Klassificering av skador på grusvägar med Maskininlärning

En undersökning av genomförbarheten och lämpligheten av att använda Computer Vision för att identifiera skador på grusvägar i syfte att automatisera skadeklassificeringsprocessen

Márk Mészáros

EC Utbildning

Examensarbete

2024-01-01 – 2024-02-09

Abstract

The quality and maintenance of gravel roads is crucial in the Scandinavian countries, and the current manual methods for quality control are very resource intensive. This thesis investigates the feasibility and suitability of using Computer Vision to identify damage on gravel roads in order to automate the damage classification process. The work carried out includes exploring the dataset and experimenting with different ways of preparing the data to find the best way to train and test a pre-trained ResNet50 model using Transfer Learning and Fine Tuning methods. The thesis also discusses the theory and methods used to solve this issue. The various challenges are also discussed in detail with particular focus on the importance of the data quality, the nature of the problem at hand and whether or not a full automation of the damage assessment process is feasible. The main conclusions show the sensitivity of the models to weather and light conditions and the importance of well-defined damage and non-damage categories.

Erkännanden

Först och främst vill jag tacka Tommy Nielsen och Nicklas Mattisson för deras arbete och engagemang i detta projekt, som skulle ha varit näst intill omöjligt utan deras bidrag.

Antonio Prgomet vägledde oss under projektet och under hela vår utbildning, vilket lade grunden för att uppnå de färdigheter och kunskaper som behövdes för detta projekt.

Tack till Harri Ahola för att han gav vår grupp förtroendet att utforska problematiken med att upptäcka skador på grusvägar.

Jag är tacksam gentemot min familj för deras oändliga tålamod och för att ge mig tid och utrymme att avsluta min utbildning.

Slutligen vill jag tacka EC utbildningar och alla våra lärare som gav oss en hög nivå och ett praktiskt tillvägagångssätt i utbildningen för att hjälpa oss att starta vår resa som Data Science.

Innehållsförteckning

1 Inledning

Det tidigaste gångstigarna av djur formade stigar som ibland gick över kontinenter och kopplade ihop essentiella resurser och fördelaktiga miljöer för överlevnaden. Mänskligheten var inget undantag. Människor har skapat stigar och senare vägar mellan resurser och deras hem, och senare mellan olika grupper av stammar och nationer.
Vägarnas betydelse ökade i takt med civilisationens utveckling och de spelade en avgörande roll för kulturell, ekonomisk och social utveckling och tillväxt. Det omfattande vägnätet under Kejsare Caesar Augustus tid bidrog i hög grad till det nybildade romerska imperiets uppkomst, fortlevnad och betydelse. Det är då när citatet "*Alla vägar leder till Rom*" föddes (Italianstudies 2011).

Grusvägar är fortfarande essentiella delar av trafikinfrastrukturen som kopplar ihop mindre orter, såsom agrikulturella-, skogs- och landsbygdsområden med industriella-, logistiska- och by orter. Grusvägarnas tillstånd påverkar i hög grad tillgängligheten och trafiksäkerheten av en ort. På grund av detta, ett visst lands trafikmyndighet kontrollerar regelbundet grusvägarnas kvalitet och skick, och försöker att upprätthålla en god vägkvalitet.

Traditionellt görs kvalitetskontrollen av grusvägar manuellt, genom att skicka ut bilar på vägarna och spela in videomaterial. Väglaget och skadorna kontrolleras och registreras manuellt av experter. Denna metod tar mycket tid och resurser. Med hjälp av den tekniska utvecklingen kom olika nya metoder för att kunna bedöma vägkvaliteten på ett mer effektivt sätt, såsom med vibrationsmätning, eller med laserteknik. Problemet med de ovannämnde metoderna är att utrustningen kan vara dyrt och även om de fångar olika aspekter av vägskador, kan de inte ge en helhetsbild av typ och allvarlighetsgrad av skadorna.

Med den uppåtgående populariteten för AI, har olika forskare försökt att ytterligare effektivisera och förbättra grusväg-analys med hjälp av Computer Vision och AI modeller. I detta dokument kommer jag att introducera vårt försök att implementera AI för att upptäcka skador på grusvägar i Finland.

1.1 Syfte och Frågeställning

Ramboll är ett konsultföretag inom teknik, infrastruktur och samhällsbyggnad, och får uppdrag i hela världen. Ramboll RST Finland fick uppdraget av Finska Trafikverket för att kartlägga grusvägar och identifiera skador på vägytan. Kunden har också frågat om möjligheterna att implementera Machine Learning i detta projekt och är intresserad av att se om identifiering av skador kan göras med Machine Learning och vilket typ av resultat det kommer att ge.

Syftet med detta projekt är att identifiera skador på vägarna med hjälp av en maskininlärningsmodell. Kunden vill nå en noggrannhet på 85% på förutsägelserna.

Frågeställning från uppdragsgivaren: Kan man skapa en Computer Vision (CV) modell som kan med 85% noggrannhet klassificera bilder på grusvägar till skadad – icke-skadade klasser?

2 Teori

För att kunna svara på frågeställningen, har vi använt oss av en förtränad modell, gjort Transfer Learning och Fine Tuning på den med datan vi fick från uppdragsgivaren och till sist utvärderade vi resultatet. I följande kapitel beskriver jag alla koncept som är relevanta till vår modell och metod.

2.1 Neurala Nätverk (NN)

När vi pratar om *NN* i IT sammanhanget, så pratar vi om *Artificiella Neurala Nätverk* (*ANN*) (s. 279 - Géron). *ANN:s* är inspirerade av hjärnans arkitektur och har syftet att lösa komplexa problem och ta intelligenta beslut baserat på existerande data. 'Magin' med Neurala Nätverket är att de behöver input (**x**) och output (**y**), och reglerna om hur man får en viss output av vissa inputs hittas och kalkyleras av nätverket (se Figur 1.) (s. 4 - Géron). En modell får nästan alltid flera inputs, som beskrivs som vektor $\vec{X}$.

2.2 Djupa Neurala Nätverk (DNN)

Som namnet 'nätverk' i *NN* indikerar, en *NN* byggs av många sammanlänkade *neuroner* som oftast är arrangerade i flera lager. Med sammankopplingen av många lager av neuroner, extremt komplexa mönstrar och problem kan lösas. Denna samling av neuronerna organiserad i flera lagrar kallas för *Djupa Neurala Nätverk (s. 289 – Géron)*. Den generella arkitekturen av ett *DNN* består av det första lagret som kallas för Input lager, sista lagret som kallas för Output lager, och alla mellanliggande lager som tillsammans kallas för Dolda lagrar. (se Figur 2.)

2.2.1 Neuroner (units)

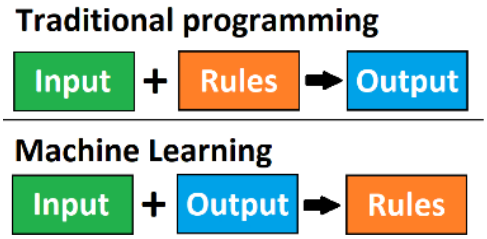
De grundläggande byggstenarna i ett ANN är neuroner som även kallas för units (s. 284 – Géron). I grund och botten tar en neuron en input, gör någon matematiskt kalkulation med den och skickar den vidare som output. Matematiska kalkylationen innebär att input datan multipliceras med en *vikt*, och en så kallad *bias* läggs till denna produkt. Efter får vi summan, den plockas in i en *aktiveringsfunktion* vilket avgör om outputen skickas vidare eller inte. I teorin, finns det en optimal kombination av *vikterna* och *bias* till alla *neuroner* av nätverket som resulterar i det önskade värdet. (se Figur 3.)

2.2.1.1 Vikt (w)

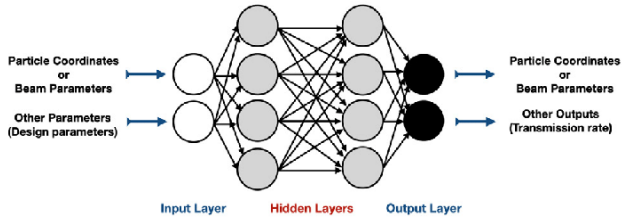
V*ikten* är en skalär som multipliceras med *neuronens* input och resulterar i en produkt. I början av en modellträning är *vikterna* slumpmässiga, och justeras under träningen med hjälp av *backpropagation* för att få det önskade värdet. Varje input **x** i en modell får en *vikt* **w**, och eftersom modellen får en vektor $\vec{X}$, så har modellen en vektor av vikter $\vec{W}$ (s. 286 – Géron).

2.2.1.2 Bias (b)

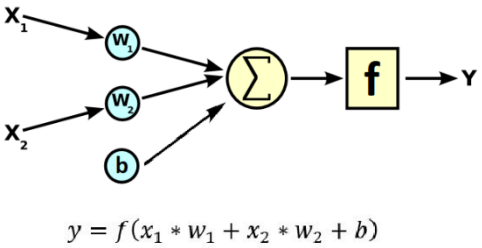
Bias är också en skalär som adderas till summan av produkter av *vikter* och inputs. Varje lager i en *NN* har bara en *bias*. *Bias* hjälper oss att kompensera resultatet med att skifta *aktiveringsfunktionen* till negativ eller positiv område. Detta gör att funktionen justeras och kommer att ge ett finjusterat resultat baserad på vikt/input produkten. *Bias* uppdateras och justeras också vid *backpropagation*. Se effekten av *bias* på Figur 4 (s. 286 – Géron).



Figur 1.Skillnad mellan Traditionell programmering och Maskininlärning



Figur 2. Basarkitektur av DNN (Researchgate 2021)



$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

Figur 3. Basarkitektur av en Neuron med matematiska beräkningens formel inom Neuronen

2.2.1.3 *Aktiveringsfunktion*

En funktion som tar in summan av $\overrightarrow{\boldsymbol{XW}} + \boldsymbol{b}$ och beräknar neuronens output. A*ktiveringsfunktionen* beter sig som en begränsare eller grindvakt som bestämmer vad outputen av *neuronen* blir. Den bestämmer om en neuron ska aktiveras och skicka datan vidare till de kommande *neuroner* eller inte. Det finns flera aktiveringsfunktioner som begränsar neuronernas output på olika sätt. Icke-linjära aktiveringsfunktioner tillämpar modeller att lära sig komplexa funktioner (Machinelearningmastery, 2021).

2.2.1.3.1 Rectified Linear Unit (ReLU)

ReLU är den mest vanliga *aktiveringsfunktion* inom *djupinlärning.* Dess popularitet ligger i egenskapen att det är mycket enkelt och snabbt att kalkylera och att den har inget maximumvärde. Formeln till *ReLU* (Se Ekvation 1.) gör att om ***z*** är större än 0, ***z*** returneras, annars returneras värdet 0. Se Figur 5. för en visuell representation av ReLU aktiveringsfunktionen på en graf (s. 292 – Géron).

$$ReLU(z) = max(0, z)$$

Ekvation 1. Formel till ReLU aktiveringsfunktion

2.2.1.3.2 Sigmoid / Logistisk aktiveringsfunktion

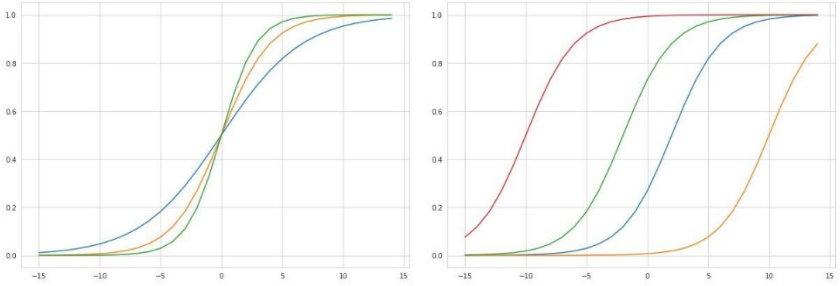
Sigmoid *aktiveringsfunktion* används oftast i output lagern i binära klassifikationsmodeller med bara två kategorier. Dess viktigaste egenskap är att den bara kan mata ut värden mellan 0 och 1, vilket gör den perfekt för att mata ut sannolikheter, och den beräknar sannolikheten om en instans tillhör till klass 1 (s. 143 – Géron).

$$\sigma(t) = \frac{1}{1+exp(-t)}$$

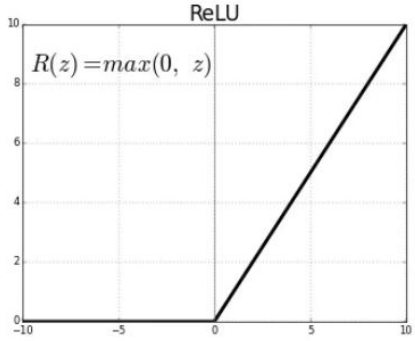
Ekvation 2. Formel till Sigmoid aktiveringsfunktion

2.2.2 Forward pass

Forward pass är när vi matar in input data till vår modell och datan går genom alla beräkningar i alla *neuroner*. Input datan multipliceras med *vikterna*, produkternas summa summeras med *bias* och detta värde går in i *aktiveringsfunktionen.* Denna output summa matas in i nästa lager tills vi får ett slutgiltigt output resultat. Eftersom *vikterna* och *bias* är initialiserade slumpmässigt, blir resultatet



Figur 4. Effekten av bias visualiserat på en logistisk funktion Vänster: konstant bias, varierande input-vikt produkt. Höger: varierande bias, konstant input-vikt produkt (Turing)



Figur 5. ReLU aktiveringsfunktion på en graf (Turing)

också slumpmässigt. Men vår fördel är att vi har båda inputs, $\vec{X}$ och outputs, **y** från början, och nu kan vi jämföra modellens resultat med verkliga (eller önskade) resultatet (s. 290 – Géron).

2.2.2.1 Loss

Skillnaden mellan modellens resultat med verkliga resultatet kallas för *loss*. För att vår modell på bästa sätt ska modellera, dvs. beskriva verkligheten med en funktion, strävar vi efter att minimera *loss*. Beroende på vilket problem man vill lösa, kan man använda sig av olika *loss* funktioner. (Machinelearningmastery, oktober 23, 2019).

2.2.2.1.1 Binary cross-entropy (log-loss)

Binary cross-entropy passade bra för vårt binära klassifikationsproblem (skada → 1 / icke skada → 0). Logaritmiska funktionen gör att om prediktionen på en träningsinstans är fel, så blir cost-värdet mycket högt, men om prediktionen är rätt så blir cost-värdet mycket lågt (s. 144 - Géron 2019). *Binary cross-entropy* räknar ut en total l*oss* för träningen som görs på många träningsinstanser, så det blir lätt att få en enda siffra som indikerar modellens loss. Se Ekvation 3. för formel till log-loss.

$$J(\theta) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

Ekvation 3. Log-loss funktion

2.2.3 Backpropagation

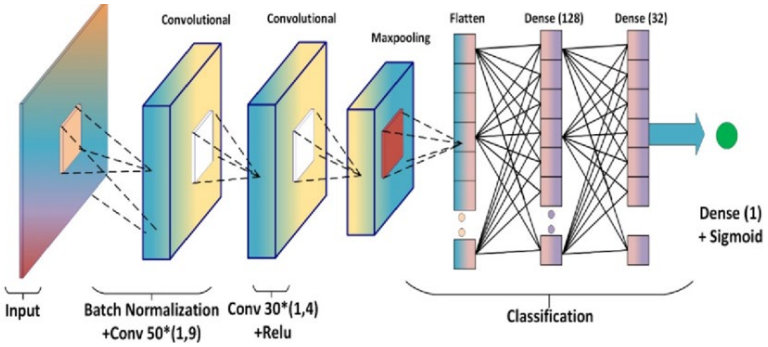
Som namnet indikerar, *backpropagation* börjar bakifrån i modellen och går till början av modellen. Den kollar *loss* och kalkylerar gradienten av nätverkets fel med avsikt på modellens alla parametrar (s. 290 - Géron 2019). Därefter görs en vanlig Gradient Descent steg som justerar *vikterna* och *bias* och modellen börjar från början igen, men nu med uppdaterade *vikter* och *bias* som kommer att ge ett nytt resultat som förhoppningsvis är närmare det sanna förväntade output värdet.

2.3 Convolutional Neuralt Nätverk (CNN)

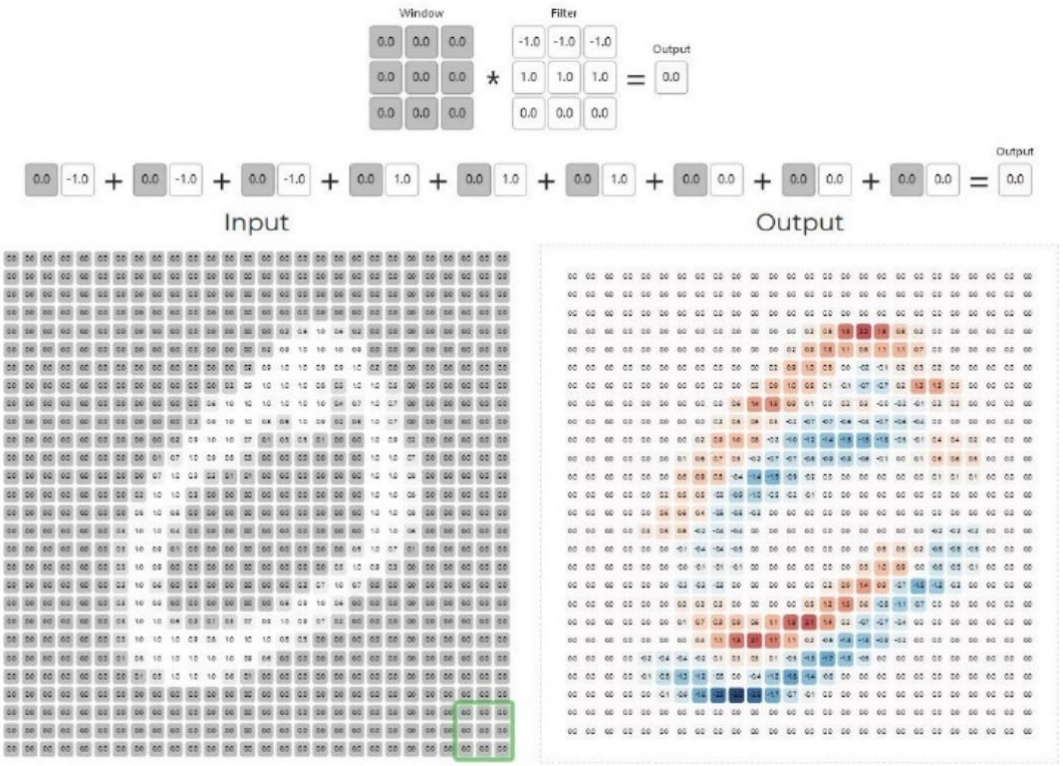
Convolutional Neurala Nätverk (CNN) är en *NN* som används mest för att lösa CV uppgifter (s. 445 – Géron). Ett CNN består oftast av flera *Convolutional lager* som tar in 2- eller 3-dimensionella tabeller av bilddatan, kollar genom pixlarna och försöker att hitta olika former, figurer och mönster i det (Se Figur 6.).

2.3.1 Convolutional lager

En *Convolutional lager* kommer att kolla på en n*n ruta på bilden, och skapar olika filter. Ett filter är en n*n ruta med olika värde i alla smårutor, som fungerar som koefficient till pixlarnas värde, och baserat på värdena i smårutorna, filtret fångar olika mönster i bilden såsom horisontala, vertikala linjer, kurvor osv. (Se Figur 7.). Därefter flyttas rutan, tills vi har gått genom hela bilden. Omfattningen av hur mycket rutan flyttas kallas för stride. Antalet och storleken på filtret vi vill ha samt hur stor striden ska vara kan vi styra och är därför i princip en hyperparameter (s. 448 – Géron).



Figur 6. Exempel av en Convolutional Neuralt Nätverk (Researchgate 2023)



Figur 7. Exempel på ett filter i en Convolutional lager (Deeplizard)

2.3.2 Pooling lager

Efter Convolutional lager har vi ett pooling lager, som ”komprimerar” bilden baserad på max-, eller medelvärde av en m*m ruta, som är resultatet av ett Convolutional lager. Vi får ju en mindre bild, men flera bilder med applicerade filter (s. 456 – Géron).

2.3.3 Flatten lager

Till slut måste man skapa en lång vektor av alla filtrerade pixelvärden som är i 2 dimensionella matrix format. Detta görs med hjälp av en Flatten lager.

2.3.4 Dense lager

Dense – fullt uppkopplat – lager är ett enkelt lager av *neuroner* där alla neuroner får input från alla neuroner i den föregående lager (s. 285 – Géron).

2.3.5 BatchNormalization lager (BN)

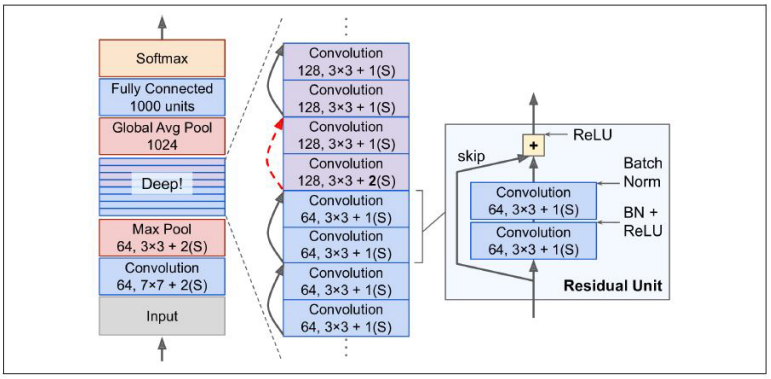
Batch-normalization standardiserar input data (medelvärde centrerat runt 0 och standardavvikelse är 1) baserat på de batch-data den får som input. *BN* används precis före eller efter en aktiveringsfunktion i varje dolda lager. Denna metod stabiliserar inlärningsprocessen och minskar antalet epoker som behövs genom att ta itu med problemet med "intern covariate shift", vilket är förändringen av inputs och kan drastiskt sakta ner och destabilisera inlärningsprocessen (Machinelearningmastery, december 4, 2019).

2.4 ResNet50

ResNet50 är ett *CNN* som var introducerad i en publikation i 2015 (Arxiv 2015). ResNet står för Residual Network och är baserad på residual learning tekniken. ResNet50 modellen består av 50 lager totalt (Se figur 8.), och lagren är organiserade i residual blocks (Se figur 9.). En residual block

layer name	output size	50-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$
	1×1	average pool, 1000-d fc, softmax
FLOPs		3.8×10^9

Figur 8. ResNet50 modellarkitektur



Figur 9. ResNet50 modellarkitektur med Residual Blocks och Residual Units (s. 473 - Géron 2019)

innehåller flera residual units. En residual unit består av flera Convolutional lager, *Batch Normalization* och *ReLU* aktivation, och använder skip connections, som gör att en residual units input läggs till i units outputen direkt, och summeras med resultatet som gick genom kalkylationerna i residual units lagren. Detta gör att även om ett lager har inte ens startat inlärningen (dvs. att det har inga uppdaterade vikter), det kommer att få värdefull signal – i form av vikter - från föregående lager. Effekten av detta gör att nätverket tvingas modellera en delvariant av modellen i stället för hela modellen. Om input till en residual unit är **x**, och output är h(**x**), så är h(x) = f(**x**) + **x**. +**x** delen kommer från skip connection och f(**x**) är värdet vi får när input **x** går genom kalkylationerna i Convolutional lagren i residual unit. Det innebär också att f(**x**) = h(**x**) – **x** (Se Figur 10.), vilket betyder att lagren i en residual unit måste lära sig att modellera en residual variation av f(**x**) funktionen, och denna teknik kallas därför för residual learning (s. 471 – Géron).

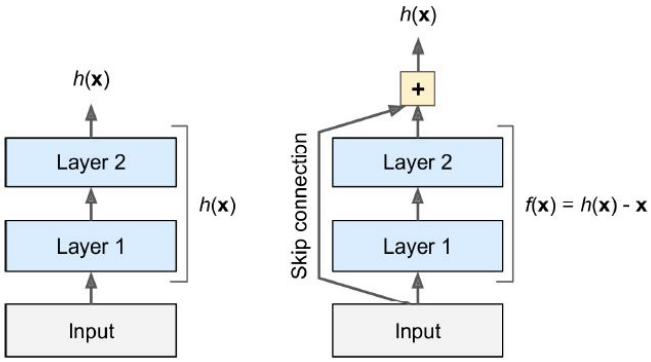
2.5 Transfer Learning och Fine-Tuning

Transfer Learning och Fine-Tuning är båda tekniker för att justera förtränade modeller till olika användningsområde, men dom har skillnader som ska adresseras (TensorFlow 2023).

2.5.1 Transfer Learning (TL)

Användning av förtränade modeller inom Computer Vision är rätt vanligt, speciellt om man har få träningsinstanser av den nya klassen. I stället för att man bygger och tränar en modell från grunden, kan man använda en redan befintlig och tränad modell som en kunskapsbas och träna upp den för att lösa ett liknande problem.

Exempel: en katt-hund klassificerare kan tränas om att bli en lodjur-varg klassificerare.



Figur 10. Exempel till Residual Learning (s. 471 - Géron 2019)

TL betyder rent tekniskt att man använder modellen bara som en feature-extraktor, dvs. att det tar ut dom olika formerna från bilderna men predikterar ingenting än. Det är vi som ska lägga till ett (eller flera) Dense lager till denna extraktor som kommer att göra den faktiska prediktionen.
Detta kan uppnås genom att ”frysa” in pretränade modellens Convolutionala lager (som har lärt sig dom olika features), ta bort eventuella sista lagret som Flatten och Dense lagret, och byta ut dom med vårt eget – än så länge inte tränade men träningsbara – Dense lager som har rätt antal outputs och träna modellen på dess nya data. Med detta blir bara vikterna till Dense lagrar uppdateras och göra prediktioner till slut.

2.5.2 Fine-Tuning (FT)

Som namnet indikerar *FT* används för att maximera prestandan av en förtränad modell. *FT* kan ske med att träna modellen med helt nya instanser av önskade klassen eller träna på gamla instanser som modellen har sett förut.

Vid FT fryser man inte hela förtränade modellen bara första delen av den, och lämnar det översta lagrar träningsbart. Det gör att översta lagrar som har lärt sig hög-nivå features från den ursprungliga dataset - vilket de var tränat på - tränas om och justeras och nu kan dessa lagrar lära sig mer specifika hög-nivå features av vårt dataset. Med detta sätt uppdateras vikterna på det översta lagret. Vid FT använder man vanligtvis en lägre learning-rate så att de redan befintliga vikterna inte uppdateras drastiskt, utan endast finjusteras – därav namnet Fine-Tuning.

2.6 Metriker

För att kunna utvärdera prestationen av vår klassificerare, har vi använt oss av flera metriker såsom accuracy, precision, recall, confusion matrix och classification report. Alla de nämnda metrikerna används för klassifikations problem.

2.6.1 Accuracy

Accuracy är metriken som visar oss hur stor andel av klassprediktionerna som var rätt.

2.6.2 Confusion matrix

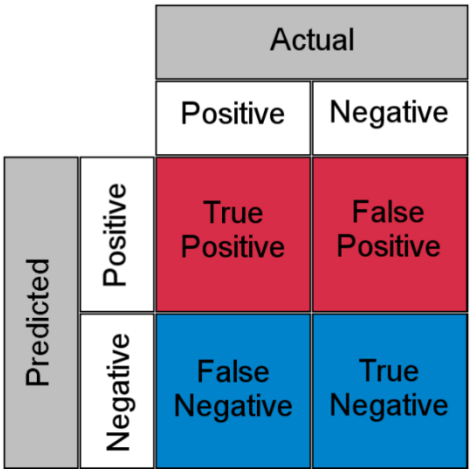
Confusion matrix är ett mycket kraftfullt sätt att visualisera klassifikationsmodellens prestanda genom att kvantifiera och visualisera predikterade klasser mot verkliga klasser i datasetet (Se Figur 11.). Eftersom vårt problem är binär klassificering, blir confusion matrixen en 2x2 matris, med fyra möjliga utkomster:

- True Positive (TP): Vi har predikterat positiva klassen, och vi hade rätt
- False Positive (FP): Vi har predikterat positiva klassen, men vi hade fel
- False Negative (FN): Vi har predikterat negativa klassen, och vi hade rätt
- True Negative (TN): Vi har predikterat negativa klassen, men vi hade fel

Ju flera prediktioner vi har på TP- och TN-diagonalen, desto bättre presterar modellen. Confusion matrix kan också visa det tydligt om vår modell tenderar att prediktera en klass över den andra. (Scikit-Learn API dokumentation)

2.6.3 Precision / Recall

Precision och recall är baserade på TP, TN, FP och FN (Se Figur 12.) (Scikit-Learn API dokumentation).
Precision svarar på: Från våra positiva prediktioner, hur stor andel var verkligen positiva.
Recall svarar på: Från alla verkliga positiva instanser, hur stor andel har vi predikterat som positiva.



Figur 11. Exempel på en Confusion matrix (Wikipedia 2019)

$$\text{precision} = \frac{TP}{TP + FP}$$

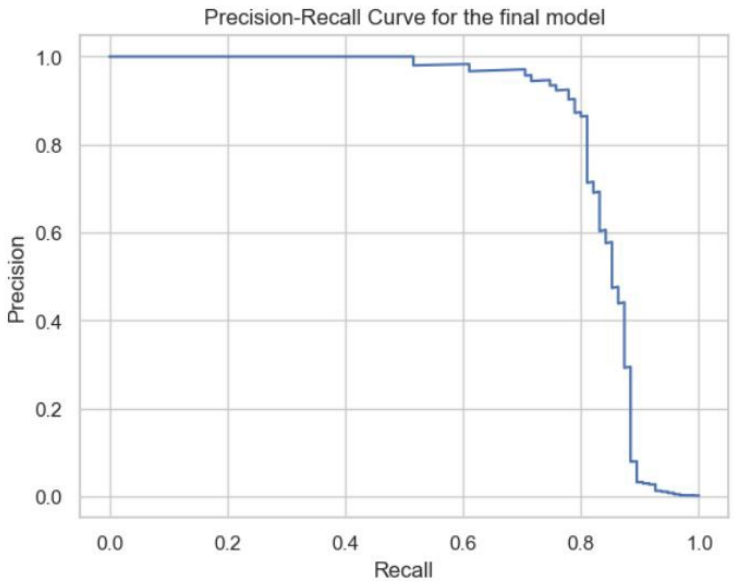
$$\text{recall} = \frac{TP}{TP + FN}$$

Figur 12. Formel till Precision och Recall

Precision och recall har ett samspel mellan varandra som manifesteras bäst i precision-recall kurvan. En klassifikationsmodell som inte predikterar klasserna direkt, men den predikterar sannolikheten av att en instans tillhör till en viss klass. Sannolikhetsgränsen av att en instans tillhör till klass 1 är som standard 0.5, dvs. 50% vid binära klassifikation. Eftersom vi får ut sannolikheter som prediktioner, kan vi också bestämma själv vart sannolikhetsgränsen ska ligga. Om vi låt säga ökar gränsen till 0.6, dvs. att modellen ska prediktera 60% sannolikhet för att en instans tillhör till klass 1, kommer detta naturligtvis påverka TP, TN, FP och FN, som i sin tur kommer att påverka precision och recall (Se Figur 13.).

2.6.4 Classification report

Classification report är en funktion i Scikit-Learn modulen som ger oss en samlad sammanfattning om prestandan av en klassifikationsmodell. Classification report innehåller de viktigaste metrikerna (precision, recall) uppdelade per klass och modellens accuracy också (Scikit-Learn API dokumentation).

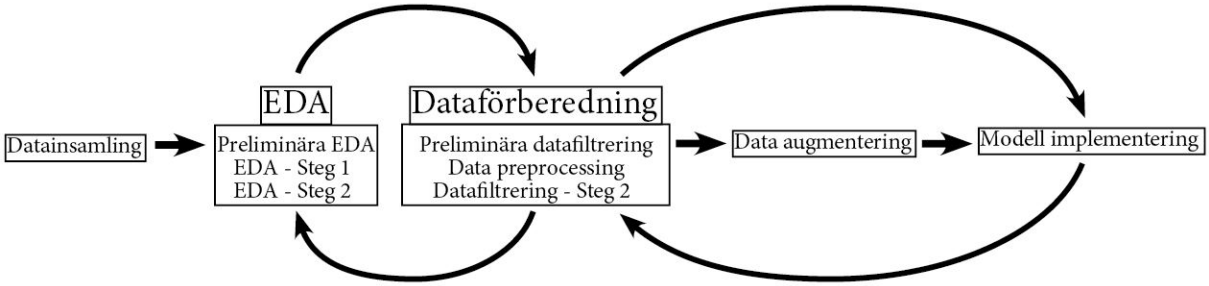


Figur 13. Precision-Recall curve

3 Metod

Bild och annoteringsdatan som vi fick från Ramboll behövde undersökas och filtreras flera gånger. Rensade datan processerades och matades in i ett DNN som hade den förtränade ResNet50 modellen som grund till Transfer Learning och Fine Tuning.

Vår process under projektets lopp var iterativt, och vi gick flertalet gånger tillbaka till ett tidigare steg i processen med bakgrund i ny information som vi fick i ett senare steg. Se Figur 14. för att få en översyn om alla steg i hela processen.



Figur 14. Projektets arbetsflöde diagram

3.1 Verktyg

I projektet har vi använt oss flera verktyg för att tillgodose våra behov för arbetet. Vi har använt oss av Google Drive för att spara våra dataset och Google Colab för att skapa notebook filer och köra modellträningar på deras gratis GPUs. Vi har använt GitHub för att versionshantera vårt kodbas och Weights and Biases för att logga våra träningsresultat. I senare steg har jag använt VSCode för att skapa och skriva kod till notebooks och för att köra modellträningar lokalt på min laptop.

3.2 Datainsamling

Datan för projektet fick vi direkt från Ramboll RST Finland i form av bilder och en csv-fil med annoteringar.

3.2.1 Bilder

Bilderna var insamlade manuellt av 6 olika bilar som körde 5–6 gånger på samma sträcka från början av oktober till början av december på totalt 27258 km lång sträcka av grusvägar i hela Finland. Bilderna var tagna av GoPro kameror som var fästa på vindrutan från insidan och positionerad så att vägen syns så bra som möjligt. Positionen av kameran per bil var konstant under hela insamlingsprocessen. Bildernas filnamn innehåller information om vilken körning (run) den är tagen ifrån, på vilken vägsträcka, tidpunkt och koordinat. Bilderna var tagna ungefär varje 20e meter.

3.2.2 Annoteringar

Annoteringar var tagna av operatören som körde bilen. Operatören har använt en applikation som heter V-tracker som kan registrera GPS koordinat och tidpunkt när knappen är intryckt.
Bild-, och annoteringsdatan är synkroniserad och upplagt som Excel filer med flera kolumner som bildens filnamn, run, vägnummer, koordinat, tidpunkt och skadans start och slutdistans.

3.3 EDA

EDA på datasetet var en iterativ process i 3 steg:

- **Preliminära EDA**: i denna steg har vi gjort EDA direkt på datasetet vi fick från Ramboll med syfte på att få en initial känsla för datan.
- **EDA - Steg 1**: baserad på preliminära EDA, har vi filtrerat datasetet. Eftersom datasetet blev förändrat har vi valt oss att göra ännu en kort EDA på det filtrerade datasetet.

- **EDA - Steg 2**: baserad på modellresultatet på den filtrerade datan samt en grundlig undersökning av publikationen Gravel road classification based on loose gravel using transfer learning av Nausheen et al. [2022], har vi bestämt oss att ytterligare filtrera datasetet, och göra ytterligare en EDA på det nyligen filtrerade datasetet.

3.3.1 Preliminära EDA

Efter en snabb genomgång av bilderna, har vi upptäckt flera utmaningar med datan (Se Figur 15.).

- **Bildernas upplösning**: Originalbilderna hade en stor upplösning på 5184 x 3888 pixlar, och innehöll mycket mer än bara själva vägen, som vi inte var nyfikna på i detta projekt. (Se Figur 16.) Detta samt att vi planerade att använda en förtränad modell som förväntade input bilder med 224 x 224 storlek gjorde att vi måste hitta en lösning på bildstorleken som jag skriver mer detaljerat om i kapitel 3.4.2 – Data Preprocessing.
- **Hinder framför kameran**: för att kunna leverera högkvalitativa bilder, var det viktigt att vindrutan var så torr som möjligt. Operatörerna har satt på vindrutetorkare vid regn, som resulterade i att på vissa bilder har vindrutetorkare som hindrade eller dolde utsikten på vägen.
- **Bilder i mörker / dimma**: en del av bilderna var tagna i mörkret eller i dimma, som minskade synligheten och hindrade upptäckten av vägskador både för operatörerna och för personer som ska kontrollera datakvalitén.
- **Bilder i soligt väder med skugga**: en del av bilderna var tagna i soligt väder som resulterade i bilder med mycket skugga på dem. Effekten av detta kommer jag att skriva mer detaljerat om i Kapitel 4.1 – Förklaring av resultat. Soliga bilder har också resulterat i reflektioner av bilens insida på vindrutan, vilket vi har noterat men inte åtgärdat.
- **Bilder i kurva**: vissa av bilderna var tagna i en kurva vilket resulterade att majoriteten av vägen inte fanns med på bilden.
- **Bilder på asfaltväg**: en del av bilderna var tagna på asfaltvägar vilket inte tillhörde till vår målgrupp av grusvägar.
- **Kameravinkel**: datainsamlingen gjordes av 6 olika bilmodeller, samt att kameror var monterad något annorlunda i varje bil vilket resulterade i att bildernas perspektiv var icke-uniforma, och vissa bilar hade mer motorhuv i bildramen än andra.
- **Bildernas annoteringar**: eftersom annoteringen gjordes manuellt av operatörerna, var annoteringarna utsatta för mänskliga fel. Förutom detta, även om operatören registrerade skadorna precist, bilderna var tagna av kameran varje 20e meter, vilket betyder att skadan kan vara långt borta, nära eller kanske redan under bilen när närmaste bilden var tagen.

Figur 15. Exempel på utmanande bilder – Skugga, mörker, torkarblad, mycket ljus, asfaltväg

Figur 16. Obearbetad bild vs. bild med intresse

Figur 17. Tvetydiga bilder på skada / icke-skada

- **Definition av vägskador**: även om annoteringar var gjorda av domänkunniga experter, vad som räknas som vägskada eller inte är fortfarande otydligt i vissa fall. (Se Figur 17.)

3.3.2 EDA - Steg 1

Under projektets lopp har operatörerna kört 4 av de planerade 6 körningarna, och vi fick därför data från run 1 till run 4, vilket är 31488 bilder totalt. En snabb analys av datan visade att run 2 och run 3 hade bara fåtal bilder med skada, och vi har därför valt att fokusera på run 1 och run 4 i stället vilket hade en vettigare distribution av bilder med skada och icke-skada. Detta resulterade i 21127 bilder, vilket vi har filtrerat enligt kapitel 3.4.1 - Preliminära datafiltrering. Efter filtreringen hade vi 15559 bilder som vi har använt till modellträningen (Se Figur 18.).
Även om vi nu hade färre bilder än i början, hade bilderna en högre kvalité, samt datan var mer balanserad.

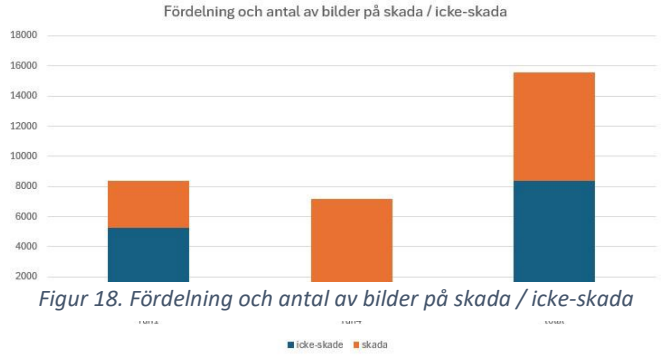
3.3.3 EDA – Steg 2

Datasetet efter Steg 2 Datafiltrering innehöll inga bilder med mycket sol eller skuggor. Bilderna var tagna i molnigt och blött väder med dämpat ljus och datasetet blev dessutom mer homogen. I detta steg har vi även skapat ett helt balanserat dataset för träning, validation och test.

3.4 Data förberedning

Dataförberedningen var baserad på preliminära EDA, ResNet50 modellens förväntade dataformat enligt Keras API dokumentation och resultatet av modellens resultat i Steg 1. Dataförberedningens olika steg:

- **Preliminär datafiltrering**: detta steg gjordes efter den preliminära EDAn med syfte att filtrera bort bilder med låg kvalitét och hinder
- **Data preprocessing**: förberedning av bilderna till ResNet50 kompatibelt format
- **Datafiltrering – Steg 2**: baserad på modellresultatet i Steg 1 och information i publikationen av Nausheen et al. [2022], har vi bestämt oss att ytterligare filtrera datasetet.



Figur 18. Fördelning och antal av bilder på skada / icke-skada

3.4.1 Preliminär datafiltrering

Den preliminära EDAn, *filtreringen* och *justeringen* av datan gjordes för att öka datakvalitén. Efter den preliminära filtreringen av datan hade vi 15559 bilder att jobba med. I detta steg har vi tagit hand om det följande problem:

- **Hinder framför kameran**: bilderna där vindrutetorkaren blockerar vyn togs bort.
- **Bilder i mörker / dimma**: bilderna tagna i mörker och dimma togs bort
- **Bilderna på asfaltväg**: bilderna tagna på asfaltvägar togs bort
- **Bildernas annoteringar**: annoteringar var kontrollerade ytterligare av domänexperter för att säkerställa att annoteringarna är rätt och att dom är i formatet som vi hade önskat för: bildens namn och skada/icke skada kolumner.

Upptäckter på datan, såsom bildernas upplösning, kameravinkel har vi behandlat i senare dataförberednings steg som jag kommer att skriva mer om i kapitel 3.4.2 – Data Preprocessing.

Resten av upptäckterna på datan, som bilder i soligt väder med skugga, bilder i kurvor och definition av vägskador har vi lämnat i dataseten, men påverkan av dem kommer jag att skriva om i Kapitel 4.1 – Förklaring av resultat.

3.4.1.1 *Train-Validation-Test dataset*

Efter den preliminära datafiltreringen, har vi delat upp hela datasetet (full_ds) i Tränings och Test dataset med 80–20 fördelning. Validation datasetet skapades under datainladdningen och var 20% av Tränings dataset. (Se Tabell 1.)

Total antal bilder	Train	Validation	Test
15559	9957	2489	3113

Tabell 1. Train-Validation-Test fördelning för full_ds

3.4.2 Data preprocessing

Innan vi ska mata in data till modellen i Steg 1, har vi delat upp bilderna i train, validation och test dataset och har förberett bilderna så att dom är i rätt format. Eftersom ResNet50 modellen var tränad på ImageNet datasetet, som standard förväntar sig ResNet50 modellen bilder i 224 x 224 x 3 format enligt Keras API dokumentation. Bilderna ska dessutom vara i BGR format med pixelvärden som är centrerade vid noll. Förutom detta, kamerors synvinkel har skiljt sig signifikant mellan de olika bilar och detta behövde åtgärdas under dataförberedningen.

3.4.2.1 Kameravinkel och bildernas upplösning

Problemet med kameravinkeln resulterade i att med vissa bilar syntes motorhuven mycket mer än med dom andra bilarna (Se Figur 19.). Förutom detta, bildernas synvinkel innehöll inte bara vägen utan omgivningen, landskapet och allt över horisonten. Detta betyder att bilderna innehöll information som inte är viktigt för vårt problem men resulterar i mycket mer data och grus i datan. För att eliminera detta problem, har vi valt att fokusera bara på de viktiga delarna av bilderna.

Detta har vi gjort med att croppa bilderna, så bilderna innehöll bara vägen och inget annat (Se Figur 20.). Vi har hittat 4 olika sätt att croppa bilderna:

Figur 19. Bilder på olika bilar var motorhuven syns

- **'original_crop'**: vi har valt placering av en kvadrat som innehöll vägen, och har använt samma placering för alla bilar.
- **'square'**: vi har anpassat placering av kvadraten per varje individuella bil. Beroende på vilken bil som tog bilden, har vi använt individuell placering av kvadraten.
- **'padded'**: i stället för en kvadrat som inte har täckt hela vägytan, har vi croppat en rektangel som hat täckt hela bredden av vägytan. Eftersom vi ville ha en kvadrat i slutänden, så har vi använt oss av padding för att få ut kvadratiska bilder.
- **'stretched'**: detta sätt byggs på 'padded' croppningen, men i stället för att använda paddig, har vi stretchat bilderna till kvadratisk form.

Efter varje croppningssätt, har vi ändrat storleken på bilderna till 224 x 224 pixelformat, så bilderna är kompatibla med det förväntade pixelformatet av ResNet50 modellen.

3.4.2.2 Preprocessning av bilder

Centrering och BGR formatering skedde med hjälp av preprocess_input() funktionen från tensorflow.keras.applications.resnet50 modul. Preprocess_input() är en inbyggd funktion för att förbereda datan till det förväntade format för ResNet50 modellen. Värt att notera är att funktionen centrerar data i caffe format, dvs. att pixelvärderna är centrerade vid noll, men inte skalade.

3.4.2.3 Preprocessning av annoteringar

Preliminära preprocessning av annoteringar med hjälp av Pandas modulen i Python resulterade i en Excel-fil som innehöll två kolumner: bildfilens namn och skada, dvs. om någon skada är synlig på respektive bild eller inte. Bilder med synliga skador fick värde 1, bilder utan skador fick värde 0.

3.4.3 Datafiltrering - Steg 2

Modellens resultat i Steg 1 var signifikant lägre än vi har förväntat oss jämfört med publikationen av Nausheen et al. [2022] där deras resultat låg över 92% accuracy. Efter en grundlig genomgång av nämnde publikation kombinerat med publikationen av Nienaber et al [2015], kom vi fram till slutsatsen att vårt dataset är betydligt mer varierat jämfört med dataset i publikationerna. Bilderna i publikationerna var tagna bara i ljust och soligt väder och var mycket homogena i kvalitét, medan vårt dataset innehöll bilder som var tagna i soligt och regnigt väder och bilderna hade varierande kvalitét. Detta, samt att gränsen för vad som ansågs vara en skada eller inte fortfarande var mycket luddiga gjorde att vi har ytterligare filtrerat vårt dataset och har skapat ett "best-of" dataset (Se Figur 21.). Best-of dataset innehåller 1000 bilder totalt. Bilderna var tagna i liknande väder- och ljusförhållande och eventuella skadorna var mycket tydliga.

Figur 20. Bilder på olika cropping / bildbeskärnings metoder

Figur 21. Bilder från best_ds

3.4.3.1 *Train-Validation-Test dataset – Steg 2*

Efter Datafiltreringen i Steg 2, har vi delat upp det helt balanserade och filtrerade datasetet (best_ds) i Tränings, Validerings och Test dataseten innehåller 640-160-200 bilder. (Se Tabell 2.)

Total antal bilder best_ds	Train	Validation	Test
1000	640	160	200

Tabell 2. Train-Validation-Test fördelning för full_ds

3.5 Data augmentering

För att förbättra modellens prestanda och göra den mer robust, har vi experimenterat med olika kombinationer av tekniker för data augmentering såsom att rotera, spegla bilder horisontellt, att justera bildernas kontrast och ljusstyrka. Vi hade 4 olika kombination av augmenteringar (Se Figur 22):

- no – ingen augmentering, bilderna är som original
- contrast – bara kontrast augmentering
- non-contrast – rotation, spegling och ljusstyrka (men ingen kontrast)
- full – alla augmenteringar

3.5.1 Steps_per_epoch

Steps_per_epoch styr hur många batcher som modellen ska lära sig ifrån inom en epok. Steps_per_epoch är oftast lika med träningsinstanser dividerat med batch_size, dvs. att modellen kommer att gå genom alla träningsinstanser en enda gång under en epok.

Om man förändrar steps_per_epoch så det blir större än träningsinstanser dividerat med batch_size, blir resultatet att modellen kommer att gå genom instanserna i datasetet flera gånger. I vårt experiment vi har även försökt att justera steps_per_epoch med att multiplicera träningsinstanserna dividerat med batch_size med 5. Detta har resulterat i att modellen har gått genom datasetet 5 gånger inom en epok, men eftersom vi har använt oss av dataaugmentering, har modellen sett en ny variation av träningsdatan varje gång (5 gånger). Den utökade träningen med augmentering har vi kallats för 5x.

Figur 22. Exempel på bildaugmentering

```python
def build_model(input_height=224, input_width=224, summary=False):
    pretrained_model = tf.keras.applications.ResNet50(include_top=False,
                       input_shape=(input_height, input_width, 3),
                       pooling='avg',classes=2,
                       weights='imagenet')
    pretrained_model.trainable = False

    resnet_model = Sequential()
    resnet_model.add(pretrained_model)
    resnet_model.add(Dense(1, activation='sigmoid'))
    if summary:
      resnet_model.summary()
    return resnet_model
```

Figur 23. build_model() funktionen

3.6 Modell implementering

Nästa steg efter data preprocessningen var att mata in data i ResNet50 modellen. Vi har delat in modellträningen i transfer learning och fine-tuning delar.

3.6.1 Transfer learning

Vi har laddat in ResNet50 modellen med vikterna som blev tränade på ImageNet datasetet, men har fryst in alla dess lager och har exkluderat sista fullt uppkopplade Dense lagret och har lagt till vår egna Dense lager ovanpå med 1 output unit, som vi har lämnat träningsbart. Träningen var 5 epoker lång med inlärningsgraden av 0.001, och syftet var att Dense lagret ska komma ifrån de initiala slumpmässiga vikterna och lära sig lite om skadorna på grusvägen (Se Figur 23.).

3.6.2 Fine-tuning

Efter detta steg tog vi transfer learning modellen och tränade ytterligare 5 epoker på datan, men denna gång har vi fryst upp det sista residual blocket som blev träningsbart. Här har vi tränat modellen på 5 epoker men med mycket lägre inlärningsgrad på 0.00005, med syftet att träningen inte ska påverka det sista residual blocket för mycket, utan bara finjustera det tillsammans med Dense lagret.

3.7 Metod sammanfattning

I slutet hade vi:

- 2 datasets (full_ds, best_ds),
- 4 olika cropping (original, square, padded, stretched),
- 4 olika augmenteringar (no, contrast, non-contrast, full) och
- 2 träningssätt med vanliga och utökade dataset (5x)
- 2 träningssätt (Transfer Learning och Fine Tuning)

Vi har tränat en modell med varje kombination och för att kunna spåra och logga våra experiment, har vi använt oss Weights and Biases plattformen för att spara alla våra resultat samt modeller.

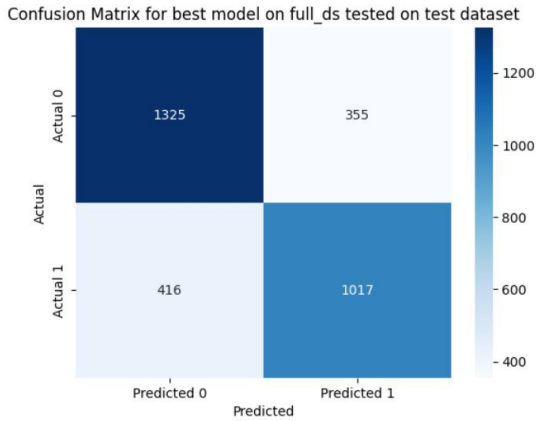
4 Resultat och Diskussion

Efter vi har gjort experiment med båda dataset på alla olika bildbeskärningar och alla olika sätt att augmentera, har vi valt ut de modellerna som hade bäst accuracy på deras respektive valideringsdataset. Vi har valt två modeller:

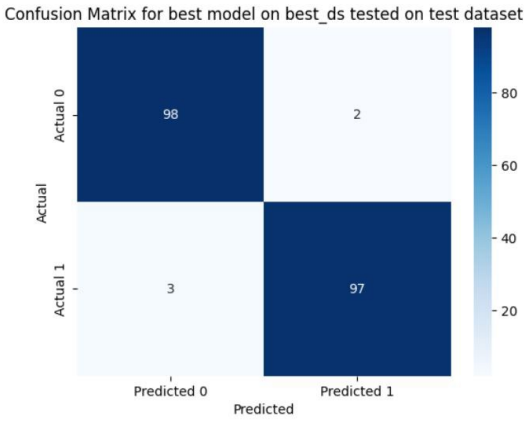
- en på hela datasetet (full_ds) som innehåller bilder i alla olika väder-, och ljusförhållande (Se Figur 24.)
- en på en utvald men mer homogen dataset (best_ds) som innehåller bilder i samma väder-, och ljusförhållande (Se Figur 25.)

Dataset	Cropping	Augmentation	Steps_per_epoch	Validation accuracy	Test accuracy
full_ds	square	contrast	1x	0.745382	0.752328
best_ds	square	full	5x	0.95625	0.975

Tabell 3. Validation och Test accuracy på full_ds och best_ds



Figur 24. Confusion matrix för test resultatet på bästa modell på full_ds



Figur 25. Confusion matrix för test resultatet på bästa modell på best_ds

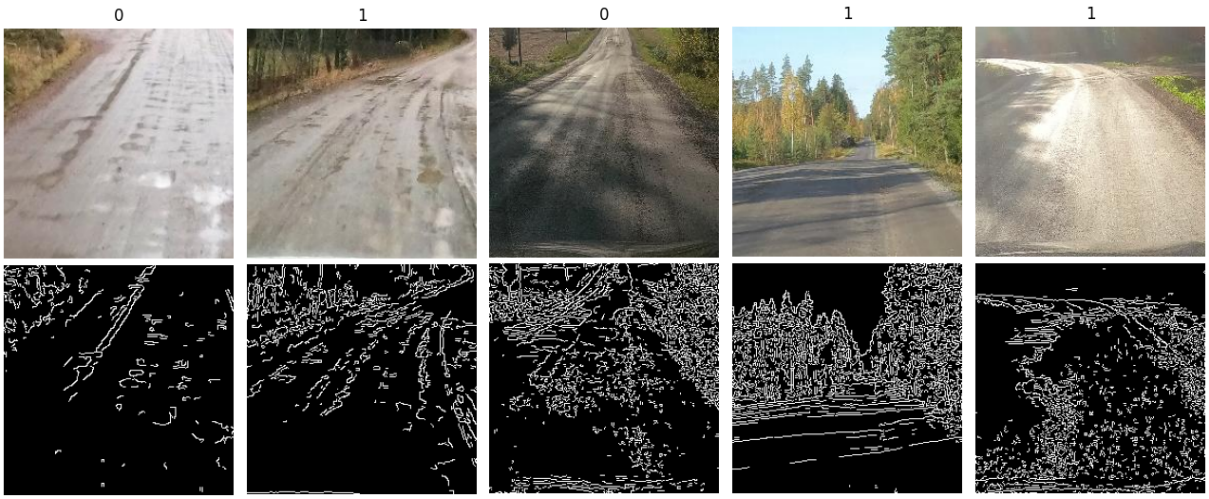
4.1 Förklaring av resultat

Som vi har förväntat oss, bästa resultatet för best_ds blev betydligt bättre än bästa resultatet för full_ds. Detta har sitt ursprung i två faktorer.

- Skador i best_ds var mycket tydliga och det var därför lättare för modellen att lära sig karakteristiker av skador.
- Bilderna hade mindre extrema skillnader mellan väder-, och ljusförhållande.

För att bekräfta vår spekulation om att homogena bilder ger bättre resultat, har vi gjort ett kort experiment, där vi provade edge detection för att få bilder med enbart svarta och vita pixlar (Se Figur 24.). När vi tittade på resultaten från edge detection upptäckte vi att bilder med starkt solljus och skuggorna hade en stor effekt på 'edges' - kanterna. Eftersom de nedersta convolutionlagret i en CNN-modell lär sig att upptäcka enkla former, kom vi fram till att vår modell har mycket svårt för bilder med många skarpa skuggor. Dessa bilder gör det inte bara svårt för modellen att lära sig de rätta mönstren för klassificering, utan är också benägna att göra klassificeringsfel i prediktionsstadiet. Förutom detta, tvetydliga bilder visar också hur svårt det kan vara för modellen att skilja dem åt.

Detta experiment har visat oss ytterligare att ett mer homogent dataset med karakteristiska egenskaper såsom väder- och ljusförhållande och tydliga skador kan ge bättre resultat.



Figur 24. Resultat på Edge detection experiment. Första två bilder är tvetydliga, sista tre bilder har stark ljus och skugga. 0→icke-skada / 1→skada.

4.2 Påverkan av olika dataförberedningar

För att kunna visa hur dom olika sätt att förbereda datan har påverkat modellens prestanda, har jag gjort grundläggande statistisk analys på resultaten. Totalt har vi tränat 40 modeller (Se Appendix A.), jag har delat in modellerna baserad på grupper i Kapitel 3.7 – Metod sammanfattning och har jämfört de olika gruppernas medelvärde.

4.2.1 Påverkan av Fine Tuning efter Transfer Learning

Om man jämför medelvärdet av accuracy resultat för Transfer Learning (TL) och Fine Tuning (FT) så ser vi att Fine Tuning resulterar i genomsnitt i högre accuracy (Se Tabell 4.).

	Medel accuracy
Transfer Learning	0.802710
Fine Tuning	0.816988

Tabell 4. Medel accuracy för Transfer Learning och Fine Tuning

4.2.2 Påverkan av datakvalité

Vi har jobbat med två dataset, full_ds och best_ds. Vi kan se definitivt att best_ds, utvald dataset med hög kvalité har en stor påverkan på accuracy resultaten både vid Transfer Learning och Fine Tuning (Se Tabell 5.).

	TL accuracy	FT accuracy	Totalt medel accuracy
Full_ds	0.713233	0.723976	0.718604
Best_ds	0.892187	0.910000	0.901094

Tabell 5. Medel accuracy för Transfer Learning och Fine Tuning per dataset

4.2.3 Påverkan av olika bildbeskärningsmetoder

Om man kollar på det olika bildbeskärningsmetoderna, visar det sig att skillnaden i medel accuracy är väldigt liten mellan original, square och stretched croppings, men padded cropping tenderar att prestera sämre än de andra 3 metoderna (Se Tabell 6.). Man kan argumentera för detta resultat att bilder med padded cropping har två stora delar med bara svarta område och pixlar, vilket gör att en signifikant del på bilderna innehåller ingen information och bidrar inte till modellträningen.

	TL accuracy	FT accuracy	Totalt medel accuracy
Original	0.804031	0.820341	0.812186
Padded	0.785058	0.787181	0.786119
Square	0.818697	0.834114	0.826406
Stretched	0.803055	0.826315	0.814685

Tabell 6. Medel accuracy per croppingsmetoder

4.2.4 Påverkan av augmentation

Enligt resultaten har dataaugmentering inte haft någon konsistent och stor påverkan på accuracyn (se. Tabell 7.).

	TL accuracy	FT accuracy	Totalt medel accuracy
No	0.813052	0.825195	0.819123
Contrast	0.806140	0.831482	0.818811
Non-contrast	0.796671	0.825769	0.811220
Full	0.798844	0.801247	0.800045

Tabell 7. Medel accuracy per augmenteringsmetoder

4.2.5 Påverkan av steps_per_epoch

Som jag skrev i Kapitel 3.5.1, steps_per_epoch styr antalet bilder hur modellen lär sig ifrån i en epok. Steps_per_epoch har störst betydelse om man använder augmentation, och på detta sätt kan man praktiskt taget utöka ett begränsat dataset med att gå genom träningsdatan flera gånger och introducera variation i det. Om vi tittar på resultatet så ser vi att utökning av dataset med att justera steps_per_epoch har en någon effekt på modellens prestanda (Se Tabell 8.).

	TL accuracy	FT accuracy	Totalt medel accuracy
1	0.785665	0.781209	0.783437
5	0.812023	0.821285	0.816654

Tabell 8. Medel accuracy per steps_per_epoch

5 Slutsats

Låt mig gå tillbaka till ursprungsfrågan: ”Kan man skapa en CV modell som med 85% noggrannhet kan klassificera bilder på grusvägar till skadad – icke-skadade klasser?”
Svaret på frågan är: Ja, det finns potential att skapa en modell som kan klassificera skadade och icke-skadade grusvägar, MEN i nuläget har vi begränsningar och förutsättningar för utveckling av en sådan modell.

Begränsningen är framför allt tiden vi har haft för projektet och att vi saknar den fullständiga domänkunskapen om skadeklassificering på grusvägar.
Modellen vi har tränat uppnår inte det önskade målet av 85% accuracy på ett dataset med mycket varierande väder- och ljusförhållande, men vi har även visat att med rätt förutsättningar, en modell kan tränas och uppnå bra resultat om underliggande datan är mer homogen. Huvudutmaningarna har jag redan nämnt i Kapitel 4.1 – Förklaring av resultat.

5.1 Rekommendationer - steg för framtida utveckling

Genom hela processen, har vi upptäckt flera olika aspekter där förbättringar kan göras för att kunna utveckla en mer effektiv lösning för grusvägsklassificering. Jag har delat in aspekterna i 3 klasser:

- Förbättra data
- Förbättra modell
- Justera målet och roll av modellen

5.1.1 Förbättra data

Grunden i alla AI projekt är datan och vi har tänkt på olika punkter om hur datakvalitén skulle kunna förbättras.

- **Uniform kameravinkel**: att kunna garantera en uniform positionering av kameror, skulle insamlingen göras av samma typ och modell av bilar, och med exakt samma positionering.
- **Extrema ljusförhållande**: för att minimera de extrema ljusskillnaderna mellan bilderna kan ett kamerafilter eller ett bildbehandlingsprogram användas. I det här steget kan en expert på fotografi och ljus inkluderas för att få ytterligare insikter och tips för hur man kan uppnå mer uniform bildkvalitet.
- **Förbättring / omdefiniering av annotationer**: att skilja mellan skada och icke-skada på en bild är på gränsen till en konstart. Även om det finns en definition till vad det räknas som skada, i verkligheten är en stor del av skadorna tvetydiga och ligger i en gråzon varför det är mycket svårt att skilja skada eller icke-skada även för en människa. Man skulle kunna definiera om vad en skada är, och dra en extremt tydlig gräns mellan klasserna skada / icke-skada när det kommer till hur modellen skulle avgöra klassifikationerna. Med detta skulle kanske fallen i gråzonen klassificeras som skador och skadade-klassen skulle kontrolleras manuellt efteråt.

5.1.2 Förbättra modell

I vårt projekt har vi inte experimenterat mycket med modellutvecklingen, och detta skulle kunna göras mer grundligt i framtiden.

- **Användning av mera Dense layers**: vi har lagt till bara ett Dense lager till ResNet50 modellen, och det är möjligt att flera Dense lager skulle kunna öka prestandan av modellen.
- **Finjustera learning rate**: i projektet har vi använt olika inlärningsgrader till TL och FT delen av träningen. När vi har bestämt inlärningsgraden, har vi utgått från inlärningsgraden som fanns

i publikationen av Nausheen et al [2022]. I publikationen har dom använt en metod som kallas för Discriminative learning, vilket är en metod att hitta optimala inlärningsgraden anpassat till data och modell, och som nästa steg skulle vi kunna hitta optimala inlärningsgrader till vårt användningsfall också.

- **Frysa upp olika lager vid FT**: I FT steget av modellträningen har vi fryst upp bara det sista residual blocket av ResNet50 modellen. Enligt publikationen av Mazor David et al. [2023] fine tuning av modellerna kan även göras genom att selektivt frigöra de olika lagren i ResNet50 modellen, i stället för att bara frigöra sista residual blocket, och denna metod kan i teorin leda till förbättrad modellprestanda.

5.1.3 Justera målet och roll av modellen

Skada klassificering på asfaltvägar är en existerande lösning, men det finns än så länge ingen liknande och bra fungerande lösning för grusvägar. Anledningen till det är mångsidig och jag har beskrivit flera av utmaningar i denna rapport. Problemet är komplext och det krävs mer tid och ansträngning för att leverera en vettig lösning.

- **Ett nätverk av modeller**: vi kunde visa i projektet att modellen kan prestera bra när bilderna är tagna i liknande väder- och ljusförhållandet, men original dataset består av bilder med mycket varierande bildkvalitet. En betydande strategi skulle kunna vara att skapa en modell som sorterar bilderna i olika segment eller klass, och sen träna en modell till varje segment / klass av bilder och träna två separata modeller som presterar bra på just den gruppen av homogena bilder. Vi har hittat liknande logik i publikation av Sonali Bhutad et al. [2022], där författarna delar upp bilderna i soliga och regniga klasser.
- **Delvis automatisering**: I stället för att skapa en klassificerare som kan automatisera utvärdering av skador på grusväg fullständigt, skulle man kunna formulera om problemet och frågeställningen och fokusera på att delvis automatisera klassificeringen, och använda modellen för att minska antalet bilder som måste kontrolleras och bedömas manuellt. Detta kan lätt göras med ’threshold tuning’ tekniken, där man justerar prediktioner av klasser baserad på prediktionernas sannolikhet.
  Man skulle kunna klassificera bilder som skadad även om modellen predikterar låg, låt oss säga 25% sannolikhet för skada (hög recall) eller vi skulle kunna prediktera skada bara när sannolikheten för det ligger över 90% (hög precision). På detta sätt kan vi filtrera bort en viss andel av bilderna som har med hög sannolikhet för skador eller icke-skador och med detta minska antalet bilder som ska kontrolleras manuellt.

Appendix A

dataset	crop	steps_per_epoch	augment	TL	FT
full	original	1	no	0.732932	0.722892
full	square	1	no	0.724096	0.723695
full	padded	1	no	0.7249	0.733735
full	stretched	1	no	0.72249	0.733735
best	original	1	no	0.9	0.9125
best	square	1	no	0.9125	0.93125
best	padded	1	no	0.88125	0.91875
best	stretched	1	no	0.90625	0.925
full	original	1	contrast	0.729317	0.748193
full	square	1	contrast	0.704016	0.745382
full	padded	1	contrast	0.700402	0.714056
full	stretched	1	contrast	0.734137	0.731727
best	original	1	contrast	0.9	0.925
best	square	1	contrast	0.9125	0.95
best	padded	1	contrast	0.85	0.90625
best	stretched	1	contrast	0.91875	0.93125
full	original	1	non-contrast	0.728112	0.746586
full	square	1	non-contrast	0.734137	0.741365
full	padded	1	non-contrast	0.697992	0.708434
full	stretched	1	non-contrast	0.694377	0.728514
best	original	1	non-contrast	0.875	0.9125
best	square	1	non-contrast	0.90625	0.9375
best	padded	1	non-contrast	0.9	0.925
best	stretched	1	non-contrast	0.8375	0.90625
full	original	1	full	0.689558	0.702811
full	square	1	full	0.698795	0.718474
full	padded	1	full	0.684739	0.68996
full	stretched	1	full	0.718474	0.719679
best	original	1	full	0.85625	0.85625
best	square	1	full	0.925	0.91875
best	padded	1	full	0.85625	0.71875
best	stretched	1	full	0.85625	0.925
full	original	5	full	0.722892	0.732932
full	square	5	full	0.719679	0.718474
full	padded	5	full	0.698795	0.694377
full	stretched	5	full	0.704819	0.724498
best	original	5	full	0.90625	0.94375
best	square	5	full	0.95	0.95625
best	padded	5	full	0.85625	0.8625
best	stretched	5	full	0.9375	0.9375

Källförteckning

Arxiv (2015). Deep Residual Learning for Image Recognition
https://arxiv.org/pdf/1512.03385.pdf hämtad: 2024-01-25

Deeplizard: Engage with Deep Learning
https://deeplizard.com/resource/pavq7noze2 hämtad: 2024-01-27

Gal Kaplun, Andrey Gurevich, Tal Swisa, Mazor David, Shai Shalev-Shwartz, Eran Malach (2023): Less is More: Selective Layer Finetuning with SubTuning
https://arxiv.org/pdf/2302.06354.pdf hämtad: 2024-01-29

Géron A. (2019): Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow

Italianstudies (2011). All Roads Lead to Rome: New acquisitions relating to the Eternal City https://italianstudies.nd.edu/news-events/news/all-roads-lead-to-rome-new-acquisitions-relating-to-the-eternal-city/ hämtad: 2024-01-28

Machinelearningmastery (oktober 23, 2019). Loss and Loss Functions for Training Deep Learning Neural Networks
https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/ hämtad: 2024-02-03

Machinelearningmastery (December 4, 2019). A Gentle Introduction to Batch Normalization for Deep Neural Networks
https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/ hämtad: 2024-02-03

Machinelearningmastery (2021). How to Choose an Activation Function for Deep Learning
https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/ hämtad: 2024-02-03

Nausheen Saeed, Roger G. Nyberg & Moudud Alam (2022): Gravel road classification based on loose gravel using transfer learning, International Journal of Pavement Engineering, DOI: 10.1080/10298436.2022.2138879

Nienaber S, Booysen M, Kroon R (2015): Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage, DOI: 10.13140/RG.2.1.3121.8408

Researchgate (2021). AI-ML developments for the Atlas ion Linac facility
https://www.researchgate.net/figure/Schematic-of-a-neural-network-surrogate-model-for-the-ATLAS-RFQ_fig2_362781067 hämtad: 2024-01-27

Researchgate (2023). A convolutional neural network-based decision support system for neonatal quiet sleep detection https://www.researchgate.net/figure/CNN-architecture-for-QS-detection_fig4_373504979 hämtad: 2024-01-27

Scikit-Learn API dokumentation.
Confusion matrix:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
Precision-Recall:
https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
Classification report:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
hämtad: 2024-02-03

Sonali Bhutad, Kailas Patil (2022): Dataset of road surface images with seasons for machine learning applications https://www.sciencedirect.com/science/article/pii/S2352340922002347 hämtad: 2024-01-29

TensorFlow (2023). Transfer learning and fine-tuning
https://www.tensorflow.org/tutorials/images/transfer_learning hämtad: 2024-01-26

Turing: What Is the Necessity of Bias in Neural Networks?
https://www.turing.com/kb/necessity-of-bias-in-neural-networks hämtad: 2024-01-27

Turing: How to Choose an Activation Function For Deep Learning
https://www.turing.com/kb/how-to-choose-an-activation-function-for-deep-learning#what-is-a-neural-network? hämtad: 2024-01-27

Wikipedia (2019). https://commons.wikimedia.org/wiki/File:ConfusionMatrixRedBlue.png
hämtad: 2024-01-27