Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Johan Martin Skavhaug

Deteksjon av trafikkskilt med dype nevrale nettverk

Bacheloroppgave i ingeniørfag, Data
Veileder: Alexander Holt
Juni 2020

NTNU
Kunnskap for en bedre verden

Johan Martin Skavhaug

Deteksjon av trafikkskilt med dype nevrale nettverk

Bacheloroppgave i ingeniørfag, Data
Veileder: Alexander Holt
Juni 2020

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

Denne rapporten er skrevet som avsluttende innlevering på studieretningen dataingeniør ved Norges teknisk-naturvitenskapelige universitet i Trondheim. Den omhandler prosjektet «Tolkning av trafikkskilt på Android med bruk av OpenCV», som etter hvert ble til «Deteksjon av trafikkskilt med dype nevrale nettverk». Rapporten gir innsikt i teknologier, rammeverk, arkitekturer og metoder som er blitt benyttet til å trene og teste nevrale nettverk som lokaliserer og klassifiserer trafikkskilt.

De viktigste grunnene til at akkurat denne oppgaven ble valgt er:

1. Undertegnede har stor interesse av maskinlæring, og da spesielt bruk av dype nevrale nettverk til å utføre kompliserte utfordringer
2. Utvikling av «open-source» alternativer til de store teknologigantene er med på å sørge for åpenhet og tilgjengelighet av nye teknologier og metoder.

Rapporten er skrevet på norsk, men for at terminologien skal forbli oversiktlig og korrekt er mye av det beholdt på engelsk. Dette fordi store deler av terminologien ikke eksisterer i norsk fagspråk, og heller ikke har klare logiske oversettelser til norsk.

Prosjektet er stort sett utført alene, og på hjemmekontor, på grunn av Covid-19. I samme periode som bølgen traff Norge fant min prosjektpartner ut at han ikke var i stand til å gjennomføre prosjektet. Oppfølging og motivering fra prosjektveileder har vært til stor hjelp i denne spesielle situasjonen. Derfor vil jeg rette en stor takk til Alexander Holt fra Institutt for datateknologi og informatikk ved NTNU, for støtten og veiledningen underveis i prosjektet.

Dokumentet er utarbeidet etter mal fra Institutt for datateknologi og informatikk ved NTNU.

Verdal, 10.06.2020

Johan Martin Skavhaug

Oppgavetekst

Originalt besto oppgaven av å lage en applikasjon på Android hvor hensikten var å benytte mobilkameraet til å kontinuerlig filme veien ut av frontruta og finne og tolke trafikkskiltene ettersom de dukket opp. Denne tolkingen skulle utføres med bruk av OpenCV og nevrale nettverk. Ved å koble de tolkede trafikkskiltene sammen med data fra mobilens GPS var hensikten i første omgang å kontribuere kartdata til OpenStreetMap, et crowd sourcet alternativ til Google Maps.

Relativt tidlig i prosjektet ble fokus satt på å finne en god løsning til å utføre tolkingen av trafikkskilt, og etter hvert ble utviklingen av en androidapplikasjon forkastet. Slik ble oppgaven i praksis å modellere og trene ett nevralt nett til deteksjon av trafikkskilt, som skal være i stand til å kjøres i tilnærmet sanntid på mobil.

Sammendrag

Object Recognition er i dag et svært akuelt og populært fagområde, hvor store framskritt har blitt oppnådd i senere tid. Det er et fagområde som kan brukes til alt fra selvkjørende biler og automatisk bildetagging til opplåsing av mobiltelefoner og masseovervåkning av mennesker. Det har blitt påstått at selvkjørende biler kan redde opptil 90% av de som ellers ville omkommet i trafikkulykker. Som et steg på veien dit er man, blant mye annet, nødt til å kunne gjenkjenne trafikkskilt.

Å detektere objekter i bilder er vanskelig, og å detektere små objekter er enda vanskeligere. I denne rapporten utforskes egnetheten til å utføre deteksjon av trafikkskilt i sanntid ved bruk av Faster R-CNN [1]. Gjennom arbeidet i dette prosjektet er det utarbeidet en grunnmodell som oppnår 34.4 mAP på testdelen av datasettet GTSDB [2], og bruker 2.68 sekunder per bilde.

Videre er det eksperimentert med augmentering av datasettet for å minske graden av overtilpasning av modellen til treningsdatasettet. Resultatene viser at forsiktig skifting, rotasjon og skalering, samt moderat tilfeldig beskjæring og strekking av treningsdata øker treffsikkerheten markant på testdatasettet.

For å gjøre modellen lettere å kjøre i sanntid ble det eksperimentert med reduksjon av regionforslag under testing av modellen. Ved å redusere antallet regionforslag oppnådde modellen en inferenstid på 0.276 sekunder uten et tilhørende tap i treffsikkerhet. Dette bestrider tidligere forsøk med Faster R-CNN på andre datasett [3].

Innholdsfortegnelse

1 Introduksjon og relevans

I dagens samfunn bruker vi kartløsninger fra store teknogiganter til å navigere i trafikken. Disse kartløsningene vil i tillegg til alle godene de tilbyr ofte ta vare på lokasjonsdataen til f.eks. mobilen de kjører på. De finnes mange grunner for å ville ha andre gode alternativer. Dessverre mangler gjerne *open-source* alternativer til disse kommersielle kartløsningene nyttig kartdata, som trafikkskilt og fartsgrenser. Ett av målene med dette prosjektet er å skaffe til veie en enkel måte å *crowd-source* slik kartdata i framtiden, ved å utføre deteksjon av trafikkskilt i sanntid på mobiltelefoner.

Ved å trene et nevralt nettverk til å utføre denne deteksjonen av trafikkskilt får man ett mer fleksibelt system en de tradisjonelle metodene for å gjenkjenne trafikkskilt. Ett nevralt nettverk vil i høy grad være i stand til å håndtere forskjellige lys- og værforhold, samt også være lite avhengig av trafikkskiltenes posisjon i bildene. Dette vil da også kunne anvendes på bilder skaffet fra andre perspektiver enn bilisten.

Det er først nylig at teknologien har kommet så langt at dype nevrale nettverk har blitt tatt i bruk til tolking av bilder. Moderne metoder drar nytte av utregningskraften til GPUer for å gjøre både trening og inferens av nevrale nettverk raskere. Mobilteknologien har gjennomgått en enrom utvikling de siste årene, slik at vi i dag ser mobiltelefoner med maskinkraft ikke langt unna en pc i midtsjiktet. Denne utviklingen har gjort det svært interessant med sanntids inferens på mobile enheter, hvor dette tidligere har blitt gjort ved hjelp av inferens-servere.

1.1 Problemstilling

Det ble valgt en problemstillinger som dette prosjektet skal finne svar på:

1. Er deteksjon av trafikkskilt med en akseptabel treffsikkerhet mulig å utføre i sanntid med bruk av Faster R-CNN?

Hypotesen er at det finnes ett sett hyperparametre hvor Faster R-CNN oppnår vesentlig bedre treffsikkerhet på deteksjon av trafikkskilt enn *single shot* detektorer som YOLO og SSD.

1.2 Dokumentstruktur

Rapporten er delt inn i 9 kapitler. Under er en kort beskrivelse av hva leseren kan forvente av innhold i hvert av dem.

3. **Introduksjon og relevans.**
   Kapitlet redegjør for hvorfor prosjektet oppsto, hva formålet med prosjektet er, samt problemstillingen som skal svares på og rapportens struktur.
4. **Teori**
   Kapitlet redegjør for relevant teori og nødvendig bakgrunnsinformasjon.
5. **Valg av teknologi**
   Kapitlet tar kort for seg hvorfor den teknologien og de metodene som er brukt i prosjektet ble valgt, samt hvorfor noe ikke ble valgt.
6. **Metode**
   Kapitlet forklarer hvordan resultatene har blitt produsert.
7. **Resultater**
   Kapitlet presenterer de ulike resultatene som prosjektet har resultert i.

8. **Diskusjon**
   Her drøftes resultatene opp mot problemstillingen og relevante tidligere forskningsresultater.
9. **Konklusjon og videre arbeid**
   Kapitlet svarer på problemstillingene fra kapittel 1, og hva som bør gjøres videre.
10. **Referanser**
    Oversikt over alle referansene i rapporten
11. **Vedlegg**
    Oversikt over vedleggene til rapporten

1.3 Forkortelser

Ord/forkortelse	**Betydning**
CNN	Convolutional Neural Networks
R-CNN	«Regions with CNN features»
FRCNN	«Faster R-CNN»
Input	Inndata
Beregningskostnad	*Computational Cost*
Metaarkitektur	En overordnet arkitektur for hele løsningen av ett Object Detection problem
IoU	*Intersection over Union*, overlapp delt på unionen.
SSD	*Single Shot Multibox Detector*
YOLO	*You Only Look Once*
Feature Map	
Fully connected	
ANN	Artificial Neural Network / Kunstige nevrale nettverk
Inferens	I maskinlæring er inferens prosessen med å ta i bruk de lærte vektene i ett nettverk til å utlede/predikere utfallet av inputen.
NMS	Non max suppresion

2 Teori

I dette kapitlet utredes det hva deteksjon av trafikkskilt faktisk innebærer. I første del får leseren en kort introduksjon til problemgruppen *Object Recognition*. Videre får leseren en kort introduksjon til kunstige nevrale nettverk og deretter en oversikt over hvordan *Faster R-CNN* fungerer.

2.1 Object Recognition

Object Recognition er et generelt begrep for samlingen av problemer som involverer alt med å identifisere objekter i digitale bilder.

I denne samlingen finner vi **Image Classification** som involverer selve klassifiseringen av et objekt, altså å tildele en klasse til et bilde som inneholder utsnittet av *ett* objekt.

Object Localization er problemet med å lokalisere objekter i et bilde og markere posisjonen til objektet med en markeringsramme. Dette kan være vanskelig fordi bilder kan inneholde flere objekter som kan være i forskjellige størrelser og posisjoner.

Object Detection er kombinasjonen av disse oppgavene, altså både lokalisering av objekter med en markeringsramme og klassifisering av objektene som er funnet. Dette er forskjellig fra *object segmentation,* hvor man klassifiserer hver eneste piksel i bildet, og dermed finner de detaljerte formene til objektene i bildet.

2.1.1 Mål på treffsikkerhet

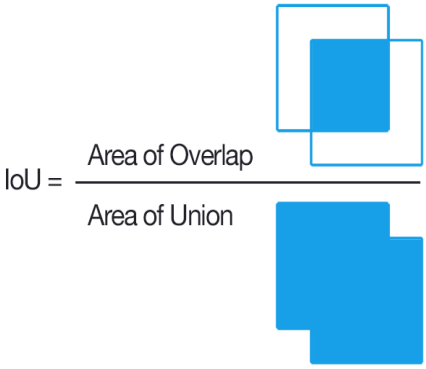
For å måle hvor godt et system løser et *Object Detection* problem brukes **Mean Average Precision** (**mAP**). For hver klasse regnes det ut gjennomsnittlig presisjon, **Average Precision** (**AP**), og gjennomsnittet av gjennomsnittene utgjør **mAP**. For å regne ut dette brukes **precision** og **recall** som igjen avhenger av antallet sanne positive- (**TP**), falske positive- (**FP**) og falske negative (**FN**) deteksjoner.

$$\boldsymbol{Precision\ p} = \frac{TP}{TP + FP} = \frac{TP}{Alle\ deteksjoner}$$

$$\boldsymbol{Recall\ r} = \frac{TP}{TP + FN} = \frac{TP}{Alle\ faktiske\ objekter}$$

$$\boldsymbol{Average\ Precision} = \int_0^1 p(r)dr$$

Precision er hvor stor andel av deteksjonene som er korrekte, og *recall* er andelen objekter som ikke ble detektert. For at en deteksjon skal være korrekt må overlappen delt på unionen (**IoU**) av markeringsrammen til deteksjonen og den faktiske rammen være over en gitt verdi. [Figur 1] viser hvordan IoU regnes ut. **Average Precision (AP)** er arealet under kurven til *precision* plottet mot *recall* av modellen. Denne *Precision-Recall* kurven finnes ved å rangere alle deteksjonene av en klasse med synkende konfidensnivå, og regne ut kumulativ *precision* og *recall* etter hver deteksjon i rangeringen.



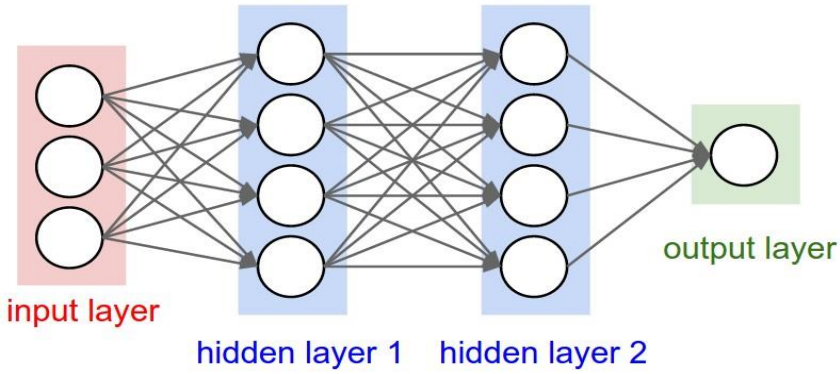
Figur 1: Intersection over Union. Kilde: pyimagesearch

2.2 Kunstige nevrale nettverk

Kunstige nevrale nettverk (ANN) er en av byggesteinene innen maskinlæring, og da spesielt *deep learning*, eller dyp læring. Disse nettverkene er bygget opp av et inputlag og et outputlag, samt noen "gjemte" lag (*hidden layers)* imellom. Det er disse gjemte lagene som analyserer bildet og finner særtrekk. Disse særtrekkene utgjør en **feature map**, en abstrakt tolkning av inputen, og det er dette som danner grunnlaget som det siste outputlaget gir en prediksjon fra.

Lagene er basert på lineære funksjoner hvor inputen blir multiplisert med et sett vekter som vektlegger forskjellige trekk i inputen. Det er koblingene mellom lagene som utgjør selve nevronene i nettverket. Nettverket ser på sannsynligheten for at inputen tilhører de forskjellige klassene det skal skilles mellom, og produserer et konfidensnivå til hver av klassene. [Figur 2] viser et nevralt nettverk med to fullt koblede gjemte lag.

Dype nevrale nettverk er nettverk som bruker flere lag til å progressivt finne særtrekk av høyere grad fra den rå inputen.



Figur 2: Kunstig nevralt nettverk. Kilde: Stanford University /CC BY-NC.

2.2.1 Trening av nettverk

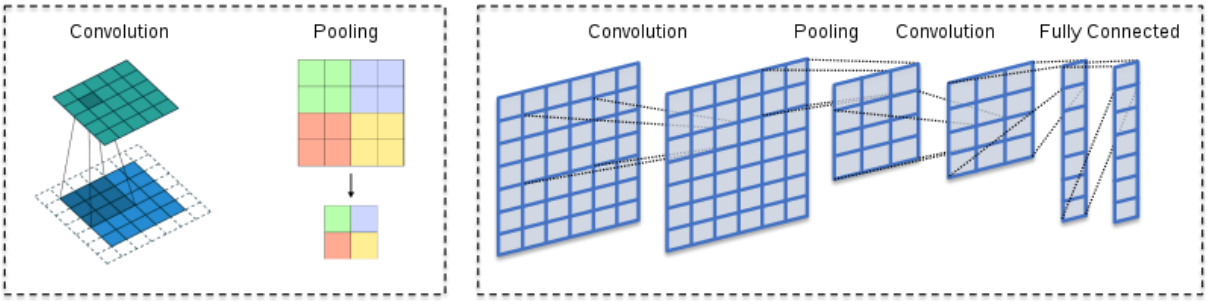
Kunstige nevrale nettverk brukes i **supervised learning**, hvor vektene i nettverket under trening justeres for at feilen mellom det kjente input-output paret og den produserte outputen skal bli minst mulig. **Backpropagation** regner ut gradienten som en funksjon av de mulige vektene i nettverket, med hensyn på nettverkets tapsfunksjon. Ved å iterativt justere hver vekt i nettverket proporsjonalt med hvor mye vekten bidrar til den totale feilen mellom produsert output og kjent output, nås til slutt en samling av vekter som produserer gode prediksjoner av inputen.

For å finne forslag til nye verdier av vektene brukes **gradient descent.** Denne metoden går ut på å iterativt forflytte seg nedover i den bratteste retningen i grafen til tapsfunksjonen av nettverket. Ved å følge denne grafen nedover er håpet å ende opp i det globale minimumet for grafen, men ulike startpunkt og stegstørrelser kan gjøre at vi ender opp i ett lokalt minimum med høyere verdi enn det globale minimumet. Koordinatene i grafen utgjør forskjellige verdier av vektene for de lineære funksjonene i nettverket.

2.2.2 Convolutional neural networks

Convolutional neural networks (CNNs), eller konvolusjonelle nevrale nettverk, er en type dype nevrale nettverk hvor en eller flere av de gjemte lagene består av spesielle lineære operasjoner kalt *convolutions*.

Det disse lagene i praksis gjør er å se på små områder av inputen om gangen ved å flytte et vindu, eller *filter*, over inputen i små steg. Dette gjør at hver del av inputen blir sett i sammenheng med dens naboer, og et filter som skal finne et spesifikt særtrekk vil kunne finne dette uansett hvor det befinner seg i inputen. Dette blir oftest kombinert med *pooling* [Figur 3] slik at neste nettverkslag kan identifisere særtrekk av høyere grad, altså se på sammenhengen mellom særtrekkene som allerede er funnet.



Figur 3: Konvolusjonslag ser bare en begrenset del av inputen om gangen. Pooling reduserer dimensjonene til dataen. Disse brukes sammen for å danne CNNs. Eksempel vist til høyre. Kilde: Andreas Maier / CC BY

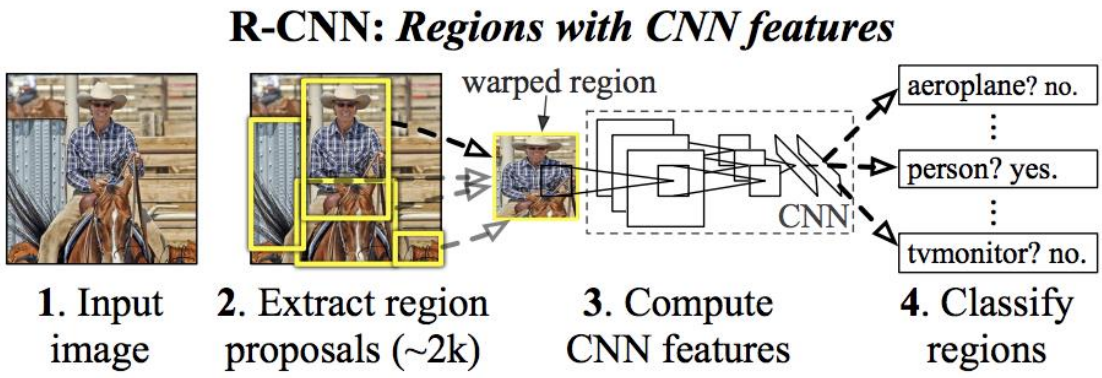
2.3 Faster R-CNN

Det finnes flere måter å utføre Object Detection med CNNs, og disse baserer seg på egne *metaarkitekturer.* For å forstå hvordan metaarkitekturen **Faster R-CNN** [1] fungerer, er det nyttig å først forstå hvordan **R-CNN** [7] fungerer.

2.3.1 R-CNN

For å detektere ett objekt i ett bilde må det velges regioner som undersøkes med CNNs. I ett enkelt bilde kan det settes sammen piksler til et svært høyt antall regioner med forskjellig lokasjon og størrelser. Beregningskostnaden for å undersøke hver av disse regionene blir for høy til at det kan gjøres i tilnærmet sanntid.

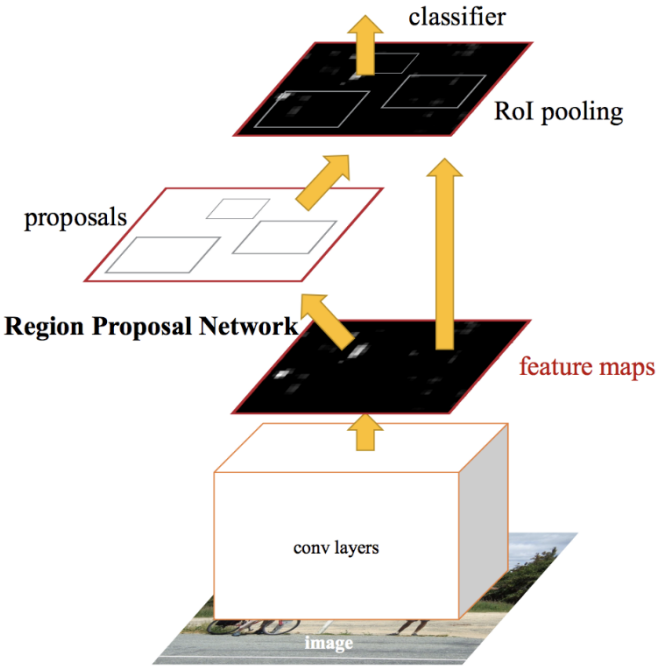
For å komme seg rundt det problemet, ble det foreslått en metode som benytter *selective search* [8] til å generere bare 2000 regioner, kalt *region proposals* (regionforslag) [7]. Alle de 2000 regionforslagene blir så kjørt gjennom CNN for å produsere en feature map av hver region, og deretter klassifiseres hver av regionene. Problemet med R-CNN er at det fortsatt er en svært høy beregningskostnad ved å undersøke 2000 regionforslag, hvor [7] rapporterte henholdsvis 13 og 53 sek/bilde på GPU og CPU. I tillegg er *selective search* en fastsatt algoritme som ikke trenes opp etter hvor gode regionforslagene er.



Figur 4: Flytskjema for RCNN. (2) utføres selective search for å finne regioner. (3) beregnes særtrekk ved hver region gjennom CNN (4) Hver region klassifiseres med SVMs (Support Vector Machines). Kilde: [7]

2.3.2 Fast & Faster R-CNN

For å løse problemet med å kjøre 2000 regioner gjennom CNN bruker Faster R-CNN selve bildet som input til CNN. Som output fra CNN produseres da en feature map som beskriver særtrekkene til bildet. Ut ifra denne feature map'en velges regionforslag, og ved hjelp av et *Region of Interest(RoI) Pooling Layer* blir disse omformet til en fast størrelse. Med den allerede beregnede feature map'en klassifiseres de omformede regionene med *fully connected* nettverkslag. Ved å anvende feature map'en til å finne regionforslag unngås å gjentatte ganger sende bildet gjennom et dypt CNN. Det er bare selve klassifiseringen av de utvalgte regionene som utføres flere ganger på bildet.



Figur 5: Oversikt over Faster R-CNN arkitekturen. Bildet kjøres gjennom CNN (conv layers) og danner en feature map. Denne blir brukt både til regionforslag og klassifisering. Kilde: [1]

Faster R-CNN bruker et **Region Proposal Network** (**RPN**) for å finne regionforslag fra *feature map*'en, i motsetning til forrige iterasjon, **Fast R-CNN** [9], som bruker *selective search* på feature map'en. Dette RPN'et har en mindre beregningskostnad enn *selective search*, og kan i tillegg gi bedre regionforslag ettersom nettverket trenes til å finne progressivt bedre regionforslag. Som standard

produserer RPN'et 300 regionforslag. Reduseres antall regionforslag under testing vil beregningskostnaden for hele prosessen bli lavere og raskere deteksjon kan oppnås [3].

3 Valg av teknologi

3.1 Datasett

German Traffic Sign Detection Benchmark [2] er ett datasett bestående av 900 bilder med variende antall trafikkskilt tatt i forskjellige vinkler og lysforhold. Dette datasettet er brukt i flere konkurranser tidligere og vil være et greit utgangspunkt for prosjektet, da det har blitt produsert gode modeller på dette. I tillegg har tyske trafikkskilt stor overlapp med norske skilt, slik at det i praksis kan være mulig å få noen resultater om det testes på trafikkskilt langs veier i Norge.

3.2 Object detection

Moderne metoder for å utføre object detection baserer seg på CNNs, i motsetning til tradisjonelle metoder hvor det har blitt brukt algoritmer tilpasset hvert enkelt problem [10]. Dette delkapitlet presenterer de ulike metaarkitekturene som brukes til object detection i dag.

3.2.1 Metaarkitekturer

De moderne metodene kan deles inn i metoder som verdsetter høyest mulig treffsikkerhet, og metoder som fokuserer på å løse problemet med minst mulig beregningskostnad. De mest nøyaktige metodene slik som **Faster R-CNN** [1], **R-FCN** [11] er regionbaserte metoder, i form av at de først finner forslag til områder i bildet som kan være et objekt og utfører klassifisering på de områdene som har høyest score. I kontrast til de tyngre metodene "ser" **SSD** [12] og **YOLO** [13] kun en gang på hele bildet og unngår dermed mye unødvendige kalkuleringer. En grunnleggende mangel ved SSD og YOLO er at de sliter med små objekter.

Avveiningen mellom hastighet og treffsikkerhet i moderne arkitekturer er diskutert i [3], hvor treffsikkerheten på små objekter (areal > 32^2 [14]) er best for Faster R-CNN. Derimot gir [15] resultater hvor SSD får høyest treffsikkerhet, men hvor treffsikkerheten er beregnet over alle objekter, hvor de fleste er mye større enn trafikkskiltene som skal detekteres i dette prosjektet.

På grunn av SSDs og YOLOs dårligere treffsikkerhet på små objekter ble det valgt å utforske Faster R-CNN i dette prosjektet.

Kort notat om tradisjonelle metoder

Tidlig i prosjektet ble det undersøkt om mulighetene til å utnytte OpenCV sitt bibliotek til *object localization* av trafikkskilt. I dette biblioteket var HAAR Cascades mest interessant, men forsøk tyder på at det ikke egner seg til bruk i sanntid [16][17], samt at denne modulen i OpenCV er foreldet og ikke lenger vedlikeholdt. Det ble derfor valgt å bruke CNNs til både *object localization* og *classification*.

3.2.2 mAP

Forskjellige utlyste konkurranser bruker ulike implementasjoner av mAP med små forskjeller for å evaluere modeller. For dette prosjektet ble valgt å bruke samme implementasjon som Pascal VOC [18]. Dette fordi datasettet som blir brukt i prosjektet inneholder klasser med tilnærmet samme reelle størrelse, og fordi selve lokaliseringen av objektene ikke trenger å være svært presis. Derfor har vi lite nytte av å finne mAP ved forskjellige verdier av IoU eller forskjellige størrelser av objekter.

3.3 Nettverksarkitekturer

I Faster R-CNN brukes et basisnettverk til å finne en feature map som brukes videre i modellen. I forskningsartikkelen ble VGG-16[4] brukt som basis og gir gode resultater uten stor beregningskostnad. Samtidig er VGG-16 enkel å implementere, slik at faren for feil minskes. Derfor er VGG-16 valgt som basisnettverk i dette prosjektet også.

For å teste innvirkningen av å bytte nettverk er også Mobilenet [5] implementert. Dette har også blitt gjort av andre [3], slik at risikoen for feil implementering også her er liten.

Mobilenet har senere fått både v2[19] og v3 [20] som begge yter bedre enn den forrige. Dessverre er disse arkitekturene ganske kompliserte, og litteratursøket gav ingen resultater på at de er implementert i Faster R-CNN med suksess. Derfor ble det besluttet å ikke implementere de nyere versjonene av mobilenet.

3.4 Rammeverk

mAP

For utregning av mAP brukes [21]. Dette er et fleksibelt bibliotek med implementasjoner av mAP som gir samme resultater som offisielle implementasjoner i COCO[14] og Pascal VOC[18]. Repoet har midlertidig en feil som var nødvendig å fikse for å bruke biblioteket på datasettet som blir brukt i dette prosjektet.

Grunnlag for kildekode

Kildekoden bygger på https://github.com/kbardool/keras-frcnn.

Tensorflow

Tensorflow [22] er et rammeverk for maskinlæring, og Keras [23] er et rammeverk for å sette sammen nevrale nettverk. Sammen danner de et komplett sett med standardiserte implementasjoner av maskinlæringsalgoritmer som brukes for å unngå implementeringsforskjeller fra andre forskningspublikasjoner. Det ble valgt å bruke tensorflow da dette er bredt støttet og brukt innen maskinlæringsmiljøet, og stadig forbedres med nye implementasjoner.

3.5 Google Colaboratory

Google Colaboratory[24], eller *colab* er googles tjeneste for gratis hosting av Jupyter notebooks for interaktiv kjøring av python kode på maskiner med kraftige GPUer. Interaktiv koding er svært hjelpsomt under implementering av maskinlæringsmodeller, da man gjerne vil undersøke dataene under flere steg for å sørge for at alt fungerer som det skal. I tillegg gjør Jupyter notebooks det enkelt visualisere og finne frem til riktig data.

Selv om koden nødvendigvis mister nesten all form av prosjektstruktur, er fordelene med interaktiv koding sammen med gratis GPU-tid uten noen form for oppsett svært attraktivt. Det ble tidlig i prosjektet besluttet å benytte colab for å trene et nettverk til applikasjonen, og det ble ikke sett på som nødvendig å portere koden til ett standard python prosjekt da prosjektet ikke hadde noe krav til dette.

Kildekoden for å trene og teste nettverkene er lagret i notebooks. Mer informasjon om installering og kjøring av denne finns i vedlegg B: Systemdokumentasjon.

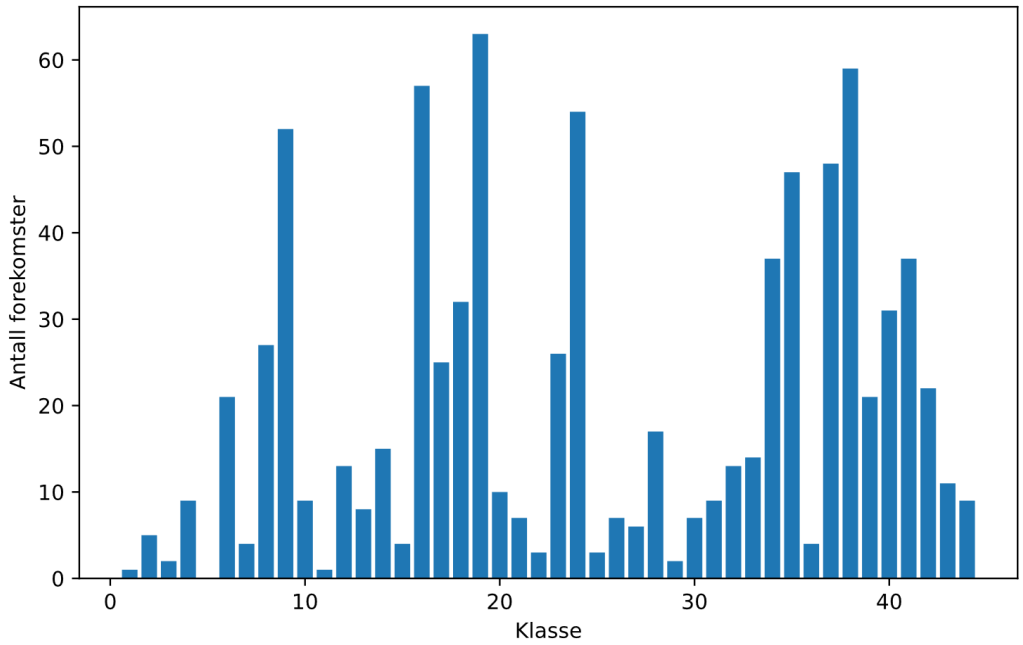
3.6 Administrativ metode

Tidlig i prosjektet ble det besluttes å bruke konseptet med sprinter fra SCRUM metodologien. Siden prosjektet bare involverte 3 deltakere, og oppgavestiller ikke stilte spesifikke krav til metodologi ble det avgjort at man ikke burde bruke en rigid implementasjon av SCRUM. Etter hvert som antall deltakere ble redusert til 2, og systemutviklingsdelen av oppgaven frafalt ble ideen med å dele opp prosjektet i sprinter islagt.

4 Metode

4.1 Datasett

GTSDB består av 900 bilder med til sammen 1213 forekomster av 43 forskjellige typer trafikkskilt. Hvert bilde er et 1360x800 bilde tatt fra frontruta i en bil, hvor 741/900 inneholder ett eller flere trafikkskilt. De forskjellige skiltene dukker ikke opp like ofte i datasettet. I figuren under er 'bakgrunn' tatt med som en ekstra klasse slik at modellen kan velge bort markeringsrammer den ikke tror er skilt.



Figur 6: Fordeling av klassene i GTSDB

4.1.1 Datainndeling

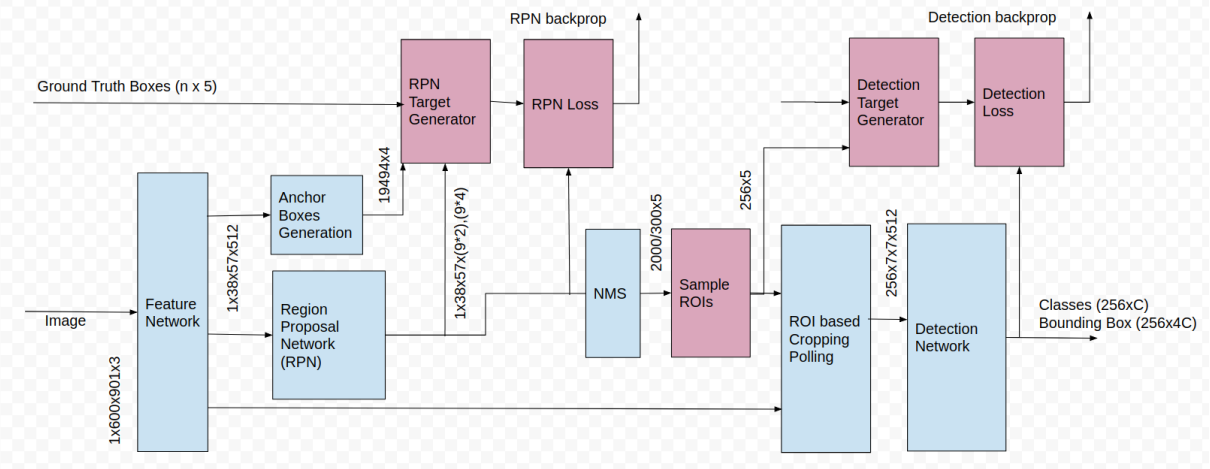
Datasettet som er brukt er delt inn i treningsdata og testdata. Treningsdatasettet blir brukt under treningen av modellen, og testdatasettet blir brukt for å evaluere de trente modellene. På grunn av den lille størrelsen av datasettet og det store antallet klasser har det blitt valgt å ikke anvende et valideringsdatasett.

For å unngå overtilpasning (*overfitting)* av datasettet som blir brukt til trening testes modellen på testdatasettet. Oppnås vesentlig dårligere treffsikkerhet på testdatasettet enn på treningsdatasettet er modellen blitt overtilpasset.

4.2 Trening av nettverket

Selve metaarkitekturen for trening av nettverket er beskrevet i delkapittel [2.3.2]

Nettverkene blir trent med *backpropagation* med hensyn på de fire tapsfunksjonene til Faster R-CNN beskrevet i forskningsartikkelen [1]. Dette betyr at modellen vil ta i bruk vektene den fant i forrige steg for å prøve å minimere summen av tapsfunksjonene. For å utføre selve optimaliseringen på tapsfunksjonene benyttes optimaliseringsalgoritmen Adam [25], som er en versjon av SGD (*stochastic gradient descent*)



Figur 7: Trening av Faster R-CNN. Kilde: @whatdhack, medium.com

Vi deler treningen av nettverkene inn i epoker på 1000 steg hver. Hvert steg i disse epokene består av følgende:

1. Henting av neste bilde fra datasettet og preprosessering av dette. Preprosesseringen består av å endre størrelse på bildet til riktig dimensjon, utregning av de beste markeringsrammene RPN'et kan foreslå, og normalisering av bildet ved å trekke fra den gjennomsnittlige fargeverdien av alle pikslene i datasettet. En vanlig praksis i fagområdet er bruke verdiene fra ImageNet, så disse verdiene blir brukt til normalisering av vårt datasett også.
2. Trening av RPN'et på bildet. Dette består av å utføre en enkelt oppdatering av grafen til RPN'et med ett steg av Adam med steglengde $1e^{-5}$.
3. Prediksjon av regions of interest. Bildet kjøres gjennom basisnettverket, RPN'et klassifiserer hvert punkt i feature map'en og de 300 beste forslagene blir funnet ved hjelp av NMS (*Non-maximum suppression).*
4. Regionforslagene sammenlignes med de kjente markeringsrammene og deles inn i positive og negative regioner. Positive regioner defineres her som regionforslag med $\mathrm{IoU} > 0.5$ med en kjent markeringsramme. Negative regioner er likedan definert som regionforslag med $0.1 < IoU < 0.5$. Regionforslag med $IoU < 0.1$ blir ikke brukt videre.
5. Fra regionforslagene velges det tilfeldig opp til 2 positive og 2 negative regioner. Er det mindre enn 2 positive regioner å velge mellom, velges flere negative slik at totalt antall blir 4. Disse 4 regionene utgjør partiet som klassifiseringsnettverket deretter blir trent på, ved å utføre en enkelt oppdatering av grafen til klassifiseringsnettverket med ett steg av Adam med steglengde $1e^{-5}$.

Etter hver epoke sammenlignes summen av tapsfunksjonen med forrige epoke og nettverksvektene blir lagret hvis de er bedre. Her burde valideringsdata blitt brukt for å evaluere om modellen faktisk har blitt bedre, siden tapsfunksjonen er fullstendig basert på treningsdatasettet og dermed er utsatt bias.

4.3 Testing av nettverket

Hvordan modellen produserer prediksjoner er beskrevet i kapittel [2.3.2].

Når nettverket testes er vi i hovedsak interessert i tre ting: mAP, inferenstid og eksempler på output. Alle de 235 bildene i testdatasettet blir kjørt gjennom nettverket, og for hvert bilde blir de predikerte markeringsrammene og tilhørende konfidensnivå tatt vare på. mAP regnes ut over alle disse prediksjonene. For mAP-verdiene av treningsdatasettet er de 200 første bildene brukt til utregning av mAP.

Inferenstiden måles fra bildet er i minnet, til NMS er utført på de predikerte markeringsrammene og disse er lagret i minnet. Dette inkluderer preprosessering av bildet. Inferenstiden regnes ut for hvert av bildene i testdatasettet, og inferenstiden som omtales er gjennomsnittet av disse.

I tillegg til mAP og inferenstid lagres også noen av bildene med de predikerte markeringsrammene tegnet inn.

Selve prediksjon av markeringsrammene til et bilde går som følger:

1. Bildet leses inn i minnet og preprosesseres. Denne preprosesseringen består av å normalisere bildet ved å trekke fra den gjennomsnittlige pikselverdien fra ImageNet, og bildet skaleres til den ønskede størrelsen.
2. Prediksjon av *regions of interest*. Bildet kjøres gjennom basisnettverket, RPN'et klassifiserer hvert punkt i feature map'en og de 300 beste forslagene blir funnet ved hjelp av NMS. Det testes flere verdier av antall forslag etter NMS.
3. Regionforslagene blir i grupper på 4 klassifisert av klassifiseringsnettverket, og de faktiske koordinatene til rammen blir regnet ut og lagret.
4. NMS blir utført på nytt for å bli kvitt overlappende markeringsrammer av samme klasse.

4.4 Fysisk

Treningen av modellene ble gjort stykkevis på colab [24] og på lokal maskin, så treningstiden til nettverkene er ikke fullt sammenlignbare. Inferens av alle modellene ble utført på en Windows maskin med Nvidia GTX1070 GPU og Intel i7-8600K prosessor. GPU ble brukt både under trening og testing av modellene.

5 Resultater

5.1 Vitenskapelige resultater

5.1.1 Grunnmodell

For å sørge for at resultatene til de forskjellige modellene kan sammenlignes tas det utgangspunkt i en grunnmodell. Alle modellene tar utgangspunkt i denne grunnmodellen, og har bortsett fra de nevnte forskjellene eksakt samme treningsdata, nettverksarkitektur og hyperparametre. Modellens parametre er gjengitt i [Tabell 1]

Parameter	Verdi
Basenettverk	VGG-16
Augmentering av treningsdata	Ingen
Oppløsning på bilder	800px[1]
Antall regionforslag under testing	300
Ankerstørrelser	[16, 32, 64, 128]
Ankerforhold	[[1, 1], [$3/6^2$, $2/6^2$], [$2/6^2$, $3/6^2$]]
Overlapsterskel for IoU ved NMS av prediksjoner	0.5

Tabell 1: Basismodellens parametre

5.1.2 Mobilenet

Det ble testet modeller med mobilenet som basisnettverk. Disse modellene opplevde en markant dårligere mAP, uten noen signifikant reduksjon i verken treningstid eller inferenstid, som vist i [Tabell 2]

Oppsett			Treningstid	Inferenstid	mAP
Arkitektur	Augmentering	Oppløsning			
VGG-16	-	800px	32.6 min	2.65 s	34.1[2]
VGG-16	-	400px	10.5 min	2.08 s	11.3
Mobnet	-	800px	38.8 min	2.69 s	22.9
Mobnet	-	400px	11.0 min	2.00 s	6.7

Tabell 2: Treningstid, inferenstid of mAP av modell med basenett Mobilnet

5.1.3 Reduksjon av regionforslag

Som beskrevet i delkapittel [2.3.2] vil en reduksjon av antall regionforslag generert fra RPN'et redusere inferenstiden til modellen. Som vist i [Tabell 3] ble det testet med 7 forskjellige antall regionforslag.

Modellsammendrag	Inferenstid	mAP
VGG-16, 800px, 300 forslag	2.65 s	34.1
VGG-16, 800px, 100 forslag	1.02 s	34.4
VGG-16, 800px, 50 forslag	0.614 s	34.8
VGG-16, 800px, 25 forslag	0.415 s	36.5
VGG-16, 800px, 15 forslag	0.317 s	37.8
VGG-16, 800px, 10 forslag	0.276 s	37.7
VGG-16, 800px, 5 forslag	0.246 s	34.0

[1] Bildet blir skaler slik at korteste side blir denne størrelsen
[2] Modellen ble trent i 41 epoker

Tabell 3: Treffsikkerhet med forskjellig antall regionforslag

5.1.4 Augmentering av datasett

For å få full effekt av augmenteringen ble disse modellene trent i 20 ekstra epoker, til totalt 60 epoker. Det ble eksperimentert med 3 forskjellige augmenteringer av treningsdataen:

1. Ingen
2. Skifting, skalering og rotering + beskjæring og strekking
3. Skifting, skalering og rotering

Vi ser av [Tabell 4] at augmentering fører til en økning av mAP på testdatasettet.

Modellsammendrag	Augmentering	mAP-40	mAP-60
VGG-16, 800px	1	34.1	-
VGG-16, 400px	1	11.3	8.1
VGG-16, 400px	2	9.6	13.6
VGG-16, 400px	3	11.3	10.4

Tabell 4: Treffsikkerhet med augmetering av treningsdata

5.1.5 Diluted convolutions

Ved omgjøring av siste blokk i VGG til **diluted convolutions**[26]. 'block4pool' er fjernet, og påfølgende konvolusjonslag har *dilation_rate = 2*. Vi ser av [Tabell 5] at mAP øker markant, samtidig som også trenings- og inferenstid gjør det.

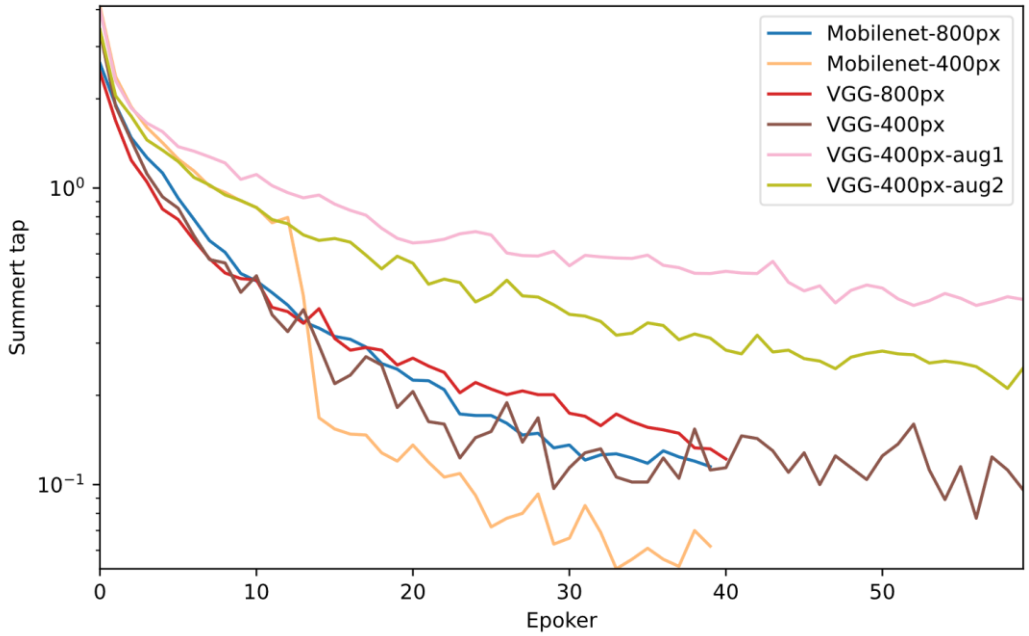
Modellsammendrag	Treningstid	Inferenstid	mAP
VGG-16, 400px, aug,	10.5 min	2.08 s	11.3
VGG-16, 400px, aug, diluted	22.9 min	3.07 s	18.0[3]

Tabell 5: Treningstid, inferenstid og mAP av modell med diluted convolutions

[3] Trent i bare 37 epoker

5.1.6 Verdi av tapsfunksjonen under trening

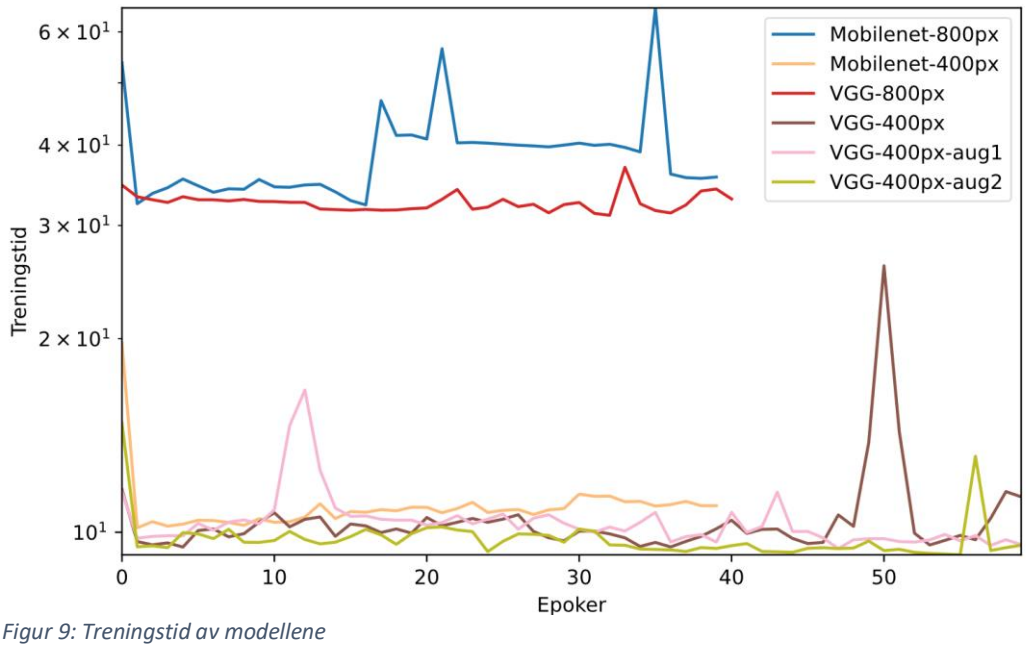
Det totale tapet til modellen etter hver epoke i treningen. Modellene trent på augmentert data oplever større tap gjennom hele modellen, som vist ved i [Figur 8]. Vi ser også at tapskurven begynner å flate ut etter 60 treningsepoker.



Figur 8: Totalt tap under trening

5.1.7 Treningstid

Tiden det tok å trene hver epoke (1000 steg) i minutter. Vi ser i [Figur 9] at modellene basert på bildestørrelse 400px trenes markant raskere.



Figur 9: Treningstid av modellene

5.2 Ingeniørfaglige resultater

I oppstarten av prosjektet ble det skrevet et visjonsdokument med blant annet mål om hvordan produktet skulle se ut etter endt prosjekt. Disse målene kan finnes i vedlegg A: Visjonsdokument, kapittel 5 og 6. Underveis i prosjektet ble store deler av dette nedprioritert og fokus ble satt på å trene en modell som kunne detektere trafikkskilt i sanntid.

Det kan likevel oppsummeres at prosjektet hadde mål om å produsere to konkrete produkter:

1. Et nevralt nettverk for å tolke trafikkskilt. I tillegg ble det stilt flere mål direkte relatert til dette.
    a. At nettverket har stor nok treffsikkerhet til å brukes til registrering av trafikkskilt
    b. At det kan utføres inferens med nettverket i sanntid.
    c. At nettverket er i stand til å klassifisere de forskjellige fartgrensene
    d. At nettverket er i stand til å klassifisere alle vanlige trafikkskilt
2. En androidapplikasjon for å ta i bruk nettverket.

Prosjektet har ført til disse resultatene for hvert av målene:

1. Det har blitt produsert ett nevralt nettverk som tolker trafikkskilt. Dette nettverket oppfyller noen av delmålene knyttet til det.
    a. Nettverket har ikke stor nok treffsikkerhet til å pålitelig registrere korrekte trafikkskilt. Ved justering av kriteriene for hvilke prediksjoner man beholder vil modellen kunne justeres til å ikke registrere mange feil, men da vil også modellen overse de aller fleste skiltene.
    b. Det er trent et nettverk med inferenstid 0.246, noe som tilsvarer opp til 4 fps
    c. Alle nettverkene er trent til å klassifisere 43 typer trafikkskilt, slik at også 1.d er oppfylt.
2. Det ble ikke utviklet en androidapplikasjon for å ta i bruk nettverket til dette prosjektet.

5.3 Administrative resultater

De administrative resultatene av prosjektet er gjengitt i vedlegg C: Prosjekthåndboka. Både ukesrapportering, møteinnkallinger og møtereferat ble gitt opp underveis i prosjektet, og erstattet med møter med veileder to ganger i uken. I Prosjekthåndboka finnes den komplette timelisten som erstatning for ukesrapportene.

Framdriftsplanen som ble satt opp i begynnelsen av prosjektet stemte ikke overens med hva som ble utført i praksis. De første ukene fulgte framdriftsplanen ganske godt, med planlegging av prosjektet og skriving av visjonsdokument.

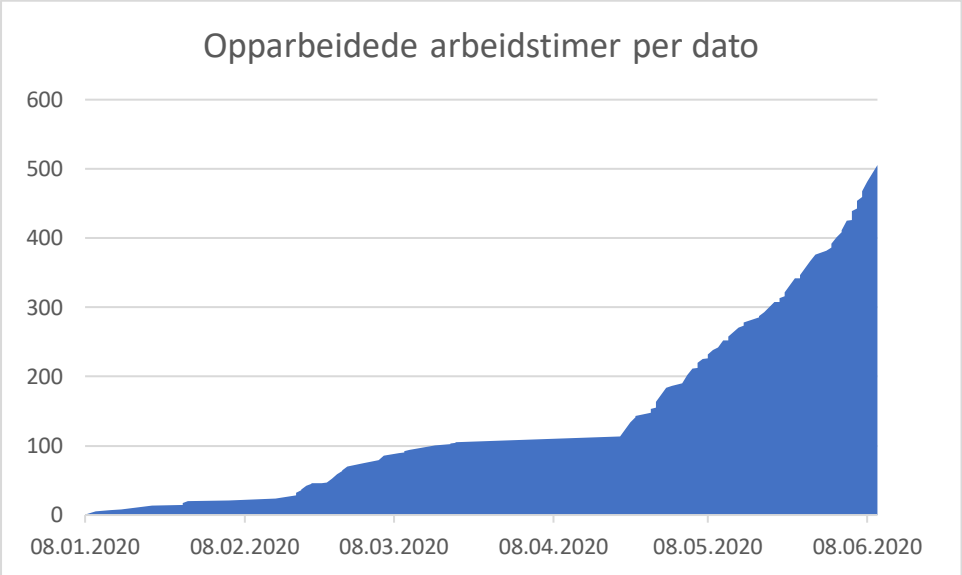
I [Tabell 6] ser vi at den reelle timebruken ikke var alt for langt unna den planlagte.

	Faktisk	Planlagt	Differanse
Implementasjon	210,5	190	+20,5
Dokumentasjon	12,5	50	-37,5
Research	73,5	80	-6,5
Planlegging	26,5	20	+6,5

Møte	8	10	-2
Rapport	174,5	150	+24,5
Sum	505,5	500	+5,5

Tabell 6: Planlagt mot faktisk timebruk per kategori

I tabell [Tabell 7] ser vi at de aller fleste arbeidstimene ble unnagjort sent i prosjektperioden.



Tabell 7: Kumulative arbeidstimer

6 Diskusjon

6.1 Diskusjon av Vitenskapelige resultater

Den beste modellen som ble trent og testet, er basismodellen med 10 regionforslag under testing. Denne modellen oppnådde 37.7 i mAP og en inferenstid på 0.276 sekunder. Det tilsvarer ca 3 fps i en reell applikasjon.

Resultatene viser at både redusering i antall regionforslag og augmentering av data øker mAP. Siden disse ikke ble kombinert og testet sammen vet vi ikke hvor god en eventuell modell kunne blitt.

Når vi augmenterer treningsdataen ser vi at vi får en markant økning i mAP. Dette tyder på at datasettet vårt er for lite, ved at modellen over tilpasser seg datasettet.

6.1.1 Underfitting

Datasettet det trenes på bærer preg av et stort antall klasser på forholdsvis lite data. Halvparten av klassene har mindre enn 10 forekomster i treningsdatasettet, og de to verste har bare 1 forekomst hver. Dette fører til at modellen ikke lærer seg de generiske trekkene ved klassen, men heller spesifikt med akkurat den forekomsten som finnes i treningsdatasettet. I tilfellet med deteksjon av trafikkskilt med Faster R-CNN vil boksen som modellen ser på som et trafikkskilt ikke havne nøyaktig på trafikkskiltet, slik at trafikkskiltets plassering i denne rammen i testsettet vil være ett av trekkene modellen lærer seg er viktig.

Dette ser vi ved at modellene presterer bedre når treningsdataen blir augmentert under treningen.

6.1.2 Ubalanse i datasettet

Den store variasjonen i antall forekomster av klassene introduserer et bias i modellen, slik at den vil foretrekke å kjenne igjen de klassene som forekommer ofte i datasettet. Dette fordi disse klassene vil ha mer å si for det gjennomsnittlige tapet i modellen. For å unngå dette bør datasettet utvides slik at alle klassene forekommer tilnærmet like ofte.

6.1.3 Spikes i treningstid

Variasjonen i treningstid for hver epoke skylles mest trolig lagringsmediets caching. For at prosjektet skulle bli mest mulig smidig og ta godt utnytte av google colab ble noen modeller trent stykkevis. Det er sannsynlig at første epoke etter hver treningsstart vil ta lengre tid fordi programmet må lese totalt 2,73 GB bildedata fra disk.

6.2 Diskusjon av Ingeniørfaglige resultater

Det ble ikke gjennomført en ordentlig oppdatering av hverken visjonsdokument eller kravspesifisering av oppgaven når situasjonen ble endret underveis. Dette har til dels først til at prosjektet ikke har hatt noe konkret mål annet enn å trene en god modell. Dette har fungert til en viss grad, i og med at prosjektet har i resultert i en modell som til dels utfører det som var håpet i starten av prosjektet.

6.3 Diskusjon av Administrative resultater

Det kommer godt frem av de kumulative arbeidstimene at den administrative siden av prosjektet har lidd litt under omstendighetene. Litt av det kan nok skyldes på den spesielle situasjonen som oppsto, både ved mistet av partner og påtvungent hjemmekontor. Likevel ser man at de planlagte

arbeidstimene ikke endte opp svært langt unna de faktiske arbeidstimene. Etter oppstart etter pausen i mars/april ble det inngått muntlig avtale med prosjektveileder å avholde korte nettmøter hver tirsdag og fredag. Ukesrapporter og møteinnkallinger og referat ble dermed nedprioritert. Dette har ført til at det ikke eksisterer god dokumentasjon på hva kravene til produktet er.

7 Konklusjon og videre arbeid

Problemstillingen som ble presentert i kapittel 1.1 stiller ingen konkrete krav til verken presisjon eller inferenstid av modellen. Inferenstiden til den raskeste modellen som ble testet, 0.246, kan benevnes som utførbar i sanntid. Samtidig peker resultatene på at enda raskere inferens er mulig ved å skalere ned inputbildet, dog på bekostning av mAP.

Det er spørsmålet om hvilken mAP som kvalifiserer til å detektere trafikkskilt som er vanskelig å svare på. Siden augmentasjon ga såpass stor økning i mAP for de modellene det ble testet på tyder det på at ett utbedret datasett kanskje kan oppnå *enda* bedre resultater. Med det datasettet og de modellene som er blitt trent i dette prosjektet er *ikke* deteksjon av trafikkskilt med Faster R-CNN mulig å utføre i sanntid med en *akseptabel* treffsikkerhet.

7.1 Videre arbeid

Datasettets begrensede omfang påvirker modellens prestasjon. For at nettverket skal bli markant bedre, bra nok til å oppnå en akseptabel treffsikkerhet, er datasettet nødt til å utvides. Samtidig er det flere iterasjoner av grunnmodellen som ikke ble testet sammen, slik at det er tenkelig å oppnå bedre treffsikkerhet med rett kombinasjon av de hyperparametrene som er blitt testet i dette prosjektet.

8 Referanser

[1] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.

[2] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, “Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark,” in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[3] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7310–7311.

[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” 2013.

[8] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, “Selective search for object recognition,” *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[9] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[10] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. Velasco-Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, “Deep learning vs. traditional computer vision,” 2019.

[11] J. Dai, Y. Li, K. He, and J. Sun, “R-fcn: Object detection via region-based fully convolutional networks,” in *Advances in neural information processing systems*, 2016, pp. 379–387.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” *Lecture Notes in Computer Science*, p. 21â€“37, 2016. [Online]. Available: http://-dx.doi.org/10.1007/978-3-319-46448-0_2

[13] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv*, 2018.

[14] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. DollÃ¡r, “Microsoft coco: Common objects in context,” 2014.

[15] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu, “Object detection with deep learning: A review,” 2018.

[16] S. Soo, “Object detection using haar-cascade classifier,” *Institute of Computer Science, University of Tartu*, pp. 1–12, 2014.

[17] P. C. U. Murillo, R. J. Moreno, and J. O. P. Arenas, “Comparison between cnn and haar classifiers for surgical instrumentation classification,” *Contemporary Engineering Sciences*, vol. 10, no. 28, pp. 1351–1363, 2017.

[18] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2018.

[20] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” 2019.

[21] S. L. N. Rafael Padilla and E. A. B. da Silva, “Survey on performance metrics for object-detection algorithms,” 2020. [Online]. Available: https://github.com/rafaelpadilla/Object-Detection-Metrics

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[23] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[24] Google. Colaboratory: Frquently asked questions. [Online]. Available: https://-research.google.com/colaboratory/faq.html

[25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

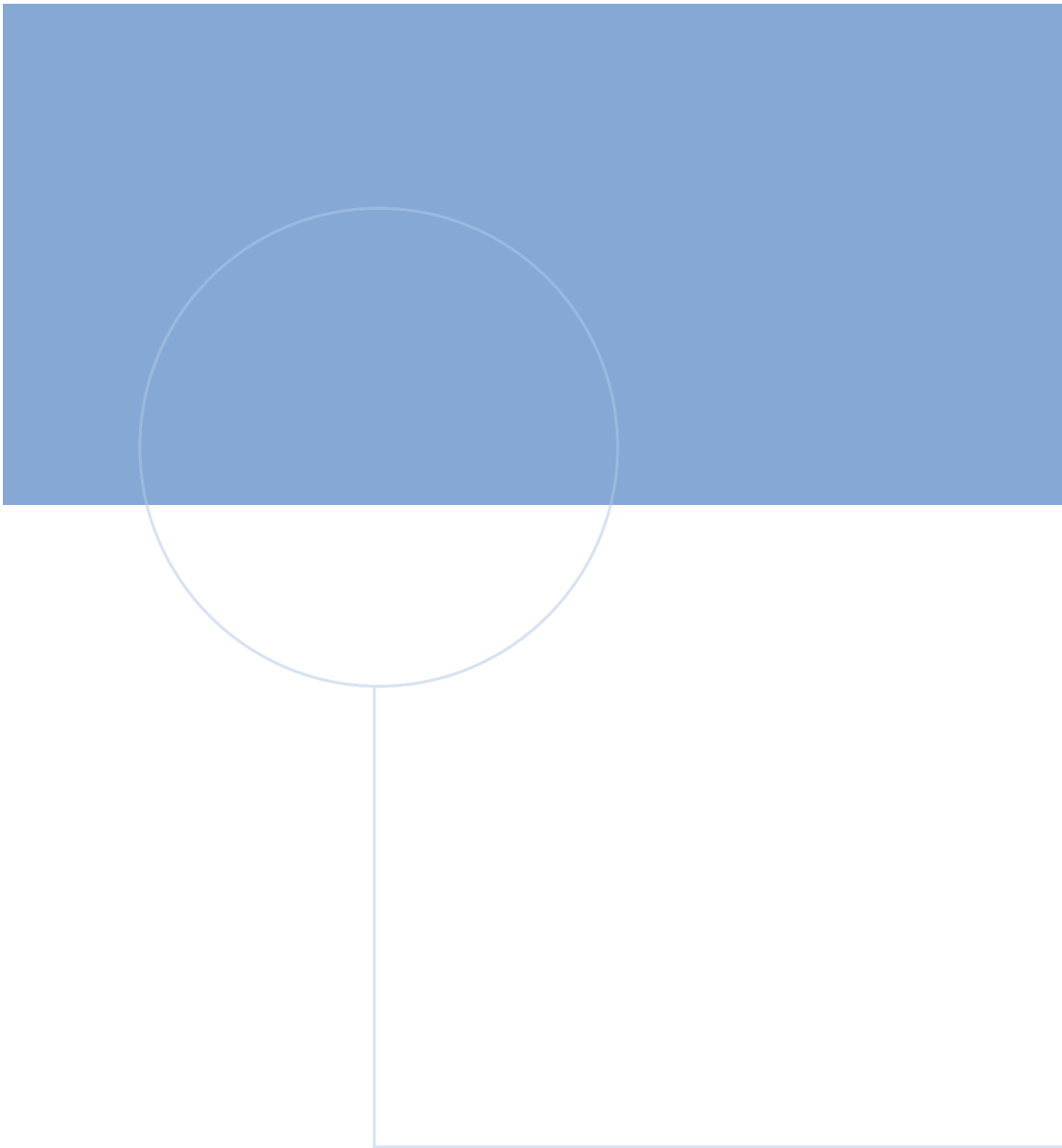
[26] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” 2015.

9 Vedlegg

Vedlegg A: Visjonsdokument

Vedlegg B: Systemdokumentasjon

Vedlegg C: Prosjekthåndbok



NTNU

Kunnskap for en bedre verden