YRKESHÖGSKOLAN
NOVIA

Defektdetektering av komponenter med hjälp av artificiell intelligens

David Ena

Examensarbete för automationsingenjör (YH)-examen

Utbildningsprogrammet för el- och automationsteknik

Vasa 2023

EXAMENSARBETE

Författare: David Ena
Utbildning och ort: El- och automationsteknik, Vasa
Inriktning: Automationsteknik
Handledare: Ray Pörn

Titel: Defektdetektering av komponenter med hjälp av artificiell intelligens

---

Datum: 30.5.2023 Sidantal: 47

Abstrakt

Syftet med examensarbetet var att testa defektdetektering på en typ av reflektor med hjälp av fritt tillgängliga AI-program. Olika typer av program och metoder skulle testas för att få en uppfattning om hur bra de presterar och vilka typer av defekter som kan detekteras.

De olika metoder som testades var bildklassificering, objektdetektering och segmentering. AI-modeller med dessa metoder tränas i fyra olika program: LandingLens, Sentisight AI och Clarifai som alla är webbaserade AI-plattformar, och Fastai som är ett programmeringsbibliotek för Python som man kan använda för att bygga upp egna modeller.

Av de metoder och program som testades så presterar bildklassificering i LandingLens bäst. Med rätt konfidensgräns så kan man nå 100 % noggrannhet på testbilderna. Segmentering fungerar inte alls medan objektdetektering och bildklassificering i de andra programmen ger varierande resultat på 80–95 % noggrannhet. Alla defekter kan detekteras med åtminstone någon av modellerna, men många modeller har problem med mörkare och ljusare bilder. Objektdetekteringen har också problem med stora defekter som vissa typer av fettfläckar.

Språk: svenska
Nyckelord: AI, maskininlärning, djupinlärning, defektdetektering

BACHELOR'S THESIS

Author: David Ena
Degree Programme: Electricity and Automation
Specialisation: Automation
Supervisor(s): Ray Pörn

Title: Defect Detection on Components with Artificial Intelligence

---

Date: 30.5.2023 Number of pages: 47

Abstract

The purpose of this thesis was to test defect detection on a specific type of reflector with freely available AI programs. Different types of programs and methods would be tested to get an idea of how well they perform and what types of defects can be detected.

The different methods used were image classification, object detection and segmentation. AI models using these methods are trained in four different programs: LandingLens, Sentisight AI, Clarifai and Fastai. LandingLens, Sentisight AI and Clarifai are all web-based AI platforms, while Fastai is a library for Python that can be used to build your own models.

Among the methods and programs tested, image classification in LandingLens performed the best. With the right confidence threshold, 100 % accuracy could be reached on the test set. Segmentation didn't work at all, while object detection and image classification in the other programs gave varying results between 80–95 % accuracy. Every type of defect could be detected with at least one of the models, but many models had problems with darker and brighter images. Object detection also had problems with larger defects, such as certain types of grease stains.

Language: Swedish
Key words: AI, machine learning, deep learning, defect detection

Innehållsförteckning

1 Inledning

Det här examensarbetet går ut på att testa möjligheter att utgående från bilder detektera defekter på reflektorer med hjälp av olika AI och maskininlärningsprogram. Arbetet gjordes åt Nordic Lights i Jakobstad.

1.1 Uppdragsgivare

Nordic Lights startades 1992 och är nu ett av världens största tillverkare av arbetsbelysning för tunga fordon. Nordic Lights tillverkar arbetsbelysning, strålkastare och markeringsbelysning för de mest kända tillverkare av byggmaskiner, bland annat Caterpillar, Hitachi, John Deere och Komatsu. Belysningarna används i många industrier, bland annat gruvdrift, konstruktion, materialhantering, skogsbruk och jordbruk. Nordic Lights komponenter köps in av underleverantörer runt om i världen och monteras ihop till en komplett produkt i Jakobstad. (Nordic Lights, u.å.).

1.2 Syfte

Monteringen av produkter vid Nordic Lights har historiskt sett varit stor del manuellt arbete, men på senare år har det blivit betydligt mer automatisering i produktionen. När produkterna monteras manuellt granskas komponenterna av människor innan montering och således kan felaktiga komponenter enkelt sorteras bort. Men när mer blir automatiserat upptäcks defekter först när produkten är färdigt monterad, vilket leder till att hela produkten måste slängas. Ett av de större problemen är reflektorer som har visuella skador på sig och syftet med detta arbete är att automatiskt upptäcka dess fel före montering.

1.3 Mål

Arbetet bestod av att först skapa en bildbank för reflektorerna och sedan testa att göra modeller med olika fritt tillgängliga AI-program. AI-programmen fick vara helt gratis program eller betalda program som har demoversioner man kan testa gratis. Modeller skulle göras med några olika program och tekniker för att kunna jämföra vad som fungerar bättre och sämre. Modellerna testades till sist på ett antal reflektorer för att få en överblick

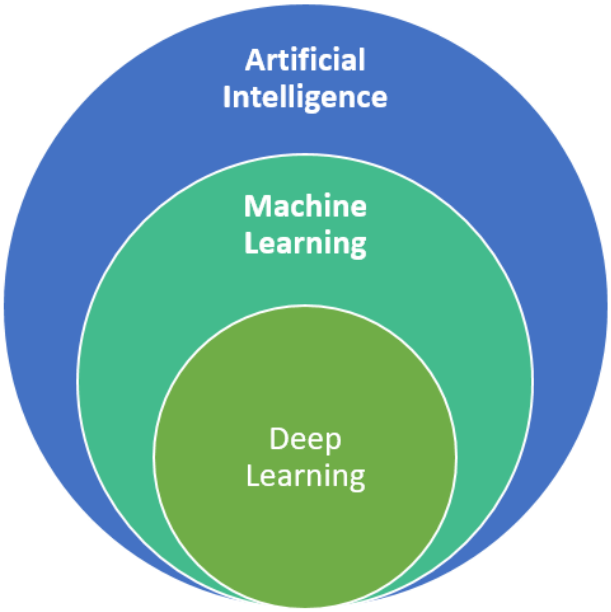
av hur stor del av defekterna som kan upptäckas och vilka defekter som inte kan upptäckas. Resultatet av arbetet kommer sedan användas som jämförelse och stöd för kommande investeringar för Nordic Lights.

2 Teori

AI har blivit mycket vanligare senaste åren, till stor del tack vare tillgång till GPU:s som kan göra parallellbehandlingar snabbare, billigare och kraftfullare. På samma gång finns det väldigt stor lagringskapacitet och enorma mängder av data. Det här har gjort att saker som drömdes om när AI först dök upp på 1950-talet börjar snart kunna bli verklighet. I det här kapitlet förklaras teorin bakom AI och maskininlärning, hur bildhantering med AI fungerar, olika typer av maskininlärning och hur man tränar en maskininlärningsmodell.

2.1 AI, maskininlärning och djupinlärning

Det kan vara värt att först definiera skillnader på AI, maskininlärning (ML) och djupinlärning (DL). De används ofta på samma sätt, men de är inte exakt samma sak. I stora drag är DL en typ av ML och ML en typ AI, de blir mera specifika om man går djupare (figur 1).



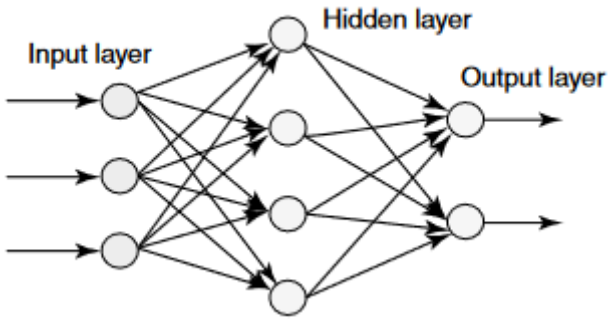
Figur 1. AI, maskininlärning och djupinlärning (Berchane & Berchane, 2018).

AI syftar i grunden på en maskin som kan tänka som en människa. Det är dock svårt att avgöra vad som egentligen menas med det, AI är ett mycket brett område. I det här arbetet

användes i huvudsak maskininlärning och djupinlärning, men i AI ingår också bland annat robotik, expertsystem och naturlig språkbehandling. Maskininlärning är en typ av AI som använder olika algoritmer för att analysera data, lära från det och sedan förutspå eller bestämma en uppsättning händelser. I stället för att skriva kod för en specifik uppsättning av instruktioner tränar man den med data och algoritmer för att lära den att utföra en specifik uppgift. Djupinlärning är en typ av ML som försöker imitera hur människohjärnan fungerar, det använder sig av neurala nätverk för att lära sig. (Berchane & Berchane, 2018).

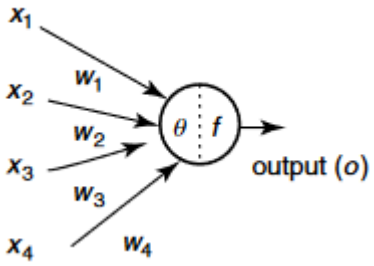
2.2 Neurala nätverk

Neurala nätverk är baserade på den biologiska strukturen av vår hjärna och sammankopplingar mellan neuronerna. Till skillnad från hjärnan där alla neuroner kan kopplas till alla andra neuroner är neurala nätverk kopplade ihop med specifika riktningar för spridning av data. Ett neuralt nätverk består av ett invärde lager, ett utvärden lager och ett eller flera gömda lager (figur 2). Varje lager har ett antal neuroner med olika funktioner som generar en utvärden (Abraham, 2005).



Figur 2. Artificiellt neuralt nätverk med ett dolt lager (Abraham, 2005).

En neuron har flera invärden med vikter associerade till dem, neuronen tar den viktade summan av alla invärden och tar det genom en funktion för att generera ett utvärde (figur 3). Själva inlärningen i ett neuralt nätverk kommer från att vikterna justeras av inlärningsalgoritmen (Abraham, 2005).



Figur 3. Artificiell neuron (Abraham, 2005).

2.3 Algoritmer

Normalt tar en algoritm invärden och använder logik och matematik för att få utvärden, ett resultat. Det speciella med en AI-algoritm är att den tar både invärden och utvärden samtidigt och lär sig från datat. Efter den blivit tränad kan den producera utvärden när den får nya invärden. (Lowe & Lawless, 2021).

De fem vanligaste uppgifterna en algoritm kan utföra är:

- Regression –används normalt för prognoser och att hitta samband mellan invärde och utvärde på variabler.
- Klassificering – när invärde datat kan bli separerade i olika grupper eller kategorier, till exempel man eller kvinna.
- Kluster – gruppering av data som är liknande, till exempel gruppering av bilder på olika djur beroende på dataattribut.
- Associering – när kännetecken av datat kan bli associerat med andra kännetecken för att länka ihop data, till exempel kunder som köper bröd och smör köper också oftast mjölk.
- Kontroll – när felmåttet mellan börvärdet och ett uppmätt värde hjälper bestämma vad som ska göras, till exempel en självstyrande fordonskontroller kan tillämpa kontrollerad bromsning eller manöver för att undvika kollision. (Lowe & Lawless, 2021).

2.4 Inlärningstyper

Det finns fyra olika inlärningstyper för AI: övervakad, semiövervakad, oövervakad och förstärkningsinlärning. Vilken av de här som är bäst att använda beror på ändamålet och vilken typ av data man har.

2.4.1 Övervakad inlärning

Systemet lär sig från märkt data med både invärden och utvärden gett som träningsdata. Det lär sig att förutspå framtida utvärden baserat på tidigare data. En vanlig tillämpning är att klassificera bilder, där träningsdata är märkta bilder. Vanliga algoritmer som används är regression och klassificering. Nackdelen med övervakad inlärning är att allt data måste vara märkt vilket kan ta väldigt mycket tid om man måste ha tusentals eller till och med miljontals med data. (Sarker, 2021).

2.4.2 Oövervakad inlärning

Systemet lär sig från endast invärden, datat är inte märkt och inga utvärden ges. Datat organiseras efter liknande mönster, det här gör att anomalier blir mer uppenbara. Ett exempel är att gruppera kunder baserat på köphistorik och bläddringshistorik. Vanligast algoritmer som används är kluster, associering och anomalidetektering. (Sarker, 2021).

2.4.3 Semiövervakad inlärning

Använder sig av både märkt och omärkt data, det fungerar som en mix av övervakad och oövervakad inlärning. I många fall finns det för lite märkt data, men kan finnas mycket omärkt data, i sådana fall kan med semiövervakad inlärning få ett bättre resultat än om man bara använde sig märkt data. Vanliga användningsområden är maskinöversättning, spårning av bedrägeri och textklassificering. (Sarker, 2021).

2.4.4 Förstärkningsinlärning

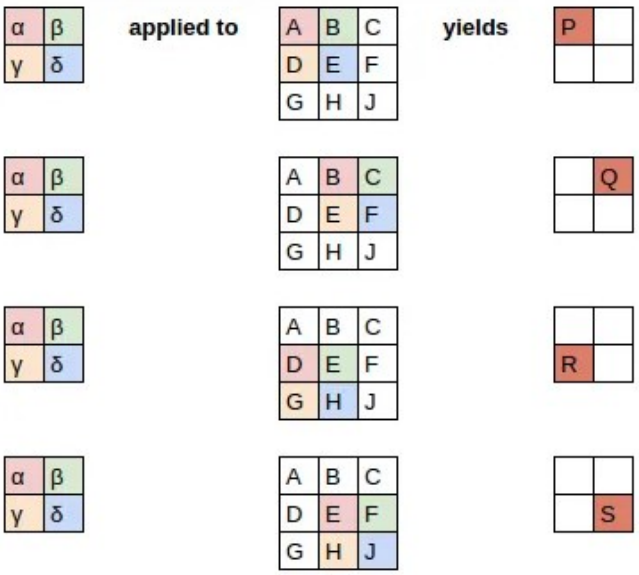
Systemet får inget träningsdata och bara lär sig genom att interagera med miljön. Systemet får positiv eller negativ belöning för sin senaste handling och från det försöker maximera sin totala belöning. Förstärkningsinlärning är en variant av djupinlärning och använder sig av neurala nätverk. En vanlig tillämpning är spel. (Sarker, 2021).

2.5 Konvolutionerande neurala nätverk

Konvolutionerande neurala nätverk eller CNN är en typ av neuralt nätverk som oftast används för att analysera och klassificera bilder. Till skillnad från vanliga neurala nätverk ersätter de en eller flera av de gömda lagrens matrismultiplikationer med konvolutioner (Goodfellow, Bengio, & Courville, 2016). Målet med CNN är att tilldela vikter och bias till olika aspekt och objekt i bilden, och skilja dem från varandra. Med träning lär sig ett CNN att tillverka filter för att hitta dessa aspekter (Saha, 2018).

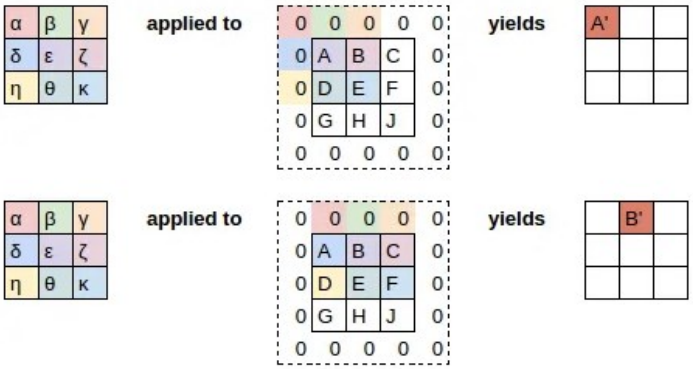
2.5.1 Konvolution

I ett CNN görs konvolutionen med att ta summan av en elementvis multiplikation av två matriser och adderar en bias vid behov. Den ena matrisen är en del av en bild och den andra är ett filter. Filtret tillämpas på alla delar av bilden så man får en ny matris (Figur 4).



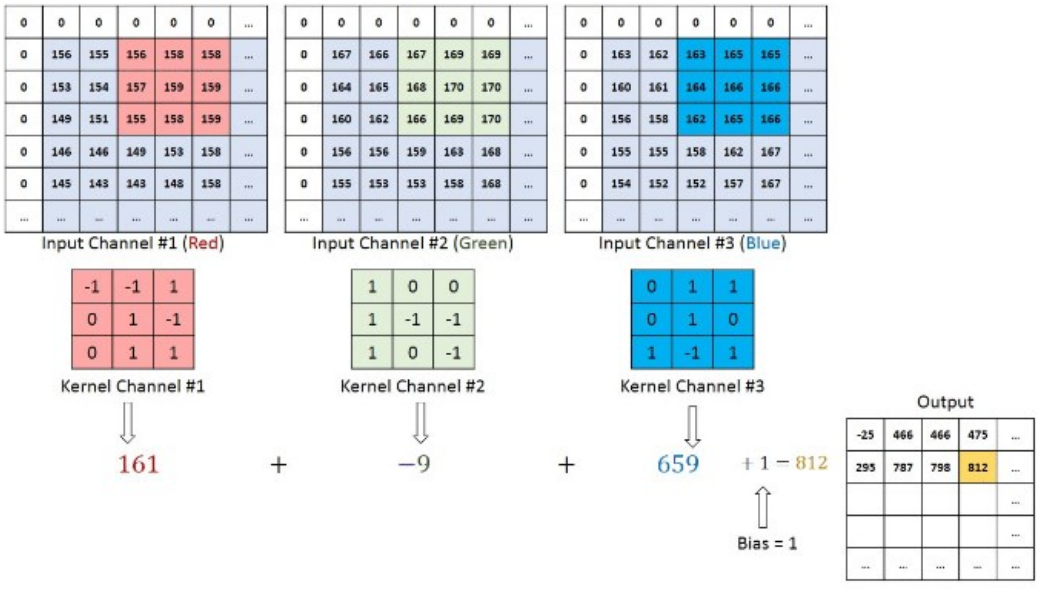
Figur 4. Exempel på konvolution (Kleinsmith, 2017).

Som man kan se gör det här gör dock att bilden blir mindre för varje gång man kör filtret över den. För att motverka det här kan man fylla i kanterna med nollor så att resultat matrisen får samma storlek (Figur 5).



Figur 5. Konvolution fyllning (Kleinsmith, 2017).

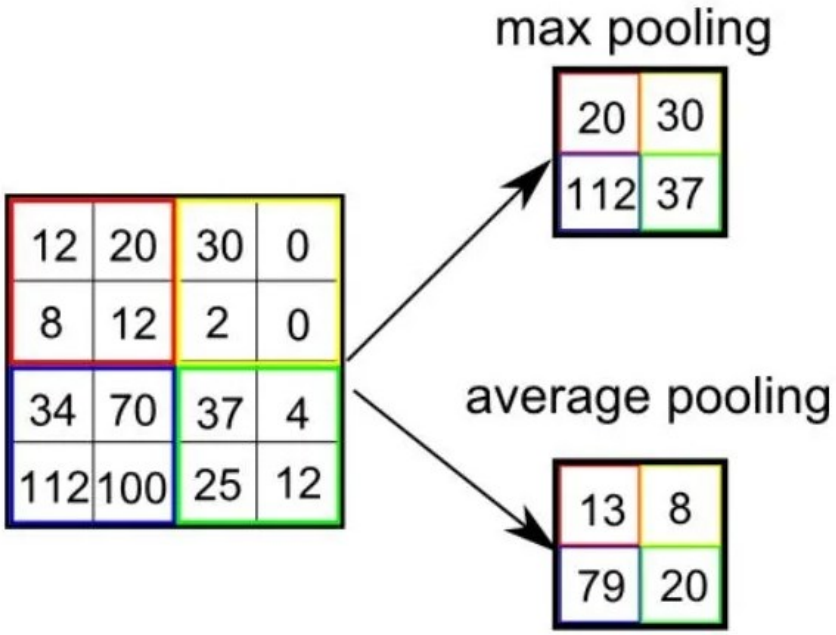
En bild har dock oftast mer en två dimensioner, den har både bredd och höjd, men också ett djup eftersom den har flera färger och till exempel i RGB är färgerna indelade i röd, grönt och blått, vilket gör att den har ett djup på tre lager. I det här fallet får också filtret en tredje dimension och den elementvisa matrismultiplikation görs för varje lager och sedan adderas ihop så vi får ett tvådimensionellt utvärde (Figur 6). På det här sättet får man sänkt ner bilderna till en form som är enklare att bearbeta utan att förlora egenskaper som kan vara kritiska för att göra bra förutsägelser. (Saha, 2018).



Figur 6. Konvolution på en MxNx3 bildmatris med ett 3x3x3 filter (Saha, 2018).

2.5.2 Pooling

Efter konvolutionen görs en pooling. Målet med pooling är att sänka storleken på matrisen för att göra datat lättare att bearbeta. Det är också användbart för att ta fram dominerande egenskaper från matrisen. Det finns två typer av pooling: Max-pooling och medelvärdes-pooling. Som hörs från namnet med max-pooling tar man det maximala värden från en del av matrisen och med medelvärdes-pooling tar man medelvärdet av all värden i en del av matrisen (Figur 7). (Saha, 2018).



Figur 7. Exempel på pooling (Saha, 2018).

I allmänhet är max-pooling bättre än medelvärdes-pooling för att upptäcka specifika aspekt i en bild. På grund av att max-pooling tar det maximala värdet fungerar det på samma gång som ett filter för störningar. Det gör också att även om det aspekt man försöker upptäcka i en bild är väldigt litet i proportion till hela bilden kommer det ändå fram, medan med

medelvärdes-pooling skulle det behöva ta upp majoriteten av bilden för att upptäckas. (Sylvain & Howard, 2020).

2.6 Bildigenkänningstekniker

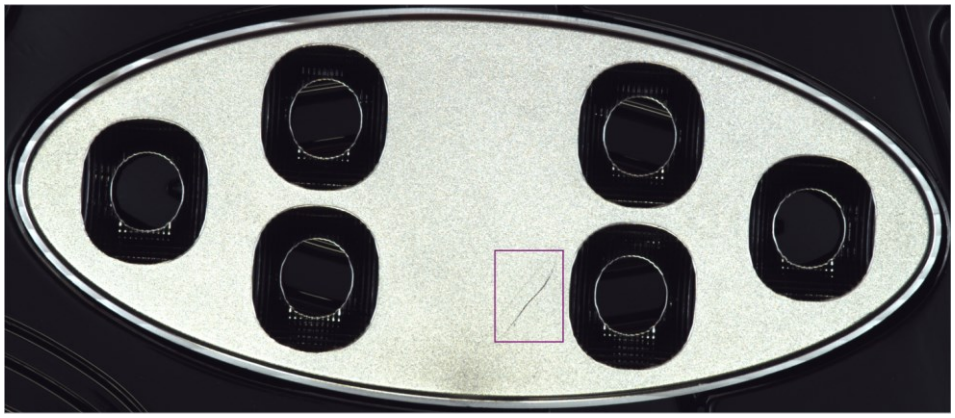
Det finns flera olika typer av tekniker för att känna igen och märka bilder. Bland dessa är de tre vanligaste typerna bildklassificering, objektdetektering och segmentering.

2.6.1 Bildklassificering

Med bildklassificering innebär att man delar in bilder i två eller flera kategorier och en AI lär sig att placera in bilder i dessa kategorier (Davies, 2012). En bildklassificeringsmodell kan vara endera enkel-label där varje bild tillhör endast en kategori eller multi-label där varje bild kan höra till flera kategorier. En bildklassificeringsmodell kommer alltid att placera in bilder i åtminstone en kategori vilket gör att om man tränar en modell att se skillnaden på hundar och katter, och man ger den en bild på en fågel kommer den fortfarande märka den som hund eller katt.

2.6.2 Objektdetektering

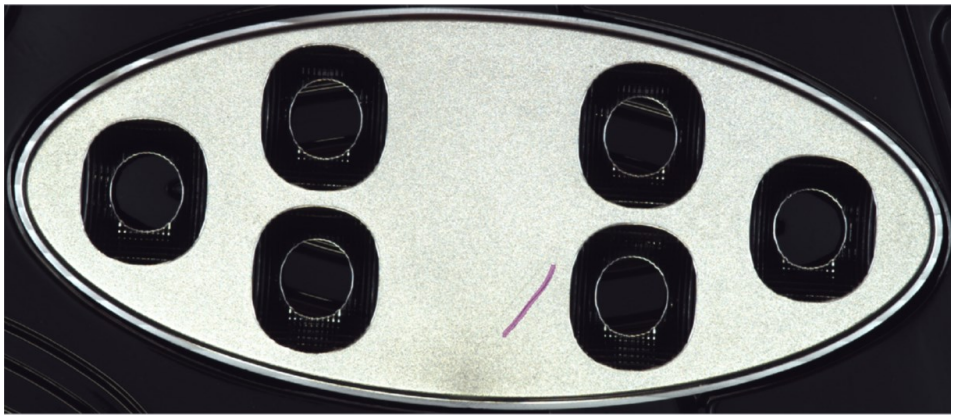
Till skillnad från bildklassificering som endast berättar vad en bild är, märker objektdetektering ut var på bilden en eller flera objekt är (Davies, 2012). Objekten märks oftast ut med rutor som kan ses i figur 8.



Figur 8. Repa markerad på reflektor med objektdetektering.

2.6.3 Segmentering

Segmentering fungerar på liknande sätt som objektdetektering, men är mycket noggrannare. I stället för att bara märka ut var objektet är på bilden, märks exakta kanter ut för att få en noggrann bild av formen på objektet som kan ses i figur 9. (Davies, 2012).



Figur 9. Repa markerad på reflektor med segmentering.

2.7 Träning av en ML-modell

När man ska träna en ML-modell finns det en hel del saker man måste tänka på, även med de simplaste AI-program går inte bara att slänga in en klunga med bilder och tro att det blir bra. Hur man delar upp data, hur man ändrar på parametrar och hur man hanterar data har stor betydelse i hur ens modell presterar.

2.7.1 Träning, validering och testning

När man ska träna en modell kan man inte bara sätta allt sitt data in i modellen för att tränas på och sedan testa modellen på samma data för då skulle man inte veta hur bra den presterar på data den inte har sett. Före man börjar med att träna sin modell måste man dela upp sitt data i tre olika grupper. Träningsdata är det data som modellen kommer träna sig på, valideringsdata är det som används för att testa modellens felprocent under träning och testdata är för att till sist testa modellen på helt okänt data. Det data man har kan delas in i olika proportioner beroende på hur mycket data man har, men i normala fall delas det i 60 % träningsdata, 20 % valideringsdata och 20 % testdata. (Sylvain & Howard, 2020).

2.7.2 Överföringsinlärning

Att träna en modell från grunden kan ta lång tid och kräva mycket data, för att göra det här smidigare har man börjat använda sig av överföringslärande. Med överföringslärande menas att man använder sig av en redan tränad modell och justerar den för det ändamål man strävar efter. I stället för att träna hela nätverket tränar man bara de några sista lagren så modellen blir anpassad för ens egna ändamål. Det här snabbar upp processen rejält och gör att man kan träna modeller med ibland bara några tiotals bilder. (Sylvain & Howard, 2020).

2.7.3 Hyperparameter i ett neuralt nätverk

Inlärningshastighet, viktminskning, epochs och batch-storlek är de vanligaste parametrarna man justerar för att få en bättre modell. En epoch är när man har skickat allt sitt träningsdata genom algoritmen en gång. Det betyder att när man ställer in antal epochs ändrar man hur många gånger modellen ska tränas på datat. Ju mer epochs man har desto mera lär sig modellen, men för varje epoch tar träningen längre och risken för överanpassning ökar. För att hitta ett optimalt antal epochs kan man börja med att sätta det så högt man är villig att vänta på att modellen tränas och om modellen då blir överanpassad kan man sänka antalet epochs till strax innan överanpassningen började. (Sylvain & Howard, 2020).

En epoch består av flera batchar. När en modell tränas körs allt data den tränas på in i minnet på GPU:n (eller CPU:n), vilket i många fall kan vara för mycket eller för stort data för att GPU:n ska kunna hantera allt på en gång. Därför delar man ofta in datat i flera batchar, och GPU:n behöver då bara hantera en batch åt gången. En batch-storlek indikerar då hur många bilder man ska ha per batch. Batch-storlek ställs normalt in med multipler av två. (Sylvain & Howard, 2020).

Inlärningshastighet är ett mått på hur snabbt en modell lär sig. Med för låg inlärningshastighet kan modellen ta allt för länge att lära sig och det finns risk för överanpassning. Men däremot om man har för hög inlärningshastighet kommer modellen tränas för snabbt varje epoch och den överskjuter det minimala felet och modellen blir bara sämre. För att hitta en optimal inlärningshastighet kan man använda sig av inlärningshastighetssökare. (Sylvain & Howard, 2020).

Viktminskning är ett sätt att motverka att vikter blir för stora och modellen börjar överanpassas. Med en högre viktminskning anpassas inte modellen lika hårt till träningsdatat men ifall den är för hög kommer aldrig modellen att bli så bra den skulle kunna vara. Man anger oftast viktminskning i multipler av tio, där 0.1 eller 0.01 oftast används. (Sylvain & Howard, 2020).

2.7.4 Olika sätt att avgöra modellens prestanda

När man tränar en modell finns det ett antal olika sätt man kan mäta dess prestanda på. Under träningen ser man på att träningsfel och valideringsfel blir så lågt som möjligt. Träningsfel indikerar hur bra modellen presterar på träningsdatat och valideringsfel indikerar hur bra den presterar på valideringsdatat. Det finns många olika typer av loss funktioner för att räkna ut dessa, men den vanligaste inom bildigenkänning är korsentropins felfunktion (Koppert-Anisimova, 2021):

$$D(S, L) = -\sum_{i} L_i * \log_2 S_i \quad (1)$$

S är de förutsagda sannolikheterna som adderar upp till ett och L är de riktiga värdena. Till exempel om S har värdena (0.7, 0.2, 0.1) och L är (1, 0, 0) blir korsentropifelet 0,515. Ju lägre loss desto bättre modell.

Efter träningen är klar mäter man oftast noggrannhet, precision, återkallelse, F1 eller någon kombination av dessa för att bestämma modellen prestanda. Vilka mått som används varierar på olika plattformar och program. Noggrannhet är antal korrekta identifikationer över antal totala identifikationer:

$$Noggrannhet = \frac{Korrekta\ identifikationer}{Totala\ identifikationer} \quad (2)$$

Precision är antal korreka identifikationer av en specifik kategori över det totala antal identifierade av den kategorien:

$$Precision = \frac{Antal\ korrekt\ identifierat\ som\ A}{Totala\ identifierat\ som\ A} \quad (3)$$

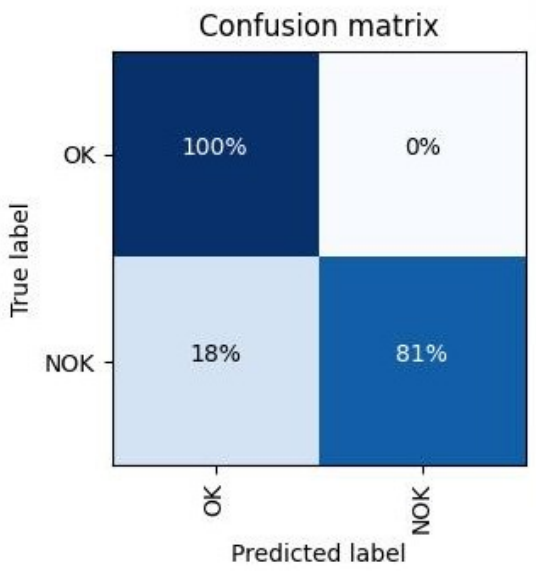
Återkallelse är antal korrekta identifikationer av en specifik kategori över antalet totala objekt av kategorin:

$$Återkallelse = \frac{Antal\ korrekt\ identifierat\ som\ A}{Totalt\ antal\ i\ kategori\ A} \quad (4)$$

F1 är ett harmoniskt medelvärde av precision och återkallelse:

$$F_1 = 2 * \frac{Precision * Återkallelse}{Precision + Återkallelse} \quad (5)$$

Ett annat sätt att få en bättre överblick över vad modellen har problem att identifiera när man använder bildklassificering är med en sammanblandningsmatris. Sammanblandningsmatrisen visar vilka kategorier som identifierades som vad (Sylvain & Howard, 2020). I figur 10 kan vi se att 100 % av bilderna i kategori OK blev rätt identifierade, men 18 % av bilder i NOK blev felidentifierade som NOK. Från sammanblandningsmatrisen får man fram vilka kategorier modellen har problem att se skillnaden på och man kan justera träningen efter det.



Figur 10. Sammanblandningsmatris.

2.7.5 Dataaugmentering

Med dataaugmentering menas att man skapar slumpmässiga variationer av invärden, så att de ser annorlunda ut men inte ändrar på meningen med datat. Det här kan vara viktigt för att få ens modell att känna igen flera typer av bilder, speciellt om man har väldigt lite data till att börja med. Vanliga dataaugmenteringstyper är rotation, vändning, beskärning,

perspektivförvrängning, ljusstyrka förändring och kontrastförändring. De valda dataaugmenteringstyperna tillämpas slumpmässigt på datat under träningsprocessen och ändras för varje epoch. Viktigt att tänka på är att inte använda augmentationstyper som kan riskera att det man vill identifiera klipps bort från bilden. Dataaugmentering är bra när man har lite data och kan motverka överanpassning. (Sylvain & Howard, 2020).

2.7.6 Överanpassning

Ett av de vanligaste problemen när man tränar en modell är att man överanpassar modellen. Under träning strävar man efter att valideringsfelet ska vara så liten som möjligt, om felprocenten börjar öka under träning har man överanpassat modellen. Det här innebär att modellen börjar anpassa sig för mycket efter det data den är tränad på vilket gör att nytt data inte längre kan identifieras. Överanpassning uppstår oftast när man har för lite data, det data man har är för specifikt eller man tränar modellen för länge. För att motverka överanpassning kan man sänka antal epochs, sätta till en viktminskning, använda sig av dataaugmentering eller någon kombination av dessa. (Sylvain & Howard, 2020).

3 Verktyg

I detta kapitel presenteras de olika verktyg och mjukvaror som används i projektet, och varför de har valts.

3.1 Kamera

Robotarna i fabriken använder sig redan av Omron kameror för placeringar och justeringar av komponenter så samma typ av kamera kommer användas för att samla bilder. En teststation har satts upp vid Nordic Lights där man kan ändra kamerans höjd, fokus och ändra ljuskällor för att få bilder med stor variation. Programmet som används för att ansluta till kameran är FH/FHV Series Vision System Ver.6.41.

3.2 Landing AI

Landing AI är en webbaserad bildigenkänningsplattform som använder sig av AI. Det är grundat av Andrew Ng som är en av de världsledande personer inom AI (Andrew Ng, u.å.). Landing AI:s verktyg kallas LandingLens och i det kan man träna tre olika typer av AI

modeller: objektdetektering, segmentering och bildklassificering. Landing AI är en betald plattform men det har en gratis demoversion som används i det här examensarbetet.

3.3 Clarifai

Clarifai är en webbaserad AI-plattform som innehåller datorsyn, språkteknologi och röstigenkänning. Det är grundat 2013 av Matthew Zeiler och plattformen kom topp fem i bildklassificering vid ImageNet 2013 Competition (Clarifai, u.å.). Clarifai har många olika typer av både betalda och gratis modeller, men i det här arbetet används endast bildklassificering som är gratis.

3.4 Sentisight AI

Sentisight AI är en webbaserad plattform som används för att göra AI-bildigenkänningsprogram. Det är fokuserat på att ha ett så smidigt och effektivt sätt att annotera och märka bilder och sedan kunna snabbt träna en modell på bilderna. Sentisight har tre olika typer av AI: bildklassificering, objektdetektering och segmentering. Sentisight är en betald plattform men har en gratisversion som används i det här arbetet.

3.5 Fastai

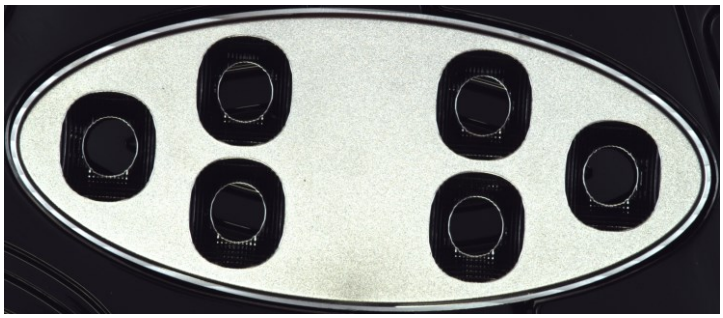
Fastai skapades av Jeremy Howard och Rachel Thomas 2016, och målet var att göra en mera lättillgänglig väg för nybörjare att lära sig använda AI (Howard, 2016). Det är ett djupinlärningsbibliotek som ger verktyg för att man snabbt och enkelt ska kunna bygga djupinlärningsmodeller. Fastai är byggt upp i Python och använder sig av PyTorch biblioteket som är helt gratis att använda. Det har färdiga modeller för bildklassificering och objektdetektering men om man vill tillverka en modell från grunden har det också verktyg för att hjälpa till med det. Fastai har också flera kurser som är gjorda för nybörjare till AI som går igenom grunderna med AI och hur man bygger upp simpla modeller.

4 Datainsamling

För att kunna träna en AI behöver man en del data, i det här fallet bilder. De flesta AI-program kräver minst 20–30 bilder per klassifikation, men oftast är ju mer bilder desto bättre. Det fanns ca 100 felaktiga reflektorer ihopsamlade och ett så gott som oändligt antal felfria reflektorer vid Nordic Lights när arbetet började.

4.1 Kamerainställningar

Inledningsvis måste kameran ställas in för att man ska få så tydliga bilder som möjligt. Kameran ansluts till med FH/FHV Series Vision System från dator och i det programmet kan man göra alla inställningar. Skärpan ställs in så bilden blir klar, bilden vitbalanseras med RGB värden på R: 1.4, G: 1.0, B:1.9 och slutarhastighet ställs till 60 000 µs så bilden får lämplig ljusstyrka. Till sist sätts ett beskäringsfilter på bilden så majoriteten av bilden tas upp av reflektorn. Man får då tydliga bilder av reflektorerna (Figur 11).

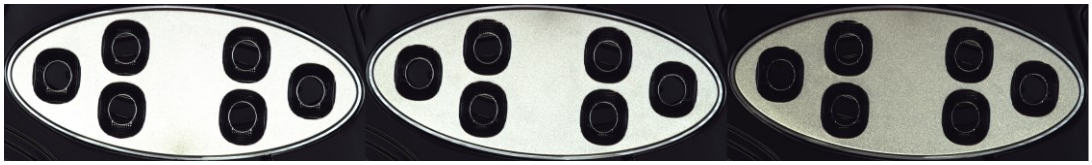


Figur 11. Bild av felfri reflektor.

4.2 Bildtagning

För att få mera variation i datat togs bilderna med varierande ljus (figur 12). Det fanns ett antal lampor uppsatta vid teststation som gick att flytta på och tända/släcka för att kunna variera ljuset. I normala fall vid produktionen skulle inte ljuset variera så mycket som det gör på de tagna bilderna, men för att vara på säkra sidan togs bilder som var lite extra

mörka/ljusa. Sammanlagt togs ca 200 bilder med varierande ljus där hälften var av felaktiga reflektorer. Bildernas upplösning är 2900 pixlar bred och 1250 pixlar hög.



Figur 12. Ljusskillnader.

4.3 Sortering och filtrering

Bilder delas in i två kategorier: defekta och felfria. De felfria bilder gås igenom manuellt och vissa bilder som hade defekter på sig som inte märktes under bildtagningen tas bort. Förutom kategorierna behöver också bilderna delas in i träningsdata, valideringsdata och testdata. Indelningen av träningsdata och valideringsdata gör de flesta AI-program av sig själv men för testdatat måste det oftast göras manuellt. Till att börja med plockades bara testdatat ut slumpmässigt men det visade sig senare att vissa typer av defekter inte kom med i träningsdatat, bilderna blev då uppdelade på nytt men den här gången gicks bilderna igenom manuellt och bilder med varierande defekter plockades ut för att få ett så överblickande testdata som möjligt. Till slut fanns totalt 181 bilder där tränings- och valideringsdatat har 76 felaktiga bilder och 66 felfria bilder, medan testdatat har 18 felaktiga och 20 felfria bilder. Senare när modellerna testades på plats togs lite flera bilder, totala testbilder i slutet var 35 felaktiga och 25 felfria.

5 Olika typer av defekter

De olika defekter reflektorerna kan ha går i stora drag att indela i fyra kategorier: repor, prickar, skrapmärken och fettfläckar. På de 76 bilder i tränings och valideringsdatat fanns totalt 66 repor, 86 prickar, 73 skrapmärken och 40 fettfläckar. De här värdena är inte 100 % exakta dock eftersom i vissa fall är det ganska otydligt om flera defekter överlappas eller det bara är en defekt.

Repor är lätta att märka ut eftersom de alltid har ganska tydliga början och slut. Ett typiskt exempel av en repa syns i figur 13.

Figur 13. Reflektor med repa.

Skrapmärken kan se lite ut som repor men är oftast bredare, det kan också ofta vara otydligare var de börjar och slutar än repor. Ett exempel på skrapmärke syns i figur 14.

Figur 14. Reflektor med skrapmärke.

Prickar är också väldigt tydliga största delen av tiden och där prickar framkommer är det oftast mer än en. Exempel på prickar syns i figur 15.

Figur 15. Reflektor med prickar.

Fettfläckar är oftast väldigt lätta att se, men kan vara svåra att märka ut exakt eftersom de har otydlig kontur. Fettfläckar tar också ofta upp väldigt stor del av reflektorn som kan ses i figur 16.

Figur 16. Reflektor med fettfläck.

6 Träning av modellerna

Fyra olika program testades att träna modeller i: LandingLens, Clarfiai, Sentisight AI och Fastai. Bland dessa tränades tre olika typer av modeller: bildklassificering, objektdetektering och segmentering. Bildklassificering gjordes i alla fyra, objektdetektering gjordes i LandingLens och Sentisight, och segmentering gjordes endast i LandingLens.

6.1 Fastai

Till skillnad från de andra tre AI-program som testas är Fastai ett programmeringsbibliotek och inte en direkt applikation så det kräver att man skriver koden för att göra en modell själv. Pythonkoden skrivs i notebooks på Kaggle i browser men den går lika bra att köra det lokalt på en dator bara man laddar ner Fastai biblioteket.

Koden som skrivs är för bildklassificering, där de olika kategorierna är NOK eller OK, beroende på om reflektorn på bilden har defekter eller inte. Koden börjar med att man importerar Fastai biblioteket för att få alla Fastai funktioner, det här biblioteket innehåller också alla PyTorch funktioner:

```
from fastai.vision.all import *
```

En sökväg till bilderna som ska användas för träningen ställs in:

```
path = '/kaggle/input/trainingsetmanualsort'
```

En funktion för att definiera kategorierna på bilderna görs. Namnet på bilderna börjar endera med OK eller NOK, funktionen är skriven så den returnerar True om bildens namn börjar på O, om inte returnerar den False:

```
def is_ok(x): return x[0].startswith('O')
```

Fastai kräver att man har en dataloader när man ska göra modell. Dataloaderns uppgift är att berätta åt modellen vad vi har för data, hur datat ska kategoriseras, hur valideringsdatat ska indelas, och vad vi ska ha för augmenteringar. Dataloadern ställs in på följande sätt:

```
dls = ImageDataLoaders.from_name_func('.',
    get_image_files(path), valid_pct=0.25,
    label_func=is_ok, bs = 32,
    item_tfms=Resize((625,1450),ResizeMethod.Crop),
    batch_tfms=aug_transforms(flip_vert=False, max_warp=0.1,
    max_zoom=1, max_lighting=0.2, max_rotate=6))
```

De olika variablerna som ställs in är:

- get_image_files(path): ställer in sökvägen till bilderna.
- valid_pct=0.25: sätter undan 25 % av bilderna till validering.
- label_func=is_ok: ställer in kategoriseringsfunktion.
- bs=32: ställer in batch size till 32 bilder. En högre batch size än 32 gjorde så GPU:n blev utan minne, men om man sätter det lägre börjar modellen prestera sämre.
- Item_tfms=Resize ((625,1450),ResizeMethod.Crop): ställer in storleken på bilderna. Om bildstorleken lämnades i original upplösning hamnade man att ställa ner batch sizen, men det gjorde modellen mycket sämre. Därför halveras både höjd och bredd på bilderna för att få en bättre modell. På grund av den höga upplösningen till att börja med inverkade det här inte mycket på själva tydligheten av bilderna.

- batch_tfms=aug_transforms(flip_vert=False,max_warp=0.1,max_zoom=1,max_lighting=0.2, max_rotate=6)): ställer in de olika augmenteringarna som ska användas under träning. Vertikal flip och zoom stängs av eftersom risken finns att defekter klipps ut ur bilden. Varp, ljusförändring och rotation används, men de har ganska låga värden för att inte riskera att defekterna blir för otydliga eller helt försvinner.

En learner skapas för att träna modellen och spara tränade värden:

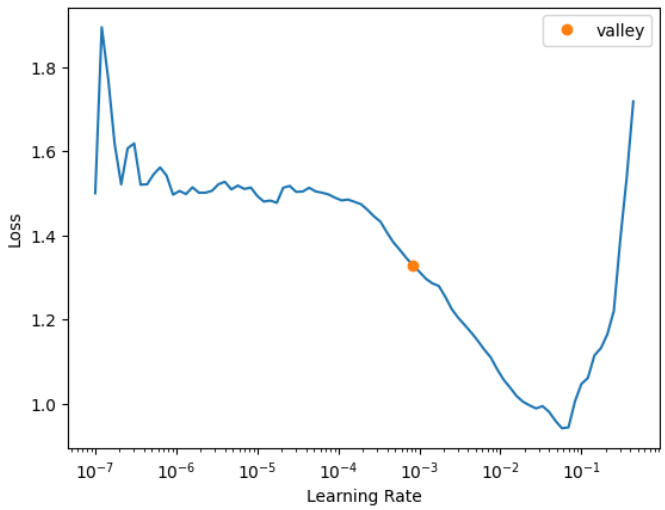
```
learn = vision_learner(dls, arch=resnet18, metrics=error_rate)
```

Learnern som skapas är en vision learner som använder sig av CNN. Learnern använder en förtränad modell som sedan anpassas till ens eget data. Den förtränade modellen som valdes att användas här är resnet18. Resnet valdes eftersom det är en av de bästa modellerna för bildklassificering (Sylvain & Howard, 2020). Det finns också några olika versioner av resnet och specifikt resnet18 användes för att det är en av de mindre modellerna så att träna det går snabbare. Resnet18 tar under 30 sekunder per epoch att tränas medan större nätverk som till exempel resnet34 kan ta nästan dubbelt längre.

Som nästa steg ska den färdigtränade modellen finjusteras utgående från egna bilder. Men för att göra det behöver man ställa in antal epochs, inlärningshastighet och viktminskning. För att hitta en optimal inlärningshastighet körs en inlärningshastighetssökare med funktionen:

```
learn.lr_find()
```

Funktionen ger en ut en graf som visar inlärningshastighet i relation till felet (figur 17). Man vill att felet ska vara så lågt som möjligt men om väljer inlärningshastigheten efter det minimala felet som syns i grafen finns risk att det slår över och felet blir väldigt högt i stället (Sylvain & Howard, 2020). Därför är den rekommenderade inlärningshastigheten ca 10 gånger mindre än där felet var som minst. Inlärningshastigheten sätts till 0,007.



Figur 17. Inlärningshastighet graf.

Viktminskning är som standard i fastai 0.01 och det verkar fungera bra så det behöver inte ändras i det här fallet. Antal epochs är svårare att uppskatta och man måste bara testa sig fram. Antal epochs sätts till att börja med till 20 och learnern finjusteras några gånger för att få en känsla av hur många som behövs:

```
learn.fine_tune(epochs=20, base_lr=0.007, wd=0.01)
```

Under de första några epochs när finjusteringen körs är felprocenten ganska ostabil men i de flesta fall börjar den stabiliseras runt ca 15 epochs som syns i figur 18. I vissa fall börjar också modellen överanpassas redan vid 15 epochs, men 15–20 verkar vara ett lämpligt värde för antalet epochs i det här fallet. Orsaker att det varierar för varje gång man kör beror på att augmenteringar, valideringsdata och även själva optimeringsalgoritmen fungerar slumpmässigt.

epoch	train_loss	valid_loss	error_rate	time
0	0.538583	0.790279	0.314286	00:16
1	0.554121	0.739716	0.285714	00:18
2	0.578339	0.610943	0.228571	00:22
3	0.475636	0.274244	0.085714	00:23
4	0.472339	1.354864	0.400000	00:23
5	0.429931	0.285151	0.085714	00:23
6	0.379245	0.376332	0.171429	00:24
7	0.363145	0.714163	0.142857	00:23
8	0.344948	0.910466	0.085714	00:23
9	0.332503	0.560700	0.171429	00:23
10	0.310247	0.361324	0.085714	00:23
11	0.289812	0.897834	0.085714	00:23
12	0.262777	1.184326	0.142857	00:23
13	0.252983	1.053142	0.142857	00:23
14	0.244549	0.697058	0.085714	00:23
15	0.231874	0.537185	0.085714	00:24
16	0.219581	0.482234	0.085714	00:23
17	0.210702	0.481870	0.085714	00:23
18	0.202233	0.490587	0.085714	00:23
19	0.195485	0.474661	0.085714	00:23

Figur 18. Finjustering av parametrar i Fastai.

Finjustering testades ännu några gånger med varierande epochs mellan 15–20 för att försöka få fram en så bra modell som möjligt. De bästa tränade modellerna som framkom var en med 15 epochs som hade en felprocent på 2.86 och en annan modell med 20 epochs som hade en felprocent på 0 (figur 19).

epoch	train_loss	valid_loss	error_rate	time
0	0.835997	3.208080	0.457143	00:31

epoch	train_loss	valid_loss	error_rate	time
0	0.703048	2.495127	0.457143	00:28
1	0.668680	0.794631	0.400000	00:27
2	0.652595	0.754916	0.371429	00:27
3	0.593941	0.535288	0.200000	00:27
4	0.538615	1.515142	0.371429	00:27
5	0.494461	0.103178	0.057143	00:26
6	0.499215	0.272903	0.085714	00:27
7	0.475851	0.655209	0.142857	00:27
8	0.437885	1.085969	0.142857	00:27
9	0.412485	0.779747	0.142857	00:27
10	0.385614	0.402571	0.142857	00:27
11	0.352369	0.155864	0.028571	00:27
12	0.320587	0.074163	0.028571	00:26
13	0.307617	0.054506	0.028571	00:27
14	0.284532	0.045570	0.028571	00:27

epoch	train_loss	valid_loss	error_rate	time
0	1.302200	0.554452	0.285714	00:25

epoch	train_loss	valid_loss	error_rate	time
0	0.878461	0.365705	0.114286	00:23
1	0.719808	0.378139	0.171429	00:24
2	0.620788	1.050475	0.428571	00:25
3	0.583561	2.408031	0.485714	00:25
4	0.512932	1.519359	0.371429	00:23
5	0.490019	0.663643	0.142857	00:24
6	0.492209	0.474735	0.200000	00:24
7	0.444845	1.871675	0.342857	00:24
8	0.460242	0.110285	0.085714	00:23
9	0.460153	0.131025	0.057143	00:24
10	0.441641	0.434830	0.171429	00:24
11	0.407669	1.209034	0.285714	00:23
12	0.382532	1.367759	0.285714	00:23
13	0.356512	0.573273	0.200000	00:20
14	0.331930	0.304095	0.142857	00:16
15	0.305227	0.146529	0.057143	00:16
16	0.284129	0.078446	0.028571	00:16
17	0.260250	0.038759	0.028571	00:16
18	0.251936	0.019661	0.000000	00:16
19	0.235550	0.016153	0.000000	00:16

Figur 19. Fastai finjustering.

Värt att notera att en felprocent på 0 betyder inte direkt att modellen är perfekt. På grund av att felprocenten är uträknad efter valideringsdatat och vilka bilder som hör till valideringsdatat är slumpmässigt kan man i vissa fall bara ha tur och modellen råkade fungera bra på ens valideringsbilder. Det här är en större risk eftersom vårt valideringsdata är endast ca 40 bilder, det här är orsaken vi också har lagt undan en del bilder till testdata för att kunna vara säkra på att modellen faktiskt är så bra som vi tror. När modellerna är färdigtränade exporteras de med funktionen:

```
learn.export('model.pkl')
```

6.2 Clarifai

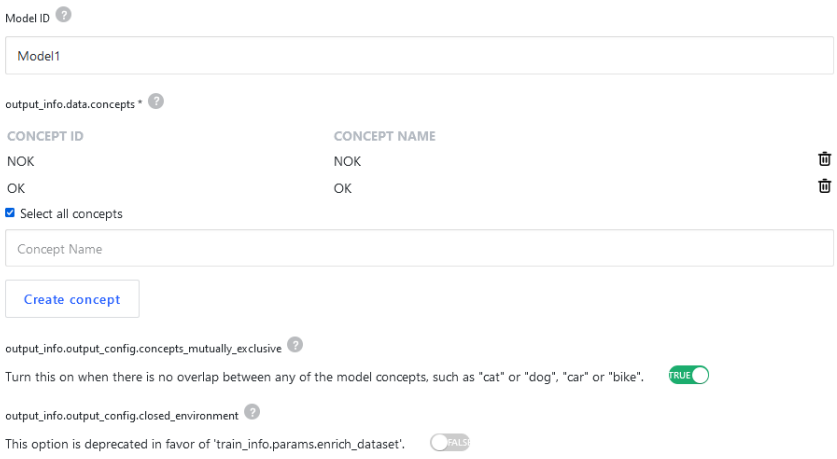
Träning av Clarifai modellen görs i webbläsare. Clarifai har två portaler för att träna AI-modeller, en legacy och en nyare portal. Den nyare portal har endast begränsad tillgänglighet eftersom den är inte helt färdig ännu så legacy portalen används i det här

projektet. Med Clarifai testas bildklassificering där bilderna är indelade i två kategorier lika som i Fastai: OK och NOK.

När man gör en modell får man först välja förtränad modell som ska anpassas efter ens eget data. Modellerna är namngivna efter vilka typer av bilder de är tränade på som mat, ansikten, djur med mera. Det finns ingen modell som är specifikt passande för det här arbetet, så den allmänna modellen ”General” används.

Som första steg måste bilder laddas upp till modellen, detta görs genom att gå in fliken ”Data Mode” och välja de bilder man vill ladda upp. Det går att välja kategori på bilderna redan när man laddar upp så det inte behövs göras senare. Uppladdning till sidan fungerar ganska dåligt eftersom statusen för uppladdningen inte syns. Enda sättet att veta när uppladdningen är klar är att ladda om sidan vilket avbryter uppladdningen om den inte är klar. Man måste bara vänta extra länge och hoppas på att det är färdigt när man laddar om.

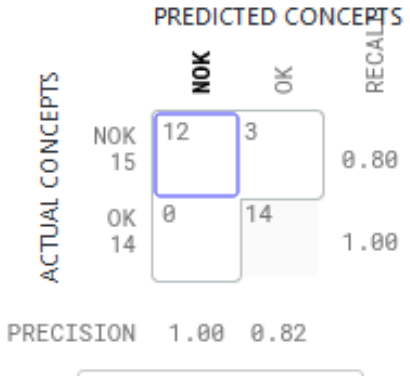
När bilderna är färdigt uppladdade kan man träna en modell. Modellen som tränas kallas Transfer Learning Classifier som är en bildklassificeringsmodell. Inställningarna för modellen är ganska minimala. Man väljer vilka koncept den ska lära sig vilket i det här fallet är OK och NOK, och väljer att det får bara finnas ett koncept per bild (figur 20).



Figur 20. Clarifai modellinställningar.

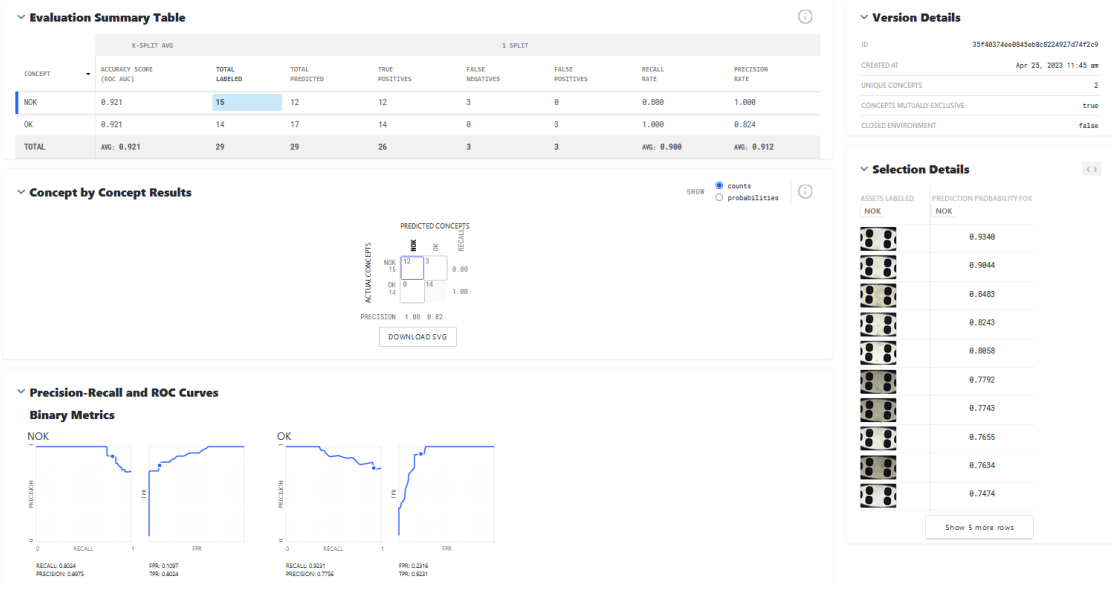
Efter det är det bara att vänta på att modellen tränas, träningen är väldigt snabbt och tar knappt en minut. Den bästa tränade modellen har en noggrannhet på 92 %, med 82 %

precision på OK-bilder och 100 % precision på NOK-bilder som man kan se på sammanblandningsmatrisen i figur 21.



Figur 21. Clarifai sammanblandningsmatris.

Om man vill veta specifikt vilka bilder som blev fel identifierade kan man välja att visa bilder märkta som NOK men identifierade som OK. Det här visar dock endast bilderna som en liten ikon och namnet på bilder visas inte, vilket gör det så gott som omöjligt att faktiskt veta vad som var problemet (figur 22).



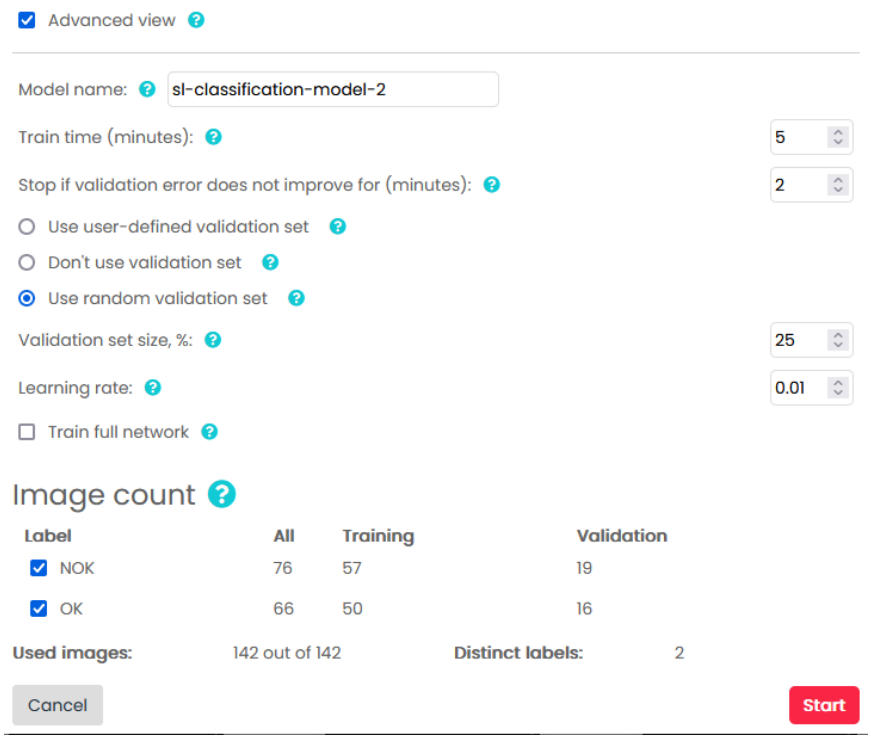
Figur 22. Clarifai resultat.

Till skillnad från Fastai där träningen av modellen hade vissa slumpmässiga variabler vilket gjorde att den tränade modellen prestation kunder variera, verkar inte Clarifai ha något

sådant. Så länge varje tränad modell har blivit given samma bilder är resultatet alltid det samma.

6.3 Sentisight AI

Träning av en AI-modell i Sentisight görs i webbläsare och både bildklassificering och objektdetekteringsmodeller tränas. Sentisight har flera inställningar man kan ändra på innan man tränar en modell än Clarifai. Man kan ställa in hur länge den ska träna och efter hur länge den ska stoppa ifall valideringsfelet inte blir bättre. Valideringsdata procenten och inlärningshastigheten går också att ställa in. Bildkvaliteten går inte att ställa in och alla bilder blir automatiskt anpassade till 299x299 pixlar upplösning. Bild av inställningar syns i figur 23.



Figur 23. Sentisight AI-modellinställningar.

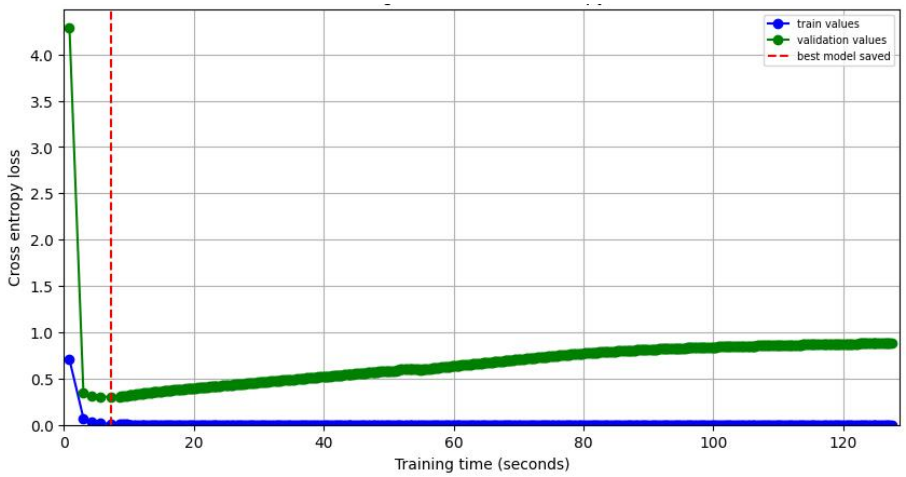
6.3.1 Bildklassificering

För bildklassificering ställs träningstiden till fem minuter med stopp efter två minuter om valideringsfelet inte blivit bättre. Till skillnad från de andra programmen som sparar modellen med de värden den hade efter att modellen körts igenom alla epochs, använder

sig Sentisight av det lite äldre systemet att spara modellen utefter den epoch där valideringsfelen var som minst. Även fastän träningstiden är inställd på fem minuter blir den bästa modellen oftast sparad redan under de första 20 sekunderna vilket gör att träningen normalt tar bara lite på två minuter.

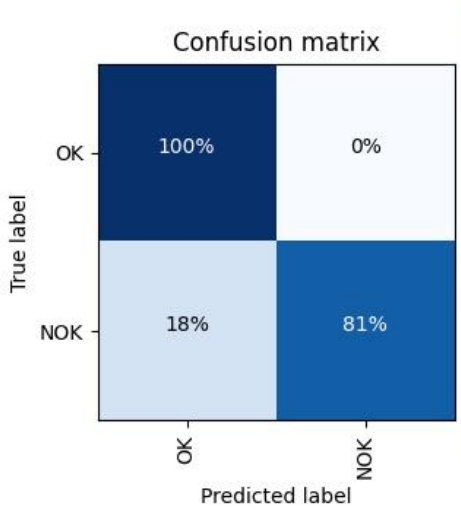
Valideringsdatat sätts till 25 % och inlärningshastigheten sätts till 0,01. Inlärningshastigheten provades med lite olika värden mellan 0,1–0,001 men det verkade inte påverka hur bra modellen presterade så det lämnades på 0,01. Bilderna som tidigare är indelade i kategorierna OK och NOK.

När modellen är tränad kan man se inlärningskurvan på modellen, i figur 24 ser man att felet var som bäst vid ca 7 sekunder och det är den modellen som blir sparad.



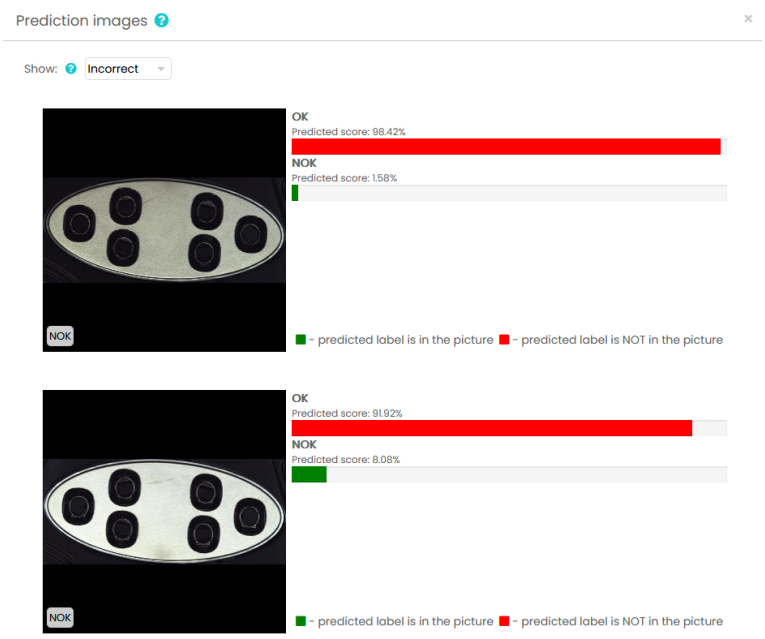
Figur 24. Sentisight inlärningsgraf.

Den bästa tränade modellen som framkom hade 90,2 % noggrannhet, 91,9 % precision, 90,2 % återkallelse och 90,2 % F1. Från sammanblandningsmatrisen i figur 25 kan man se att den har en 100 % precision på OK-bilder och 81 % precision på NOK-bilder.



Figur 25. Sentisight sammanblandningsmatris.

Till skillnad från Clarifai, har Sentisight ett bättre sätt att se vilka bilder som modellen hade problem med. Man kan direkt välja att visa alla identifikationer och sedan sortera efter felidentifierade bilder. På de felidentifierade bilderna syns då en procent på hur säker modellen var på sin identifikation (figur 26).



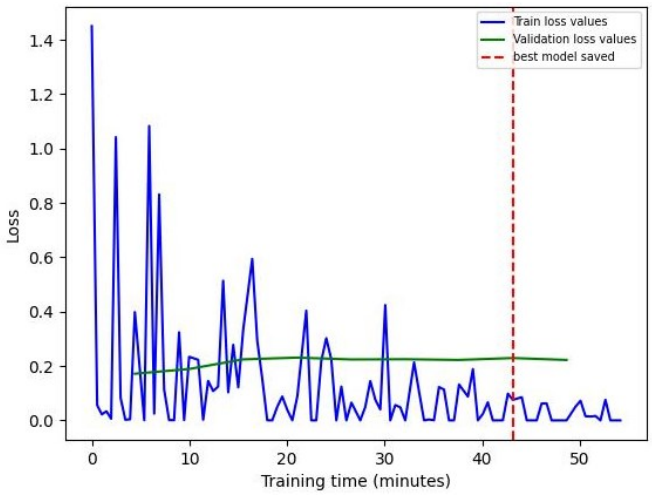
Figur 26. Sentisight AI felidentifikationer.

6.3.2 Objektdetektering

Sentisight har ett eget verktyg för att markera objekt i bilder. Objekten markeras ut med en rektangel runt felen i bilderna och alla fel hålls inom en klass.

Objektdetektering kräver mycket mera tid att tränas än bildklassificering, träningstiden ställs in till en timme med ett stop efter 15 minuter om inte modellen förbättras. Valideringsprocenten sätts till 25 % och inlärningshastigheten hålls på det rekommenderade 0,001. Modellen väljs också att tränas på omarkerade bilder för att få mera felfritt data till träningen.

Den bästa tränade modellen kommer fram vid ca 43 minuter som syns i figur 27 och har 62,9 % precision, 44,7 % återkallelse och 52,2 % F1.

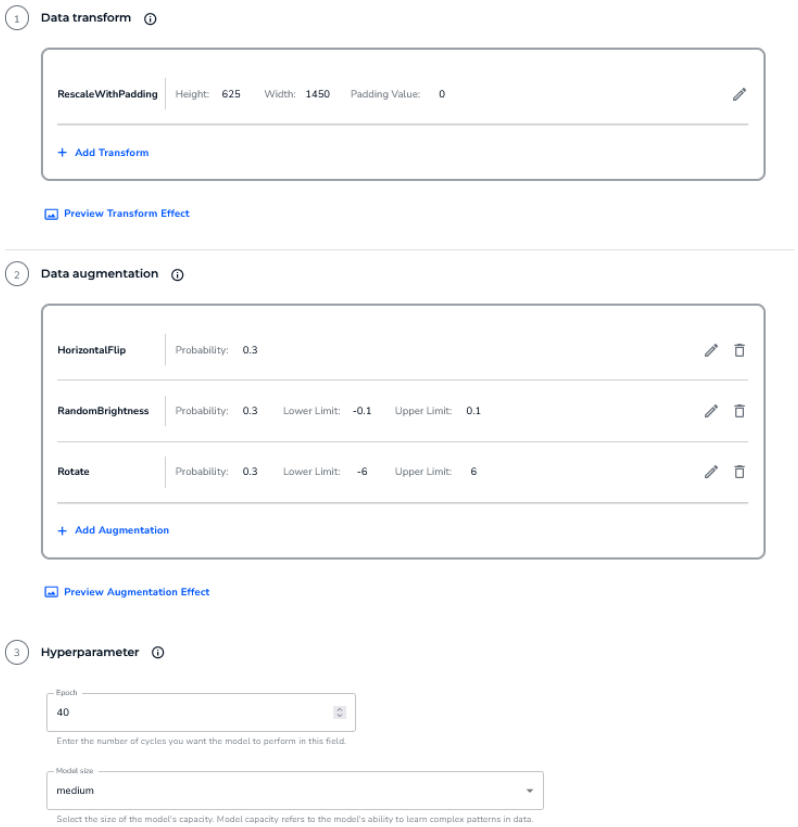


Figur 27. Sentisight inlärningsgraf.

6.4 LandingLens

LandingLens modeller tränas i webbläsare och både bildklassificering, objektdetektering och segmentering provas. Inställningar som går att göra före man tränar modellen är att ändra storlek på bilderna, lägga till dataaugmenteringer, ändra antal epochs och ändra modellens storlek. Bildernas storlek får max vara 1500x1500 pixlar, så storleken sätts till 625x1450 för att bildernas ska behålla samma bildförhållande. Dataaugmenteringer som används är horisontell flip, slumpmässig ljusförändring och rotation. Lika som i Fastai sätts dessa ganska låga för att undvika att defekter försvinner ur bilden. Ljusförändring sätts att vara mellan -1 till 1 och rotation sätts till max 6°. Antal epochs är som standard 40 och det fungerar bra så det ändras inte på. Modellstorleken går att välja mellan eller liten. En större

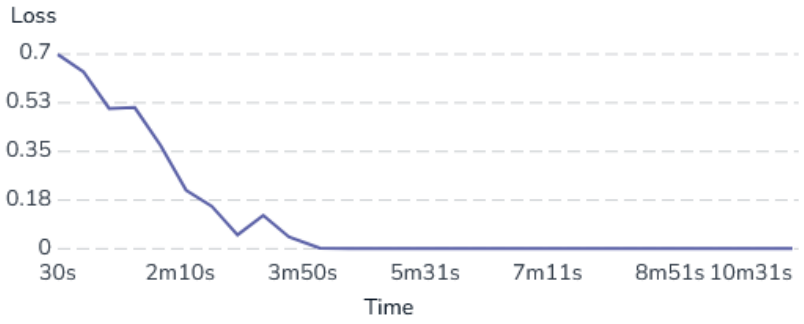
modell tar i allmänhet längre att träna men ger bättre resultat, och i LandingLens är skillnaden i träningstiden inte stor, modellens storlek väljs till mellan. Inställningarna syns i figur 28.



Figur 28. LandingLens modellinställningar.

6.4.1 Bildklassificering

Som i tidigare program är bilderna indelade i två kategorier: OK och NOK. När modellen tränas kan man se en inlärningskurva där valideringsfelen sjunker över tid (Figur 29).



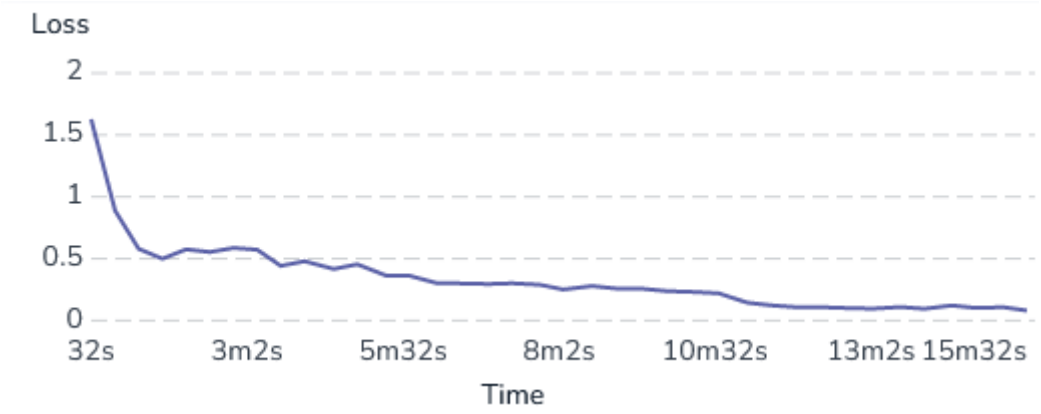
Figur 29. LandingLens inlärningsgraf.

När modellen är färdigtränad har den ett valideringsfel på noll med en precision på 99,5 % och en återkallelse på 99,4 %. Till skillnad från de andra program som gav prestandan på modellen efter valideringsdatat ger LandingLens prestandan efter allt data, inkluderat träningsdata och testdata. För att kunna jämföra det bättre med de andra modellerna kan man filtrera efter att bara visa valideringsdata och då är precisionen 100 % och återkallelsen 100 %. Statistiken på hur modellerna presterar på testdatat kommer upp senare.

6.4.2 Objektdetektering

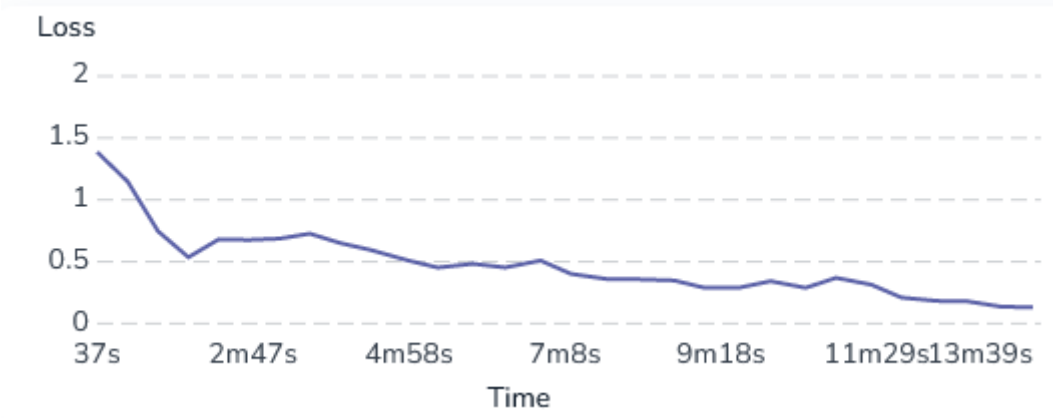
I LandingLens tränades med två olika objektdetekteringsmodeller. En modell där alla fel var i en klass och en modell där felen var uppdelade i fyra klasser: repor, prickar, skrapmärken och fettfläckar. Objektdetekteringsmodeller tar lite längre att träna än bildklassificering men jämfört med Sentisight som krävde upp mot en timme behöver LandingLens endast runt 15 minuter.

När modellen med en klass tränas får vi en inlärningskurva som i figur 30, där man kan se att valideringsfelet sjunker till ca 0,1. Vid en konfidensgräns på 0,3 har modellen en precision på 94,5 % och en återkallelse på 88,6 %. Modellens prestanda på endast validerings datat är 86,8 % precision och 73,3 % återkallelse.



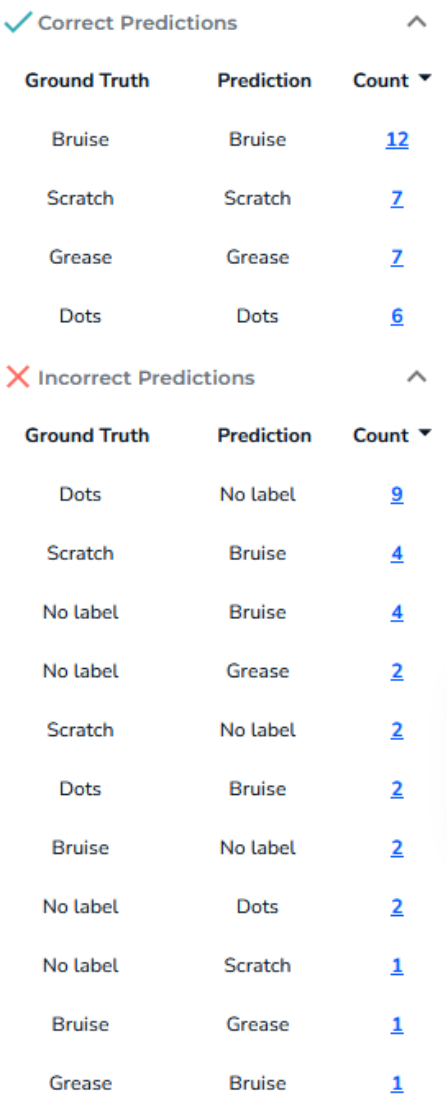
Figur 30. LandingLens inlärningsgraf.

För att se hur modellen skulle prestera ifall de olika defekterna var mera grupperade tränades en annan modell där defekterna är indelade i fyra olika klasser. Den modellen får en inlärningskurva som i figur 31 och har en valideringsfel på ca 0,2. Modellens prestanda vid en konfidensgräns på 0,23 är på alla bilder är 76,3 % precision och 70,7 % återkallelse. Prestandan på endast valideringsdatat är 65,3 % precision och 60,4 % återkallelse.



Figur 31. LandingLens inlärningsgraf.

I figur 32 kan man se hur defekterna identifierades. Det intressanta är att den hade störst problem med att identifiera prickar med nio som inte alls blev identifierade medan bara sex blev rätt identifierade.



✓ Correct Predictions

Ground Truth	Prediction	Count
Bruise	Bruise	12
Scratch	Scratch	7
Grease	Grease	7
Dots	Dots	6

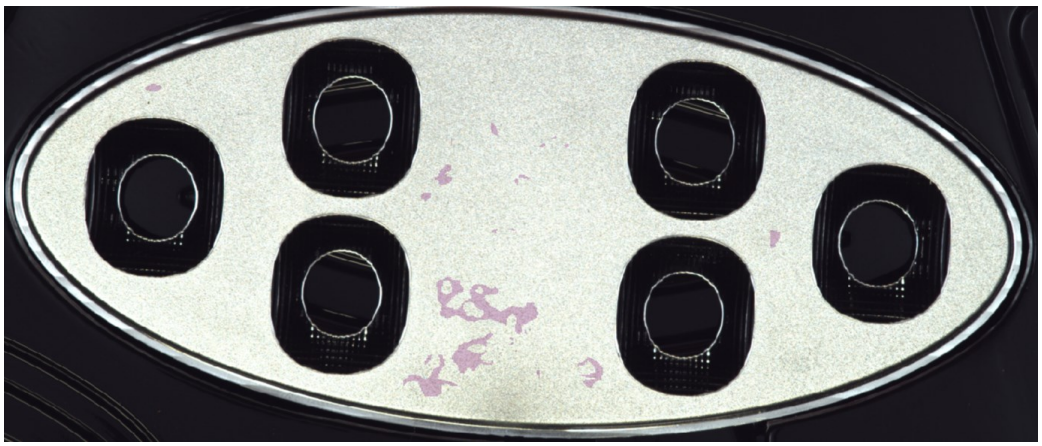
✗ Incorrect Predictions

Ground Truth	Prediction	Count
Dots	No label	9
Scratch	Bruise	4
No label	Bruise	4
No label	Grease	2
Scratch	No label	2
Dots	Bruise	2
Bruise	No label	2
No label	Dots	2
No label	Scratch	1
Bruise	Grease	1
Grease	Bruise	1

Figur 32. LandingLens identifikationer.

6.4.3 Segmentering

Segmentering testades också i LandingLens, men det märktes ganska snabbt att segmentering inte alls fungerar i det här fallet. Segmentering har samma problem som i objektdetektering där vissa defekter är svåra att märka exakt ut, men skillnaden är att objektdetektering är mera skonsam eftersom defekterna bara behöver vara inom en ruta. I segmentering måste defekterna markeras så konturen runt defekten är tydlig. Även om man är noggrann är det här så gott som omöjligt att göra på stor del av defekterna. Resultatet av att träna en modell är att modellen börjar märka ut en massa saker på helt felfria reflektorer som man kan se i figur 33.



Figur 33. LandingLens segmentering.

7 Testning

När modellerna är tränade måste de ännu testas på helt nytt data. Av denna orsak har ett antal bilder satts undan som testdata. Modellerna testas på totalt 60 bilder varav 35 har defekter av varierande typer och 25 är felfria.

7.1 Fastai

För att använda den tränade modellen behöver man bara några rader kod. Man måste importera fastai biblioteket, definiera alla funktioner som fanns i modellen och sedan importera den med funktionen load_learner:

```
from fastai.vision.all import*
def is_ok(x): return x[0].startswith('O')
learn = load_learner('/kaggle/input/model-save/model.pkl')
```

Sedan körs en simpel for loop för att checka igenom alla bilderna i en mapp:

```
import os
directory = '/kaggle/input/test'
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    is_ok,_,probs = learn.predict(f)
    print(filename, f" is ok: {is_ok}.")
    print(f"Probability it's ok: {probs[1]:.4f}")
```

Det här ger ett utvärde som visar vad bilderna blev identifierade som och hur säker modellen var på sin identifikation. Som man kan se i figur 34 blev till exempel NOK (9) identifierad som inte ok med en sannolikhet att den är ok på 0,05 %, medan OK (3) blev identifierad som ok med en sannolikhet på 94,32 %. Som standard är konfidensgränsen 50 % för att identifiera en bild som ok.

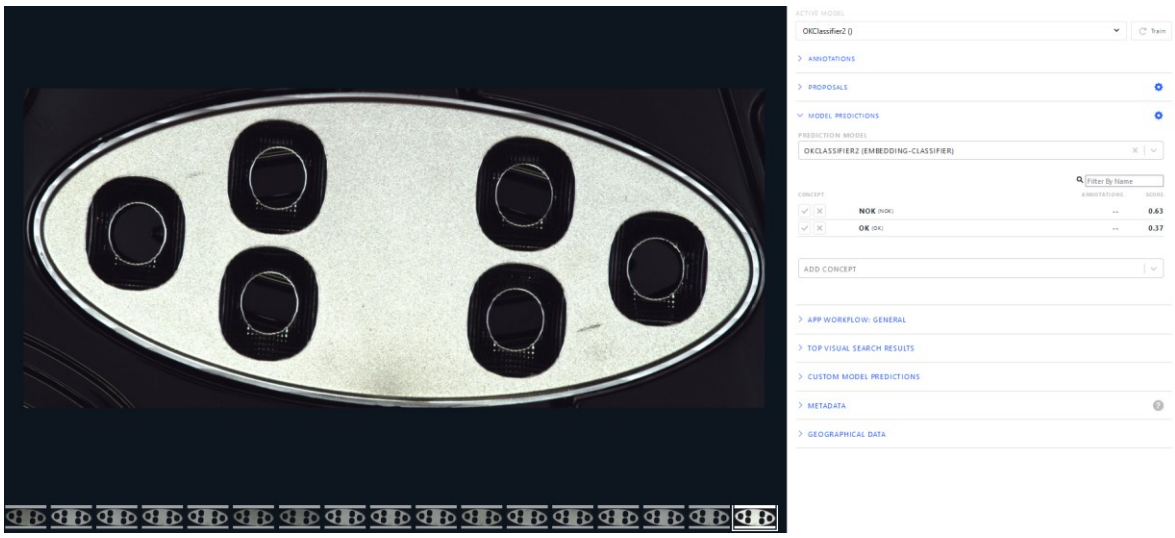
```
NOK (9).jpg  is ok: False.
Probability it's ok: 0.0005

NOK (5).jpg  is ok: False.
Probability it's ok: 0.0003

OK (3).jpg  is ok: True.
Probability it's ok: 0.9432

NOK (4).jpg  is ok: False.
Probability it's ok: 0.0000

OK (1).jpg  is ok: True.
Probability it's ok: 0.9822
```

Figur 34. Fastai identifikationer.

Vi tränade två modeller i Fastai, modell1 med felprocent på 2,9 % under validering och modell2 med 0 %. Med konfidensgräns på 50 % gav modell1 en noggrannhet på 90 % medan modell2 gav en noggrannhet på 85 %. Med modell1 var de felidentifierade bilderna ganska splittrade med fyra defekta och två OK-bilder felidentifierade, medan modell2 hade en defekt och åtta OK-bilder felidentifierade. Eftersom modell2 hade mycket mer OK-bilder felidentifierade kan man pröva sänka konfidensgränsen för att få ett bättre resultat. Det här hjälpte bara lite och vid en optimal konfidensgräns på 6–10 % ökade noggrannheten till 87,77 %. I det här fallet fastän modell2 hade bättre resultat under validering presterade den sämre än modell1 på testdatat.

7.2 Clarifai

Clarifai har ingen specifik funktion för att testa modeller, man måste bara ladda upp bilder på samma sätt som till träningen och sedan gå igen bilderna en och en för att se resultaten på modellen (figur 35).



Figur 35. Clarifai identifikation.

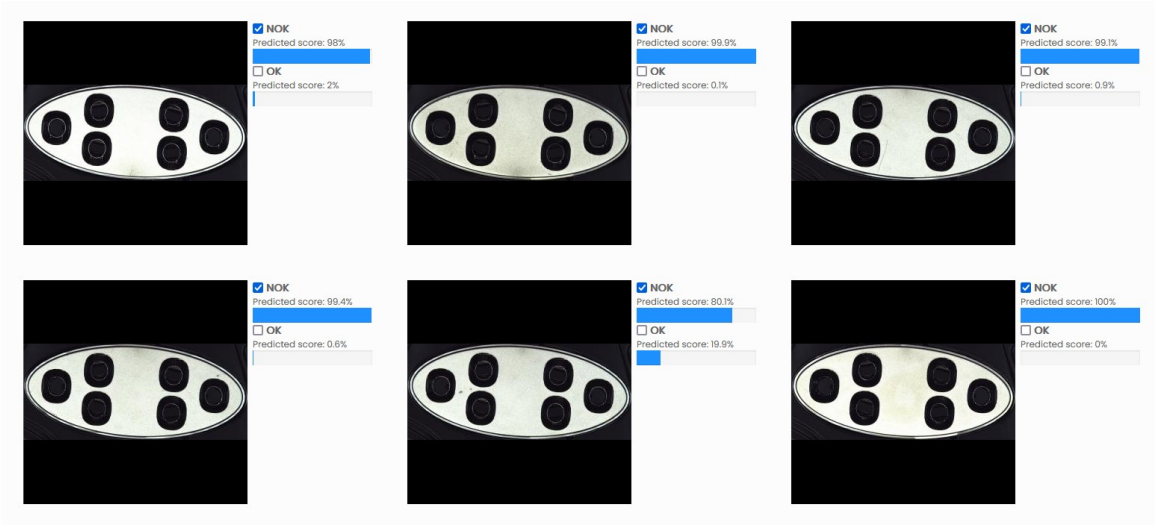
Clarifai bildklassificeringsmodellen gav en 81,67 % noggrannhet, med 20/25 OK-bilder och 29/35 NOK-bilder korrekt identifierat. Clarifai var också mycket osäkrare med sina identifikationer än de andra modellerna. Medan de andra modellerna var oftast 90–100 % säkra på sina korrekta identifikationer hade Clarifai väldigt få bilder som var identifierade med över 90 % säkerhet.

7.3 Sentisight AI

I Sentisight kan man enkelt göra identifikationer på nytt data genom att gå till Make a prediction på modellen man vill testa och ladda upp bilderna där.

7.3.1 Bildklassificering

När man laddat upp bilderna kan man enkelt se bilderna med identifikation och deras säkerhet vid sidan om (figur 36).

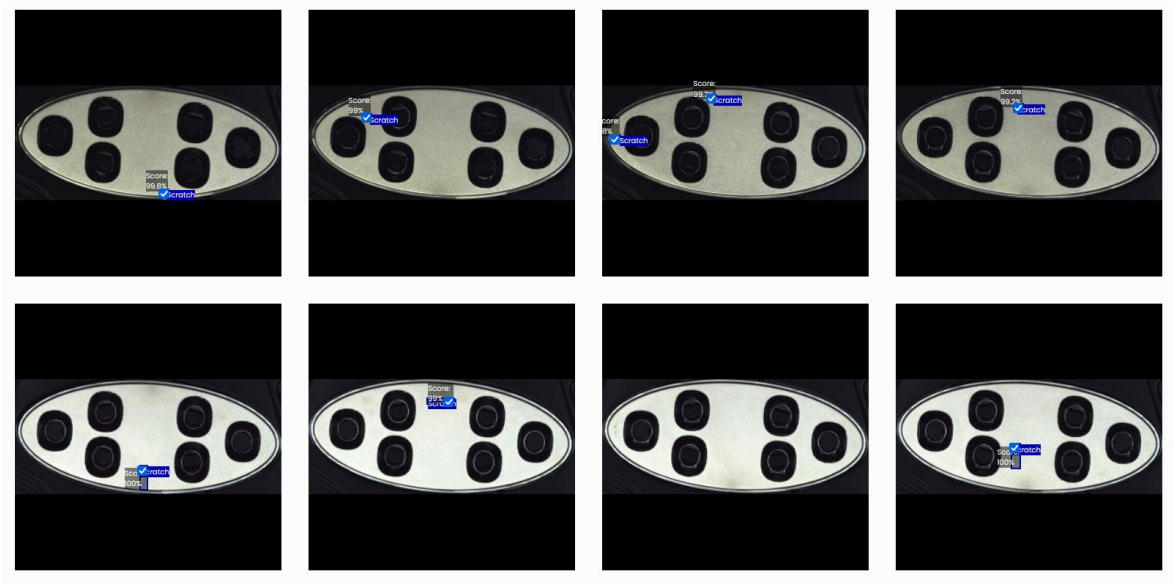


Figur 36. Sentisight AI identifikationer.

Bildklassificeringen gav en noggrannhet på 81,67 %, med 19/25 OK-bilder och 30/35 NOK-bilder korrekt identifierat.

7.3.2 Objektdetektering

När man laddat upp bilderna kan man se märkta defekter med säkerhet på bilderna (figur 37). Bildkvaliteten är dock mycket låg och det är svårt att se vilka defekter som är märkta ut. Om man bara tar i beaktande om bilden en hade defekt eller inte, hade 34/35 defekta bilder märkt åtminstone en defekt, medan 20/25 OK-bilder inte hade märkt ut någon defekt, vilket är lite bättre än bildklassificeringen.



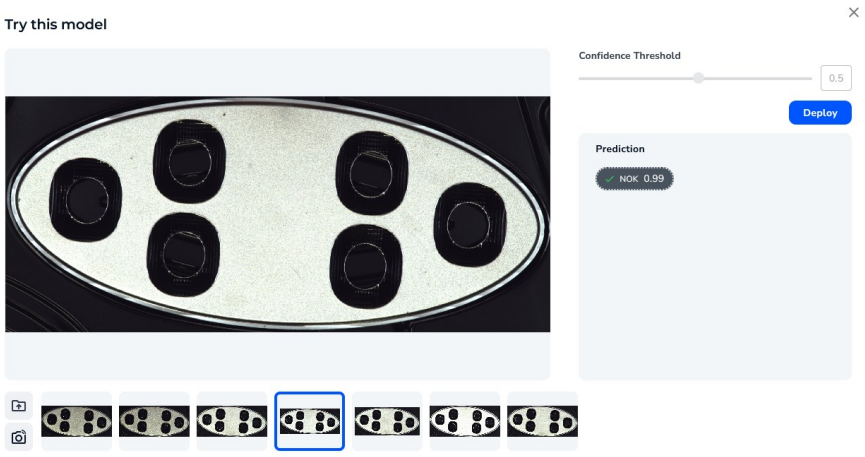
Figur 37. Sentisight AI objektdetektering.

7.4 LandingLens

I LandingLens kan man välja att göra nya identifikationer genom att trycka på Predict och ladda upp bilder.

7.4.1 Bildklassificering

Med bildklassificering ser man identifikationen av bilden och dess säkerhet (figur 38). Det finns också en konfidensgräns som är inställd på 50 %, den här går inte att ändra i gratisversionen av programmet.

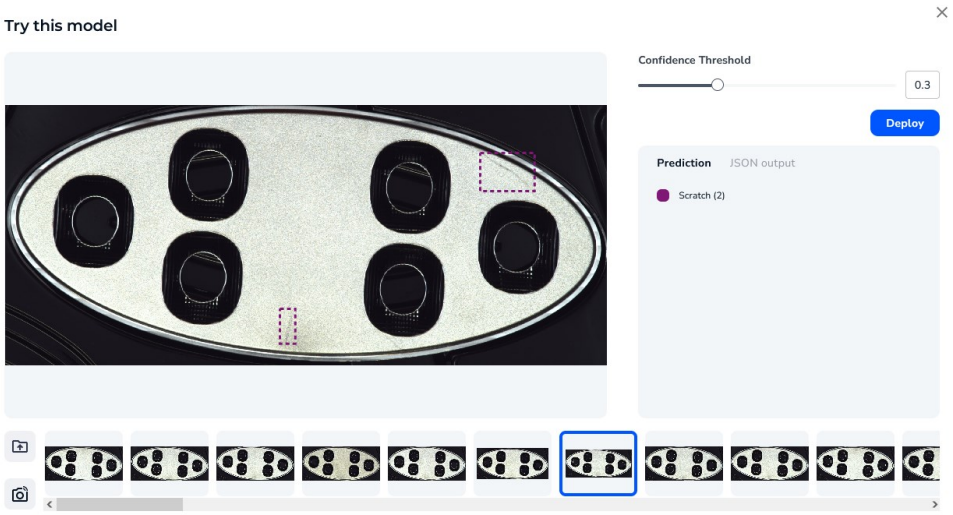


Figur 38. LandingLens identifikation.

Av de 60 testbilderna hade LandingLens överlägset bästa resultat med en noggrannhet på 98,33 % med endast en defekt bild felidentifierad. Den felidentifierade bilden hade dock en säkerhet på 80 % medan den mest osäkra OK bilden hade en säkerhet på 85 %. Det här betyder att om man skulle kunna ändra konfidensgränsen till något mellan 80–85 % skulle modellen ha 100 % noggrannhet på testdatat.

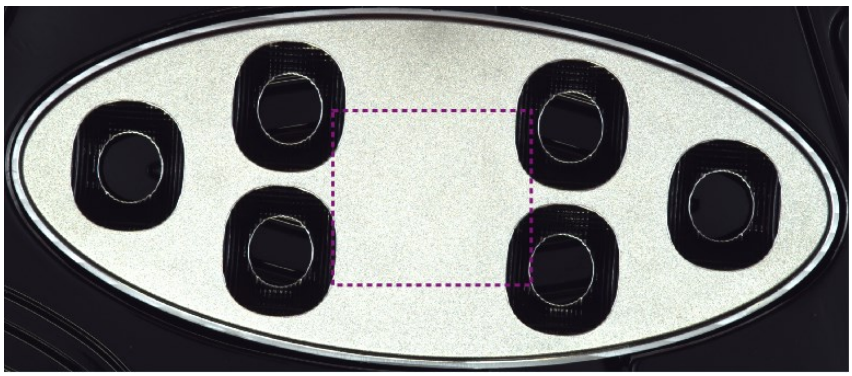
7.4.2 Objektdetektering

Med objektdetektering ser man märkta defekter på bilderna och man kan enkelt ändra på konfidensgränsen (figur 39).



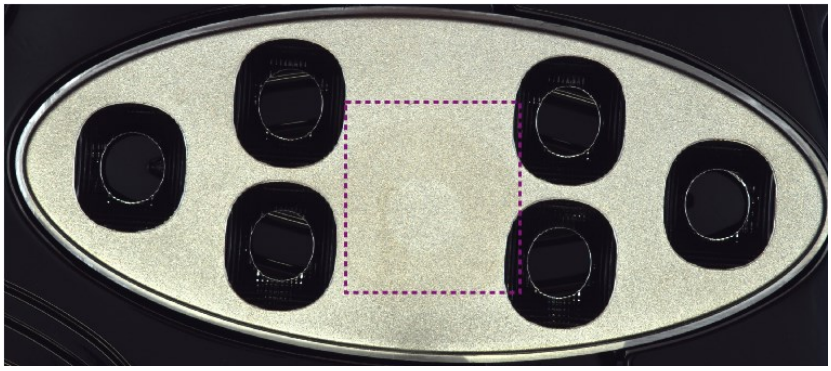
Figur 39. LandingLens objektdetektering.

Vid den optimala konfidensgränsen på 30 % hittar modellen defekter på 35/35 defekta bilder, men det är ändå vissa defekter som mindre prickar och små skrapmärken i mörkare bilder som inte märks ut. För att modellen ska märka ut alla defekter måste konfidensgränsen dras ner till så lågt som 17 %, men vid den här nivån märker också modellen ut många saker som inte är defekter, bland annat en vanlig märkning är en stor ruta mitt på reflektorn (figur 40).



Figur 40. LandingLens objektdetektering felmärkning.

Redan om man ökar konfidensgränsen till 32 %, börjar märkningar falla bort och vissa defekta reflektor blir nu utan märkningar. De första defekter som slutar märkas ut är de stora fettfläckarna som syns i figur 41. Det intressanta är att det här är en liknande märkning som i figur 40, det verkar som modellen har problem att förstå större defekter.



Figur 41. LandingLens objektdetektering fettfläck.

Vid konfidensgräns på 30 % märkte modellen ut defekter på 2/25 OK-bilder. Problemet här är att modellen har tendens att märka ut väldigt små prickar som i normala fall inte skulle klassificeras som en defekt (figur 42). Det här verkar vara mest på bilder med starkare belysning.

Figur 42. LandingLens liten prick.

8 Resultatsammanfattning

Resultaten för varje metod och program har redan kommit fram i tidigare kapitel, men här är en sammanfattning för att få bättre överblick över hur de presterade jämfört med varandra och vilka typer av defekter som var problematiska.

8.1 Bildklassificering

Resultaten för bildklassificering är ganska självklara som man kan se i tabell 1. Tabellen går ut efter den bäst presterande modellen för varje program. Clarifai och Sentisight AI presterade ungefär lika bra, Fastai presterade bättre, men LandingLens presterade överlägset bäst.

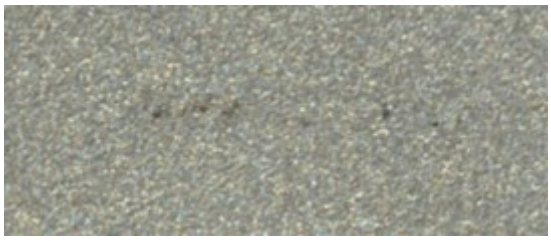
Tabell 1. Bildklassificering prestanda.

	Fastai	Clarifai	Sentisight AI	LandingLens
OK precision	85 %	77 %	79 %	96 %
NOK precision	94 %	85 %	83 %	100 %
OK återkallelse	92 %	80 %	76 %	100 %
NOK återkallelse	89 %	83 %	86 %	97 %
Totala noggrannhet	90 %	82 %	82 %	98 %

Vilka typer av defekter modellerna hade problem med är intressant. Clarifai, Sentisight AI och Fastai hade nästan exakt samma bilder felidentifierade, medan den enda bilden som LandingLens felidentifierade, fick alla de andra korrekt identifierat. Den bild som LandingLens felidentifierade syns i figur 43 och det är svårt att säga varför den här var ett problem. Den har flera tydliga prickar och många andra bilder som blev korrekt identifierade hade mycket otydligare och mindre prickar än den här.

Figur 43. Reflektor med prickar.

De bilder som Clarifai, Sentisight AI och Fastai felidentifierade kan bli bättre förklarade. De flesta av bilderna var mörkare bilder av reflektorer med små defekter, ett exempel syns i figur 44.



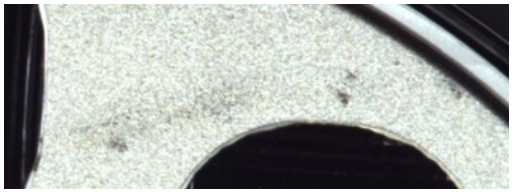
Figur 44. Reflector med defekt.

Två av bilderna hade väldigt tydliga defekter. Orsaken att en av de här blev felidentifierade kan bero på att den är en mera unika defekt som inte fanns i träningsdatat. Defekten i figur

45 har en väldigt tydlig repa men den är tjockare och har en mera kurvad form än andra repor.

Figur 45. Reflektor med tjock repa.

Defekten i figur 46 är också en svår att säga varför den blev felidentifierad eftersom liknande defekter definitiv finns i träningsdatat.



Figur 46. Reflektor med felidentifierad defekt.

8.2 Objektdetektering

Objektdetekteringen testades i två program: Sentisight AI och LandingLens. Sentisights objektdetektering presterade lite bättre än bildklassificeringen i samma program, men var fortfarande mycket sämre än både bildklassificering och objektdetektering i LandingLens. Sentisight hade också problemet att bilderna blev minskade, bildkvaliten blev väldigt låg och det var svårt att se vad som märktes ut på bilderna.

Största skillnaden jämfört med bildklassificeringen var prestandan på fettfläckar. I bildklassificeringen blev alla bilder med fettfläckar korrekt identifierade och alla modellerna var väldigt säkra på sina identifikationer. I objektdetekteringen försvann markeringen av fettfläckar nästan direkt man började höja på konfidensgränsen och sänkte man konfidensgränsen för lågt började det markera stora rutor mitt på reflektorerna där det inte fanns någon defekt. Däremot repor och skrapmärken hade inte

objektdetekteringen problem med, även de defekter bildklassificeringen hade problem med i figur 45 och 46, märkte objektdetekteringen bra ut.

I allmänhet presterade objektdetekteringen lite sämre än bildklassificeringen, men ifall man också vill kunna se var i bilden defekten är, är nog objektdetekteringen fortfarande ett bra alternativ. Att dela in defekterna i olika kategorier gjorde bara att modellen presterade sämre, det här kan bero på att det då var mycket mindre träningsdata per kategori och många defekter var också ganska otydliga till vilka kategorier de hörde.

9 Diskussion

Syftet med arbetet var att testa några olika program och tekniker för att få en uppfattning av vad som fungerar bättre och sämre, hur stor del av defekter som kan detekteras och vilka defekter som inte går att detektera. Av de program som användes kom det fram ganska tydligt att LandingLens presterade bäst, både i bildklassificering och objektdetektering.

Av metoderna som användes blev det klart väldigt tidigt att segmentering inte alls är en möjlighet för det här ändamålet. Objektdetektering och bildklassificering fungerade båda betydligt bättre, med bildklassificering i LandingLens kunde man nå upp till en 100 % noggrannhet på 60 testbilder. Det här är förstås ändå en ganska liten bildbank så i praktiken kan man inte förvänta sig att modellen presterar med 100 % noggrannhet, men det är ändå klart att klassificeringen kan utföras relativt pålitligt.

Vilka typer av defekter som kan detekteras är inte lika självklart eftersom alla defekter kunde detekteras av åtminstone någon modell. I stora drag hade objektdetekteringen problem med fettfläckar och många modeller hade problem med bilder som var mörkare och ljusare.

9.1 Förbättringar

Även om resultatet av arbetet var bra finns det en del saker man skulle kunna förbättra. I början av projektet när bilderna samlades in, togs de med varierande belysning. Tanken var att i praktiken kommer belysningen variera, bilderna togs med lite ljusare och mörkare belysningen än förväntat för att säkert få tillräckligt med variation i datamaterialet. Det här

gjorde dock vissa defekter svårare att se och modellerna hade också egna inbyggda dataaugmenteringar för att ändra ljusstyrkan som användes. Det skulle kanske varit bättre att ta alla bilder med identisk belysning eller använda sig av normalisering för att hålla defekterna mera tydliga.

Majoriteten av bilden tas upp av reflektorn och även fastän bakgrunden i teorin inte ska ändra skulle det kunna vara bra att filtrera ut bakgrunden för att minimera risken att modellen observerar irrelevanta aspekt i bilden.

Mera data är också en sak som troligen skulle förbättra resultaten. Det togs ca 200 bilder totalt och av de användes under 120 som träningsdata. Ju fler bilder man har desto flera olika defekter får man modellen att lära sig. Om man skulle få samlat ihop några 1000 bilder skulle man också kunna testa att träna en modell helt från grunden utan att använda överföringslärande.

9.2 Personliga åsikter

Jag tycker arbetet har varit väldigt intressant och lärorikt. Innan jag började med arbetet var den enda erfarenhet jag hade med AI lite Fastai som jag lekt med på fritiden. Nu har jag mycket mera kunskaper om både hur AI fungera och hur man ska träna en modell för att få bättre resultat. Jag tycker slutresultatet av projektet var väldigt bra och de här AI-programmen är definitivt en möjlighet att använda vid defektdetektering för Nordic Lights.

10 Källor

Abraham, A. (2005). Artificial neural networks. i *Handbook of measuring system design* (ss. 901-908). Stillwater, OK, United states of America: John Wiley & Sons, Ltd. doi:10.1002/0471497398

Andrew Ng. (u.å.). *About*. Hämtat från https://www.andrewng.org/about/ den 25 4 2023

Berchane, N., & Berchane, N. (2018). *Artificial Intelligence, Machine Learning, and Deep Learning: Same context, Different concepts.* M2 IESC. Hämtat från https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/

Clarifai. (u.å.). *About*. Hämtat från https://www.clarifai.com/company/about den 25 4 2023

Davies, E. R. (2012). *Computer and Machine Vision : Theory, Algorithms, Practicalities.* Elsevier Science & Technology.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press. Hämtat från https://www.deeplearningbook.org/

Howard, J. (2016). Launching fast.ai. *fast.ai*. Hämtat från https://www.fast.ai/posts/2016-10-07-fastai-launch.html

Kleinsmith, M. (2017). CNNs from different viewpoints. *ImpactAI*. Hämtat från https://medium.com/impactai/cnns-from-different-viewpoints-fab7f52d159c

Koppert-Anisimova, I. (2021). Cross-Entropy Loss in ML. *unpack*. Hämtat från https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0

Lowe, A., & Lawless, S. (2021). *Artificial Intelligence Foundations: Learning from experience.* (u.o.): BCS Learning & Development Limited.

Nordic Lights. (u.å.). *About Us*. Hämtat från https://www.nordiclights.com/about-us/ den 2 5 2023

Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Towards Data Science*. Hämtat från https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science, 2*(160). doi:https://doi.org/10.1007/s42979-021-00592-x

Sylvain, G., & Howard, J. (2020). *Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD.* O'Reilly Media, Inc. Hämtat från https://github.com/fastai/fastbook