NTNU
Kunnskap for en bedre verden

Hovedrapport - deteksjon og klassifikasjon av trafikkskilt

Av Thomas Bjerke og Trym Grande 23.11.2020

Innholdsfortegnelse

Innholdsfortegnelse **2**

Introduksjon **3**

Tidligere relevant arbeid **4**

CarND Traffic Sign Classisier Project 4

hoanglehaithanh/Traffic-Sign-Detection 5

Metode **5**

Resultat **9**

Resultatmåling 9

Trening 9

Test 10

Diskusjon **12**

Konklusjon og referanser **13**

Introduksjon

Denne oppgaven er symbiotisk med Systemutviklingsfaget som foregår parallelt med maskinlæringsfaget. Noen av forutsetningene og valgene i dette prosjektet kommer derfor fra prosjektet i systemutvikling. Oppgaven i systemutvikling er følgende:

"Oppgaven går ut på å lage en applikasjon på Android (Java eller Kotlin), hvor man har en telefon på oppe på dashboardet i en bil og bruker kameraet til å kontinuerlig filme veien og tolke trafikkskiltene (eller fotobokser) ettersom de dukker opp. Samtidig skal koordinatene til skiltene/fotoboksene bli lagret sammen med skiltet/boksen vha. telefonens GPS. Tolkningen av de forskjellige skiltene skal skje ved bruk av OpenCV og trene opp et nevralt nett; vi konsentrerer oss i første omgang kun om fartsgrenser og vil utvide etter hvert. Det er en fordel om studenten(e) har tilgang til egen bil, men ikke et krav, og vi vil være behjelpelig med å skaffe rådata til opptrening av modellen dersom det som finnes åpent ute på nett ikke er tilstrekkelig.

Bruksområdene til en slik applikasjon er nær sagt uendelige, men i første omgang er det et håp om kunne bruke data til å bidra til å føre opp skilt/bokser i OpenStreetMap, som er et crowd sourcet alternativ til løsninger som Google Maps. Bare det å kunne vite fartsgrensen når en bruker OpenStreetMap vil være til stor hjelp."

Oppgaven har blitt todelt, og målet med dette prosjektet innebærer da kun maskinlæringsdelen. Dette vil si detektering og klassifisering av skilt. Problemet som skal løses i dette prosjektet handler altså om hvordan man kan tolke et forbipasserende trafikkskilt med fartsgrense mest mulig effektivt. Dette er både med tanke på at det skal kjøre i sanntid, og at det skal kunne prossesseres på en mobiltelefon. I denne rapporten ser vi på hva som har blitt gjort og hva vi har kommet fram til i løpet av prosjektet.

Tidligere relevant arbeid

Det første vi gjorde ved starten av prosjektet var å se gjennom lignende prosjektarbeid som var gjort tidligere. Her fant vi mye forskjellig med varierte implementasjoner og mål. I vårt prosjekt krevde vi OpenCV til deteksjon, video framfor individuelle bilder, datasett med fartsgrenseskilt, og rask nok maskinlæringsmodell til å kunne kjøre på mobiltelefon. Følgende prosjekter ble brukt til inspirasjon:

- CarND Traffic Sign Classisier Project
- ENPM 673 Traffic Sign Recognition
- vamsiramakrishnan/TrafficSignRecognition
- hoanglehaithanh/Traffic-Sign-Detection
- jacobssy/Traffic_Sign_detection

CarND Traffic Sign Classisier Project

I dette prosjektet har åtte personer jobbet sammen for å bruke konvolusjon for klassifisering av trafikkskilt. Input til klassifiseringsmodellen er et bilde, og prosjektet inneholder ikke kode for detektering, altså fungerer det ikke på video. Med en treffsikkerhet på 97% var det likevel interessant for oss å gå gjennom prosjektet for å finne ut om vi kunne bruke noen av de samme metodene for klassifiseringsdelen.

Det første vi la merke til var at modellen var veldig komplisert. Vi hadde ganske dårlig tid i forhold til hva vi prøvde å oppnå, så det passet dårlig for oss. I tillegg, siden CarND-prosjektet ikke hadde som mål å fungere på video, så hadde de ikke tatt noe særlig hensyn til hvor rask modellen var, da dette ikke var nødvendig for dem. Vi kom derfor fram til at det ikke var mye fra dette prosjektet vi kunne bruke i vår klassifisering.

Selve datasettet som ble brukt derimot, virket nesten perfekt for oss. Skiltene var veldig like norske skilt, og hvert skilt i datasettet var merket med skilttype. Vi endte derfor opp med å bruke samme datasett i vårt eget prosjekt.

hoanglehaithanh/Traffic-Sign-Detection

Dette prosjektet inneholder både deteksjon og klassifisering av trafikkskilt, og fungerer derfor på video. Klassifiseringen er gjort ved bruk av en SVM-modell (Support Vector Machine). Denne modellen så ganske mye enklere ut å jobbe med, og etter å ha undersøkt litt videre, fant vi ut at SVM kunne være aktuell for oss, da den også er veldig rask.

En stor mangel i forhold til vårt mål, var at modellen i dette prosjektet ikke skilte mellom ulike fartsgrenseskilt, noe som var en hovedprioritet for oss. I tillegg fant vi ut etter å ha testet den på en video som vi tok selv at den rett og slett fungerte ganske dårlig. Etter å ha gjort litt videre undersøkelser, hadde vi mange ideer til ting vi kunne gjøre for å få den til å fungere bedre for vår tiltenkte bruk, og kom fram til at denne klassifiseringsmodellen kunne være et godt utgangspunkt for oss.

Metode

Vår metode for tolkning av trafikkskilt består at to faser; deteksjon og klassifisering. I deteksjonsfasen prøver vi å oppdage skilt i videoen, men tar ikke hensyn til hvilket skilt det er. Det er det vi gjør i klassifiseringsfasen. Vi deler derfor opp dette kapittelet i to underkapitler.

Deteksjon

I deteksjonsfasen bruker vi OpenCV i fem trinn for å oppdage regioner i hver videoframe som ligner på trafikkskilt. OpenCV står for Open Source Computer Vision Library og er, som navnet tilsier, et open source-bibliotek for datasyn og maskinlæring. OpenCV har grensesnitt til mange forskjellige programmeringsspråk, og vi valgte å bruke Python. Grunnen til det var hovedsakelig at man i Python har tilgang til en del biblioteker som ville komme godt med i løsningen av prosjektet, som for eksempel NumPy og forskjellige maskinlæringsbiblioteker. I tillegg virket det naturlig å bruke samme programmeringsspråk som vi hadde brukt i undervisningen og øvingene i TDAT3025. Vi skal nå se nærmere på hvert av de fem trinnene i deteksjonsfasen.

I første trinn tar vi et bilde fra inputvideoen og øker kontrasten. Dette er både for å enklere kunne skille mellom komponenter med ulik farge, og for å klargjøre for trinn to.

Andre trinn består av å fjerne farger som vi ikke trenger å ta hensyn til videre. Vi fjerner for eksempel svart og grønn fordi ingen norske trafikkskilt har konturer med disse fargene, så vi trenger derfor ikke å sjekke om disse regionene er trafikkskilt, da vi allerede vet at det ikke kan være det.

I tredje trinn bruker vi Laplacian of Gaussian for å finne kanter i bildet, altså for å skille ulike objekter i bildet fra hverandre. Laplacian of Gaussian er en algoritme som detekterer kanter basert på den andrederiverte av et bilde. Altså hvis vi har en graf over intensitetsverdiene til hver piksel i et gråtoneskalabilde, vil LoG se på hvor den andrederiverte av grafen krysser null, og tolke dette området som en kant.

Deretter, i fjerde trinn, gjør vi bildet binært for å enklest mulig kunne finne konturer. Det vil si at vi bruker kantene som vi fant med Laplacian of Gaussian, og gjør kantene hvite, mens resten blir svart. Det gjør at konturer kommer veldig tydelig fram.

Femte og siste trinn består av å gå gjennom konturene fra trinn 4, og finne ut hvilke av disse som ligner på sirkler eller ellipser, nettopp fordi norske fartsgrenseskilt alltid har sirkel-form. Grunnen til at vi også leter etter ellipser er fordi sirkler blir til ellipser dersom de er sett på skrå. Når en sirkel eller ellipse-lignende kontur er funnet, konkluderer vi med at regionen består av et skilt, og sender da den samme regionen fra originalbildet videre til klassifisering.

Klassifisering

Det er i klassifiseringsfasen at skiltene som blir funnet i deteksjonsfasen faktisk blir tolket. Input til klassifiseringen er altså et bilde av noe som deteksjonsfasen tror er et skilt. For å tolke skiltene bruker vi en opptrent SVM-modell. Hovedgrunnen for dette valget var først og fremst hastigheten, i og med at sluttmålet er å få den til å fungere i sanntid. Modellen er trent på et datasett bestående av tyske fartsgrenseskilt som er veldig like, men ikke identiske med norske fartsgrenseskilt.

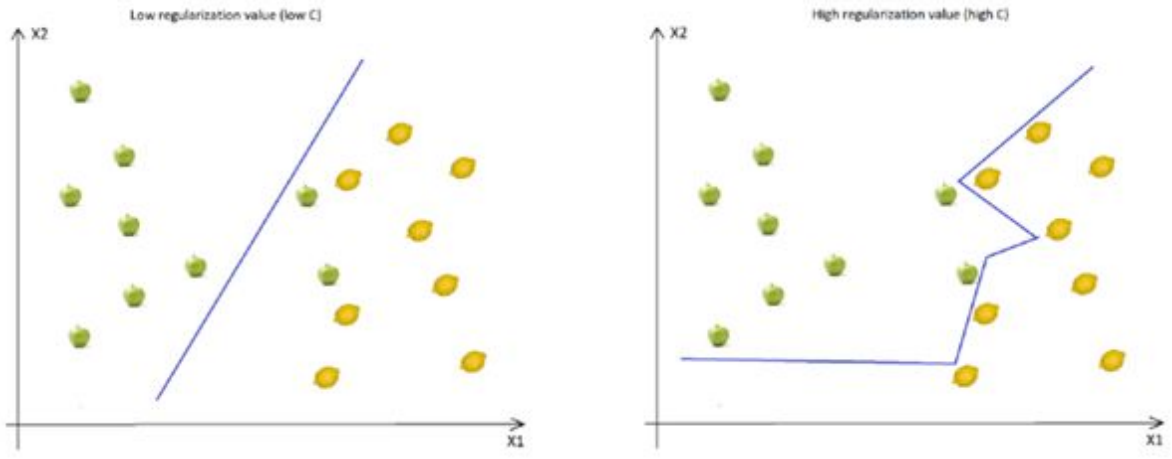
SVM står for Support Vector Machine, og er en supervised maskinlæringsmodell for klassifisering av kategorier. Modellen fungerer slik at den representerer dataen i et vektorrom, og prøver å finne det hyperplanet som på best måte skiller dataen inn i ulike klasser. Det er vanlig blant maskinlæringsalgoritmer å prøve å lære seg de mest vanlige karakteristikkene for hver klasse, for så å basere klassifiseringen på de karakteristikkene. SVM gjør på en måte motsatt. Den finner de eksemplene som er mest like hverandre fra ulike

klasser, og forsøker å finne det optimale hyperplanet for å skille disse. De eksemplene som ligner mest på andre klasser blir kalt support vektorer, og ligger i et n-dimensjonalt vektorrom, hvor n er antall attributter for hvert eksempel. Det optimale hyperplanet er det som gir størst avstand til nærmeste support vektor fra hver klasse. SVM finner to hyperplan som ideelt sett ikke har noen supportvektorer mellom dem, og snittet av disse to blir optimalt hyperplan, ofte kalt decision boundary.

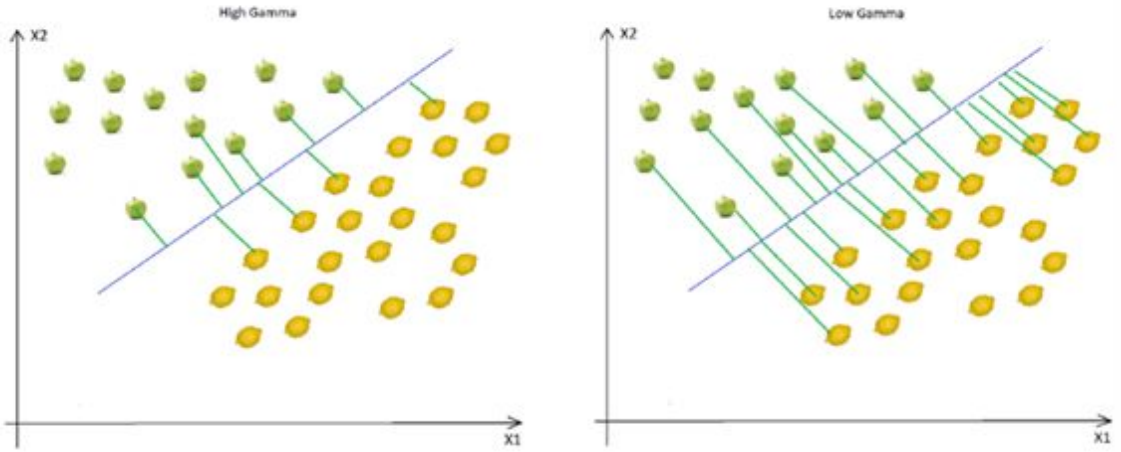
Det er ikke alltid like lett å finne et tydelig skille mellom slike support vektorer, og dette løser SVM ved bruk av kjernefunksjoner. Kjernefunksjoner er mappingsfunksjoner som har som mål å skille de ulike klassene mest mulig fra hverandre. Noen av de vanligste kjernefunksjonene er; polynomisk kjernefunksjon, Gaussisk kjernefunksjon, Radial Basis Function (RBF), Sigmoid, og lineær kjernefunksjon.

Selv med kjernefunksjoner vil det ofte være vanskelig å finne to hyperplan uten noen datapunkter mellom dem, så vi trenger derfor noen trade-offs, toleranse for unntak. Dette styres i SVM av to parametere; regulariseringsparameteren, og gamma.

Reguleringsparameteren kalles i Python for C, og forteller modellen hvor viktig det er å unngå misklassifiseringer. Desto høyere C man bruker, desto mindre blir distansen mellom hyperplanet og supportvektorene. Eksempel med epler og sitroner:



Gammaparameteren bestemmer hvor langt unna det plausible hyperplanet en supportvektor kan ligge og fortsatt bli tatt med i beregningen av hyperplanet. Hvis man bruker høy gammaverdi, vil SVM kun ta med punkter som ligger nærme det plausible hyperplanet i beregningen:



De grønne strekene på bildet brukes for å vise hvilke eksempler som blir tatt med i beregningen.

I oppsettet av SVM-modellen var det altså tre variabler vi måtte ta hensyn til: kjernefunksjon, reguleringsparameteren, og gammaparameteren. I starten testet vi bare med ulike verdier som virket å gi mening ut ifra vår forståelse av hvordan disse variablene påvirket modellen. Etter hvert fant vi ut at det ville være lurt å lage et gridsearch for å optimalisere.

Gridsearchet vi lagde tok maskinlæringsmodellen og en tabell for hver av variablene som input. Den kryssjekket da variablene mot hverandre, og sjekket hvilken kombinasjon som ga best treffsikkerhet. I starten hadde vi kun med RBF som kjernefunksjon i gridsearchet. Grunnen var at etter å ha lest om de ulike kjernefunksjonene kom vi fram til at det ga mest mening å bruke RBF.

Etterhvert fant vi en vitenskapelig rapport hvor de testet ulike kjernefunksjoner for å se hvilken som ga best nøyaktighet for skiltklassifisering med SVM (Shi 2008). Der fant de ut at det var lineær kjernefunksjon som ga best nøyaktighet, noe de også ble overrasket over ("Surprisingly, the basic linear kernel performed better than other kernels."). Vi bestemte oss derfor for å likevel ta med andre kjernefunksjoner i gridsearchet, og da den var ferdig å kjøre, viste det seg at det stemte at lineær kjernefunksjon ga best treffsikkerhet. Ved bruk av lineær kjernefunksjon trenger ikke SVM de andre variablene, så de ble derfor ikke tatt hensyn til videre. Lineær kjernefunksjon er også det som gir raskest trening og klassifisering, noe som passet veldig bra for oss i og med at målet var å få modellen til å kjøre på mobiltelefon i sanntid.

Resultat

Resultatmåling

For å få til et mål på hvor god modellen var, implementerte vi en metode som viste både treffsikkerhet for datasettet og en confusion matrix. Treffsikkerheten regner ut antall frames der modellen gjetter riktig kontra feil.

Confusion matrixen viser hvordan treffsikkerheten er fordelt over de forskjellige skiltene i datasettet. Den viser label/target for hver rad og prediksjon for hver kolonne. Her ser vi altså nøyaktigheten for hvert skilt. Skiltene er sortert slik: [ukjent skilt-lignende objekt, 20, 30, 40, 50, 60, 70, 80, 100, 120].

Trening

Funksjonen trener først opp modellen på datasettet, og tester deretter på en ny, separert del av datasettet. Dette gjøres for å unngå overtilpasning der modellen "husker" bildene i datasettet. Resultatet måles i prosent, og ble i vårt tilfelle 93.56%. Dette reflekterer hvor god modellen er for sitt eget datasett, men ikke nødvendigvis i en reell setting med andre forhold.
Her ser vi stort sett verdier langs diagonalen, som viser god nøyaktighet.

confusion matrix:
[[296 0 0 0 0 0 0 0 0 0]
[0 19 0 0 0 0 0 0 0 0]
[2 0 202 0 8 0 0 2 0 0]
[1 0 0 116 0 0 0 0 0 0]
[0 0 19 0 213 2 0 9 1 0]
[1 0 0 0 2 125 0 10 0 0]
[0 0 0 0 0 0 181 0 0 1]
[1 0 3 0 4 10 0 154 4 1]
[0 0 1 0 2 1 0 4 124 4]
[0 0 0 0 4 0 0 2 7 110]]

Test

Vi gjorde det samme for nøyaktigheten for et tenkt produksjonsmiljø. I dette tilfellet tok vi en video modellen ikke hadde sett før, og lagra alle skiltene i videoen med antatt label. Etter litt manuelt arbeid med å korrigere riktig label, kunne vi da kjøre modellen på nytt med dette test-datasettet, og få nøyaktigheten på samme måte som med funksjonen vi brukte på testdatasettet.

accuracy: 86.31 %
confusion matrix:

```
[[  1   0   0   0   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0   0   0]
 [  0   1   0  65   0   0   0   0   0   0]
 [  1   1   7   0 109  11   0   5   0   0]
 [  0   0   1   0   7  63   0   0   0   0]
 [  1   1   4   0   4   0  69   2   0   0]
 [  0   0   0   0   0   0   0   1   0   0]
 [  0   0   0   1   0   0   0   0   0   0]
 [  0   0   0   0   1   0   0   0   0   0]]
```

I dette tilfellet ser vi at 50 og 60-skilt blir forvekslet med hverandre omtrent 1/10 av gangene. Grunnen til at 20, 30, 80, 100, og 120 mangler her er fordi vi ikke rakk å samle inn nok testdata i produksjonsmiljø for disse. Disse har allikevel blitt brukt under trening, og mangler kun testing. I og med at dataen allerede har blitt brukt for trening, vil testing med den samme dataen gi en overdreven nøyaktighet grunnet overtilpasning, og er derfor ikke aktuelt.

Se vedlegg for demovideo.

Her er noen bilder fra noen av demovideoene:


40
40


50
50
706
E 6

Diskusjon

Resulterende nøyaktighet avhenger av parametrene vi har brukt. Dette er noe vi kunne ha optimalisert mer ved å utvide gridsearch-et vi brukte for å finne optimale parametre. Hadde vi hatt bedre tid, kunne vi altså søkt gjennom et større kombinasjonsområde med parameter. Da kan det hende det hadde vist seg at en annen kjernefunksjon hadde fungert bedre enn en lineær. Dette hadde muligens gitt bedre nøyaktighet, selv om vi egentlig tviler i og med at den vitenskapelige rapporten også kom fram til at lineær kjernefunksjon ga best treffsikkerhet.

Datasettet inneholder varierende mengde data for hver skilttype. Dette varierer mellom ca. 2200 og 1200 bilder per type, med unntak av 20-skilt som kun har omtrent 200. Bildene varierer også mye i lysforhold. Spesielt mangler skiltene fra 60 og opp like lyse og klare bilder som de andre. Dette kan hindre treffsikkerheten til å nå fullt potensial. I tillegg, ser vi at det er ulikt antall mørke bilder i hver klasse, som kan føre til at modellen feiltrenes og blir mer utsatt for å feilklassifisere mørke skilt ved å utelukke klassene med få mørke skilt. Det

bør også nevnes at skiltene i datasettet er tyske, men testingen blir gjort mot norske skilt, som igjen kan føre til unøyaktige resultater. De tyske ligner på norske, men er ikke helt identiske. Datasettet vi brukte mangler også 90- og 110-skilt.

I starten hadde vi problemer med at modellen klassifiserte en del objekter som ikke var skilt som fartsgrenseskilt. Vi bestemte oss da for å legge inn i koden at alle bilder som ble tolket som skilt skulle legges inn i en egen mappe, og deretter la vi inn objektene som ikke var skilter i UNKNOWN SIGN-LIKE-mappa i datasettet. Det løste problemet med at ikke-skilt ble tolket som skilt. Det var stort sett de samme type ikke-skilt-objektene som ble tolket som ble tolket som skilt på kryss av forskjellige testvideoer vi brukte, så etter å ha lagt inn ikke-skilt fra èn video i UNKNOWN SIGN-LIKE-mappa, så ble resultatet også mye bedre på andre videoer.

Deretter bestemte vi oss for å også utvide og forbedre resten av datasettet ved å kjøre gjennom mange forskjellige videoer, og legge alle tolkede skilt i sine respektive mapper i datasettet. Treffsikkerheten ble da betydelig bedre for andre videoer.

Som videre arbeid, vil det være relevant med et bedre og mer "balansert" datasett der hver skilttype er representert likt, der norske skilt blir brukt, og der kvaliteten og lysforholdet er godt nok. Bedre finstilling av parameter hadde også vært relevant for å forbedre modellen utenfor det vi fikk tid til.

Konklusjon og referanser

Alt i alt har gjennomføringen av prosjektet vært en svært lærerik prosess. Vi har gått i dybden på Support Vector Machine-modellen, og føler vi har fått en helhetlig forståelse av den, og alt som inngår av kjernefunksjoner og parametre. Vi har lest mange vitenskapelige artikler og rapporter om SVM, og har på den måten fått oversikt over styrker og svakheter, samt diverse observasjoner gjort av andre. Videre har vi fått testet mange av disse observasjonene, og fått merke modellens styrker og svakheter i praksis.

I tillegg har vi også lært mye om andre maskinlæringsmodeller gjennom alle undersøkelsene som ble gjort under oppstarten av prosjektet hvor vi skulle finne ut hvilken modell som passer best for oppgaven.

Når det kommer til datasett, har vi fått bedre forståelse for hvor svakhetene kan ligge, samt metoder man kan bruke for å rette opp i disse.

Gitt tiden vi hadde på å fullføre prosjektet, er vi godt fornøyde med resultatet. Vi har oppnådd høy treffsikkerhet ved å bruke en rask modell med rask kjernefunksjon, og ser stort potensial for å få den til å fungere på mobil, noe vi fortsetter å jobbe videre med i systemutviklingsprosjektet.

- Shi, M., Wu, H., Fleyeh, H. *Support vector machines for traffic signs recognition* (2008) https://ieeexplore.ieee.org/document/4634347 [lest 06.11.2020]
- Prosjekt nevnt i "tidligere relevant arbeid"
- Gudigar A., Jagadale B.N., P.K. M., U. R. (2012) Kernel Based Automatic Traffic Sign Detection and Recognition Using SVM. In: Mathew J., Patra P., Pradhan D.K., Kuttyamma A.J. (eds) Eco-friendly Computing and Communication Systems. ICECCS 2012. Communications in Computer and Information Science, vol 305. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-32112-2_19 [lest 06.11.2020]
- Dawson, C. *SVM Parameter Tuning* (2019) https://towardsdatascience.com/a-guide-to-svm-parameter-tuning-8bfe6b8a452c [lest 04.11.2020]