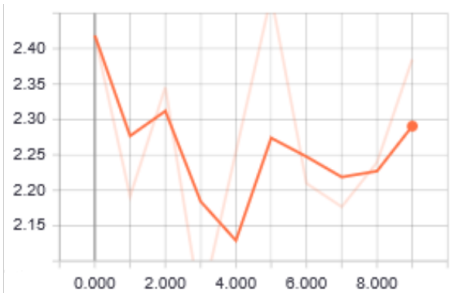
Bilaga E

Resultat från experiment med Mask R-CNN

Experiment 1

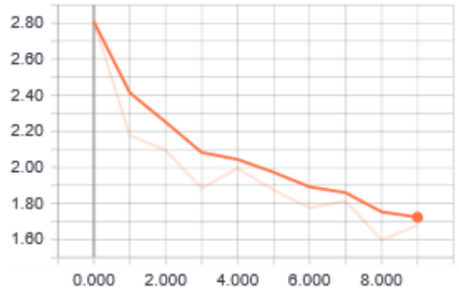


Figur E.1: Träningsförlust

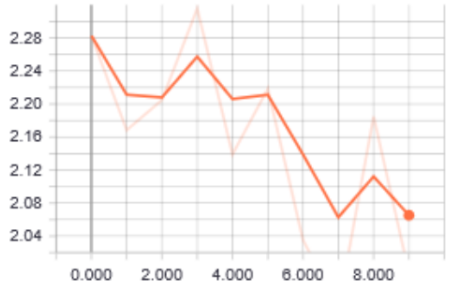


Figur E.2: Valideringsförlust

Experiment 2.1

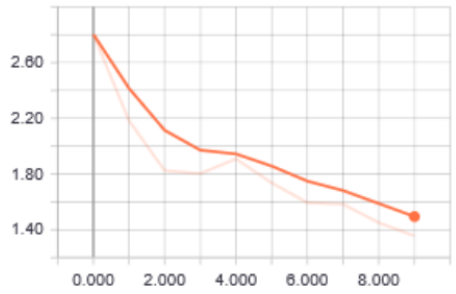


Figur E.3: Träningsförlust

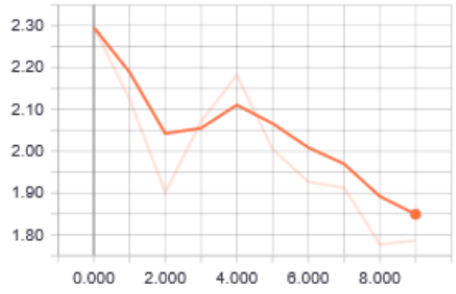


Figur E.4: Valideringsförlust

Experiment 2.2

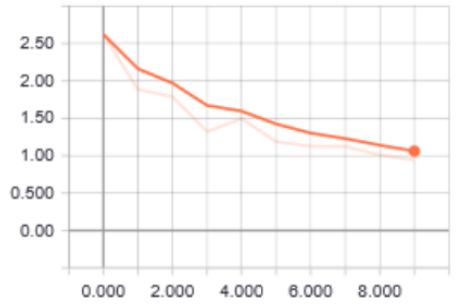


Figur E.5: Träningsförlust

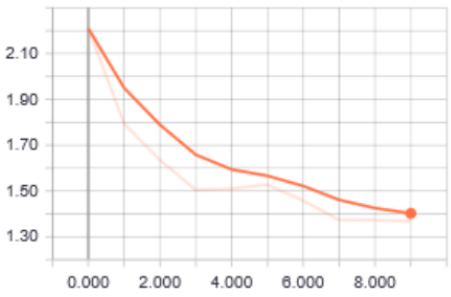


Figur E.6: Valideringsförlust

Experiment 2.3

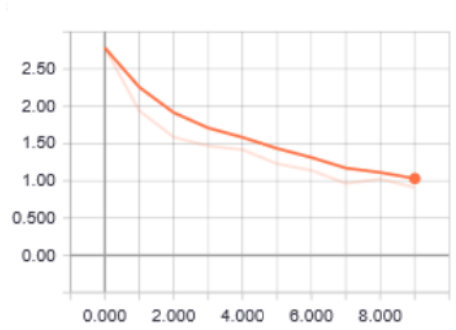


Figur E.7: Träningsförlust

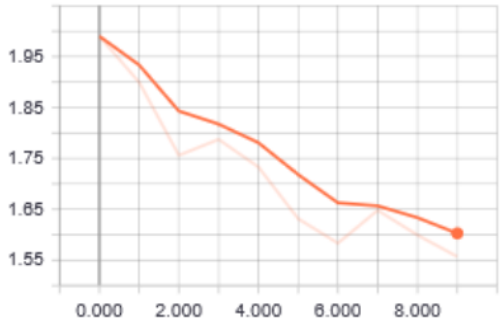


Figur E.8: Valideringsförlust

Experiment 3



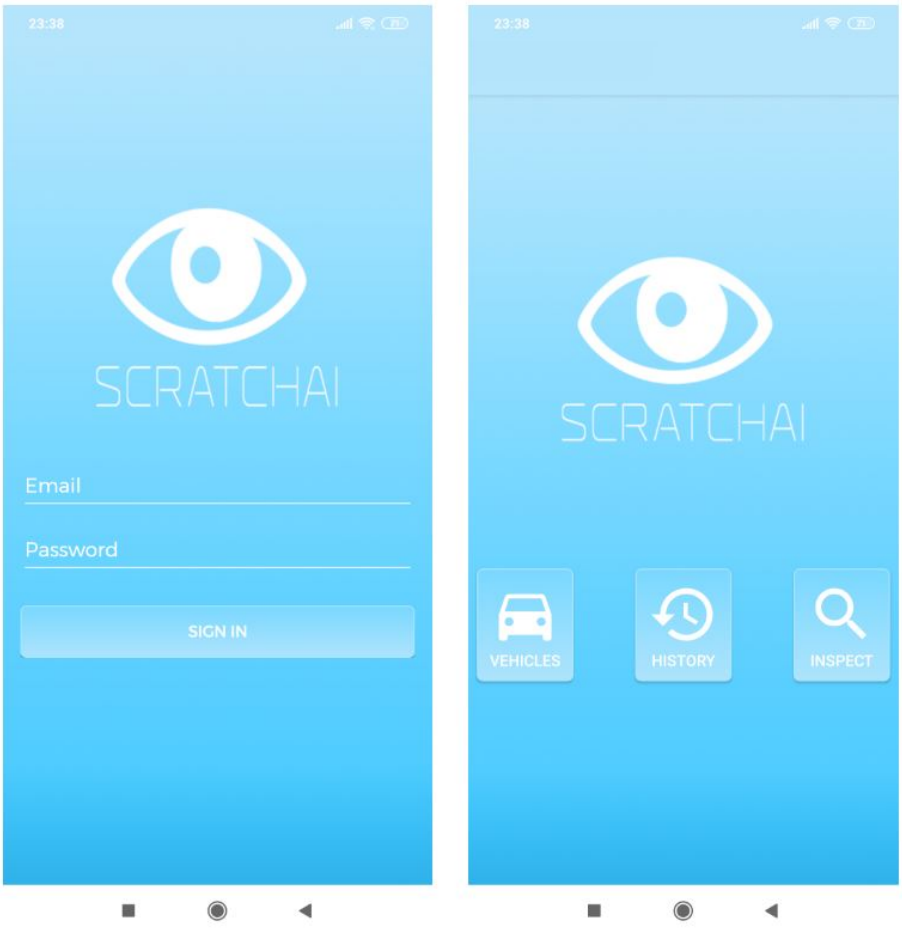
Figur E.9: Träningsförlust

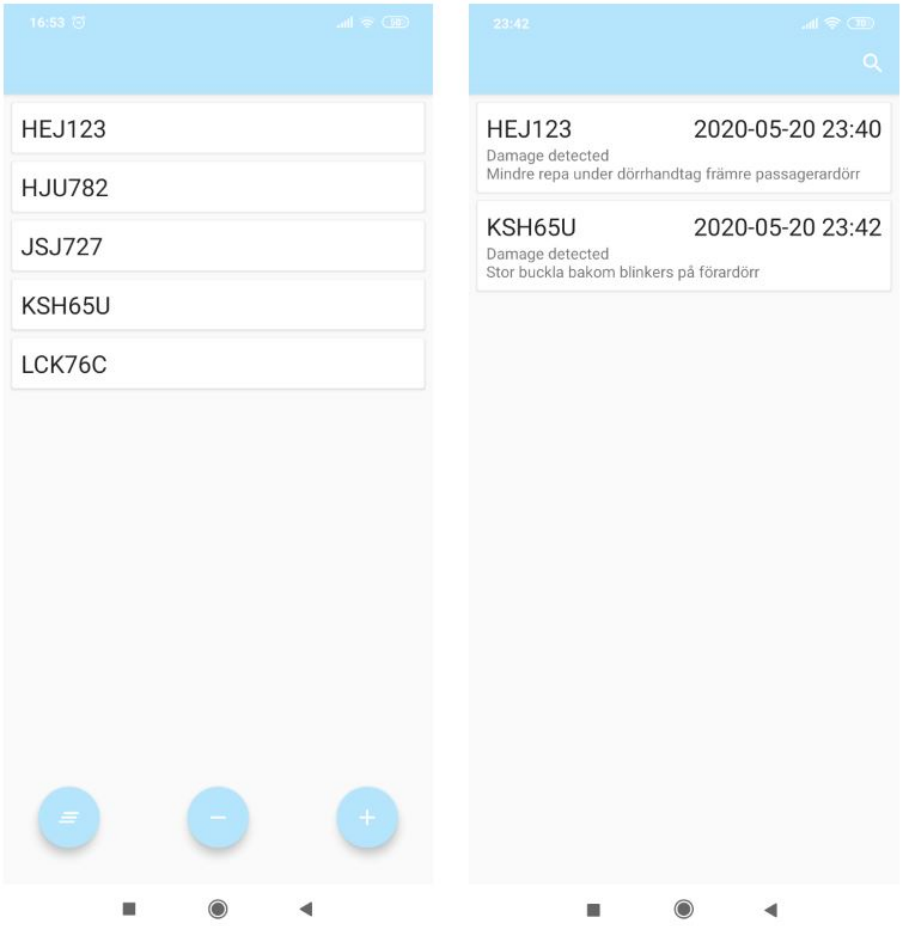


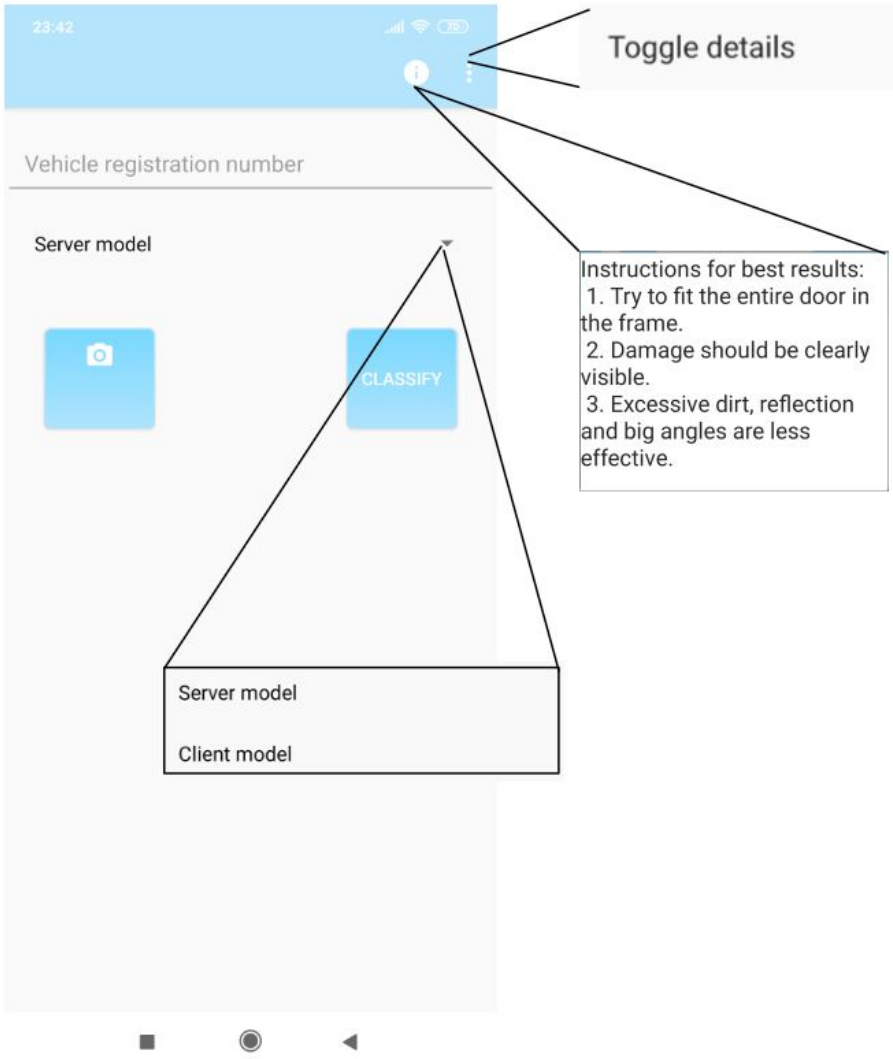
Figur E.10: Valideringsförlust

Bilaga F

Användargränssnitt

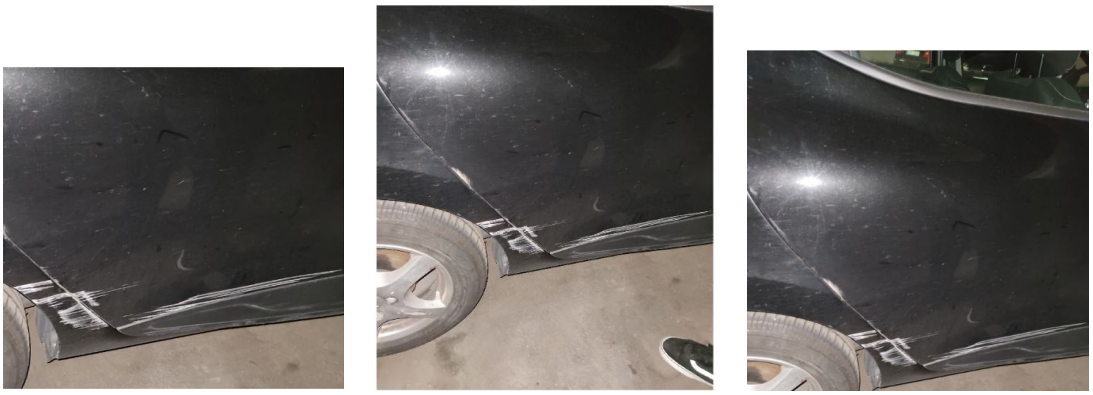



16:53
HEJ123
HJU782
JSJ727
KSH65U
LCK76C
23:42
HEJ123
2020-05-20 23:40
Damage detected
Mindre repa under dörrhandtag främre passagerardörr
KSH65U
2020-05-20 23:42
Damage detected
Stor buckla bakom blinkers på förardörr


23:42
Toggle details
Vehicle registration number
Server model
CLASSIFY
Instructions for best results:
1. Try to fit the entire door in the frame.
2. Damage should be clearly visible.
3. Excessive dirt, reflection and big angles are less effective.
Server model
Client model

Bilaga G

Gränsfall för skada



(a) 0.99, 0.01 **(b) 0.945, 0.055** **(c) 0.999, 0.001**

(d) 0.571, 0.429 **(e) 0.062, 0.938**

Figur G.1: (Konfidens skada, konfidens intakt)

Litteraturförteckning

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[2] S. Martin, "What is transfer learning?" , 2019. [Online]. a https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/, hämtad: 2020-05-20.

[3] S. Dabeer, M. M. Khan, and S. Islam, "Cancer diagnosis in histopathological image: Cnn based approach," *Informatics in Medicine Unlocked*, vol. 16, p. 100231, 2019. [Online]. Tillgänglig: http://www.sciencedirect.com/science/article/pii/S2352914819301133

[4] Q. Zhang, X. Chang, and S. B. Bian, "Vehicle-damage-detection segmentation algorithm based on improved mask rcnn," *IEEE Access*, vol. 8, pp. 6997–7004, 2020.

[5] M. Dake, "A simplified view of an artificial neural network," Tillgänglig: https://sv.m.wikipedia.org/wiki/Fil:Neural_network.svg hämtad: 2020-06-13.

[6] G. Lindsay, "Convolutional neural networks as a model of the visual system: Past, present, and future," *Journal of Cognitive Neuroscience*, p. 1–15, Feb 2020. [Online]. Tillgänglig: http://dx.doi.org/10.1162/jocn_a_01544

[7] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang, "Recent advances in convolutional neural networks," *CoRR*, vol. abs/1512.07108, 2015. [Online]. Tillgänglig: http://arxiv.org/abs/1512.07108

[8] Aphex34, "Typical cnn," Tillgänglig: https://en.wikipedia.org/wiki/File:Typical_cnn.png hämtad: 2020-06-04.

[9] B. Mehlig, "Artificial neural networks," *CoRR*, vol. abs/1901.05639, 2019. [Online]. Tillgänglig: http://arxiv.org/abs/1901.05639

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[12] Y. Srivastava, V. Murali, and S. R. Dubey, "A performance comparison of loss functions for deep face recognition," *CoRR*, vol. abs/1901.05903, 2019. [Online]. Tillgänglig: http://arxiv.org/abs/1901.05903

[13] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Tillgänglig: http://arxiv.org/abs/1609.04747

[14] S. Salman and X. Liu, "Overfitting mechanism and avoidance in deep neural networks," *CoRR*, vol. abs/1901.06566, 2019. [Online]. Tillgänglig: http://arxiv.org/abs/1901.06566

[15] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Tillgänglig: http://arxiv.org/abs/1506.01497

[16] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Tillgänglig: http://arxiv.org/abs/1504.08083

[17] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Tillgänglig: http://arxiv.org/abs/1703.06870

[18] Oracle, "Java language and virtual machine specifications," , 2020. [Online]. Tillgänglig: https://docs.oracle.com/javase/specs/index.html, hämtad: 2020-05-20.

[19] JetBrains, "Intellij idea," , 2020. [Online]. Tillgänglig: https://www.jetbrains.com/idea/, hämtad: 2020-05-20.

[20] Google, "Android studio," , 2020. [Online]. Tillgänglig: https://developer.android.com/studio/index.html, hämtad: 2020-05-20.

[21] ——, "Save data in a local database using room," , 2020. [Online]. Tillgänglig: https://developer.android.com/training/data-storage/room, hämtad: 2020-05-20.

[22] ——, "Deploy machine learning models on mobile and iot devices," , 2020. [Online]. Tillgänglig: https://www.tensorflow.org/lite, hämtad: 2020-05-20.

[23] P. S. Foundation, "Python is a programming language that lets you work quickly and integrate systems more effectively," , 2020. [Online]. Tillgänglig: https://www.python.org/, hämtad: 2020-05-20.

[24] Google, "An end-to-end open source machine learning platform," , 2020. [Online]. Tillgänglig: https://www.tensorflow.org, hämtad: 2020-05-23.

[25] Keras, "Keras," , 2020. [Online]. Tillgänglig: https://keras.io, hämtad: 2020-05-23.

[26] Nvidia, "Train models faster," , 2020. [Online]. Tillgänglig: https://developer.nvidia.com/cuda-zone, hämtad: 2020-05-23.

[27] ——, "Nvidia cudnn," , 2020. [Online]. Tillgänglig: https://developer.nvidia.com/cudnn, hämtad: 2020-05-23.

[28] Anaconda, "Your data science toolkit," , 2020. [Online]. Tillgänglig: https://www.anaconda.com/products/individual, hämtad: 2020-05-23.

[29] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," Tillgänglig: https://github.com/matterport/Mask_RCNN, hämtad: 2020-05-23, 2017.

[30] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Tillgänglig: http://arxiv.org/abs/1704.04861

[31] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions," *CoRR*, vol. abs/1808.05517, 2018. [Online]. Tillgänglig: http://arxiv.org/abs/1808.05517

[32] Keras, "Keras applications," , 2020. [Online]. Tillgänglig: https://keras.io/api/applications/, hämtad: 2020-05-20.

[33] P. with Code, "Instance segmentation," , 2020. [Online]. Tillgänglig: https://paperswithcode.com/task/instance-segmentation, hämtad: 2020-05-23.

[34] U. of Oxford, "Vgg image annotator," , 2020. [Online]. Tillgänglig: http://www.robots.ox.ac.uk/ vgg/software/via/via-1.0.6.html, hämtad: 2020-05-23.

[35] Keras, "Callbacks api," , 2020. [Online]. Tillgänglig: https://keras.io/api/callbacks/, hämtad: 2020-05-21.

[36] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Tillgänglig: http://arxiv.org/abs/1405.0312

[37] Google, "Activity," , 2020. [Online]. Tillgänglig: https://developer.android.com/reference/android/app/Activity, hämtad: 2020-05-20.

[38] ——, "Guide to app architecture," , 2020. [Online]. Tillgänglig: https://developer.android.com/jetpack/docs/guide, hämtad: 2020-05-21.

[39] ——, "Livedata overview," , 2020. [Online]. Tillgänglig: https://developer.android.com/topic/libraries/architecture/livedata, hämtad: 2020-05-21.

[40] M. Spitsin, "Singletons in android," , 2016. [Online]. Tillgänglig: https://medium.com/@programmerr47/singletons-in-android-63ddf972a7e7, hämtad: 2020-05-21.

[41] Google, "Asynctask," , 2016. [Online]. Tillgänglig: https://developer.android.com/reference/android/os/AsyncTask, hämtad: 2020-05-21.

[42] Nvidia, "Cuda toolkit 9.0 downloads," , 2020. [Online]. Tillgänglig: https://developer.nvidia.com/cuda-90-download-archive, hämtad: 2020-05-23.

[43] ——, “Cuda installation guide for microsoft windows,” , 2020. [Online]. Tillgänglig: https://docs.nvidia.com/cuda/archive/9.0/cuda-installation-guide-microsoft-windows/index.html, hämtad: 2020-05-23.

[44] ——, “cudnn archive,” , 2020. [Online]. Tillgänglig: https://developer.nvidia.com/rdp/cudnn-archive, hämtad: 2020-05-23.