Norges miljø- og biovitenskapelige universitet

Masteroppgave 2019 30 stp
Fakultet for realfag og teknologi
Ivar Maalen-Johansen

Pikselbasert arealklassifisering av urbane områder med hyperspektrale flybilder og maskinlæring

Pixel-based area classification of urban areas with hyperspectral airborne images and machine learning

Annette Primstad og Åsmund Stemme
Geomatikk

Sammendrag

Kartlegging av arealtyper med fjernmåling kan være utfordrende i tettbebygde byområder på grunn av store variasjoner innen arealtyper. Flybårne hyperspektrale bilder er på vei inn i fjernmålingsindustrien, men disse inneholder så mye informasjon at det kan være vanskelig å utnytte potensialet i dataene optimalt. Mange peker på at maskinlæring kan være løsningen på dette. I denne oppgaven ble flere hyperspektrale datasett over Oslo brukt til å teste ut pikselbasert arealklassifisering, og vurdere noen utvalgte maskinlæringsalgoritmer opp mot hverandre.

Bildene som ble benyttet var et radians- og et reflektans-datasett, begge i områdene VNIR og SWIR. VNIR hadde en romlig oppløsning på 0,3 m og bestod av 186 bånd. SWIR hadde romlig oppløsning på 0,7 m og bestod av 288 bånd. Vi hadde dermed totalt 474 bånd å jobbe med. Målet var å undersøke hvor bra de tre maskinlæringsalgoritmene CNN, SVM og logistisk regresjon presterte på arealklassifisering med datasettene. De ulike datasettene ble og sammenlignet for å se hvilket som ga best resultat på vårt testområde.

Vi valgte å plukke ut to områder, ett for trening og testing av modeller fra algoritmene, og ett for endelig validering. Siden alle de tre utvalgte algoritmene gjør styrt klassifisering, utarbeidet vi bakkefasiter for de to områdene. Det ble valgt å se på permeable og impermeable flater, og skille noen klasser innen vegetasjon og steinbasert materialer. Disse klassene ble asfalt, betong, busk, gress, grus og tre. Vi forholdt oss kun til enkeltpiksler i selve klassifiseringen, og det ble satt en nedre grense for minst 15000 piksler i hver klasse i fasiten.

Radians med 466 bånd presterte best i treningen, og ble valgt for endelig validering. CNN ga en nøyaktighet på 51 % og en kappa-koeffisient på 38 % for å skille de seks klassene. SVM ga en nøyaktighet på 74 % og en kappa-koeffisient på 65 %, og logistisk regresjon ga en nøyaktighet på 86 % og en kappa-koeffisient på 80 %. Dette resultatet viser at det er mulig å gjennomføre pikselbasert arealklassifisering med hyperspektrale bilder, og med dette datasettet på de 6 utvalgte klassene presterte den simpleste algoritmen, logistisk regresjon, best.

Abstract

Mapping area types with remote sensing may be challenging in densely populated urban areas because of large variations in area types. The technique of airborne hyperspectral images is entering the remote sensing industry, but these types of data contain so much information that it may be difficult to fully obtain the potential such data provides. Many looks to machine learning for a solution to solve this obstacle. In this thesis, hyperspectral datasets over the city of Oslo were used to test out pixel-based area classification, and to consider and compare the potential of a few chosen machine learning algorithms.

The images used for this study were a radiance and a reflectance dataset, in both the VNIR and SWIR range. VNIR had a spatial resolution of 0.3 m and consisted of 186 bands. SWIR had a spatial resolution of 0.7 m and consisted of 288 bands. We then had a total of 474 bands to work with. The goal was to investigate how well the three machine learning algorithms CNN, SVM and logistic regression performed on area classification with the datasets. The various datasets were also compared to see which one gave the best results for our chosen test area.

We picked two areas of interest, one for training and testing the models of the algorithms, and one for final validation. Because the selected algorithms all were supervised classifiers, we prepared ground truth for the two areas. It was chosen to inspect permeable and impermeable surfaces and distinguish between a few classes within vegetation and stone-based materials. These chosen classes were asphalt, concrete, grass, gravel, shrub and tree. For classifications single pixels only was used, a lower limit for each class was set to at least 15,000 pixels.

A radiance set of 466 bands performed best and was therefore selected for final validation. For classification with six classes, CNN gave an accuracy of 51% and a kappa coefficient of 38%. SVM gave an accuracy of 74% and a kappa coefficient of 65% and logistic regression an accuracy of 86% and a kappa coefficient of 80%. This result shows that it is possible to classify urban areas with hyperspectral images. On the 6 selected classes with this dataset, the simplest algorithm, logistically regression, performed best.

Forord

Med denne oppgaven markeres avslutningen på vår masterutdanning innen geomatikk ved Norges miljø- og biovitenskapelige universitet (NMBU). Denne oppgaven er skrevet våren 2019 i samarbeid med Plan- og bygningsetaten i Oslo kommune.

Først ønsker vi å takke den hyggelige gjengen ved Plan- og temakartenheten i Geodataavdelingen på PBE i Oslo, hvor vi har sittet og skrevet store deler av denne våren. Dere har vært veldig inkluderende og oppmuntrende underveis, selv om det til tider har vært vanskelig å forstå hva vi har drevet med. Det har vært hyggelig å vite at vi alltid har være velkommen til gode lunsjsamtaler og kakefredag. I tillegg ønsker vi å gi en ekstra takk til vår bi-veileder fra PBE, Webjørn Finsland for gode råd og innspill underveis, og Mario Gil Sanchez for tilgang maskinpark og software som ble helt essensielt for å kunne gjennomføre denne oppgaven.

En stor takk rettes også til vår andre biveileder Ingunn Burud, som har bidratt til å øke vår forståelse for hyperspektrale bilder, og for at du alltid har satt av tid til å diskutere spørsmål med oss, samt gitt oss gode tips angående skriving.

Videre ønsker vi å takke vår hovedveileder Ivar Maalen-Johansen som har gitt god veiledning når det har vært nødvendig. Til slutt rettes det også en takk til dataleverandøren Terratec som har vært behjelpelige med å gi klarhet i spørsmål rundt de hyperspektrale dataene.

Norges miljø- og biovitenskapelige universitet

Ås, 14.mai 2019

Annette Primstad & Åsmund Stemme

Forkortelser

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CSV	Comma Separated Value
LR	Logistisk regresjon
MIR	Mid InfraRed
ML	Maskinlæring
NDVI	Normalised Difference Vegetation Index
NIR	Near InfraRed
NN	Nevralt nettverk
nm	Nanometer (=10^{-9})
PCA	Principal Component Analysis
ReLU	Rectified linear unit
RGB	Red, Green and Blue
SVM	Support Vector Machine
SWIR	Shortwave Infrared
TIFF	Tagged Image File Format
VNIR	Visible Near Infrared
μm	Mikrometer (=10^{-6})

Innholdsfortegnelse

Figurer

1 Innledning

1.1 Bakgrunn

Byer endres hele tiden, og allerede på 90-tallet ble det klart at byenes grøntarealer blir påvirket negativt av fortetting. Derfor har en etter den tid jobbet med å bevare slike grøntområder, siden disse områdene er med på å sikre rekreasjonsområder, det biologiske mangfoldet, håndterer flom og overvann osv. (Thorén and Nyhuus, 1994, Mueller, 2019).

For å dokumentere utviklingen og kvantifisere endringene er fjernmåling et viktig verktøy.
Byrådet i Oslo har et ønske om å kontinuerlig vite status på grøntareal i kommunen. I Oslo tas det flyfoto hvert år for konstruksjon av grunnkart. Her kan man til en viss grad identifisere grønne områder, men på grunn av at disse bare registrer synlig lys i 3 bånd (vanlige RGB-fargebilder) er det begrensninger. Ny teknologi gir mulighet for å ta bilder med mange bånd, og de registrerer også lys med andre bølgelengder enn det som er synlig for øyet. Dette gir nye muligheter for arealklassifisering, og man kan tenke seg både sikrere klassifisering og nye muligheter for å skille mellom ulike materialer.

Bakgrunnen for denne oppgaven er at Oslo kommune i 2017 kjøpte et sett med hyperspektrale flybilder over deler av Oslo. Dataene ble kjøpt inn for å undersøke bruksområdene for hyperspektrale data med tanke på grøntkartlegging.
Våren 2018 ble det skrevet to masteroppgaver med utgangspunkt i dette datasettet. Den ene oppgaven tok for seg klassifisering av tretyper (Røstad, 2018) og den andre estimering av biomasse for trær (Barane, 2018). Hyperspektrale flybilder utgjør enorme datamengder ettersom de består av flere hundre bånd. Dette betyr at man ikke kan bruke klassiske bildeanalyseteknikker, men bør ty til multivariat analyse og maskinlæringsbaserte metoder (Barane, 2018, Røstad, 2018).

I Oslo er det på grunn av fortetting store områder med impermeable flater som asfalt og byggmaterialer som gjør det vanskelig for vann å trekke ned i bakken. Dette fører til uønskede vannmengder og i verste fall flom. Oslo kommune er opptatt av å kunne forutsi flomutsatte områder ved hjelp av flommodellering slik at en kan vite hvor i byen slike problemer vil oppstå. I tillegg til å se på forholdet mellom vegetasjon og ikke-vegetasjon er det derfor også viktig for byplanleggere å vite hvilke typer vegetasjon og hvilke typer overflatematerialer som finnes i byrommet.

I denne oppgaven har det blitt fokusert på mulighetene for å klassifisere ulike typer vegetasjon og ulike typer steinbaserte materialer fra de hyperspektrale flybildene ved bruk av algoritmer innen maskinlæring. Ruhet er et av parameterne i en flommodell, og påvirker hvor fort vannet vil renne ved en gitt helning, dybde og mengde. For å velge de ulike klassene av materialer har vi tatt utgangspunkt i noe som kalles Mannings ruhetstall (som referert i Vassdragshåndboka (Fergus et al., 2010)), et verktøy byplanleggere bruker for å simulere vannavrenning i byrom. Vi har også valgt å begrense oss til en pikselbasert analyse.

Følgende punkt har blitt undersøkt i oppgaven vår*:*

- Hvordan skille mellom forskjellige typer vegetasjon fra hyperspektrale bilder
- Hvordan skille mellom forskjellige typer steinbaserte materialer
- Hvilke typer algoritmer egner seg best for denne type klassifisering
- Hvilke problemer støter man på med den enorme mengden informasjon slike data inneholder
- Teste ulike versjoner av datasettene mot hverandre.

1.2 Tema og problemstilling med bruk av hyperspektrale flybilder

Vi har valgt å jobbe med følgende problemstilling:

- *Hvor bra fungerer tre utvalgte maskinlæringsalgoritmer (CNN, SVM og logistisk regresjon) til pikselbasert arealklassifisering av hyperspektrale flybilder over urbane områder?*

Hyperspektrale flybilder har vært vanlig å bruke til klassifisering av store homogene område (Dalponte et al., 2013), men er i Norge foreløpig ikke vanlig å bruke i urbane byområder med stor romlig variasjon/inhomogene arealdekke.
Hyperspektrale bilder har lenge vært forbundet med store datamengder som nesten er umulig å håndtere for datamaskinene. I tillegg har det vært kostbart å skaffe disse bildene. Det har også vært vanskelig å få god romlig oppløsning på slike bilder, men med ny forskning har det kommet bedre sensorer og maskiner som nå gjør det mulig å vurdere å bruke disse bildene, selv om det fremdeles er en vei å gå for brukervennligheten.
Hyperspektrale bilder har mye vansker med atmosfæreforhold og spektral spredning fra naboobjekter som fører til glidende overganger mellom materialer. Derfor kan det være vanskelig å skille ut de uavhengige og viktigste variablene fra datasettet. Ghamisi et al. (2018) er av den oppfatningen at maskinlæring kan være til hjelp med å trekke disse variablene ut fra hyperspektrale bilder.
Derfor er det spennende å undersøke hvordan de tre mye brukte algoritmene logistisk regresjon, SVM og CNN presterer på et ikke-homogent urbant område i Oslo med høy romlig dataoppløsning og opp mot 500 bånd.

1.3 Arbeidsfordeling

Siden vi er to studenter som har jobbet med denne oppgaven, så har vi valgt å dele opp oppgaven slik at hver enkelt student har sitt hovedansvar for en bestemt algoritme og skrevet resultater og teori om de valgte algoritmene. Innsamling av data fra feltarbeid, diskusjon av resultater og alt utenom de spesifiserte temaene er gjort i fellesskap. Det har vært tett samarbeid underveis i arbeidet.

Oppdelingen er som følger:

- Annette Primstad har hatt hovedansvaret for teori og resultater for CNN.
- Åsmund Stemme har hatt hovedansvaret for teori og resultater for SVM og logistisk regresjon.

1.4 Oppsett og struktur

Masteroppgaven har følgende struktur:

Kapittel 1: Introduksjon

- Beskriver bakgrunnen for oppgaven, tema og problemstilling og arbeidsfordeling.

Kapittel 2: Teori

- I denne delen skal det teoretiske grunnlaget for oppgaven forklares. Først beskrives lys, elektromagnetisk stråling og sammenheng mellom materialer og stråling. Deretter blir fjernmåling, sensorer og hyperspektrale bilder beskrevet. Til slutt blir de benyttede maskinlæringsalgoritmene gjennomgått.

Kapittel 3: Metode

- Her gjennomgås først materialet som blir brukt i denne oppgaven. Dette omhandler databestilling, prosjektområde, feltarbeid, fasitarbeid, de ulike datasettene og programvaren. Deretter forklares hvordan dataene har blitt jobbet med og hvilke valg som har blitt tatt underveis. Til slutt blir det forklart hvordan maskinlæringsalgoritmene (CNN, SVM og LR) har blitt jobbet med for å klassifisere hyperspektrale flybilder.

Kapittel 4: Resultater og diskusjon

- Dette kapittelet presenterer parallelt resultatene og diskusjon av klassifiseringene. Datasettene radians og reflektans vil bli sammenlignet for alle maskinlæringsalgoritmene. CNN blir brukt til å utforske klasseinndeling, variasjon og fasit for klassene innenfor gruppene vegetasjon og steinbasert. Resultatene vil danne grunnlaget for de endelige 6 klassene som vil bli testet på treningsområdet Valle Hovin og et ukjent område på Tøyen, dette er en endelig test av modellene. Til slutt blir resultatene for CNN, SVM og LR diskutert og sammenlignet. Erfaringer og problemer med fasit blir og diskutert. I slutten av kapittelet skal en kunne svare på hvor bra de tre utvalgte maskinlæringsalgoritmer (LR, SVM og CNN) fungerer til pikselbasert arealklassifisering av hyperspektrale flybilder.

Kapittel 5: Konklusjon

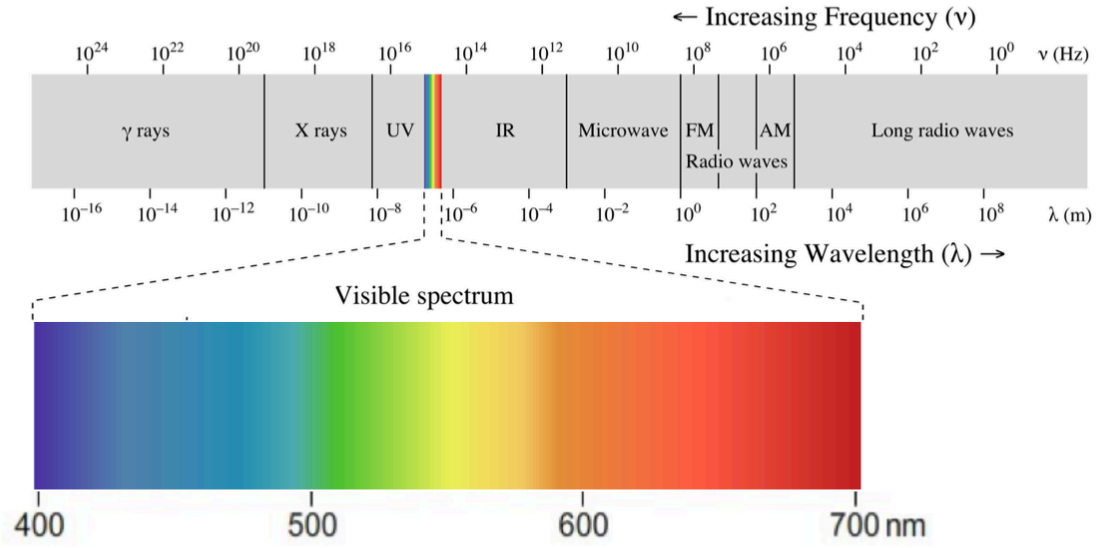
- Konklusjonen oppsummerer funnene og svarer på problemstillingen. Til slutt presenteres forslag til videre arbeid.

Vedlegg

2 Teori

2.1 Lys og elektromagnetisk stråling

Lys er en form for elektromagnetisk stråling som faller innenfor det synlige spektret (cirka 400nm-750nm). Elektromagnetisk stråling består av fotoner som kan oppfattes både som partikler og bølger med en viss energi. Bølgelengdene skiller seg fra hverandre ved energien fotonene bærer med seg (NASA, 2013). Det er vanlig å oppgi elektromagnetisk stråling i bølgelengder, der enheten vanligvis varierer mellom nanometer (10^{-9}m) og mikrometer (10^{-6}m).
Det elektromagnetiske spekteret deles gjerne inn i ulike grupper bestemt ut fra bølgelengden. Fra lavest bølgelengde til høyest er rekkefølgen gammastråler, røntgenstråler, ultrafiolett lys, synlig lys, nærinfrarødt lys, mikrobølger og radiobølger (se Figur 2.1).

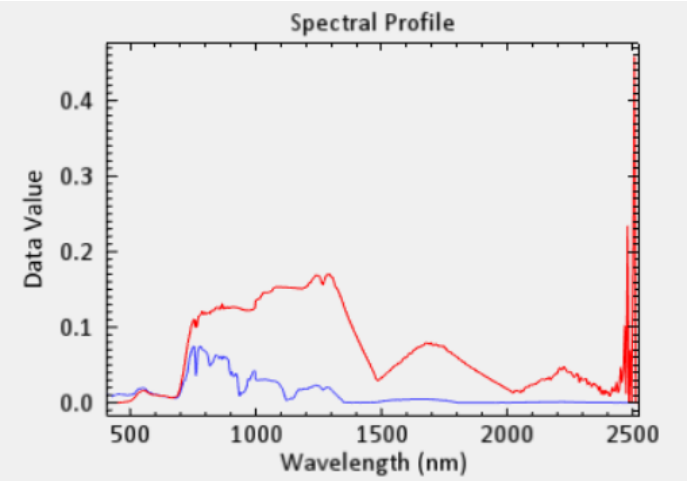


Figur 2.1: Elektromagnetisk spekter (Ronan, 2007).

Lys blir ikke oppfattet før det blir reflektert til øynene våre. Det vil si at det bare er lyset som blir reflekterte som blir sett. Når noe blir oppfattet rødt, betyr dette at de resterende bølgelengdene har blitt absorbert før de reflekteres til øynene. Ulike materialer absorberer og reflekterer ulikt. Dersom det brukes en sensor som er i stand til å fange opp elektromagnetisk stråling utenfor det synlige lyset, eks nærinfrarødt, vil en lære om materialegenskaper som vi ikke kan oppfatte med våre øyne.

2.2 Radians

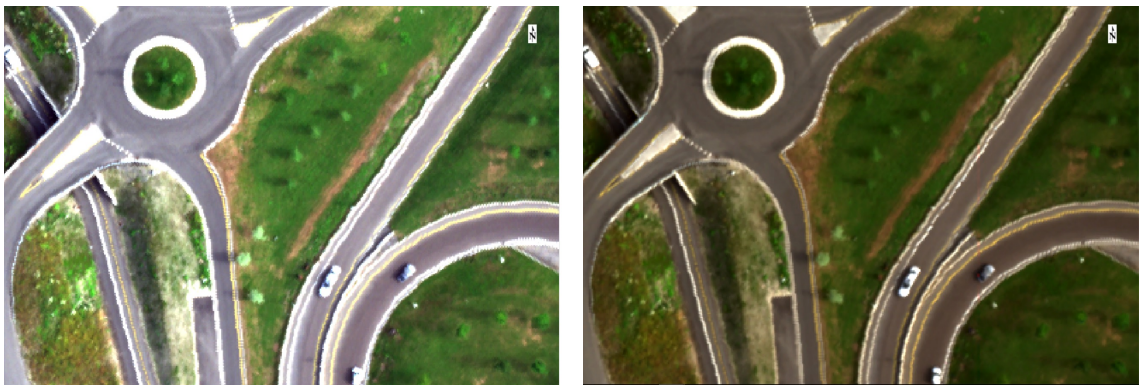
Radians er mengden reflektert lys i hvert bølgelengdebånd (Smith, 2012) i et hyperspektralt bilde. Radians inkluderer all stråling reflektert fra overflaten, inkludert det som eventuelt reflekteres fra nabopiksler, og i tillegg refleksjon fra atmosfæren. Dette betyr at det kan være utfordrerne å jobbe med radiansbilder, da det kan forekomme mye støy fra andre piksler. Det er vanlig at radiansbilder blir transformert til reflektansbilder når det skal gjøres kvantitative analyser av hyperspektrale bilder (Shippert, 2013a). Figur 2.2 som sammenligner reflektans og radians.



Figur 2.2: Spektralsignaturen til et gresspiksel vist i både radians (blått) og reflektans (rødt), for å vise hvor forskjellige de to signaturene ser ut.

2.3 Reflektans

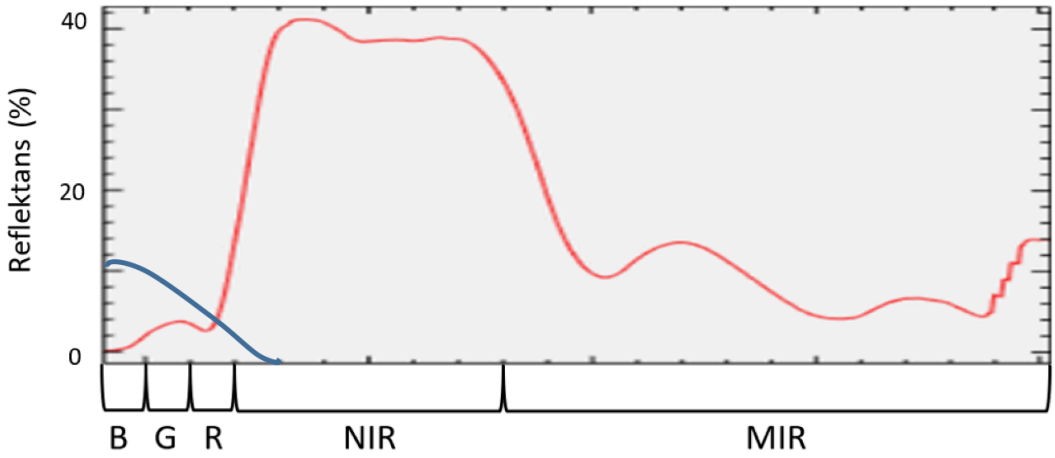
Reflektans blir definert som mengden elektromagnetisk stråling som treffer en overflate i forhold til strålingen som blir reflektert. I et reflektansbilde skal alle atmosfærekomponenter være fjernet, det vil si at det er blitt gjort en atmosfærekorreksjon. (Shippert, 2013a). Et radiansbilde og et reflektansbilde av samme område er vist i Figur 2.3. Reflektans bør brukes til å sammenlikne bilder fra ulike tidspunkt eller ulike steder, siden atmosfærepåvirkning varierer med sted og tid.



Figur 2.3: Et utsnitt av samme område fra et reflektansbilde til høyre og et radiansbilde til venstre. Bildene inneholder akkurat det samme båndene, men ser likevel ulike ut. Radiansbildet ser ut til å være overeksponert i noen av de hvite områdene Reflektansbildet har ikke det samme problemet, men til gjengjeld ser bildet mye mørkere ut.

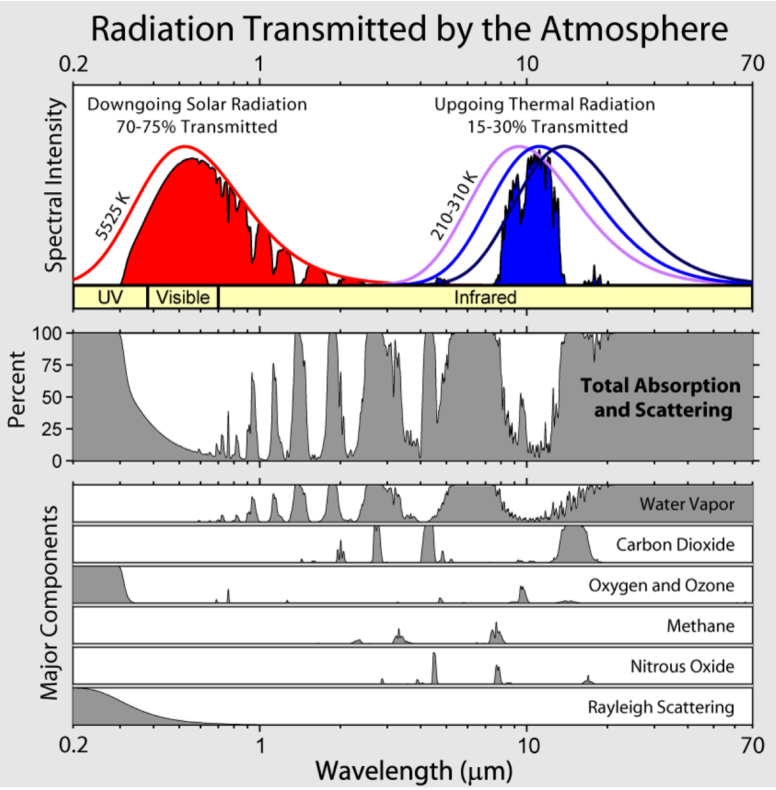
2.4 Materialer og spektralsignaturer

Ulike materialer reflekterer og absorberer stråling ved forskjellige bølgelengder. Hvordan dette skjer er helt avhengig av materialets oppbygning og tekstur (Ghamisi et al., 2017). Ethvert materiale vil reflekterer unike stråleverdier, samlet kan disse kalles en spektralsignatur. Spektralsignaturen kan oppnås ved å ta et bilde med veldig mange smale bølgelengdeområder av et materiale. Resultatet blir signaturen. Setter man sammen disse verdiene til en graf får man et visuelt inntrykk av spektralsignaturen. Materialer av samme type har lignende spektralsignaturer, akkurat som materialer av ulik type har forskjellige. Det er derfor mulig å bruke disse spektralsignatuerne til å klassifisere materialer. I Figur 2.4 kan en se et plott over de reflekterte verdiene til et piksel med vegetasjon (rød linje i figuren), og et piksel med vann (blå linje).



Figur 2.4: Figuren viser den reflekterte verdier fra ett vegetasjonspiksel, illustrert med en rød linje, og tilsvarende for vann i blått. Den vertikale aksen representerer refleksjon angitt i prosent, mens den horisontale aksen viser hvor i de elektromagnetiske spekteret en befinner seg, synlige lys og det infrarøde spektre er inkludert. I dette plottet er blå satt i området 400µm-500µm, grønn 500µm-600µm, rød 600µm-700µm, nær-infrarødt (NIR) 700µm-1300µm og midler infrarød (MIR) 1300µm-2500µm.

Elektromagetiske stråling blir ikke bare absorbert eller reflektert av faste materialer på selve jordoverflaten, det er og mange stoffer i atmosfæren som påvirker strålingen underveis. Figur 2.5 viser noen av disse atmosfæriske komponentene. Det kan derfor være hensiktmessig å utføre en atmosfærekorreksjon for å fjerne denne støyen.



Figur 2.5:Viser elementer som påvirke strålingen i atmosfæren (Rohde, 2007).

Vegetasjon absorberer mesteparten av det synlige lyset og reflekterer det nærinfrarøde. Grunnen til at vegetasjon oppfattes som grønn på et bilde er fordi det er mest grønt lys igjen, siden nærinfrarødt lys ikke kan oppfattes uten spesielle kamera.
Et vannområde vil ikke avgi mye varme og de nærinfrarøde båndene vil derfor få veldig lave verdier. Vann er i utgangspunktet gjennomsiktig, men på bilder tatt på avstand ser det ofte blått ut, dette kommer av at blått lys har de korteste bølgelengdene av alt synlig lys. Der det blå lyset vil bli spredt når det kræsjer med partikler i atmosfæren, vil rødt og grønt lys passere. Det blå bølgelengdene bli så reflektert av vannet slik at dette deror ser blått ut.

2.4.1 Vegetasjonsindekser

Vegetasjonsindekser blir brukt til å skille ut ulike typer vegetasjon fra et flerkanalsbilde. Indeksene er forholdstall som blir beregnet ut fra reflekterte strålingsverdier. Verdiene som blir brukt til å sette vegetasjonsindekser kommer fra bølgelendeintervaller innenfor synlig lys, nærinfrarød (NIR) og midlere infrarød (MIR) stråling. En svært mye brukt vegetasjonsindeks er NDVI (normalisert differanse vegetasjonsindeks), se formel 2.1.

$$\mathrm{NDVI} = \frac{(\mathrm{NIR} - \mathrm{R})}{(\mathrm{NIR} + \mathrm{R})} \qquad (2.1)$$

der R er bølgelengde området innenfor de røde synlige lyset. NDVI varierer mellom -1,0 og 1,0. Frisk vegetasjon ligger mellom 1,0 og 0,7, mens vegetasjon som ikke er frisk vil ligge under denne verdien. Verdier under null vil vanligvis bety at materialet er ubevokst (Dick, 2009).

2.5 Fjernmåling

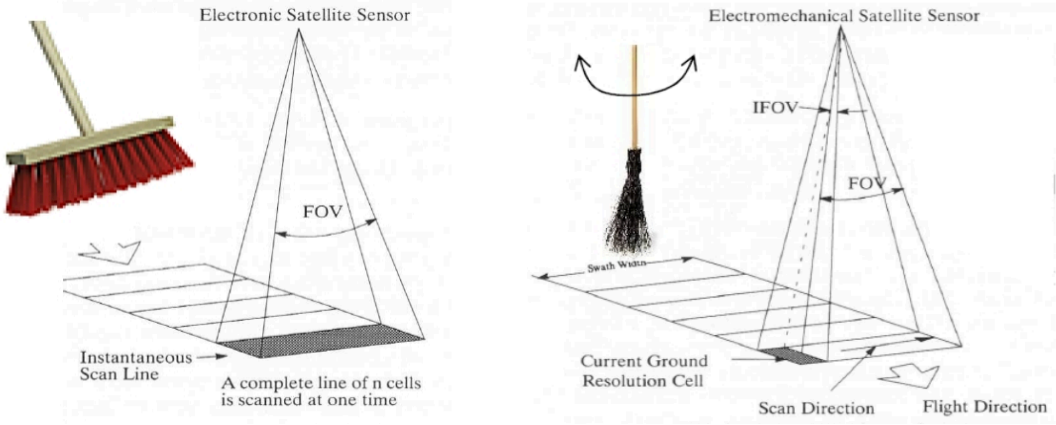
Fjernmåling handler om å måle objekter og områder som ikke er i fysisk kontakt med måleinstrumentet, til forskjell fra det som gjerne ses på som tradisjonell «nærmåling», der en måler direkte på stedet (Lied, 2018). En vanlig form for fjernmåling er registering av reflektert stråling på eller nær jordens overflate ved bruk av fly, droner eller satellitter med ulike typer sensorer. Under opptak blir den elektromagnetiske strålingen registret i ulike bølgelengdeområder i det elektromagnetiske spekteret. Disse bølgelengdeområdene blir også kalt bånd eller kanaler. Det optiske spekteret er mye brukt, dette spektret inkluderer både det synlige lyset og det nærinfrarøde (400nm-2500nm). Ved å kombinere bånd fra synlig lys og nærinfrarødt kan disse settes sammen til bilder med informasjon om omgivelsene som det blotte øye ikke kan se.

2.6 Hyperspektrale sensorer

Hyperspektrale avbildningssystemer bruker sensorer som for det meste opererer fra de synlige til de mellom- og nærinfrarøde bølgelengdene, og kan simultant samle inn opptil hundrevis av smale spektrale kanaler fra det samme området på jordoverflaten. Disse sensorene samler data i piksler som er representert som vektorer, der hvert element er en måling som korresponderer til en spesifikk bølgelengde. Størrelsen på hver vektor er lik det antall spektrale kanaler med data som er samlet inn av sensoren. For en vanlig multispektral sensor er det vanlig med inntil ti slike kanaler, mens når det er snakk om hyperspektralt er det ofte opp til flere hundre kanaler. Definisjonen på hva som er multispektralt og hyperspektralt er veldig flytende, og ofte er det ikke antall bånd som bestemmer dette, men intervallbredden til båndene. Multispektralt har brede bånd mens hyperspektralt har smale bånd. Den detaljerte spektralinformasjonen man får ut av hyperspektrale sensorer øker sjansen betraktelig for å få til et nøyaktig skille mellom materialer man ønsker å skille fra hverandre. Sensorene har derfor et vidt anvendelsesområde, og bør være godt egnet til arealklassifisering av jordoverflate (Benediktsson and Ghamisi, 2015). En negativ side med de smale båndene hyperspektrale bilder leverer er at dataene blir mer utsatt for støy siden smale bånd gir lite energi, og man bør derfor gå opp i pikselstørrelse for at signalet skal skille seg fra støyen.

2.6.1 Pushbroom-skanner

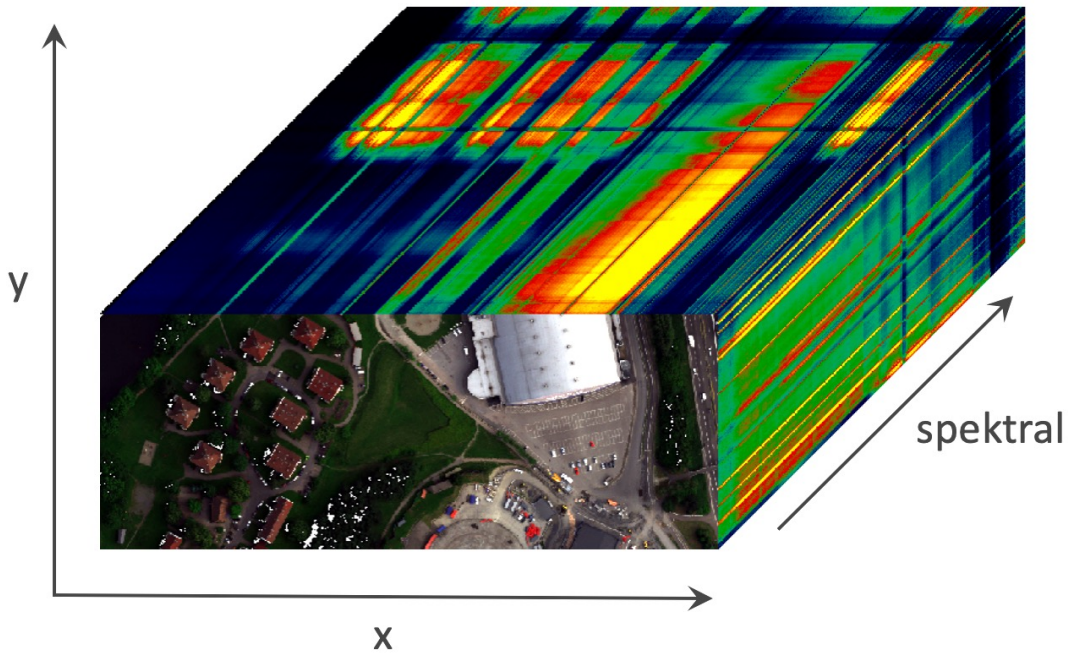
En pushbroom-skanner inneholder en linje med detektorer som er satt på tvers av flyveretningen (Graham, 2000). Hver detektor fokuserer på ett piksel på flystripen hver. Skanneren kan dermed skanne en hel linje med piksler samtidig. Det er fordeler og ulemper med disse sensorene. En ulempe er at alle detektorene må ha samme følsomhet, dersom dette ikke er tilfellet kan en ende opp med uønskede striper i bildet. Fordelen med denne skanneren er at den gir kraftigere signal fra hvert område, siden den bruker mer tid på hver piksel i forhold til andre skannere, som eks whiskbroom-skannere. Slike tar bare ett og ett piksel på en stripe om gangen (Shippert, 2013b). Figur 2.6 illustrerer forskjellen mellom disse sensorene.



Figur 2.6: Prinsippet for de to ulike skannertypene pushbroom (venstre) og whiskbroom (høyre) (Schiewe, 2006).

2.6.2 Hyperkube

Hyperspektrale bilder kan betegnes som en stabel av bilder med forskjellige bølgelengdeintervaller fra samme området på jordoverflaten. Akkurat som i et vanlig digitalt fargebilde (RGB) er det en romlig x-dimensjon, en romlig y-dimensjon, men i tillegg kommer en spektral dimensjon. Til forskjell fra et vanlig RGB-bilde som har 3 fargekanaler, består et hyperspektralt av en stabel av opptil flere hundre lag oppå hverandre, som en kube. Denne stabelen er den spektrale dimensjonen. Et eksempel på en slik kube kan ses i figur 2.7.



Figur 2.7: Eksempel på en hyperkube.

Når det blir tatt hyperspektrale bilder ved bruk av for eksempel en pushbroom-skanner mistes tidsdimensjonen, dette siden billedtagningen blir gjort i flere operasjoner. Det er nødvendig å sette disse stripene sammen, og det betyr at bildet endre opp med å bestå av bildestriper fra ulike tidspunkter under billedtagningen. Det er derfor viktig å ta bilder av materialer som står i ro, for at de skal fanges opp riktig.

2.7 Hydraulisk modellering

Det finnes mage ulike hydrauliske modeller, disse modellene blir laget for å simulere hvordan vann beveger seg, og modellene blir tilpasset etter hvordan modellen skal brukes. Det finnes egne modeller for rør, elver og andre terrengoverflater.

Disse modellene har tre hoved-input som avgjør hvor fort vannet renner. Dette er helning som kommer fra terrengmodell, samt permeabilitet og Mannings ruhetstall som begge avgjøres av egenskaper ved terrengoverflaten. Her defineres permeabilitet som evnen et materiale har til å transportere væske, også kalt gjennomstrømmelighet, mens Mannings tall er et empirisk bestemt mål på en overflates ruhet. Det betyr at denne verdien vil endre seg i forhold til overflaten. Figur 2.8 viser en tabell med noen foreslåtte Mannings tall for en kanal.

Kanaler med sider og bunn av:		Vanlige M-verdier
Asfalt	glatt ru	75-80 60-65
Vegetasjon	planerte sider beskyttet med grasvekst	20-33
Tre	høvlet, umalt uhøvlet planker	70-83-90 66-77-90 55-67-83
Betong	glatt puss på grus delvis sprøytet samme på råfjell	63-70-90 50-59-66 40-50-60 37-45-58
Støpt bunn med sider av:	glatt steinmur stein i mørtel betongblokker steinplastring	50-59-66 40-50-60 33-40-50 28-33-50
Grusbunn med sider av:	betong stein i mørtel steinplastring	40-50-59 38-43-50 28-30-43

Figur 2.8: Tabellen i figuren viser de empiriske verdiene for Mannings ruhetstall M, gjengitt etter Ven Te Chow (1959). I engelsk litteratur brukes n i stedet for M, der er n = 1/M (Fergus et al., 2010).

Ved urbane områder blir det ofte brukt to strategier for å sette Mannings tall.
Metode en bruker områdets urbaniseringsgrad til å sette verdier, mens nummer to, som er brukt i denne oppgaven, ser på den spesifikke ruheten for veier, bygg, parkeringsplasser, gress, skog, grus osv.

2.8 FKB – felles kartdatabase

FKB er et begrep som brukes om det mest detaljerte grunnkartdataene. Der grunnkartdata beskriver generelle objekter i det fysiske landskap. Dette kan være data over vann, bygninger, eiendommer, veier, steder, høyde, arealbruk, ledninger og mye mer (Geonorge).

2.9 Maskinlæring

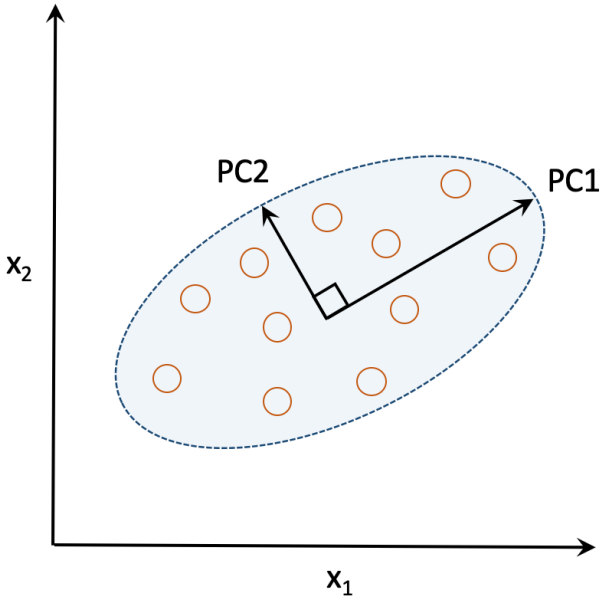
Maskinlæring er å gi datamaskiner en mulighet til å lære ved å gjøre data om til kunnskap. Det finnes ulike typer for maskinlæring, to veldig vanlige er styrt og ikke-styrt læring. Disse har veldig ulike bruksområder og krav til forarbeid.

Styrt maskinlæring krever styringsinput i form av f.eks. fasitdata, en må på forhånd bestemme hva modellen skal finne. Dataene som brukes til å trene algoritmen må kunne kobles til en merkelapp (fasiten), som sier noe om hvilken klasse eller verdi som er den korrekte for den aktuelle sample slik at en underveis kan gi algoritmen en tilbakemelding på hvordan den har prestert. Da kan den lære av feilene underveis og forbedre seg. Denne typen maskinlæring er brukt for å predikere et utfall på usette data (Raschka and Mirjalili, 2017).

Ikke-styrt maskinlæring krever ingen fasit. Noen metoder tillater brukeren å bestemme hvor mange klasser algoritmen skal forholde seg til, mens andre lar algoritmen gjøre dette selv. Det kreves derfor veldig lite forarbeid ved disse algoritmene, men til gjengjeld blir det ikke gitt noen tilbakemelding på prestasjonene til algoritmen. Denne typen maskinlæring blir vanligvis brukt til å finne skjulte mønstre i data. Det er derfor vanlig å bruke denne type algoritmer til å undersøke dataene før en vet hva en leter etter i et datasett (Raschka and Mirjalili, 2017).

2.10 PCA

PCA (Prinsipal Component Analysis) er en populær algoritme for ikke-styrt, lineær transformasjon. Den er mye brukt til feature extraction, dimensjonsredusering og eksplorative dataanalyser. PCA bruker korrelasjonen mellom variabler i et datasett til å gjenkjenne sammenhenger og mønstre. Målet til PCA er å finne retningens maksimale varians, og denne maksimale variansen blir satt til å være den første komponenten (PC1) (Raschka and Mirjalili, 2017). Den neste komponenten (PC2) vil stå vinkelrett på første komponent, og inneholde nest mest variasjon. Dette kan ses i figur 2.9. Slik kan det fortsette videre med flere komponenter. De første komponentene inneholder nesten all informasjonen i hele datasettet, det er derfor mulig å bruke disse til å beskrive mesteparten av hele datasettet.



Figur 2.9: De originale variabelaksene x_1 og x_2 er blitt plottet mot hverandre som et spredningsplott. PC1 og PC2 er prinsipalkomponentene i dette datasettet.

2.11 K-means clustering

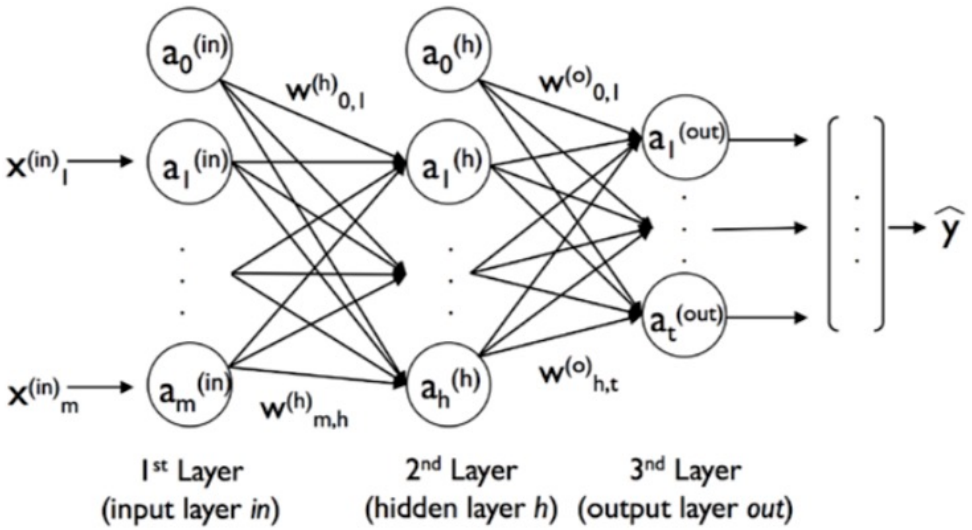
K-means clustering er en av de mest brukte ikke-styrte maskinlæringsalgoritmene. Algoritmen går gjennom et datasett og finner data som ligner på hverandre for å generere grupper. Det dannes *k* antall grupper i datasettet, i forhold til hvor mange klasser som på forhånd er valgt av bruker. For hver gruppe beregnes et tyngdepunkt (gjennomsnitt). I starten vil disse gruppene bli plassert tilfeldig, men for hver nye iterasjonsrunde der det beregnes et nytt tyngdepunkt for gruppene, vil dataverdiene bli flyttet til den gruppen med den verdien som er nærmest tyngdepunktet. Dette blir repetert til det ikke lenger er noe data å flytte, eller til oppgitt grense for maks iterasjoner er nådd (Richards, 2013).

2.12 ANN – kunstige nevrale nettverk

CNN er et nevralt nettverk, og for å forstå CNN er det nødvendig å forstå grunnlaget for oppbygningen til nevrale nettverk.

Et kunstig nevralt nettverk er basert på hypoteser og modeller av hvordan den menneskelige hjernen arbeider for å løse avanserte oppgaver. De første studiene av nevrale nettverk ble gjort allerede på 1940 tallet av Warren McCulloch og Walter Pitt (Raschka and Mirjalili, 2017). Men det var ikke før i moderne tid nevrale nettverk slo til på grunn av mangel på datakraft.

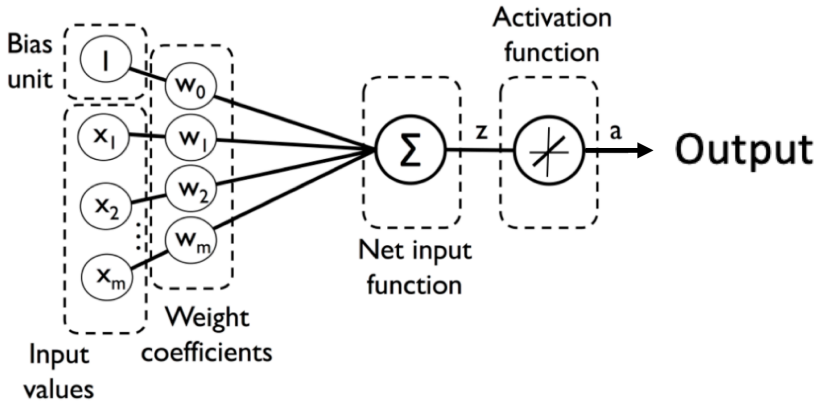
I figur 2.10 kan en se konseptet ved et nevralt nettverk med mange lag. Dette nettverket består av et input-lag, ett eller flere skjulte lag (hidden layers) og et output-lag. I hvert lag kan en se flere nevroner. Disse kobler seg til alle nevronene i det neste laget. Denne typen fullt koblet nettverk blir også kalt Multilayer Perceptron (MLP) (Raschka and Mirjalili, 2017). Om denne typen nettverk har to eller flere skjulte lag blir nettverket kalt et dypt kunstig nevralt nettverk. Nevrale nettverk faller under styrt klassifisering.



Figur 2.10: Multilayer Perceptron (MLP) med tre lag. For å kunne skille mellom hvilket lag hvert element hører til, er det brukt parenteser over elementene, (in) står for inputlag, (h) står for skjult lag og (out) outputlag. Raschka and Mirjalili (2017) side 384.

2.12.1 Kunstige nevron

Kunstige nevroner er basert på ekte nevroner, altså koblede nerveceller i hjernen, som behandle og overføre signaler mellom seg. Figur 2.11 gir et visuelt innblikk i hvordan et nevron fungerer i et kunstig nevralt nettverk.



Figur 2.11: Hentet fra Raschka and Mirjalili 2017, side 382. Figur viser hvordan et kunstig nevron fungerer.

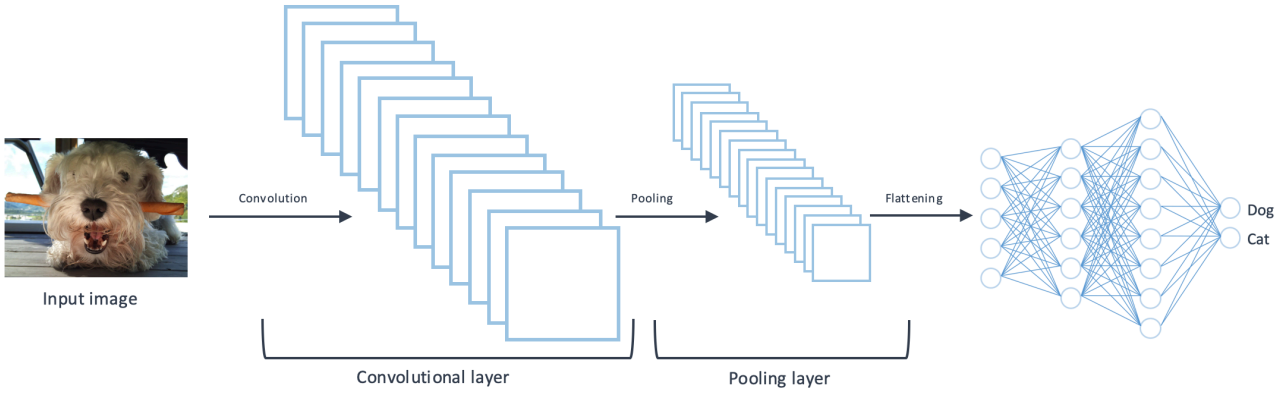
Nevronet mottar inputverdier fra nevronene fra forrige lag i nettverket. Disse inputverdiene x multipliseres med sine tilhørende vektkoeffisienter w, deretter summerer nettofunksjonen disse sammen og legger samtidig til en konstant (bias I).

$$z = \sum_i w_i x_i + I \tag{2.2}$$

En aktiveringsfunksjon vil så ta denne netto summen z og bestemme hvor mye informasjon som skal sendes videre til nevronene i neste lag i nettverket. Resultatet etter arkiveringsfunksjonen blir kalt aktivering a. Det er vanlig å bruke både lineære og ikke-lineære aktiveringsfunksjoner. Ikke-lineære kan være veldig nyttige, siden de fanger opp informasjon som en lineær modell ikke har grunnlag for å oppdage (Raschka and Mirjalili, 2017).

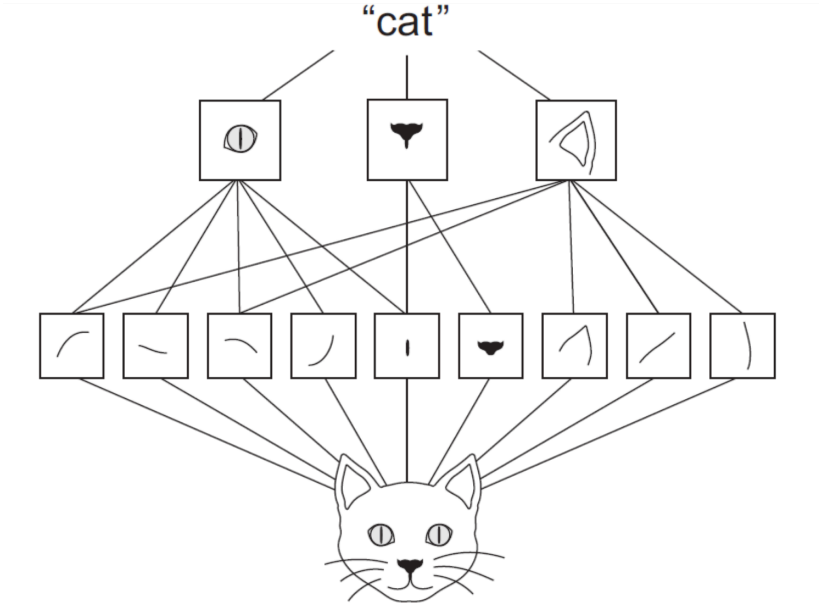
2.12.2 CNN – Convolutional Neural Network

CNN er en variant av nevrale nettverk. Det som skiller et CNN fra et vanlig ANN er bildebehandlingsdelen i starten av nettverket (se figur 2.12), siden CNN bruker bilder som input i stedet for tabeller med data. Bildebehandlingsdelen består av ett eller flere "convoluton"- og "pooling"-lag, disse blir påført bildet for å finne karakteristiske trekk i dette slik at nettverket vil være i stand til å identifisere hva som befinner seg i bildet.



Figur 2.12: Illustrasjon av et eksempel på strukturen til et"Convolutional Neural Network". Nettverket tar inn et bilde og påfører bildet ulike bildeoperasjoner ved bruk av convolution og pooling. Til slutt blir bildets ulike lag slått sammen og ført inn i et ordinært nevralt nettverk.

CNN er en svært mye brukt metode for bildegjenkjenning og objektdetektering. Dette kommer av at nevrale nettverk er dyktige til å lære seg hvilke variabler som er nyttige for en bestemt oppgave, og "nyttige" variabler i et datasett er essensielt for alle maskinlæringsalgoritmer for å kunne prestere. Nevrale nettverk blir av noen sett på som en type variabel utvelger. Raschka and Mirjalili (2017) forklarer årsaken til nettverkenes bra egenskaper med utvelging, at nevrale nettverk lager variabelhierarkier. Dette skjer ved å kombinere lavnivå-variabler med hverandre for å skape høynivå-variabler. En kan se på lavnivå-variabler som det første laget i nettverket, mens høynivå er de siste lagene. I et CNN ville lavnivå-variablene gjerne vært symbolisert som kanter eller klumper, figur 2.13 illustrere dette.



Figur 2.13: Hentet fra Chollet (2017), side 123. Illustrerer hvordan variabelhierarkiet i en CNN modell i teorien kan se ut. Her er alle kantene og segmentene lavnivå-variabler, disse kombineres til øye, nese og øre som er høynivå-variabler i hierarkiet. Deretter brukes disse høynivå-variablene til å bestemme at dette er en katt.

Convolutional layer

Et "convolution"-lag bruker filtre som føres over bildet og utfører bildefilteroperasjoner, dette gjøres for alle dimensjonene i bildet. Disse filtrene har vekter som må trenes, og etter hvert som filtrene blir trent lærer nettverket å kjenne igjen kanter og mønstre i bildet. Et dypt nettverket vil være i stand til å kjenne igjen avanserte mønstre fra bildene.

Pooling layer

Pooling layer har ingen trenbare parameter/vekter, men bruker et filter som føres over bildet for å trekke ut en verdi innenfor området filteret dekker. Det er vanlig å trekke ut gjennomsnittsverdien eller maksverdien, og tilegne denne verdien til et nytt bilde.

Strides

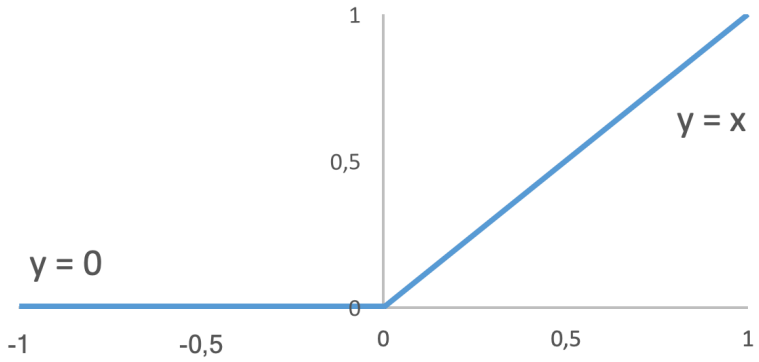
"Strides" bestemmer stegene til filteret, med dette menes hvor mye et filter skal forskyves i bildet for hver gang det har blitt brukt på et område i bildet. Dette påvirker om bildet øker eller minker i størrelse.

2.12.3 Aktiveringsfunksjoner

En aktiveringsfunksjon er en funksjon som har som hovedformål å aktivere et nevron, slik at dette sender informasjon videre. Det vanligste er å bruke ikke-lineære aktiveringsfunksjoner, men i teorien kan en bruke en hvilken som helst funksjon som aktiveringsfunksjon, til og med lineære funksjoner, men det vil ikke være nyttig å bruke lineære aktiveringsfunksjoner i både det nevrale nettverkets skjulte lag og outputlaget. Dette siden en trenger ikke-linearitet for å løse komplekse problemer, og summen av lineære funksjoner vil gi en ny lineær funksjon. Derfor må en bruke en ikke-lineær aktiveringsfunksjon for å gjøre modellen ikke-lineær (Raschka and Mirjalili, 2017). Det er også andre maskinlæringsalgoritmer som bruker aktiveringsfunksjoner, f.eks. logistisk regresjon.

ReLU

ReLU er den mest brukte aktiveringsfunksjonen i nevrale nettverk, spesielt i CNN. Denne funksjonen setter alle negative verdier til 0 og er lineær for alle verdier over 0, figur 2.14 illustrerer dette.



Figur 2.14: En graf som viser hvordan funksjonen ReLU fungerer på et område x = [-1, 1].

Dersom en ikke vet hvilken aktiveringsfunksjon som skal brukes, pleier ReLu å være et bra valg (Hao, 2017). I tillegg er den billig å bruke med tanke på tidsbruk, siden den ikke opererer med kompliserte matematikk. En annen positiv side med ReLu er at den ikke lider av "the vanishing gradient problem" som andre aktiveringsfunksjoner eks. sigmod (Raschka and Mirjalili, 2017).

Dying ReLU

En negativ side med å gi alle negative verdier null, er at problemet kalt "Dying ReLU" kan oppstå. ReLU-nevronet er "dødt" når det sitter fast på den negative siden (se figur 2.14) og aldri får en annen verdi enn null (ML Cheatsheet, 2018). Et negativt nevron vil ha store problemer med å komme seg ut av denne sonen, siden gradienten er så lav. Etter en lengre periode vil en kunne ende opp med et nettverk der mesteparten av nevronene i nettverket ikke gjør noe. Det er stor sjanse for at dette problemet opptrer når læringsraten er høy, eller det er en stor negativ bias (Hao, 2017). Lav læringsrate minker ofte dette problemet. Andre funksjoner som har en liten økning i den negative sonen unngår dermed problemet, eks. Leaky ReLU.

Leaky ReLU

Aktiveringsfunksjonen Leaky ReLU har en liten stigning på negativ side, og defineres som

$$\phi(z) = \begin{cases} 0{,}01z, & z < 0 \\ z, & z > 0 \end{cases} \tag{2.3}$$

Funksjonen fikser "The dying ReLU problem", siden den ikke har null stigning i negativ sone. Den negative siden med denne funksjonen er at den er lineær (ML Cheatsheet, 2018).

Sigmoid- Logistic

Det er vanlig å bruke denne aktiveringsfunksjonen på outputlaget ved binærklassifisering, siden funksjonen tar en verdi og presser den inn mellom 0 og 1, dette medfører at høye negative tall blir satt til null og høye positive tall til 1, som beskrevet i formel 2.4.

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

Denne funksjonen lider også av "the vanishing gradient problem", siden gradienten til sigmoid er så lav. "The vanishing gradient problem" oppstår under backward-propagation dersom den lokale gradienten er veldig nær null når den skal multipliseres med output-gradienten. Da vil gradienten bli "drept", og den lave verdien vil gjøre at nesten ingen signal går videre gjennom nevronet til dens vekter og igjen videre som aktivering (Hao, 2017).

Softmax
Softmax-funksjonen beregner sannsynlighetsfordelingen mellom *n* antall hendelser, med hensyn på alle de andre hendelsene. Den blir derfor brukt i output-lag for å bestemme klassefordelinger (ML Cheatsheet, 2018). Se formel 2.5.

$$\phi(z) = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}} \tag{2.5}$$

Funksjonen gir ut en verdi mellom 0 og 1, dette er sannsynligheten for hver klasse. Hvis verdiene for hver klasse summeres, er verdien lik 1 (Hao, 2017).

2.12.4 Skjulte lag (hidden layer)

Skjulte lag er de lagene i et nevralt nettverket som ikke er input-lag eller output-lag. Det som er spesielt med de skjulte lagene er at en ikke har kontroll på hva slags informasjon disse lagene inneholder. Dette er en av grunnene til at nevrale nettverk ses på som "svarte bokser". Det er ingen grense for hvor mange skjulte lag en kan i et nettverk, men den negative siden med å ha for mange av disse er feilgradienten som blir beregnet i backward-propagation. Gradienten vil minke for hvert lag som blir lagt til nettverket, og "The vanishing gradient problem" gjøre modellens læring mer utfordrende. Det er derfor nødvendig å bruke algoritmer som ikke er utsatt for dette problemet, eks. ReLU (Raschka and Mirjalili, 2017).

2.12.5 Forward propagation

Forward propagation er prosessen som aktiverer det nevrale nettverket. Prosessen starter med å kjøre treningsdataene gjennom alle lagene, fra input- til output-lag. Dette kan illustreres ved å ta utgangspunkt i figur 2.10, som viser et nevralt nettverk med tre lag. Den matematiske formel 2.2 utvides fra å gjeldende bare for et enkelt nevron, til å gjelde for et helt lag.

Det starter med en matrisemultiplikasjon mellom $A^{(in)}$og $W^{(h)}$

$$Z^{(h)} = A^{(in)}W^{(h)} \tag{2.6}$$

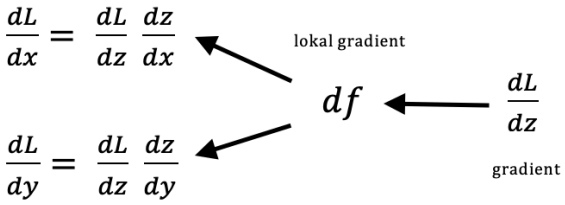
der $A^{(in)}$er en variabelmatris på formen $n\ x\ m$, der n representerer alle $x^{(in)}$verdiene i tillegg til bias-konstant, mens m er antallet ulike variabler. $W^{(h)}$er vektmatrisen til det skjulte laget, denne matrisen har dimisjonene $m\ x\ d$, der d er antall nevroner i det skjulte laget. Når disse multipliseres sammen får en netto input-matrisen $Z^{(h)}$ på formatet $n\ x\ d$. Deretter blir den valgte aktiveringsfunksjonen $\phi(\dots)$ brukt på alle elementene i $Z^{(h)}$for å få aktiveringsmatrisen $A^{(h)}$ for de skjulte lagene. $A^{(h)}$ er på formatet $n\ x\ d$. Denne prosessen repeteres for alle lagene i nettverket. Når en kommer til output-laget multipliseres $A^{(h)}$og $W^{(out)}$, vektmatrisen har dimensjon $d\ x\ t$, der t er formen på output. Det vil si at størrelsen på output avhenger av om en har et regresjonsproblem, binært eller flere klasser.

$$Z^{(out)} = A^{(h)}W^{(out)} \quad (2.7)$$

dette gir ut matrisen $Z^{(out)}$ med formatet $n\ x\ t$. Til slutt brukes aktiveringsfunksjonen på elementene i $Z^{(out)}$for å få ut outputverdiene (Raschka and Mirjalili, 2017).

2.12.6 Backward propagation

Backward propagation er en av de mest populære algoritmene for å trene nevrale nettverk. Algoritmen brukes for å beregne gradienten for hvert lag, slik at disse videre kan kalkulere vektkoeffisientene i nettverket, se figur 2.15.



Figur 2.15: Hovedprinsippet som repeteres over alle kjerneposisjonene.

Den beregnede feilen fra outputlaget blir brukt til å kalkulere en feil for hvert nevron i de skjulte lagene (går bakover lag for lag). Kjerneregelen blir repeterende brukt gjennom alle lagene med nevroner i nettverket. Gradienten for hvert lag blir beregnet ved å bruke output-gradienten. Dette gjøres ved å oppdatere vektene trinnvis, og utnytte "gradient descent", som er en optimaliseringsalgoritme.

Grunnen til at denne prosessen går bakover (fra høyre mot venstre), er at en da starter å multiplisere en matrise med en vektor, noe som gir ut en vektor. Denne igjen multipliseres videre med en matrise. Dette er veldig mye mindre beregningskrevende i forhold til å gjøre det i forward-modus, der en først starter med matrisemultiplikasjoner og til slutt multipliserer resultatet med en vektor (Raschka and Mirjalili, 2017).

2.12.7 Optimalisering

Læringsrate
Læringsrate handler om hvor fort en modell lærer, altså tar til seg informasjon. Det er viktig at denne verdien ikke er for høy eller lav. Ved en for lav verdi vil en risikere at modellen setter seg fast i et lokalt minimum, dermed vil ikke modellen lære noe mer og den mister muligheten til å bli så bra som den optimalt kunne blitt. Samtidig vil en for høy læringsrate kunne føre til overtilpasning av modellen.

Momentum
Momentum handler om å bruke tidligere gradienter til å gi en dytt på den nåværende gradienten, slik at den kommer seg forbi et lokalt minimum.

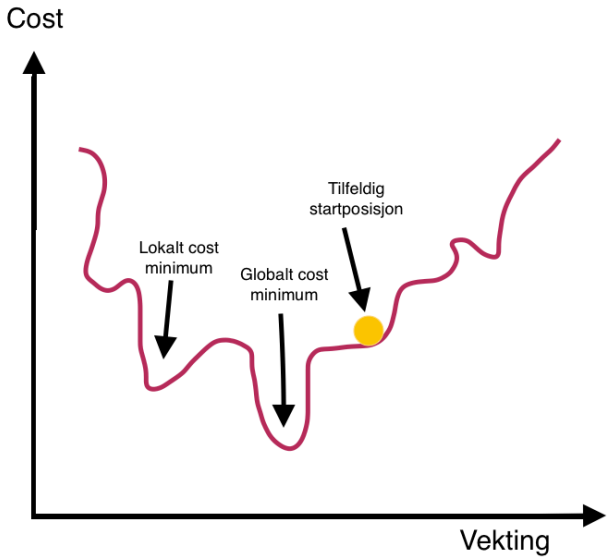
Decay
Decay handler om å bremse gradienten, slik at den ikke mister det globale minimumet.

Batchstørrelse
Det er vanlig å dele opp et datasett i flere batcher med data for å minke minnet og tidsbruken som kreves av datamaskinen for å prosessere dataene. Én batch inneholder en del av datasettet, og definerer hvor mange bilder som vil bli sendt gjennom nettverket samtidig, for eksempel ved trening. Mange batcher betyr også at nettverket vil få oppdatert vektene sine mange ganger, siden dette skjer hver gang en batch er ferdig kjørt gjennom nettverket. Det er negativt med for små batcher siden dette kan føre til ubalansen inne i batchen, noe som gir store svingninger og dårlig accuracy for den aktuelle klassifiseringen.

Epoke
Én epoke tilsvarer en treningsøkt for alle dataene. Det vil si at det har gått en epoke når alle batchene er kjørt gjennom forward propagation og backward propagation en gang. For få epoker gir undertilpasning for klassifiseringen, og for mange fører til overtilpasning. Grunnen til at det må kjøres mange epoker er at alle vekter starter med tilfeldige verdier, og det er nødvendig med mange itereringer for at verdien skal konvergere.

Gradient descent

Gradient descent er en optimaliseringsalgoritme som brukes i treningen for å få et nettverk til å lære. Algoritmen går ut på å justere vekten og bias i nettverket. Målet er å optimalisere alle vektene og bias i modellen. Cost-funksjonens output-verdi skal minimeres slik at den når det globale minimumet, se figur 2.16. Siden alle vekter starter med tilfeldige verdier må dette repeteres til funksjonen konvergerer. Cost-funksjonen blir beregnet etter hver gang treningsdata har gått gjennom nettverket (ML Cheatsheet, 2017).



Figur 2.16: Et eksempel på hvordan det ser ut med en vekt på x-aksen. Verdien til cost-funksjonen representerer y-aksen. I dette bildet er det bare brukt én vekt, siden hver vekt trenger en ny dimensjon, og siden det er vanlig med tusenvis av vekter er dette umulig å visualisere.

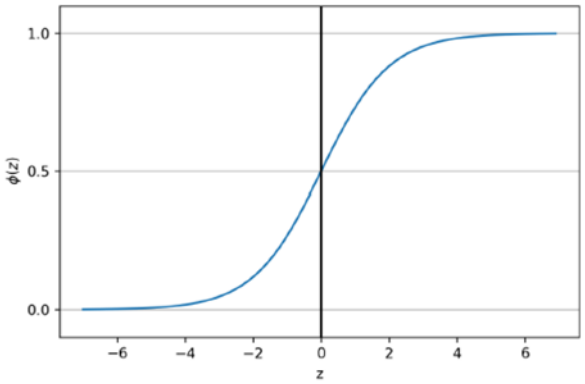
2.13 Logistisk regresjon - LR

Logistisk regresjon (LR) er en enkel, men kraftig maskinlæringsalgoritme for binære og lineære klassifiseringsproblemer. Til tross for navnet er altså LR en metode for klassifisering, ikke regresjon. Metoden tar utgangspunkt i et kunstig nevron, beskrevet i avsnitt 2.12.1. LR jobber med en binær, avhengig variabel, og ser på denne mot andre uavhengige variabler (Raschka and Mirjalili, 2017). Variabelen har da enten 0 eller 1 som mulige verdier, siden den er binær, men den kan enkelt utvides til å gjelde for multiklasser også.

Ideen bak LR tar utgangspunkt i en sannsynlighetsmodell, der sannsynligheten for at en hendelse p skal inntreffe er

$$p = \frac{1}{1-p} \tag{2.8}$$

Den logistiske funksjonen (som metoden er oppkalt etter), også kalt sigmoidfunksjonen, se formel 2.4, er en S-formet kurve som bruker aktiveringsfunksjonen til å presse en hvilken som helst reel verdi til å bli en verdi mellom 0 og 1, se figur 2.17. Den konverterer altså sannsynlighet til et binært utfall med en tersklingsfunksjon.



Figur 2.17 Viser en logistisk funksjon (Mueller, 2019)

Det finnes flere parametere for å finkalibrere en logistisk regresjonsmodell i maskinlæring. De vanligste å bruke (og som har blitt brukt i denne oppgaven) er C, lambda og L1/L2.

C-verdi og lambda-verdi

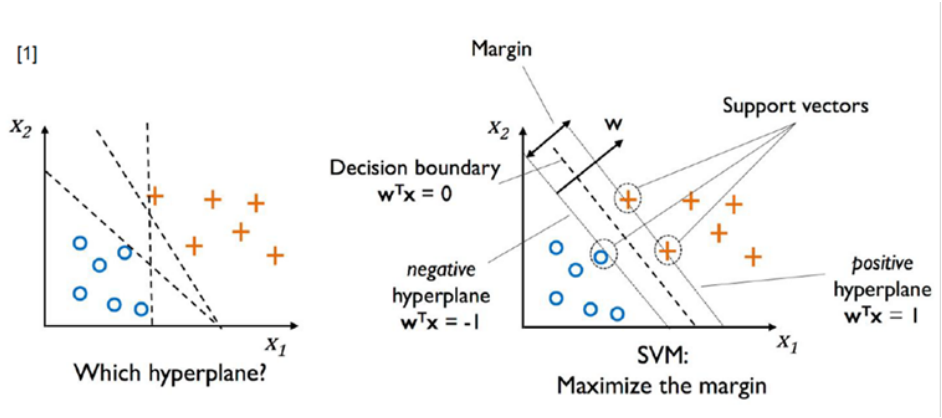
C-verdien er en regulariseringsparameter som kan justeres for å begrense LR-modellen. For små C-verdier øker regulariseringen, noe som igjen vil føre til simplere modeller som gjerne undertilpasser data. For store verdier av C senkes påvirkningen fra regulariseringen. Kompleksiteten i modellen vil da øke, og sjansen for overtilpasning er større. En høy verdi betyr altså at modellen får stor frihet til å tilpasse seg dataene.
C er den inverse av lambda, $C = 1/\lambda$. Lambda kontrollerer modellens mulighet til å øke sin kompleksitet via vektene, samtidig som den prøver å holde seg enkel.

L1 og L2

Reguleringsparameterne L1 og L2 straffer kompleksiteten til LR-modell, og blir brukt til å forhindre over- eller undertilpasning

2.14 Support Vector Machine - SVM

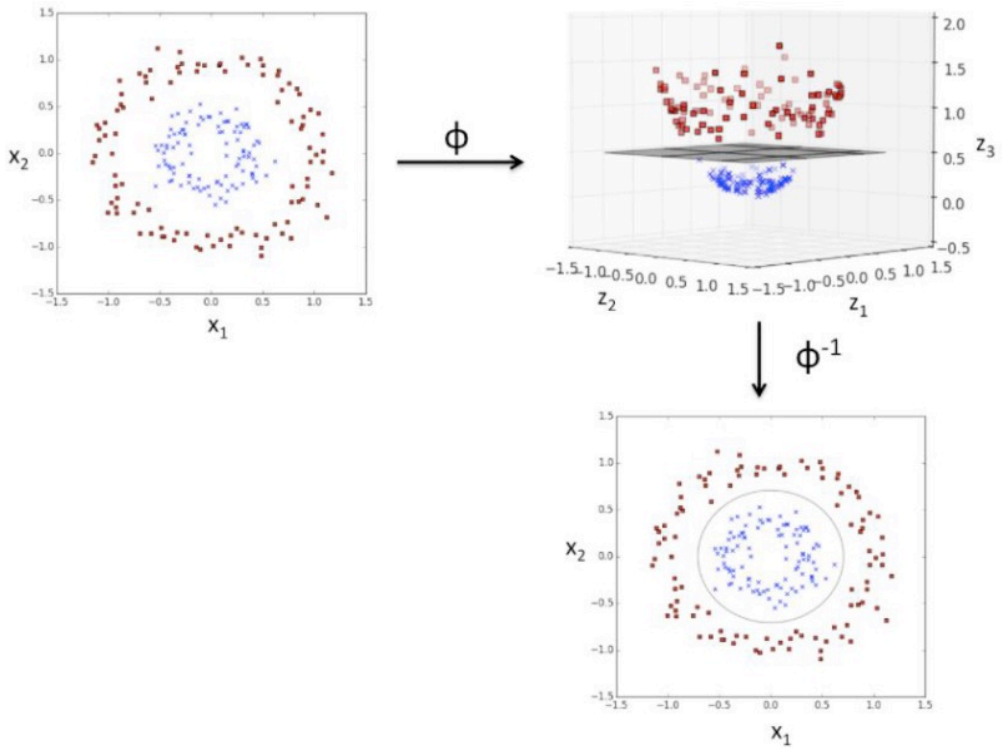
Support Vector Machine (SVM) er den mest brukte algoritmen for hyperspektral dataanalyse(Utsav B. Gewali, 2018). Der LR jobber med å minimalisere feilklassifiseringsfeil, tar SVM utgangspunkt i å maksimere marginer. Marginer er definert som avstander mellom separerende hyperplan, og trenings-samplene som befinner seg nærmest disse separerende hyperplan kalles support vektorer (se figur 2.18).



Figur 2.18 Illustrerer hvordan SVM jobber i ett plan. Når dette fortsetter oppover til et hyperplan blir det ikke mulig å illustrere med en figur (Raschka and Mirjalili, 2017).

Det optimale hyperplanet vil maksimere avstanden til dataene det trenes på, slik at det blir størst mulig separasjon mellom dem. I to dimensjoner er dette hyperplanet den stiplede linjen (figur 2.18) som deler planet i to deler, og marginen SVM jobber med å maksimalisere er den vinkelrette avstanden mellom linjen og de samplene som ligger nærmest. Dette er vanskelig å illustrere i høyere dimensjoner enn to.

SVM kan justeres til å løse ikke-lineære klassifiseringsproblemer ved å innføre kernels (kjerner). Da transformeres lineært useparable data opp i høyere dimensjoner, for slik å finne separerende hyperplan (figur 2.19).



Figur 2.19: Illustrasjonen viser SVM i et hyperplan, og hvordan kernelen jobber for å lage en separasjon (Mueller, 2019)

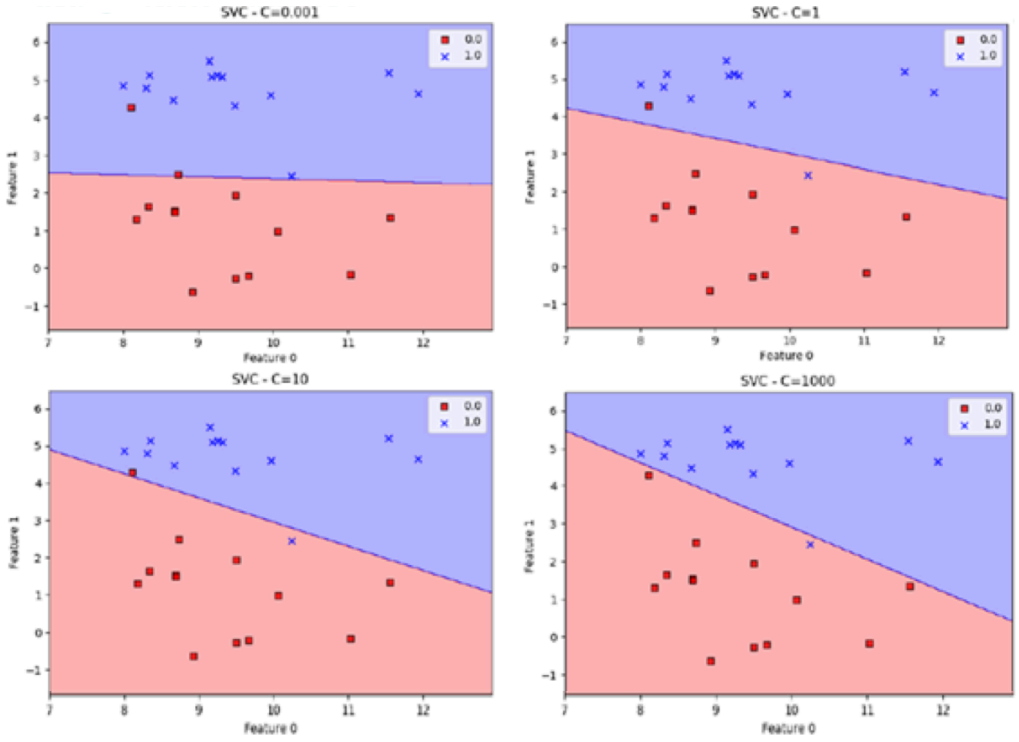
Det finnes flere parametere for å justere og kalibrere en SVM-modell i maskinlæring. De vanligste å bruke (og som er brukt i denne oppgaven) er C, gamma og valg av kernel.

Kernelen

Kernelen er en likhetsfunksjon som transformerer data mellom dimensjoner. Hvilken kernel som velges å bruke, kommer veldig an på datasettet, men tre forskjellige som er vanlige å teste ut er lineær, polynomiell og RBF som er radiell.

C- og gamma-parametere

C-parameteren kan betegnes som en kostnad ved feilklassifisering, der en lav verdi gir hyperplanet større marginer enn en høy C-verdi. Større verdier øker også kostnaden ved å klassifisere samples feil, og fører til en nøyaktigere modell (figur 2.20).

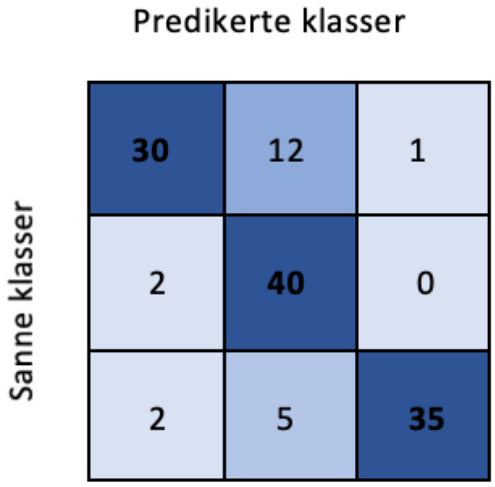


Figur 2.20 Figurene viser hvordan C-parameteren justerer hvor separeringsplanet kan velges. Grensene er generert i python.

2.15 Valideringsparametere

Forvirringsmatrise

En forvirringsmatrise er vanlig å bruke når maskinlæringsmodeller skal evalueres, siden denne viser hvor bra en maskinlæringsalgoritme har prestert. I tillegg kan matrisen brukes til å beregne andre valideringsparametere som accuracy, precision, kappa, recall og F1-verdien, og slik undersøke klassene som har blitt predikert innad. Matrisen er kvadratisk og plotter predikerte verdier mot de sanne verdiene. Diagonalen representerer elementene som har blitt klassifisert rett, alle verdier utenfor denne diagonalen er elementer predikert feil, se figur 2.21.



Figur 2.21: Forvirringsmatrise med tre klasser.

Accuracy – nøyaktighet

Accuracy angir andelen elementer som har blitt predikert rett i en klassifisering, og er svært mye brukt for å gi en generell evaluering av klassifiseringer. En skal ikke stole blindt på denne accuracy-verdien, dette siden den ikke gir informasjon om klassene individuelt. Dersom en klasse tar opp 90% av datasettet og klassifiseringen evalueres med en accuracy på 90%, så betyr dette at modellen ikke nødvendigvis har lært noe nyttig fra variablene sine, selv om en får en score på 90%. Grunnen til dette er at dersom flertallsregelen gjør det bedre enn klassifisereren, er noe feil. En bør derfor vurderer å bruke andre valideringsparametere i tillegg når en trener på ubalanserte datasett (Raschka and Mirjalili, 2017). Accuracy kan beregnes ved å dele summen av diagonalen til forvirringsmatrisen $cm_{diagonal}$ på antall elementer klassifisert totalt cm_{total}, angitt med formel 2.9.

$$acc = \frac{\sum(cm_{diagonal})}{cm_{total}} \tag{2.9}$$

Vi har i denne oppgaven for det meste valgt å ikke angi accuracy som prosentverdier, men som desimaltall mellom 0 og 1, slik standardsettingen er i programspråket vi jobber med.

Kappa-koeffisienten

Kappa-koeffisienten er svært mye brukt innen analyse av fjernmålingsdata for å evaluere hvor bra en klassifisering er (Richards, 2013). Om en tar utgangspunkt i forvirringsmatrisen, kan kappa beregnes ved å sammenlikne predikert accuracy acc_0 med tilfeldig accuracy acc_e (dersom den samme klassifiseringen var tilfeldig). Kappa beregner graden av enighet mellom tilfeldig og predikert (Scikit learn, 2018), som beskrevet i formel 2.10.

$$kappa = \frac{acc_0 - acc_e}{1 - acc_e} \tag{2.10}$$

Siden kappa tar hensyn til tilfeldighet, er det mindre misvisende å bruke accuracy sammen med kappa. Kappaverdien er alltid mindre eller lik 1. (Richards, 2013). Vi har i denne rapporten for det meste valgt å angi kappa som desimaltall mellom 0 og 1, og ikke som prosentverdier.

Tabell 2.1: Tabell hentet fra Richards (2013) gir en forklaring på hva kappaverdien betyr for klassifiseringen

Kappa-koeffisienten	Klassifiseringen kan tolkes som
Under 0,4	Dårlig
0,41 – 0,60	Moderat
0,61 – 0,75	Bra
0,76 – 0,8	Meget god
0,81 – 1,0	Nesten perfekt

Precision – presisjon

Til forskjell fra accuracy ser precision på hvordan det har blitt klassifisert innenfor en og en klasse. Precision kan derfor være nyttig å bruke når det er stor ubalanse i antall enheter klassene imellom i datasettet. Precision defineres som mengden korrekt klassifisert innenfor en klasse $c_{korrekt}$ i forhold til antallet predikert som denne klassen c_{pred}(Raschka and Mirjalili, 2017), beskrevet i formel i formel 2.11.

$$prec = \frac{c_{korrekt}}{c_{pred}} \tag{2.11}$$

Recall – gjenkalling

Akkurat som precision, evaluerer recall hvordan en klassifisering har prestert ved å undersøke hver klasse innad. Detter gjøres ved å ta antallet som har blitt korrekt klassifisert innenfor en klasse og dele på alle elementene som tilhører denne klassen c_{total} (Raschka and Mirjalili, 2017), beskrevet i formel formel 2.12.

$$rec = \frac{c_{korrekt}}{c_{total}} \tag{2.12}$$

F1-verdi

F1-verdi er et vektet gjennomsnittet mellom recall og precision, beskrevet i formel 2.13.

$$F1 = 2\frac{prec \times rec}{prec + rec} \tag{2.13}$$

F1-verdien ligger mellom 1 og 0, der 1 betyr perfekt og 0 betyr elendig (Raschka and Mirjalili, 2017).

Loss-funksjon

For å undersøke CNNs prestasjoner blir det brukt en loss-funksjon, denne beregner ut cross-entropien. Ved å minimere denne verdien, vil en også optimalisere nettverket (SuperDataScience Team, 2018). Under kan en se to eksempler på en slik funksjon, formel 2.14 blir brukt på nettverk med flere klasser, der y er en liste med alle klassene. Den andre formelen 2.15 blir vanligvis brukt ved binære klassifiseringer.

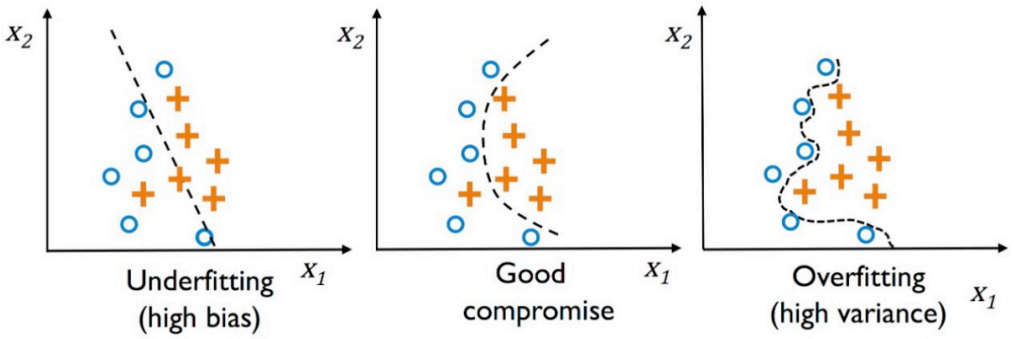
$$D_{categorical}(\hat{y}, y) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{2.14}$$

$$D_{binary}(\hat{y}, y) = -(ylog(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{2.15}$$

Her er y de sanne klassene til dataene, mens $\hat{y}$ er de predikerte verdiene. K er antall rader med data. Loss funksjonen tar utgangspunkt i gjeldende parametere og sier hvor flink modellen er til å predikere (ML Cheatsheet, 2017).

Cost-funksjon
Cost-funksjonen er veldig lik loss-funksjonen, men skiller seg fra denne ved at cost- funksjonen er mer generell. Den forteller hvordan en skal endre parameterne for å gjøre modellen mer nøyaktig. Cost-funksjonen blir bruk i algoritmen gradient descent (ML Cheatsheet, 2017).

Over- og undertilpassing
Når en modell presterer bra på treningsdata og gjør det dårlig på usette testdata, indikerer dette at modellen er overtilpasset. Dette betyr at modellen ikke har greid å generalisere informasjonen i variablene. Modellen vil ha stor varians som kan komme av for mange parametere i modellen. Dette gjør at modellen blir for komplisert og dermed tilpasser seg for bra til det aktuelle datasettet. Undertilpassing oppstår der modellen ikke kompleks nok og ikke greier å lagre all den nyttige informasjonen om mønstre og kjennetegn i datasettet. Dette fører til en stor bias-verdi om gjør at modellen presterer dårlig på usette testdata. Se figur 2.22.



Figur 2.22:Illustrerer undertilpassing, godtilpasning og overtilpassing (Raschka and Mirjalili, 2017).

3 Metode

3.1 Databestilling

Terratec AS produserte og leverte i 2017 flere datasett med hyperspektrale flybilder for Plan- og bygningsetaten i Oslo kommune. Det var bestilt data av et utvalgt testområde over byen, og det ble det fløyet 9 striper over et område på ca. 17 km^2, se figur 3.1. Stripene hadde en linjeavstand på 300 meter og en bredde på 506 meter, og sideoverlapp på stripene var ca. 23 %. Preprosesseringen som ble gjort var ortorektifisering og georeferering de ble levert med transformasjonen WGS84, UTM32. Ortorektifisering handler om å korrigere laveste og høyeste punkt i flybildet slik at størrelsesforholdene blir like, og slik hindre at høye bygninger/fjell ikke ta for mye plass i bildet. Ortorektifiseringen ble gjort basert på et 20 cm digital overflatemodell innhentet 19.07.2017 fra en ALS70 laserskanner.

Det er i leveranserapporten beskrevet noen pitch-problemer i stripe 4 og noen roll-problemer i stripe 3. De ble derfor gjort korreksjon i offset på heading, roll og pitch ut fra manuelt utvalgte kontrollpunkt fra et ortofoto som ble brukt som referanse for posisjoneringen (Terratec, 2017a). Det er også nevnt at det er noe ulik solinnstråling på ulike flylinjer, på grunn av forsinkelse under billedtagningen. Dette er vanlig med flybilder over Oslo, da flylinjene ligger i innflygningssonen til Gardermoen flyplass.

Figur 3.1: Den 19.07.2017 gjennomførte Terratec AS den flybårne hyperspektrale datainnsamlingen i form av 9 flystriper. Bildet til venstre tatt fra flyet viser værforholdene under billedtagningen. Bildet til høyre viser flystripene i den rekkefølgen de ble flydd (Terratec, 2017a)

Først ble det levert ferdig mosaikk av VNIR-bildene og SWIR-bildene, samt rå-filer i form av flystriper. Figur 3.2 viser VNIR-mosaikken. Det er disse to mosaikkene vi i denne oppgaven kaller for radians-bildene. Med hvert bilde fulgte en HDR-fil som åpnes samtidig med bildet. I disse ligger alle spesifikasjoner som trengs for videre bildebehandling, og også viktige metadata.

Figur 3.2: VNIR-mosaikken

Senere ble en tilleggsleveranse overlevert, denne inneholdt blant annet atmosfærekorrigerte data og normaliserte data, begge i både VNIR og SWIR. Det ble og levert NDVI-maske for hele settet (Terratec, 2017b). De atmosfærekorrigerte data er de vi i denne oppgaven kaller reflektans-bildene. Disse ble ikke levert som en mosaikk, men i form av enkeltstriper. Ut fra radians- og reflektans-settene ble det etablert et utsnitt over et prosjektområde på ca. 13 km^2.

3.1.1 Sensor-spesifikasjoner HySpex

I flyet Terratec brukte til datainnsamlingen var det montert to hyperspektrale sensorer, HySpex VNIR-1800 og HySpex SWIR-384. Sensorene er produsert av Norsk Elektro Optikk AS, et norsk firma etablert i 1985, med utspring fra Forsvarets Forskningsinstitutt. Sensorene ble originalt laget for fjernmålingsbruk, men passer og til laboratoriebruk med andre typer optikk (Norsk Elektro Optikk AS, u.å). Disse sensorene fungerer etter pushbroom-prinsippet, når data samles inn, skanner kameraene samtidig både all spektralinformasjon fra en og en smal linje, og det romlige området foran/under kameraet. Det ferdige datasettet inneholder slik både det romlige planet i x- og y-retninger, og den sammenhengende spektrale informasjonen for hvert piksel. (Pu, 2017) Til sammen former dette en hyperkube. VNIR-sensoren er en CMOS-detektor (Complementary Metal-Oxide Semiconductor), mens SWIR-sensoren er en MCT-detektor (Mercury Cadmium Telluride).

3.1.1.1 VNIR og SWIR

VNIR står for Visible Near InfraRed. Det er mange ulike definisjoner på hva som skal være inkludert i VNIR-spekteret, men det som går igjen er at det synlige spekteret og deler av den nærinfrarøde delen inkluderes. Påstandene splittes når en skal bestemme hvor i det nærinfrarøde denne skal stoppe. I denne oppgaven er definisjonsområdet for VNIR satt fra 400nm til 1000nm, etter HySpex-sensoren som er benyttet. VNIR-datasettene består av 186 smale bånd i spekteret 407 til 997 nanometer. Bildene har en romlig oppløsning på 30cm, slik at hver piksel dekker et bakkeareal på 30x30 cm.

SWIR står for ShortWave InfraRed, og er i denne oppgaven definert den som bølgelengdeområdet 1000nm til 2500nm etter HySpex sensoren SWIR-384. SWIR-datasettene består av 288 smale bånd i spekteret 955 til 2523 nanometer. Bildene har en romlig oppløsning på 70cm, slik at hver piksel dekker et bakkeareal på 70x70 cm. Spesifikasjonene for hver sensor kan ses i tabell 3.1.

Tabell 3.1: Gir informasjon om HySpex-spesifikasjoner for to sensorer (Norsk Elektro Optikk AS, u.å).

	VNIR - 1800	SWIR - 384
Bølgelengdeområde [nm]	407 - 997	955 - 2523
Antall romlige piksler	1800	384
Antall spektrale bånd	186	288
Spektral oppløsning [nm]	3.26	5.45
Romlig oppløsning (GSD) [cm]	30	70
Maks åpningsvinkel (FOV)	17 grader	16 grader
Pixel FOV across/along [mrad]	0.16/0.32	0.73/0.73
Radiometrisk oppløsning	16 bit	16 bit
Dynamisk rekkevidde	20000	7500
Maks bildefrekvens	260 fps	400 fps

3.2 Testområder

Mosaikkene som ble levert hadde en total størrelse på 274GB (VNIR) og 78GB (SWIR). Det å jobbe med filer på denne størrelsen i en datamaskin byr fort på store problemer, så løsningen på dette var å velge seg ut noen mindre arbeidsområder, vi kaller det videre testområder, og klippe til disse ut fra rådata/mosaikk. Det var viktig at disse områdene ikke ble for store i utstrekning og dermed filstørrelse pga ovennevnte problem.
Det ble brukt fire testområder, se figur 3.3. Det som var mest interessant for studiefeltet i denne oppgaven var egentlig tett bebygde og asfalterte urbane områder, slik som Grünerløkka og Gamlebyen. Et stort problem her når det kommer til feltarbeid for fasitproduksjon, var tilgangen til bakgårder. De aller fleste er private og låst, og ikke tilgjengelig for publikum.

Figur 3.3: Alle testområdene er markert med et rødt rektangel. Viser Tøyenområdet øverst til vestre. Vallehovinområdet til høyre. Gamle Oslo ligger under Tøyenområdet, og helt nederst i en veldig liten rød firkant er Ekeberg.

Siden oppgaven ikke skulle startes på før i januar 2019, og det da pleier å ligge mye snø i Oslo, ble det konkludert med at et feltarbeid kunne være umulig å gjennomføre på dette tidspunktet. Derfor ble det tatt noen valg senhøsten 2018, slik at en fikk dokumentert

arealdekket da. Det ble bestemt å velge to store, noenlunde like områder med stor variasjon innen arealtyper, men også nok av hver aktuell type til at det var nok til å produsere en fasit. Valget falt til slutt på to ganske like områder, slik at en kunne bruke ett som "arbeidsområde" og ett som valideringsområde til slutt. Begge områdene var preget av klassemangfold og tilgjengelighet. De hadde et rikt utvalg av gress, trær, buskas, kommunale og private asfalterte veier, grusområder og vannflater.

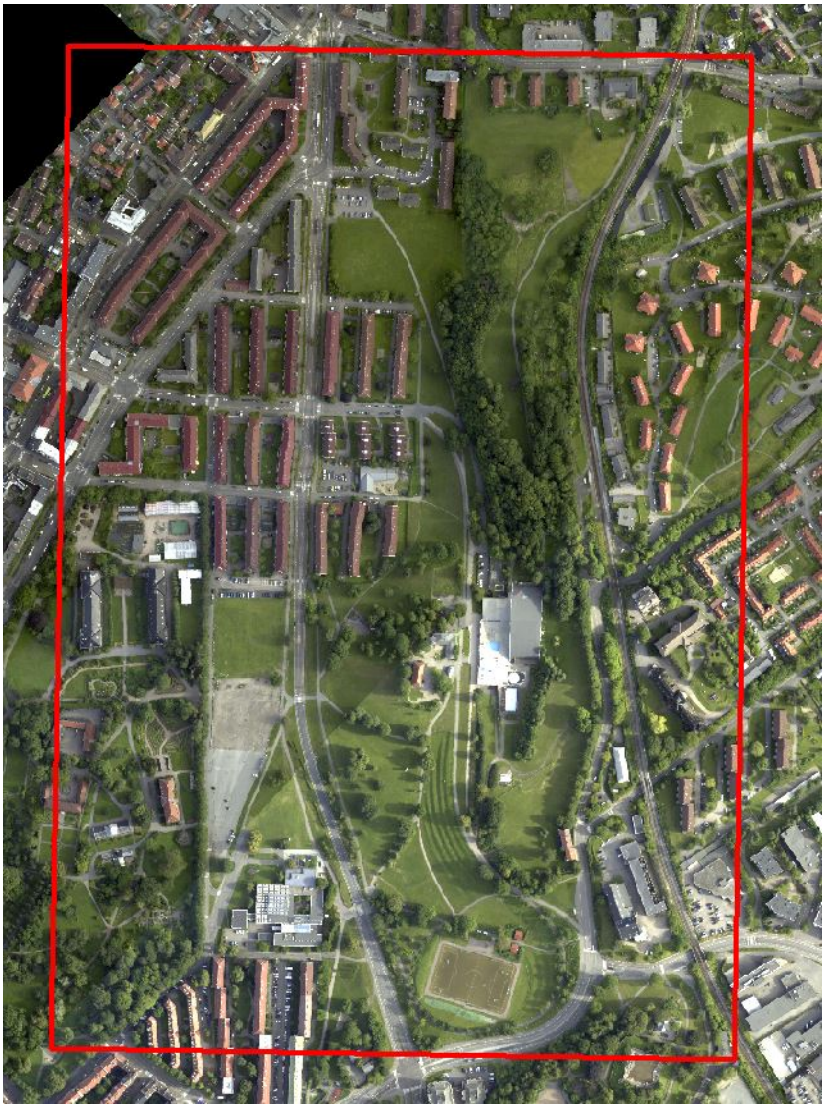
3.2.1.1 Valle Hovin

Det første området som ble valgt var treningsområdet, kalt Valle Hovin i denne oppgaven, se figur 3.4. Dette området har blitt brukt til å trene våre modeller, teste disse modellene og produsere diverse varianter av en bakkefasit

Figur 3.4: Bildet viser utsnittet for trening- og testområdet Valle Hovin

3.2.1.2 Tøyen

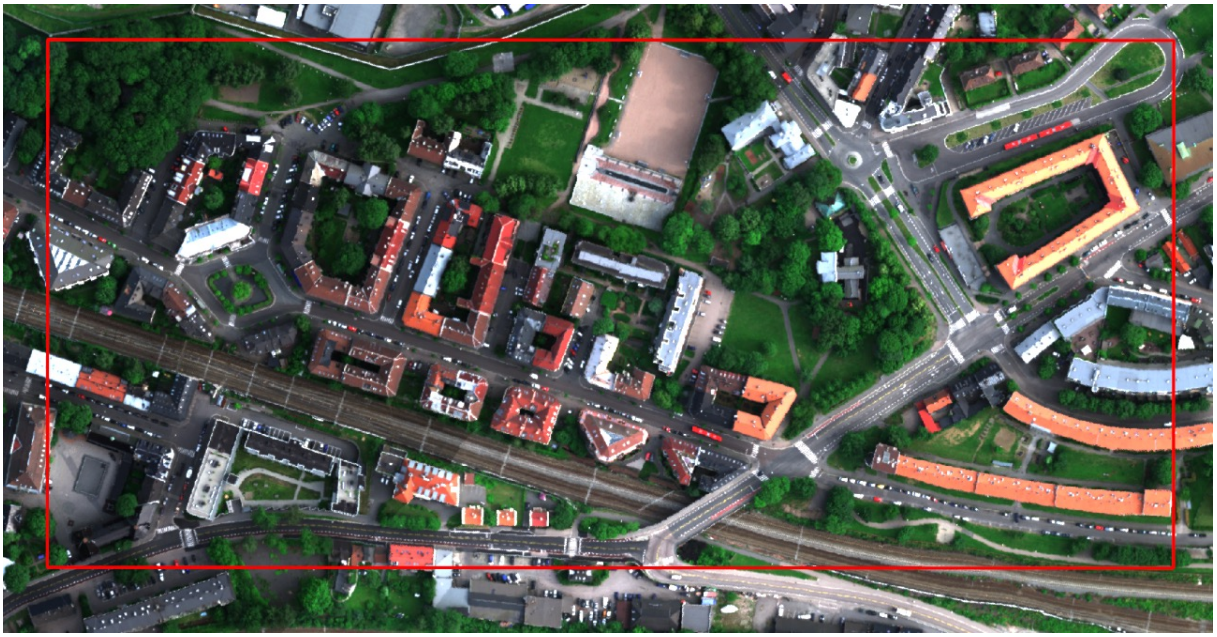
Det andre store området var et område over Tøyenparken og Keyserløkka, dette området blir kalt Tøyen i denne oppgaven, se figur 3.5. Etter at alle modellene var ferdig trent og optimalisert, ble Tøyen-området brukt til å gi et endelig svar på hvor bra modellene egentlig var. Det ble ikke gjort noe mer med modellene etter at det ble testet på dette området. Det har derfor vært viktig at Tøyen- og Valle Hovin-området hadde visse likheter i forhold til hverandre, men samtidig at de ikke har hatt noen geografisk kontakt med hverandre.



Figur 3.5: Valideringsområdet. Et område over Tøyenparken og Keyserløkka.

3.2.1.3 Gamle Oslo

Det ble i tillegg valgt ut et område i Gamlebyen, området blir i denne oppgaven kalt Gamle Oslo, se figur 3.6. Området representerer et typisk Oslo-område, det vil si leilighetskomplekser med bakgårder og mye veier og andre asfalterte arealer. Området har i denne oppgaven blitt brukt som et visualiseringssett.



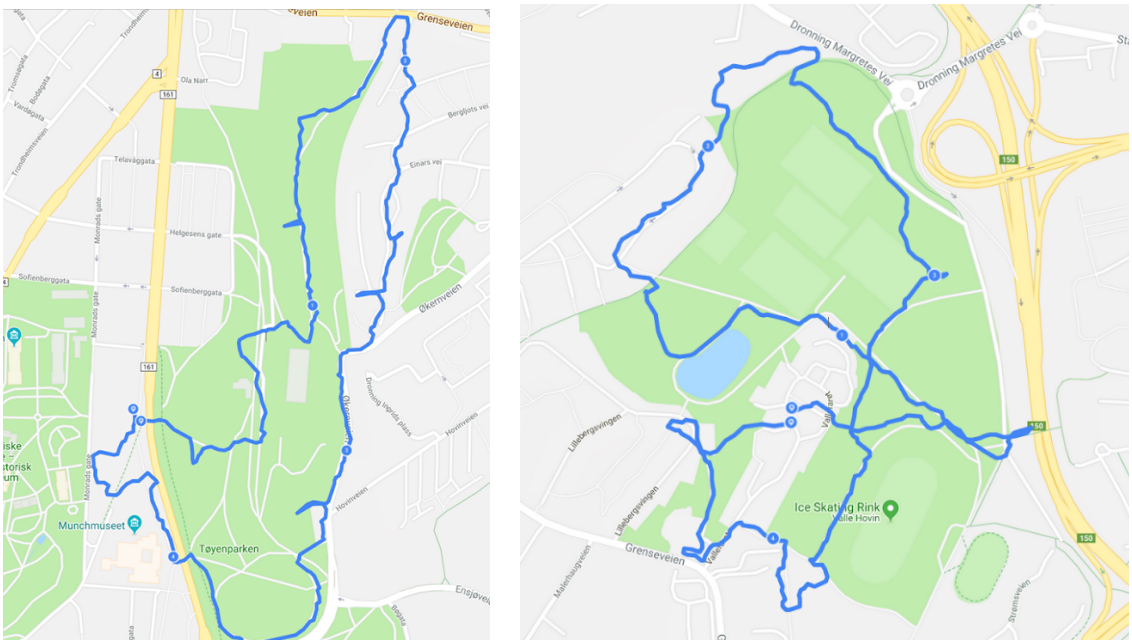
Figur 3.6: Visualiseringsområdet Gamle Oslo.Utklippet ligger rett sør for Oslo fengsel og Galgeberg, i bydel Gamle Oslo.

3.2.1.4 Ekeberg

Det ble på et tidspunkt i oppgaven et problem med at en trengte mer gruspiksler til trening. Det var mye snø på dette tidspunktet, så feltarbeid var utelukket. Den eneste store "gruskilden" som ble funnet var Sirkustomta, en parkeringsplass med grusdekke ved Tøyenparken, men denne måtte spares til selve valideringen. Derfor ble en grusbane på Ekeberg valgt, denne var utenfor de valgte testområdene. Ved en inspeksjon senere da noe av snøen hadde smeltet, viste det seg at dette var en hesteridningsbane med sand/grusdekke, altså ikke bare grus.

3.3 Feltarbeid

Det ble høsten 2018 utført feltarbeid på Tøyenområdet og Valle Hovin-området, arbeidet ble utført etter anvisninger beskrevet i URBAN EEA VEGETATION SURVEY (Skarpaas, 2017). Feltarbeidet som ble utført i denne oppgaven har hatt to formål. Den ene var å dokumentere landskapet og arealtyper dette inneholdt, det andre var å bli godt kjent med området, altså få det litt "under huden". Det var for eksempel veldig nyttig å gå rundt og betrakte all asfaltvariasjonen som fantes, noe som ble nyttig etterhvert når klassifiseringsproblemer oppsto. Det ble det utført grundig bildedokumentasjon av områdene, under billedtagningen ble det benytte GPS og geotagging av bildene slik at en etterpå kunne plassere alle bildene og ruten på et kart. I tillegg ble ruten tracket med en sporingsapp for ved hjelp av denne å raskt kunne finne rekkefølgen bildene ble tatt i måneder etterpå. Det ble tatt bilder i alle retninger av de forskjellige arealdekkene, og nærbilder av detaljer. Ruten for Tøyen kan ses i figur 3.7.



Figur 3.7: Rutene som ble gått under feltarbeid på til venstre Tøyen og til høyre Valle Hovin..

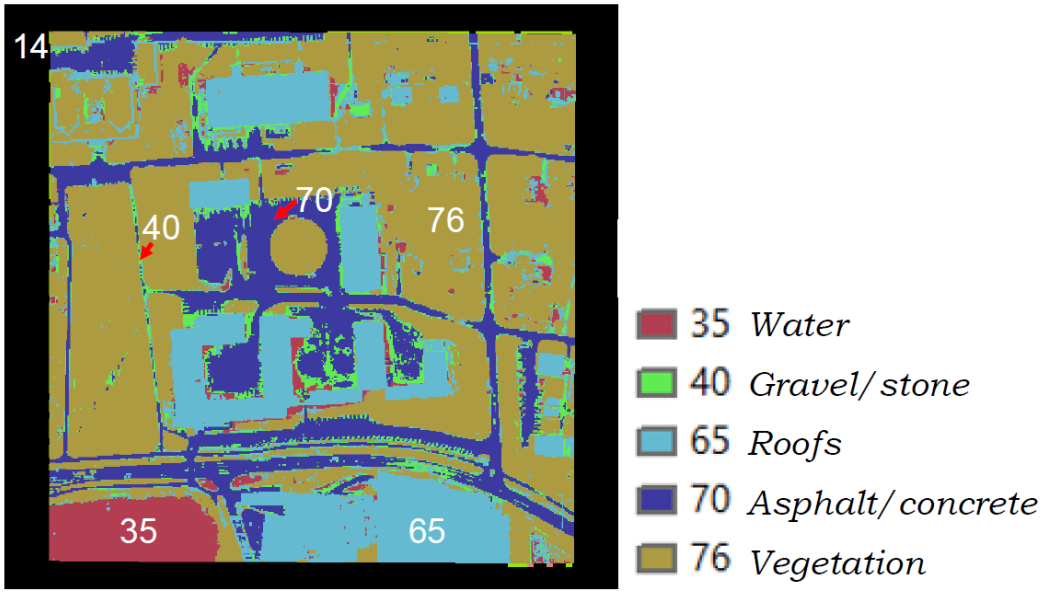
Feltarbeidet som ble gjort dokumenterer ikke 100% korrekt forholdene, dette siden det hadde gått 1,5 år fra bildene blir tatt til dette feltarbeidet ble utført. Mye har skjedd på noen av områdene, for eksempel har nedre del av Valle Hovin-området vært et byggefelt de siste årene. Det var derfor ikke mulig å kartlegge deler av disse områdene. På både Tøyen og Valle Hovin har det flere steder blitt lagt helt ny asfalt i denne perioden på 1,5 år.
Det var altså umulig for oss å vite hvordan bakken og vegetasjonen faktisk så ut på nært hold på opptaksdagen for de hyperspektrale bildene. Hvor mye grus fantes det for eksempel på asfalten denne dagen? Ved en ny inspeksjon av noen av de aktuelle klassifiserte veiene i mars 2019 var asfalten dekket av grus. Etter vårrengjøring i gatene er mye av slik "vintergrus" feiet bort, men hvor mye? Og gjaldt dette også på gangfeltene? Det er altså mye usikkerhet knyttet til det som ble dokumentert underfeltarbeidet, og det er derfor viktig å huske på at denne usikkerheten ble videreført da det ble produsert arealfasit for områdene.

3.4 Fasit

For å kunne styre en klassifisering må man ha sikker informasjon å mate modellen med. I arealklassifisering heter dette på engelsk "ground truth", altså bakkefasit, og blir i denne oppgaven bare kalt fasit. Det er veldig viktig med en god fasit, da slurv fører til feillæring, det er viktig å gi klassifiseringsmodellen et bra læringsgrunnlag. I vårt tilfelle er da fasiten at vi sier noe om hva et enkelt hyperspektralt piksel inneholder, altså hvilket materiale som er avbildet. Modellen lærer seg så karakteristikken til disse fasitpikslene med pikselinformasjonen den blir matet med. Når den er ferdig opplært blir modellen testet på nye piksler som den skal klassifisere, slik at resultatet kan sjekkes opp mot fasitverdien, og det kan bestemmes om modellen har valgt riktig eller ikke. Dersom det er mye feil i fasiten, kan dette ødelegge læringen. Konklusjonen er jo bedre modell man lager, dess bedre rustet er modellen til å velge riktig klasse for et helt ukjent datasett.
Datasettene som ble brukt hadde veldig høy romlig oppløsning (piksler på enten 30x30 eller 70 x70 cm), men selv med denne oppløsningen bestå en stor andel av pikslene av en blanding av forskjellige materialer, såkalte "mixed pixels". Det var ikke ønskelig å ha flere materialer blandet i ett fasitpiksel, siden dette fører til kombinerte strålingsverdier for ulike materiell, noe som forvirrer modellen. Under klassifiseringer er mixed pixels noe en må forholde seg til, men siden det i vår oppgave ble testet ut forskjellige arbeidsmetoder på veldig få klasser og det ikke skulle leveres et ferdig produkt i form av et klassifisert område, ble det derfor valgt å fokusere på å lage en fasit som unngikk mixed pixels, for å få så "rene" klasser som mulige.

3.4.1 Klasser

Siden de to nevnte masteroppgavene som ble skrevet for Oslo kommune våren 2018 bare forholdt seg til trær (Røstad, 2018, Barane, 2018) ble det bestemt å bruke flere typer klasser i denne oppgaven. Det var samtidig ønskelig med ikke for mange klasser, slik at arbeidet ble unødvendig komplisert i forhold til den tiden en hadde tilgjengelig. Det ble valgt å ta utgangspunkt i Mannings ruhetstall og noen få aktuelle klasser utledet fra disse. I utgangspunktet ble det foreslått fem klasser, disse var vegetasjon, asfalt, tak, vann og annet permeabelt (grus). Disse klassene kan ses i figur 3.8.



Figur 3.8: Et raster med Mannings ruhetstall laget til et forskningsprosjekt ved NMBU (Thiis et al., 2018). Dette ble brukt til inspirasjon for våre klassevalg.

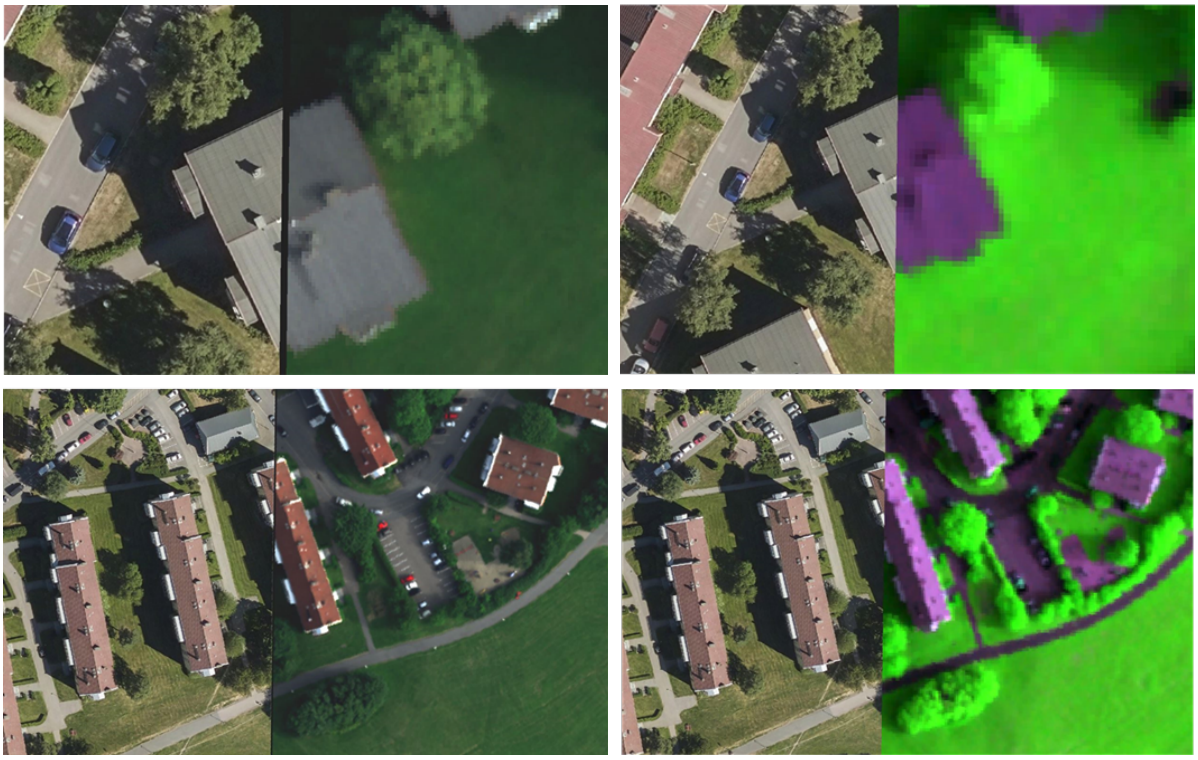
Tak-klassen var utfordrende siden det finnes så veldig mange ulike tak-materialer. I tillegg vanskeliggjorde takklassen feltarbeid for fasitproduksjon på grunn av tilgang i høyden, og siden det var vanskelig å gjenkjenne tak-materialer fra bilder og ortofoto. Det finnes allerede nøyaktige datasett for takoverflater og vann i Oslos FKB-data, derfor ble ikke disse klassene prioritert å bruke videre.

Klassene det ble valgt å jobbe videre med ble derfor vegetasjon, grus og asfalt/betong. Disse klassen ble delt plassert i nye grupper kalt steinbasert og vegetasjon. Den steinbaserte gruppen inneholdte etterhvert klassene asfalt, betong, grus og sand, siden betong ble skilt fra asfalt og sand fra grus. Dette ble gjort siden det ble antatt at disse klassene ville ha en litt forskjellig spektralsignatur. Sand ble senere fjernet som egen klasse på grunn av for få sandpiksler i treningssettet.

Vegetasjon ble delt opp i flere klasser, både 3 og 4. Disse klassene var tre, gress, naturlig busk og plantet busk. Buskklassene ble etter hvert slått sammen til bare til busk. Busk ble definert som alt høyere enn gress (eng) og lavere enn trær (fem meter). Tanken bak vegetasjonsklasseoppdelingen var å undersøke om det var mulig å skille disse klassene fra hverandre ved bruk av bare hyperspektrale flybilder.

3.4.2 Fasitproduksjon

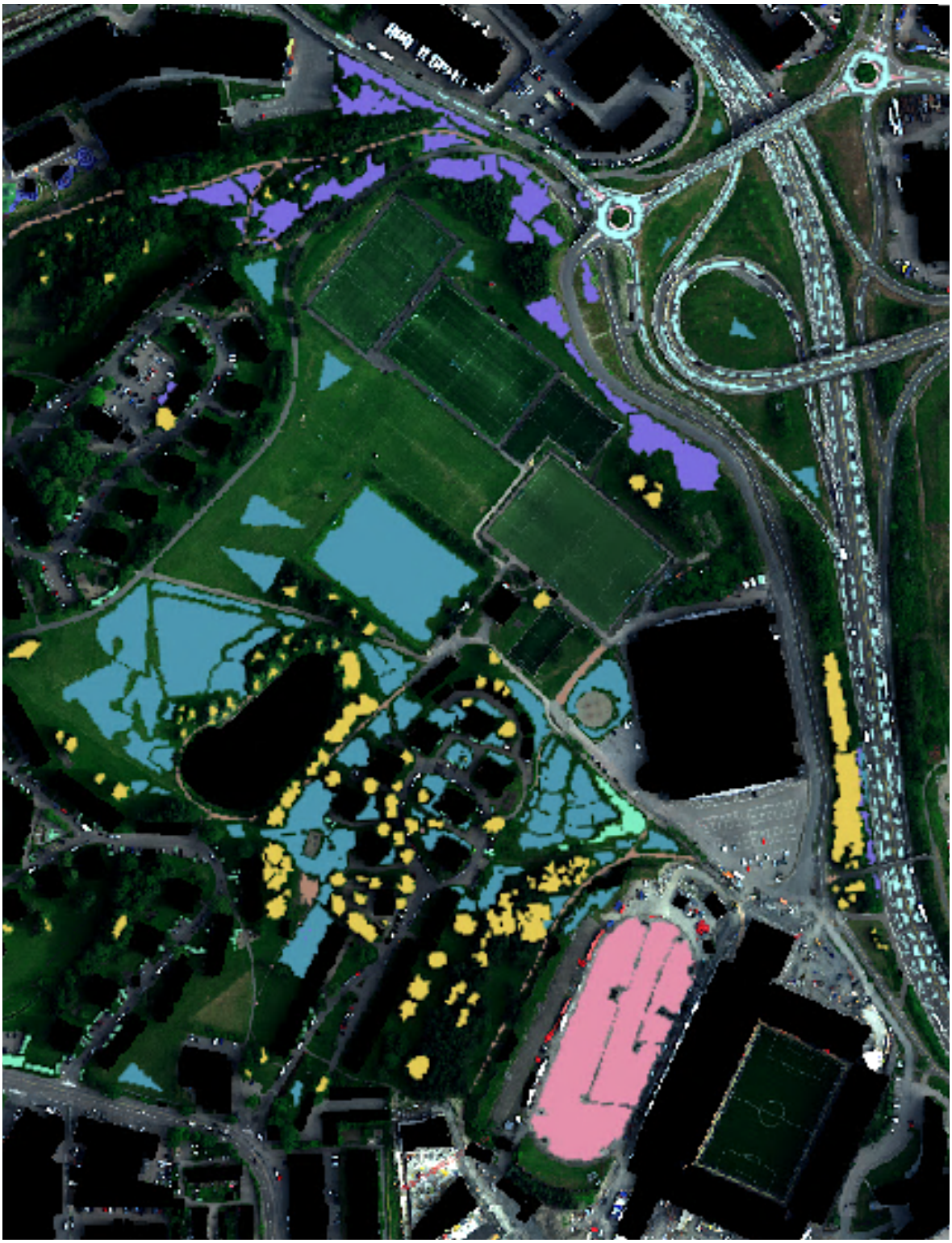
Det er ekstremt viktig å klassifisere med en bra fasit, fasitarbeidet har derfor vært noe som det har blitt lagt ned veldig mye tid i å produsere. Da fasiten ble produsert ble det tatt utgangspunkt i flere forskjellige kilder. Bildene fra feltarbeidet og ruten som hadde blitt gått i kartet (fra sporings-app) var hovedkilden. De ble brukt for å finne arealdekke på testområdene. I tillegg ble de hyperspektrale bildene av hele området benyttet, der VNIR ble brukt med RGB-farger og SWIR i en fargekombinasjon som skilte ut vegetasjon (grønt) og alt annet som lilla. Disse var veldig viktige for å sjekke at området var likt i forhold til feltarbeidet, se figur 3.9. Det ble i tillegg også brukt ulike ortofoto fra de siste årene og Google Street View som et supplement.



Figur 3.9: Består av fire bilder, der de til venstre er to bildeutklipp som viser forskjellen på oppløsningen til VNIR og et ortofoto. Bildet til høyre viser forskjellen i oppløsningen mellom SWIR og et ortofoto. Bildene ble lagt oppå hverandre under fasitproduksjonen, slik at det var mulig å sammenlikne bildene i forhold til hverandre, og samtidig ikke bevege seg utenfor gjeldene pikseloppløsning.

Under fasitproduksjonen ble bildet med dårligst oppløsning, altså SWIR, og denne pikselstørrelsen brukt til å diktere hvor grensene for de forskjellige klassene skulle gå når klassegrensene ble tegnet opp. Produksjonen av fasit ble gjort i ArcGis Pro, der de forskjellige flybildene ble lagt som lag oppå hverandre. Det ble opprettet forskjellige feature classes, en for hver klasse. Det ble tegnet polygoner over de utvalgte pikslene som omfattet den enkelte klasse.

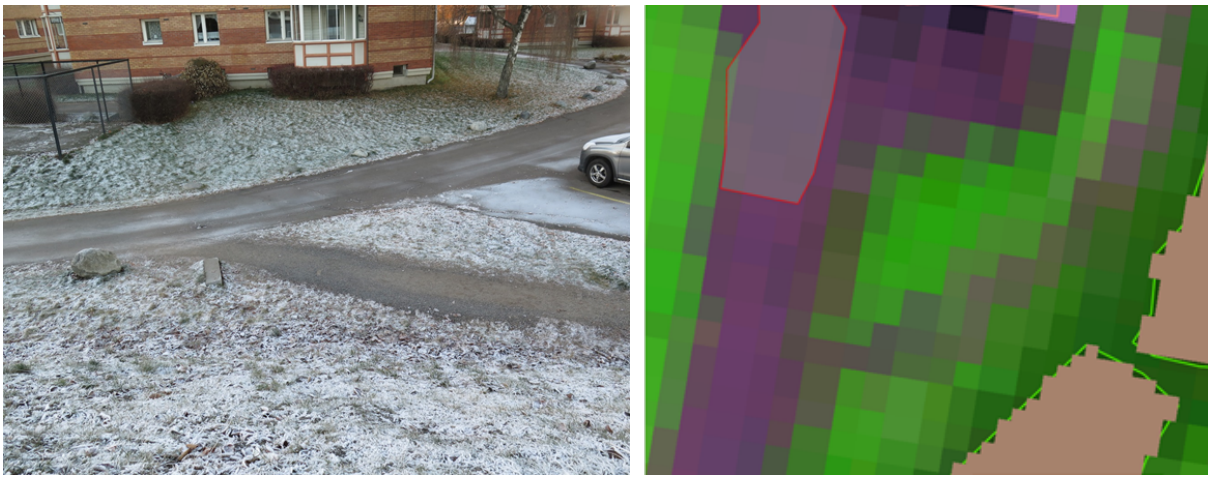
Det ble laget flere versjoner av fasiter for Valle Hovin-området. Dette ble gjort da det skulle undersøkes hvor lett det var å skille klassene innenfor gruppene steinbasert og vegetasjon fra hverandre. I figuren 3.10, kan en se et utklipp fra en av fasitene som ble brukt i oppgaven.



Figur 3.10:Bildet gir et overblikk over Valle Hovin, med fasit lagt over.

3.4.3 Antall piksler per klasse i fasit

Det er oftest tryggest å bruke balanserte klassifiseringer, at det befinner seg et likt antall piksler i hver klasse, siden ubalanse mellom de ulike klassene kan ha negativ påvirkning på styrt klassifisering. Det er også nødvendig med et stort antall med treningsdata i styrt maskinlæring for å kunne utnytte de hyperspektrale bildedataene (Ghamisi et al., 2017). Det ble derfor bestemt at hver klasse måtte inneholde minst 15000 piksler i fasiten. For klassene gress og trær var dette enkelt, siden det fantes store områder med begge typer representert, men for klassen grus var det mer komplisert. Det var vanskelig å finne nok gruspiksler til treningssettet, og de områdene som fantes besto oftest av tynne grusveier, med få piksler. Siden det var så lite grus måtte grusstiene inn i fasiten. Det er derfor en viss fare for "mixed pixels" i grusklassen, da disse stiene knapt var bredere enn en meter. I figur 3.11 kan en se hvordan pikslene til SWIR så ut for en slik sti.



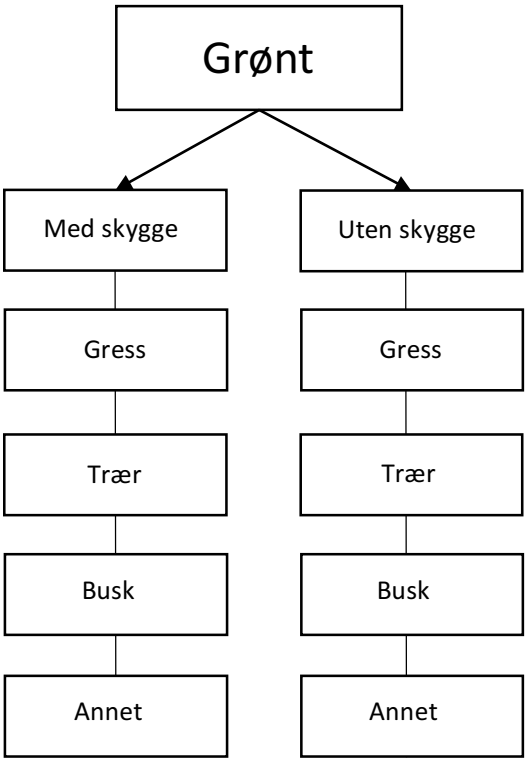
Figur 3.11: Viser et kryss der en grusvei møter en asfaltvei. Bildet til venstre er tatt under feltarbeidet, mens bildet til høyre er SWIR-bildet over samme område, under fasitarbeidet. SWIR-bildet illustrerer vanskeligheten med å få rene gruspiksler til grusklassen.

3.5 Ulike datasett

Det fantes tre ulike datasett vi hadde mulighet til å jobbe med i denne oppgaven. Disse hadde Terratec generert fra data samlet i samme opptakssituasjon (se avsnitt 3.1). Datasettene var originalsettet kallet radians, et atmosfærekorrigert kalt reflektans og et normalisert sett. I det normaliserte sette hadde hver pikselverdi blitt dividert på summen av verdiene i alle bånd for samme piksel. Målet med dette var å minimere skyggeforskjellen, men dette førte heller til at det ble overeksponert i skyggeområder og undereksponert i områder uten skygge. Det normaliserte settet ble ikke brukt i denne oppgaven.
Det var svært vanskelig å bestemme seg for hvilket av datasettene radians eller reflektans som skulle brukes til klassifiseringene i denne oppgaven. Dette grunnet usikkerhet mellom de som produserte dataene (Terratec) og forskermiljøene (Utsav B. Gewali, 2018) om hvilket som var mest optimalt til klassifisering av arealtyper. Det ble derfor beslutte å sammenligne radians- og reflektanssettet, og så jobbe videre i resten av oppgaven med det settet som presterte best, se avsnitt 4.2.1 for resultat.

3.5.1 Radians

Det første settet som ble levert var radianssettet, i form av både mosaikk og enkeltstriper. Det er og det settet som ble produsert nærmest i tid i forhold til opptakstidspunktet, og også det som er påført færrest korreksjoner i preprosessering. Grunnen til at dette blir poengtert er at enhver matematisk bearbeiding av et datasett fører med seg mer usikkerhet og slik flere potensielle feilkilder. Det er derfor trygt å si at dette settet inneholder mest informasjon, men samtidig også mest støy. Dette settet er ikke atmosfærekorrigert og kan derfor ikke benytte seg av spektralbiblioteker, noe et reflektanssett har mulighet til. Det er heller ikke blitt gjort noe med skyggeproblematikken i radianssettet.
Skygger reduserer mengden lys i et område. Det betyr at to like materialer med og uten skygge, vil gi ulike spektralinformasjon. Dette siden skygger reduserer lysinnstrålingsverdiene i alle båndene til de påvirkede pikslene (Smith, 2012). En løsning som ble vurdert for dette problemet ved klassifisering, kan være å lage egne skyggeklasser og ikke-skyggeklasser, se figur 3.12.



Figur 3.12: Vises et eksempel på en mulig klasseinndeling vi lenge vurderte, der det lages klasser med og uten skygge.

Det å doble klassene, betyr også og doble antall piksler, noe som ikke var aktuelt på grunn av det allerede eksisterende problemet med å få nok piksler innenfor hver klasse. Derfor ble denne løsningen forkastet selv om dette sannsynligvis hadde hjulpet i forhold til skyggeproblematikken i bildene.

3.5.2 Reflektans

Reflektanssettet ble levert som 9 flystriper for VNIR og SWIR, og hadde en total størrelse 4,7 TB. Det var derfor svært tungt å jobbe med settet, selv om maskinene som ble brukt til dette var kraftige og hadde godt med lagringsplass. Det var nødvendig å sette alle flystripene sammen til en mosaikk før eventuelle testområder kunne klippes ut. Dette var en tungvint og tidkrevende prosess på grunn av de store datamengdene. Akkurat som radianssettet hadde settet heller ikke blitt korrigert for skygger.

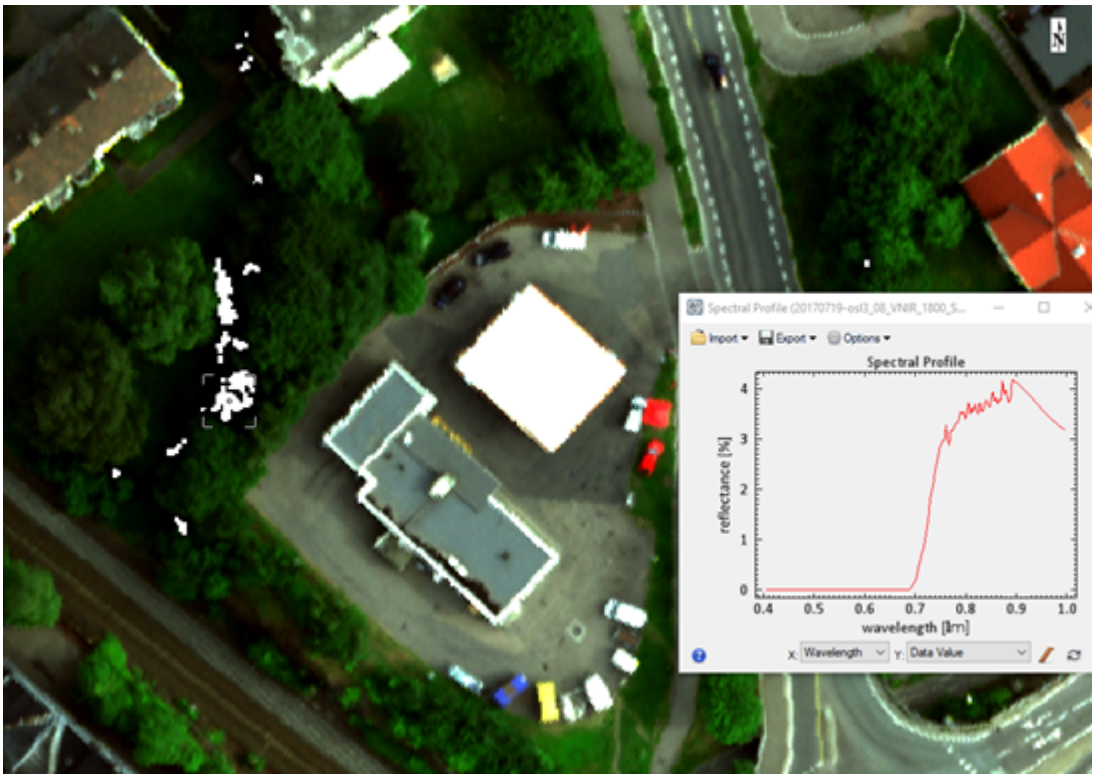
Et av hovedargumentene for å bruke reflektanssettet var at settet ga de faktiske stråleegenskapene til materialet. Noe som er verdt å merke seg er at reflektanssettet som ble benyttet ikke er absolutt reflektans som gjerne blir målt i laboratorier, men atmosfærekorrigerte radiansverdier. Det vil si at man har tatt utgangspunkt i radianssettet og påført settet en atmosfærekorreksjon. Det var ATCOR4 som ble benyttet til.

Atmosfærekorreksjonen fjerner absorbsjonsartefakter fra atmosfæren, og viser dermed spekteret fra bakken relativt til innkommende solstråling (Terratec, 2017b). ATCOR er en serie med atmosfæriske og topografiske korreksjonskoder som har blitt kontinuerlig oppdatert gjennom 90- og 2000-tallet. Den siste versjonen, ATCOR4, bruker en stor database som inneholder resultater fra utregninger av strålingsoverføringer basert på MODTRAN5-modellen (Pu, 2017).

En ulempe vi veldig tidlig oppdaget i reflektanssettet, var store hvite områder i bildet (vist med naturlige farger RGB). Disse hvite områdene opptok store deler av grøntareal i bildet, se figur 3.13.

Figur 3.13: Viser en del av en flystripe fra reflektans-settet i RGB-farger. Store deler av vegetasjonen er her forsvunnet. Problemet ser ikke ut til å påvirke områder som ikke er vegetasjon like mye.

Årsaken er at solinnstrålingen er for lav for de blå bølgelengdene. Usikkerheten ble derfor veldig høy når atmosfærekorreksjonen ble utført. Dette er en direkte konsekvens av smale bånd, energien blir for liten i forhold til støyen. Dette kunne blitt kompensert for ved å ha større pikselstørrelse, men det ville samtidig bety lavere oppløsing. Derfor besluttet Terratec å sette alle verdier innenfor båndene som hadde for høy støy til null, det vil si piksler som bare hadde bidrag fra atmosfæren (Terratec, 2017b). Et problem med dette er at man ikke har vært konsekvent når en har korrigert båndviddene i det blå feltet. Båndene som er satt til null varierer fra piksel til piksel. Noen steder er verdiene satt til null fram til 0.5nm, mens andre ganger er det satt til 0 helt fram til 0.7nm, se figur 3.14.



Figur 3.14: Bildet viser spektralkurven til et piksel i et skyggeområde. Alle verdiene fram til 0.7nm er her satt til null.

Dette skapte et dilemma: hvordan hente ut spektralinformasjon uten å inkludere nullverdiene. Dersom disse båndene skulle fjernes måtte store deler av VNIR-båndene vurderes fjernet for å unngå nullverdier. Det ville hatt stor påvirkning på de steinbaserte klassene som ikke har nullverdier i starten av det blå båndene. Spesielt CNN-modellene ville fått problemer dersom alle bånd som inneholdt null hadde blitt fjernet, siden alle bilder må ha samme størrelse i CNN-metoden. Det vil si at dersom ett bånd blir fjernet fra et piksel, må dette gjøres for alle piksler. Det ble derfor besluttet å beholde disse nullverdiene i datasettet.

3.6 Maskinvare, programvare (software og hardware)

En presentasjon av viktige programmer, filformater og maskinvare som har blitt brukt i denne oppgaven.

3.6.1 Programmer og tilleggsmoduler

ArcGIS
ArcGIS er en programsamling som er utviklet av Esri. Innenfor denne programsamlingen er det i denne oppgaven blitt brukt ArcMap og ArcGis PRO. Disse programmene er geografiske informasjonssystemer som er laget for å jobbe med kart og geografiske data. ArcMap og ArcGIS har i den oppgaven blitt brukt om hverandre. De har blitt brukt til å sette reflektanssettet sammen til mosaikk, klippe ut datasettene, lage polygoner for hver av klassene under fasitarbeidet, produsere raster som kunne brukes som fasit for CNN-metoden, og produsere CSV-filer med alle pikslenes hyperspektrale bånd og fasitverdiene koblet mot hverandre for bruk av LR og SVM.

ENVI
ENVI er et program som blir brukt innenfor fjernmåling og bildeanalyse. Programmet er spesielt bra til å trekke ut fornuftig informasjon fra flerdimensjonale data, siden det er mulig å undersøke verdiene innenfor hvert spektralbånd. I vår oppgave har dette programmet blitt brukt til visualisering og preprosessering av dataene.

Python
Python er et objektorientert programmeringsspråk som er veldig brukervennlig og lett å lære. Den støtter utallige pakker, moduler og biblioteker. Dette gir brukeren stor valgfrihet. Python er my brukt innenfor feltene vitenskapelige applikasjoner, datavitenskap og maskinlæring. Det er Python versjon 3.6 i denne oppgaven har blitt brukt til å bearbeide, visualisere og klassifisere dataene. I tillegg ble hele strukturen til maskinlæringsalgoritmene satt opp i dette språket. Under kommer noen utvalgte Python-moduler som ble mye brukt i oppgaven.

Pandas
Pandas er et gratis bibliotek for behandling av tabelldata for Python. Det er svært enkelt å bruke og veldig effektivt til store dataanalyser. I tillegg er det svært enkelt å eksportere og importere data fra ulike filformater med pandas. Pandas format dataframe ble brukt til å behandle CSV-filer i denne oppgaven.

Scikit-learn
Scikit-learn er en maskinlæringsmodul til Python som er åpen og gratis for alle. Dette er et enkelt og effektivt verktøy med mange bruksområder innen preprosessering, analyse, klassifisering og regresjon. LR og SVM er implementert med Scikit-learn.

Keras
Keras er et gratis og veldig brukervennlig bibliotek for nevrale nettverk, skrevet i Python. Keras

er bygget opp på en veldig enkel måte, for å gi brukeren muligheten til å eksperimentere med nevrale nettverk (Keras Documentation, u.å). Det er dette programmet som er brukt for å bygge og trene CNN-modellen i denne oppgaven.

3.6.2 Filformater

CSV
CSV er et tekstformat, og er mye brukt som datautvekslingsformat til å frakte eller lagre store mengder med data i tabeller mellom enheter. Som navnet tilsier brukes komma til å separere ulike verdier fra hverandre, det er også mulig å skille verdier med semikolon eller mellomrom.

Py
Py er et filformat som er laget for å kjøre kode i programmeringsspråket Python. Filen kan redigeres ved bruk av tekstprogram, men dersom koden skal kjøres må dette gjøres i et program som kan tolke Python språket. I denne oppgaven er det brukt Jupyter notebook og Spyder til dette.

Shape
Shape er et vektorformat som representerer data som punkt, linje eller polygon i geografiske informasjonssystemer. Formatet lagrer ikke-topologisk geometri og attributtinformasjon for romlige egenskaper i et datasett. Dette formatet er utviklet av ESRI (ESRI, 1998). I denne oppgaven er shape-filer brukt til å lage og eksportere testområder og fasitpolygoner, og til behandling av koordinater for hyperspektrale enkeltpiksler ved overføring av data til CSV-fil

TIFF
TIFF er et mye brukt rasterformat. Dette filformatet er veldig fleksibelt og kan inneholde flere ulike bilder samtidig. Filen har også utallige komprimeringsalternativer, og den kan rekonstrueres uten å miste bildedata. Derfor kan filen bestå av flere bilder med ulike størrelser og egenskaper. Tiff-filer består av to deler: en IFH (image file header) og IFD (image file directories), hvert bilde som er lagret i filen har en egen IFD, disse inneholder tagger som gir plasseringen til hvor ulik informasjon ligger i filen (Burger and Burge, 2016).

DAT
DAT er et ENVI-bildeformat for multi- og hyperspektale bilder. DAT har alltid en tilhørende header fil (hdr). Bildeinformasjonen blir lagret binært som en strøm av bytes som enten BSQ, BIP eller BIL. Disse bestemmer hvordan bildene skal bli lest.

BIP - Leser ett piksel av gangen. Starter med å lese inn første piksel for alle båndene. Deretter starter den på piksel to for bånd en, deretter bånd to osv. til alle piksler for alle båndene er lest inn.

BIL - Leser en linje av gangen. Starter med første linje for bånd en, tar deretter bånd to osv. til den først linjen er lest inn for alle bånd. Starter deretter på linje to.

BSQ- Leser ett bånd om gangen. Data blir lest inn ved å starte med det første båndet, der blir en og en linje med data lest inn. Når all data fra ett bånd er lest inn blir det samme gjort for neste bånd.

3.6.3 Hardware/kapasitet/maskinpark

For å kunne prosessere dataene og kjøre maskinlæringsalgoritmene vi jobbet med, måtte vi ha vi ha en kraftig maskinpark tilgjengelig, se tabell 3.2 for oversikt over kapasitet på disse.

Tabell 3.2: Oversikt over hardwarekapasitet på maskinene vi jobbet med.

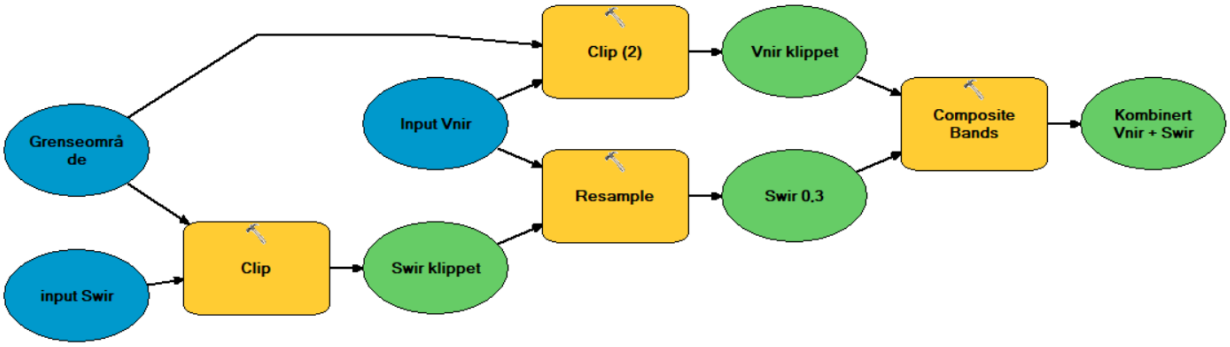
Oversikt over kapasitet på datamaskinene vi jobbet på			
Maskinnavn	RAM	CPU	Prosessornavn
Egne	8 GB	2.40 GHz	Intel Core i3-7100
1 (Å)	32 GB	3.50 GHz	Intel Xeon CPU E5-1620 v4
2 (A)	32 GB	3.60 GHz	Intel Core i7-7700
3	32 GB	3.40 GHz	Intel Core i7-6700
4	48 GB	3.40 GHz	Intel Core i7-6700

Våre egne bærbare pc-er kom raskt til kort, men vi fikk først tilgang til hver vår stasjonære maskin hos Plan- og bygningsetaten (1 og 2 i tabell). Da vi etter hvert begynte å preparere data og teste ut algoritmene, fikk vi et kapasitetsproblem. De hyperspektrale dataene fylte via Python opp RAM-et på maskinene maksimalt. I tillegg kjørte prosessorene konstant på tilnærmet full kapasitet. Vi fikk tilgang på maskin 3 i tillegg, men kapasiteten var fortsatt sprengt og klassifiseringsjobbene sto i kø Da vi endelig også fikk tilgang på maskin 4 med 48 GB RAM kunne vi kjøre all koden vi trengte, selv om vi fortsatt måtte planlegge godt for å rekke gjennom alt som skulle prosesseres. Det meste av tilgjengelig maskinpark har stått og jobbet kontinuerlig i opp mot 2 måneder på ganske høy kapasitet for å komme gjennom alt.

3.7 Preprosessering av hyperspektrale data

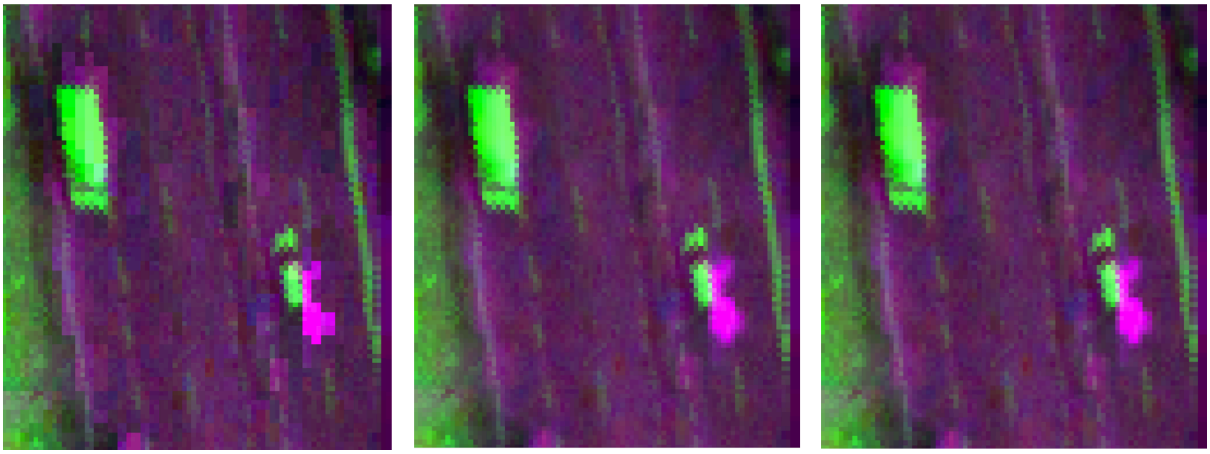
3.7.1 Sammensetting av bilder

I arbeidet med de hyperspektrale bildene var det ønskelig å benytte både VNIR og SWIR. Et overblikk over prosessen med å klippe ut og sette bildene sammen kan ses i figur 3.15. De to datasettene hadde ulik romlig oppløsning, så ble det derfor gjennomført en resampling, slik at datasettene kunne kombineres. Siden VNIR hadde høyest romlig oppløsning ble det besluttet å endre SWIR ned til 0,3m. Dette for å minimere informasjonstapet.



Figur 3.15: Illustrerer prosessen med å klippe ut og sette sammen et testområde. Grenseområde er rektangelutsnitet det klippes etter for begge datasettene VNIR og SWIR. Denne prosessen ble gjort i ArcMap og ENVI. Det ble klippet i ArcMap, mens SWIR ble resamplet og kombinert med VNIR i ENVI. Siste del ble gjort i EVNI, siden det er lettere å velge ut de aktuelle båndene i dette programmet.

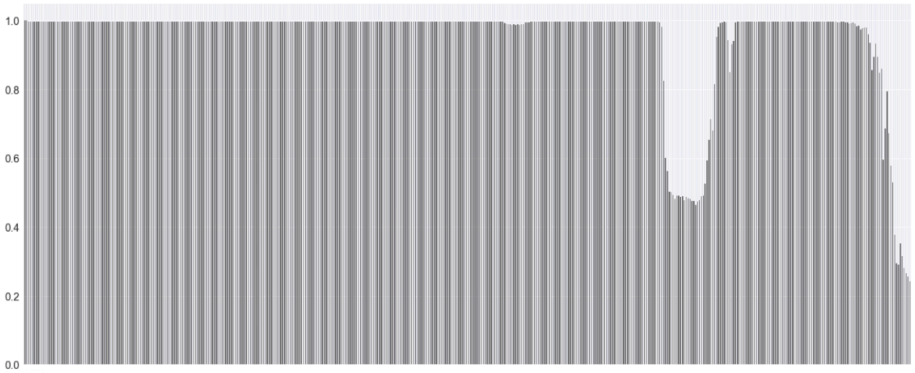
Det ble testet flere resamplingsmetoder, og nærmeste nabo-metoden ble tilslutt valgt siden denne bevarer pikselverdiene, ved å velge nærmeste verdi, mens metodene bilinear og cubic tar med flere av nabopikslene og gjennomfører en midling, se figur 3.16.



Figur 3.16: I figur til venstre er resamplingsmetode nærmeste nabo brukt, denne gir verdien som kommer fra det nærmeste pikselet. Figuren i midten har benytte bilinear, som interpolerer verdiene fra de 4 nærmeste pikslene. Figuren til høyre har brukt resamplingsmetoden cubic, her blir verdiene fra de 16 nærmeste pikslene interpolert.

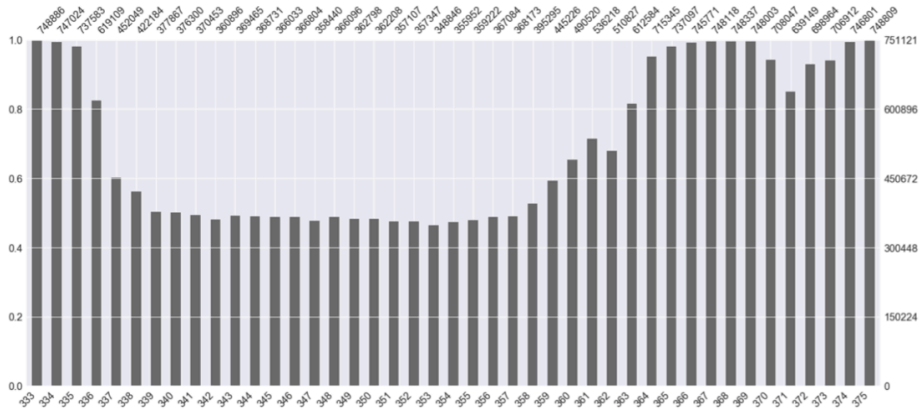
3.7.2 Valg av bånd

Etter at bildene var klippet og satt sammen, ble båndene undersøkt for støy og no-data, slik at en kan vurdere om noen av båndene burde kuttes. Se figur 3.17.

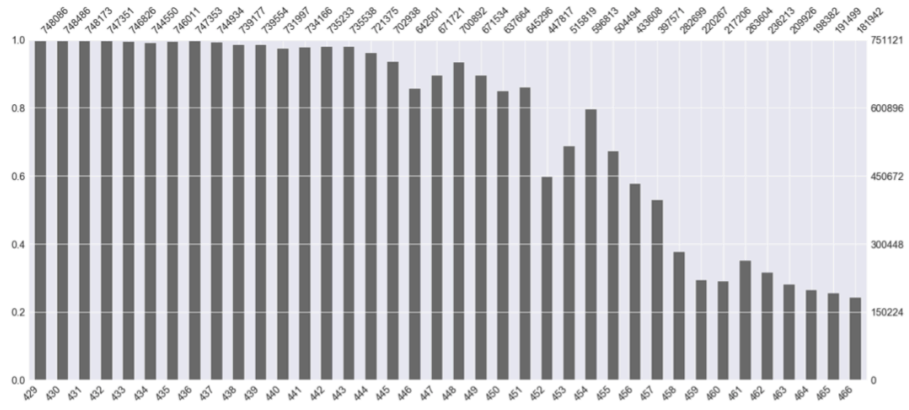


Figur 3.17: Illustrasjon av hvor mye data og no-data som ligger i dataene. Y-aksen representerer antall prosent uten no-data, x-aksen forteller hvilket bånd det gjelder, hvor vertikale stripe er ett bånd. Dersom et bånd går helt opp betyr det at det ikke er no-data i dette.Man ser to store områder med no-data i dette søylediagrammet, begge ligger i SWIR-båndene. Det er viktig å presisere at nullverdiene fra VNIR ikke er tatt med i betegnelsen no-data.

Det ble oppdaget at det var store mengder med no-data i området 1800nm-2000nm og 2300nm og ut, se i figur 3.18 og 3.19.



Figur 3.18:Her er det zoomet inn på et av områdene som så ut til å ha mye no-data. Viser at store deler av området mellom bånd 334-374 (1800nm-2000nm) har mer enn 40% med no-data i sine bånd.

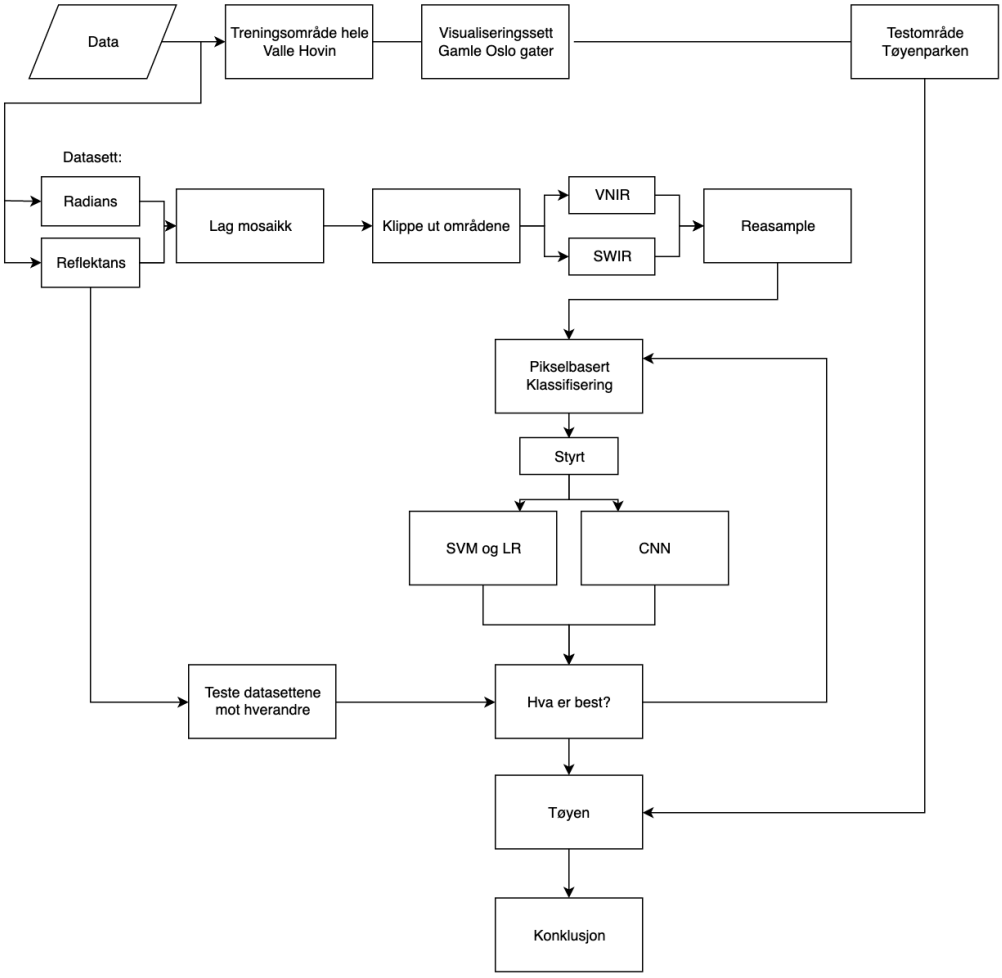


Figur 3.19:Her er det zoomet inn på det andre området som hadde mye no-data. Det er tydelig at det er veldig lite nyttig informasjon i båndene i området 438-466 (2300nm-2500nm).

Det ble også vurdert hensiktsmessig å droppe bånd i 1400nm til 1900nm siden de ligger i et område med mye atmosfærisk vannabsorpsjon (Harris Geospatial Solutions, 2018), men siden hovedfokuset til bildene som skulle brukes i CNN-modellen var å bevare hele spektralverdikurven, ble ingen av båndene fjernet selv om de inneholdt mye støy. Det ble heller ikke fjernet støy-bånd bånd for LR og SVM. Det eneste som ble fjernet var de første åtte båndene fra SWIR. Disse ble fjernet siden de overlappet med de siste båndene til VNIR. Siden VNIR hadde høyere oppløsning enn SWIR, ble det valgt å beholde VNIR-båndene.

3.8 Overblikk klassifisering

Hovedtrekkene til klassifiseringsprosessene kan ses i figur 3.20. Første steg i klassifiseringene var å sammenligne de ulike datasettene radians og reflektans. Her ble det brukt flere ulike klasseinndelinger innenfor både vegetasjon og steinbaserte klasser. Det datasettet som presterte best av radians og reflektans ble brukt videre i oppgaven. Det ble deretter undersøkt hvor lett det var å skille ulike materialer fra hverandre ved å eksperimentere med ulike antall klasser innenfor hver av hovedgruppene steinbasert og vegetasjon. For hver klassifisering ble det vurdert om klassefordeling, fasit eller parameterne i modellen skulle endres. Dette var en veldig eksplorativ del av oppgaven, der det ble oppdaget mye om klasseinndeling og variasjon innenfor klasser. Til slutt ble klassifiseringen gjort på alle klassene innenfor steinbasert og vegetasjon samtidig. Som en endelig test ble de beste modellene fra klassifiseringene kjørt på testområdet Tøyen. Tøyen ble altså kun brukt som en endelig validering for å gi svar på hvor bra de forskjellige ferdige modellene til algoritmene presterte på et ukjent område.



Figur 3.20: Flytdiagram som beskriver hovedtrekkene i prosessen fram til og med endelig klassifisering.

3.8.1 Om valg av klassifiseringsalgoritmer

Det er blitt benyttet tre ulike maskinlæringalgoritmer i denne oppgaven. Det ble valgt å følge «No free lunch-teoremet» som oppsummert sier at ingen algoritme er best for alle typer situasjoner. Skal man jobbe aktivt med maskinlæring er det derfor viktig med kjennskap til mange forskjellige metoder, og hvordan disse forholder seg til forskjellige typer data. Valg av maskinlæringsmetode for en spesiell type oppgave kan være krevende, ikke bare fordi det finnes et vidt spekter av algoritmer tilgjengelig, men også fordi den rikholdige faglitteraturen snakker mot hverandre om hva som virker best til hva (Maxwell et al., 2018).

Vi ville ha et vidt spenn i kompleksiteten til algoritmene vi valgte. Derfor falt valget på CNN som den mest avanserte, SVM som en middels avansert og LR som er relativt lite komplisert. CNN ble valgt siden nevrale nettverk har vært veldig i vinden for tiden. Dette er en "svart boks"-algoritme, det vil si at brukeren bare ser input og output, ikke hva som skjer med dataene underveis. Den ble og valgt siden den er bildegjenkjenningsbasert, og dataene er hyperspektrale bilder. SVM og LR ble valgt siden disse algoritmene er svært populære i faglitteraturen (Maxwell et al., 2018).

3.9 CNN

CNN har i denne oppgaven blitt brukt litt annerledes enn det som er vanlig. Alle modellene som har blitt brukt har kun blitt trent en gang. Det er vanlig å trene CNN-er flere ganger etterhvert som en får mer fasitdata, slik at modellen blir bedre og bedre. Dette har ikke blitt gjort i denne oppgaven, siden det bare ble brukt ett område til trening (Valle Hovin). Siden formålet har vært å finne ut hvor bra CNN fungerer til arealklassifisering innenfor klassene i gruppene vegetasjon, steinbasert og disse gruppene kombinert, har prosessen handlet om å prøve og feile. Det kan derfor hende at resultatene i denne oppgaven ville blitt annerledes dersom det hadde blitt brukt flere treningsområder for å trene opp modellen.

Bildene som har blitt brukt til å trene de nevrale nettverkene er i størrelsen 1 x 466, og 1 x 186. Dette siden det har blitt valgt å fokusere på bilder med 186 (VNIR) og 466 (VNIR + SWIR) ulike bånd. Båndverdiene ble lagt nedover som egne piksler. Denne formen for oppsett er inspirert av artikkelen «Tree species classification in Norway from airborne hyperspectral and airborne laser scanning data», som gjorde det samme med 160 spektralbånd i synlig og nærinfrarødt spekter (Trier et al., 2018). Denne bildestørrelsen er veldig spesiell, siden det representerer alle spektralverdiene til kun ett piksel. I CNN er det vanlig å bruke bilder med flere piksler i hver retning, til forskjell fra stripen med piksler som blir brukt i denne oppgaven. I denne delen av metoden vil det bli nevnt tre ulike sett. Disse gjelder bare for CNN, og en beskrivelse står under.

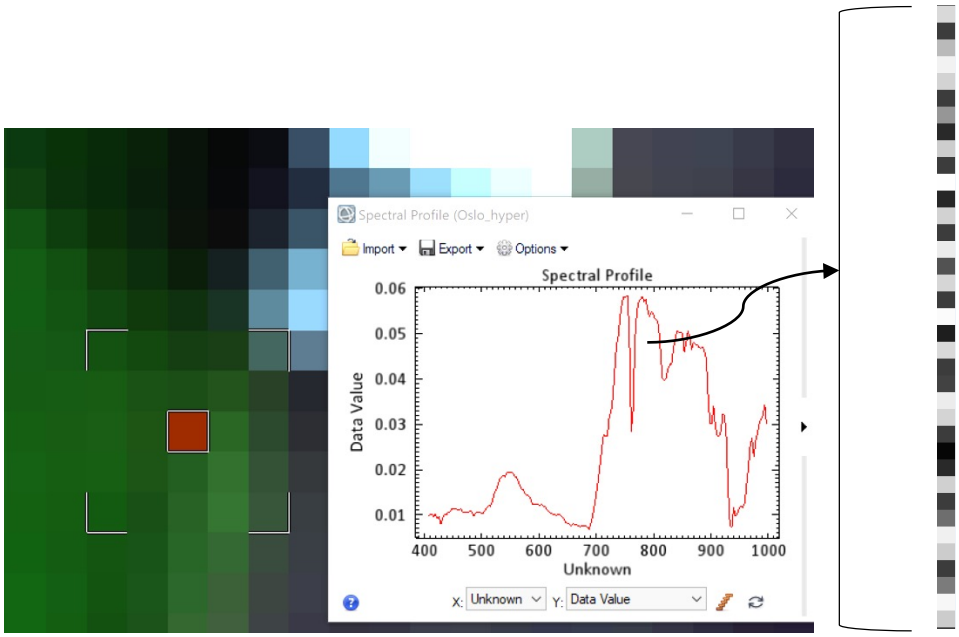
Treningssettet
Dette settet blir brukt til å trene opp modellen. Det er derfor viktig at denne klassen inneholder mesteparten av dataene, slik at modellen får et godt treningsgrunnlag. Det er bare brukt data fra Valle Hovin og Ekeberg i dette datasettet.

Valideringssettet
Dette settet skal gi en indikator på hvordan treningen går. Om scorene fra treningen avviker mye fra resultatet er dette trolig en indikasjon på at modellen er overtilpasset eller undertilpasset. Dataene i dette settet er hentet fra de samme områdene som treningssettet.

Testsettet
Dette settet blir brukt når modellen er ferdig trent, og er en endelig test for å se hvor godt modellen predikerer. Det er viktig at settet ikke har vært i kontakt med modellen under treningen. Testsettet bruker enten data fra området på Valle Hovin som er samme område treningen foregår, eller Tøyen-området som er et uavhengig område adskilt fra treningsområdet.

3.9.1 Bildegenerering

I utgangspunktet var ikke de hyperspektrale bildene på formatet 1 x 466 og 1 x 186. Det var derfor nødvendig å dele opp de aktuelle områdene til mange småbilder ved bruk av Python. Bildene ble først gjort om til et array, slik at båndverdiene for hver piksel kunne hentes ut. Hvert piksels båndverdier ble deretter lagret som et eget bilde, se figur 3.21.



Figur 3.21: Over kan en se et piksel med dens spektralsignatur, til høyre kan en se et utklipp av et av de genererte bildene. Spektralverdiene til hver piksel er blitt gjort om til et bilde. Hvert bånd har blitt et individuelt piksel med en gråverdi i det nye bildet.

For å ikke miste posisjonen til pikselet, ble navnet på bildet gitt posisjon fra det aktuelle testområdet (eks. pix22_700.tif, som vil si at det er piksel 22 nedover og 700 til høyre). Pikselets plassering er nødvendig når bildet skal settes sammen igjen etter klassifiseringen. Bildegenereringen må repeteres om bildenes størrelse skal endres, det vil si om en ønsker å redusere eller øke antall bånd som skal brukes i klassifiseringen. Siden det ble valgt å bruke datasettene radians og reflektans, måtte det genereres fire forskjellige sett med bilder for ett testområde.

3.9.2 Filoppdeling

Hvordan håndtere store datamengder var en utfordring som stadig dukket opp i arbeidet med disse bildene. Pikselstørrelsen i datasettet var på 0,3m, det ble derfor generert veldig mange bilder, siden hvert piksel ble ett bilde. Et område på 500m x 500m tilsvarer omtrent 2 780 000 bilder. Dersom alle disse skal lagres i en katalog, ville dette tatt nærmere en uke å gjennomføre. Ved nærmere undersøkelse viste det seg at lagringshastigheten ble betraktelig redusert når katalogens innhold passerte 900 000 bilder. Dersom bildene fordeles jevnt ut over flere kataloger, tok denne prosessen ikke lengre tid enn en arbeidsdag. På bakgrunn av dette ble antall bilder i hver katalog bestemt til å ligge mellom 600 000 - 900 000. Dette problemet kom ikke av maks antall elementer i en mappe, siden det viste seg å være langt over 1 million filer, men det ble mistenkt at årsaken kunne komme av at det var mange filer med nesten identiske navn (Microsoft, 2009).

3.9.3 CNN-modellene

Det ble brukt tre ulike nettverk med forskjellig oppbygning for CNN, disse blir kalt modell 1, 2 og 3. Den første modellen tok utgangspunkt i modellen som ble benyttet for pikselbasert klassifisering i artikkelen av Trier et. al. (Trier et al., 2018). I tillegg til denne modellen ble det også brukt to egenproduserte modeller, siden det er stor forskjell på å sammenlikne tresorter og arealtyper fra hverandre. Modellen i artikkelen var også kun beregnet på bilder med 160 bånd fra VNIR. Det vil si at ingen bånd fra SWIR var inkludert, det ble derfor besluttet å lage to egne nettverk med større kapasitet, som skulle testes på 6 klasser. Disse modellene kan ses i figur 3.22, 3.23 og 3.24.

```
def model_1():
    model = Sequential()
    model.add(Conv2D(20, 17, 1, input_shape=(466, 1, 1), activation="relu"))
    model.add(MaxPool2D(pool_size=(4,1), strides=1))
    model.add(Flatten())
    model.add(Dense(output_dim=100, activation= "relu"))
    model.add(Dense(activation="softmax", units=6))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()
    return model
```

Figur 3.22: Figuren viser oppbygningen til modell.

Modellen i figur 3.22 ble kjørt for seks klasser og 466 bånd. Den tar inn et bilde med størrelsen 466 x 1 og lager 20 konvolusjonlag med konvolusjonkjerner i størrelse 17 x 1. Neste linje plukker ut høyeste verdi innenfor et område på 4 x 1, med en forskyvning på en piksel, siden strides er lik 1. Slik dannes et nytt bilde for hvert lag. Deretter blir pikslene i bildene gjort om til nevroner i et nevralt nettverk. Det neste laget har 100 nevroner og aktiveringsfunksjonen ReLU, det betyr at alle aktiveringsverdier som er mindre en null bli satt til null. Output-laget har 6 nevroner, der aktiveringsfunksjonen softmax blir brukt til å gi en prosentverdi til hver av disse seks nodene. Den noden med den høyeste verdien blir den predikerte klassen. Modell 2 og 3 har samme grunnstrukturen, med noen enderinger.

```
def model_2():
    model = Sequential()
    model.add(Conv2D(30, 20, 1, input_shape=(466, 1, 1), activation="relu"))
    model.add(MaxPool2D(pool_size=(7,1), strides=1))
    model.add(Conv2D(50, 42, 1, activation="relu"))
    model.add(MaxPool2D(pool_size=(8,1), strides=2))
    model.add(Flatten())
    model.add(Dense(output_dim=100, activation= "relu"))
    model.add(Dense(activation="softmax", units=6))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()
    return model
```

Figur 3.23: Bildet viser modell 2 som ble kjørt for seks klasser og 466 bånd. Denne modellen ligner litt på modell 1, men parameterne til Conv2D og MaxPool2D er endret, i tillegg til at det er lagt inn en ekstra Conv2D og MaxPool2D.

```
def model_3():
    model = Sequential()
    model.add(Conv2D(70, 42, 1, input_shape=(466, 1, 1), activation="relu"))
    model.add(MaxPool2D(pool_size=(7,1), strides=1))
    model.add(Conv2D(50, 20, 1, activation="relu"))
    model.add(MaxPool2D(pool_size=(8,1), strides=2))
    model.add(Conv2D(30, 15, 1, activation="relu"))
    model.add(MaxPool2D(pool_size=(20,1), strides=2))
    model.add(Flatten())
    model.add(Dense(output_dim=200, activation= "relu"))
    model.add(Dense(output_dim=100, activation= "relu"))
    model.add(Dense(activation="softmax", units=6))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()
    return model
```

Figur 3.24: Bilde viser oppsettet til modell 3, som ble kjørt for seks klasser og 466 bånd. Akkurat som modell 2 tar denne utgangspunktet i modell 1, men denne har tre Conv2D og MaxPool2D. Og det har blitt lagt til et ekstra skjult lag med 200 nevroner.

3.9.4 Gjøre data klar til trening

Alle nettverk som har blitt trent i denne oppgaven har blitt trent med balanserte klasser. Det er lurt å trene et CNN balansert, for å unngå problemer med at modellen lærer seg hvilken klasse det er mest av i treningsdataene (Kubat and Matwin, 1997), og predikerer etter flertallet. Det eneste unntaket med jevn fordeling gjelder klassifiseringene der det ble brukt grus fra Ekeberg-området. Her ble det brukt mer gruspiksler i trenings- og valideringssettet, i forhold til de andre klassene, men til gjengjeld var det færre gruspiksler i testsettet. Dette ble gjort siden en ikke skulle teste nettverket på grusområdet på Ekeberg, bare trene. Fasiten ble brukt til å plukke ut dataene, det ble plukket ut like mange piksler fra hver klasse.

Etter utvelgelsen av pikslene ble settet delt i to, 20% ble plukket ut tilfeldig og satt til testsett. De resterende 80% ble satt til treningssett, rekkefølgen på dataene i settene ble randomisert. Deretter ble treningssettet lagt inn i en bildegenerator der de ble reskalert, og 30% av bildene ble satt til valideringssett. Bildegeneratoren er vanligvis brukt til å generere nye versjoner av allerede eksisterende bilder ved å rotere, flippe, zoome, horisontere, og dermed øke antall testbilder betraktelig. I denne situasjonen har dette ikke blitt gjort, siden bildene bestod av en lang rekke med piksler. Om bildestørrelsen hadde vært annerledes, det vil si hadde vi hatt bilder på eks. 32x32 ville dette vært mer interessant å prøve ut. Men siden bildene skal representere spektralverdiene til det enkelte piksel er ikke dette aktuelt.

Bildene ble plassert i batcher med data. Prosessen med å lage batcher er der for å hindre maskinen i å kjøre all dataen i minne samtidig, siden det ville krevd mer minne enn det en vanlig datamaskin har. Under en normal treningsøkt kan det bli brukt flere hundretusen ulike bilder. Derfor er løsningen å kjøre bare litt og litt av dataene om gangen, en dataframe-iterator blir brukt til å hente inn en og en batch. Det er filstien til bildene som blir brukt til å holde kontroll på bildene.

3.9.5 Trening

Før treningen startet ble arkitekturen til nettverket bestemt, dette ble gjort ved å lage en ny modell. Det hadde også fungert å bruke en allerede trent modell og trent den mer, men dette ble ikke gjort i denne oppgaven. Det har derfor blitt fokusert på å produsere nye modeller for hver klassifisering, og ta vare på de som presterer bra. Under kan en se hovedklassifiseringene som har blitt utført:

- Skille grønt og ikke-grønt
- Klassifisering av steinbaserte materialer
    - Skille asfalt, betong, grus og sand
    - Skille asfalt, betong og grus
- Klassifisering av grøntarealer
    - Skille gress, tre, hagebusk og kratt/eng (naturlig busk)
    - Skille gress, busk og tre
- 6 klasser: asfalt, betong, busk, gress, grus og tre

Skille grønt og ikke-grønt

Det ble gjennomført en klassifisering for å undersøke hvor lett det var å skille vegetasjon fra ikke-grønt materiell. Vegetasjonsklassen inneholdt gress, tre og busk, mens ikke-grønt bestod av de steinbaserte klassene asfalt, betong, grus og sand. Her er det viktig å nevne at klassene innad i grønt og ikke-grønt hadde et ujevnt antall piksler innad. Da datasettet ble satt sammen, ble det plukket ut 270 000 piksler for hver klasse (grønt og ikke-grønt). Det var totalt 540 000 piksler som ble brukt til trening, validering og testing tilsammen.

Ved denne klassifiseringen ble bare strukturen til modell 1 brukt til trening, se figur 3.22. Denne strukturen ble brukt til å trene fire modeller for datasettene reflektans VNIR, reflektans VNIR + SWIR og radians VNIR og radians VNIR + SWIR. Det ble ikke gjort noen justeringer på fasit eller modeller under denne klassifiseringen, siden hovedformålet var å se hvor lett det var å skille mellom grønt fra ikke-grønt materiell ved hjelp av hyperspektrale bilder. Samtidig som en ønsket å se om det ulike datasettene skilte seg fra hverandre i resultatet. Det tok flere dager å trene disse modellene.

Klassifisering av steinbaserte materialer

Det ble utført mange ulike klassifiseringer for å undersøke mulighetene til å skille steinbasert materiell fra hverandre. Den første modellen som ble brukt hadde fire klasser. Her ble samme fasit som under grønn og ikke-grønn brukt, men bare for radianssettet med VNIR og SWIR. Siden det bare var 7300 piksler med sand og 8600 piksler med grus, ble det besluttet å fjerne sand-klassen. Det ble utført forskjellige tiltak for å øke antallet piksler i grusklassen, disse vil bli nevnt i avsnitt 4.2.4.

Det ble også eksperimentert med de ulike modellene og klassefordelinger innad i klassen. Det vil si hvor stor variasjon som er tillat innenfor hver klasse. Ikke alle datasettene ble brukt for hvert forsøk innenfor klassifisering av steinbaserte materialer. Klassifiseringen med tre klasser med grus fra Ekeberg var eneste klassifiseringen som ble kjørt for alle datasettene. Det er derfor det er denne som blir brukt til sammenligning av radians- og reflektanssettene.

Klassifisering av grøntarealer
Under undersøkelser av muligheter for å skille grøntarealer fra hverandre ble det kjørt fire ulike typer klassifiseringer en klassifisering med fire klaser og tre klassifiseringer med ulike modeller ved tre klasser. Det ble gjort lite endringer innad i klassen, siden gress og tre var tydelige klasser. De største endringene ble gjort i buskklassen, på grunn av forvirring rundt klassen. Klassifiseringene som ble brukt til sammenligning mellom datasettene, var fire og tre klasser ved modell 1.

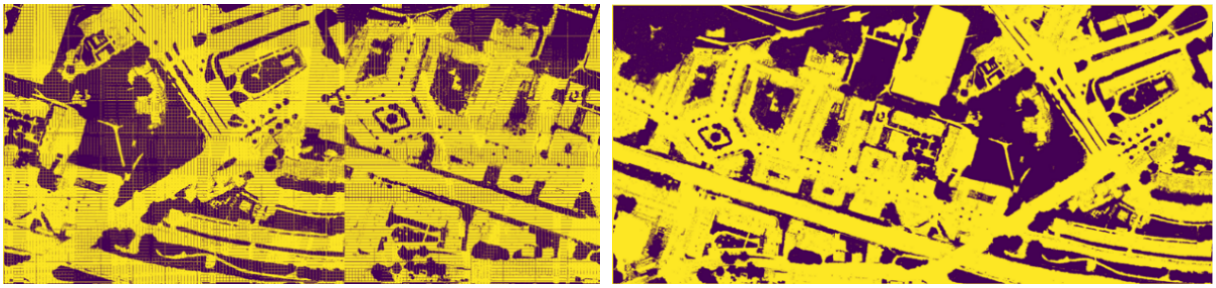
6 klasser
Da alle klassen for gruppene steinbasert
og vegetasjon var ferdig undersøkt, ble de klassefordelingene som hadde prestert best under klassifiseringen av grøntarealer og steinbasert brukt videre. Disse klassene ble asfalt, betong, busk, gress, busk og tre. Disse ble så kjørt for modell 1, 2 og 3. Tilslutt ble modellene testet på det endelige valideringsområdet Tøyen.

3.9.6 Manningsraster

Det ble produsert et raster for Mannings tall ut av et av resultatbildene, dette ble gjort for å illustrere de ulike Mannings tall-klassene som ble presentert i avsnitt 3.4.1. Manningsrasteret ble produsert ved å bruke et resultatbilde fra en klassifisering. Området som ble brukt var testområdet Gamle Oslo, alle pikslene i bildet ble klassifisert innenfor seks klasser. For å rekonstruere bildet etter klassifisering ble filnavnene til bildene koblet sammen med resultatene. Som nevnt i avsnitt 3.9.1 var filnavnet til bildene posisjonen i testområdet, før pikslene ble lagret som egne bilder. Under bilderekonstruksjonen blir hver klasse gitt hver sin unike farge.
Da bildet var ferdig rekonstruert, ble programmet ENVI brukt til å kombinere vann og tak med resultatbildet. Vann og tak var ferdig klippet fra FKB-dataene i samme størrelse som resultatbildet. Deretter ble bildet georefferert i ENVI. Bilde ble konvertert fra TIFF- fil til DAT-fil og deretter tatt inn i ArcGIS der rasteret ble omgjort til polygoner (Shp-fil). Hver klasse ble ett eget polygon. Kolonnen Mannings tall ble lagt inn i attributt-tabellen til datasettet, med de utvalgte Mannings tallene. Til slutt ble vektorfilen konvertert tilbake til rasterformat.

3.9.7 Sorteringsproblematikk

Det er svært viktig å ha kontroll på hvilken rekkefølge bildene ble sendt inn og hvordan de kommer ut. De første gangene det skulle produseres et resultatbilde oppstod det et problem, se figur 3.25. Det var mulig å kjenne igjen deler av bildet, men det var tydelig at det var noe som hadde gått galt i rekonstrueringsprosessen. Ikke bare var deler i bildet plassert feil i forhold til originalbildet, men det var også dannet et slags rutenett i bildet. Det kunne nesten virke som det hadde skjedd en type forskyvning. Det viste seg at bildet var satt sammen feil.



Figur 3.25: Disse bildene er fra et utklipp av Gamle Oslo-området, og er produsert etter å ha blitt klassifisert som enten grønt eller ikke-grønt. Bildet til venstre er det feilsammensatte bildet, mens til høyre er bildet korrekt sammensatt. Denne feilen oppstod da den predikerte klassen skulle kombineres med pikselplasseringen. Det ble antatt at filplasseringen til den predikerte var lik testsettet, dette var ikke tilfelle.

Da bildet ble rekonstruert ble pikselets filnavn brukt til å finne posisjonen i bildet. Det viste seg å være en utfordring å kombinere de predikerte verdiene med testsettets filnavn, siden en ikke visste hvilken piksel som kom først i det predikerte settet. I figur 3.26 kan en se hvordan filene ble sortert (P, 2018).

	id	1	2
0	0_0	5.525310e-22	1.0
1	0_1	5.525310e-22	1.0
10	0_10	5.525310e-22	1.0
100	0_100	5.525310e-22	1.0
1000	0_1000	5.525310e-22	1.0

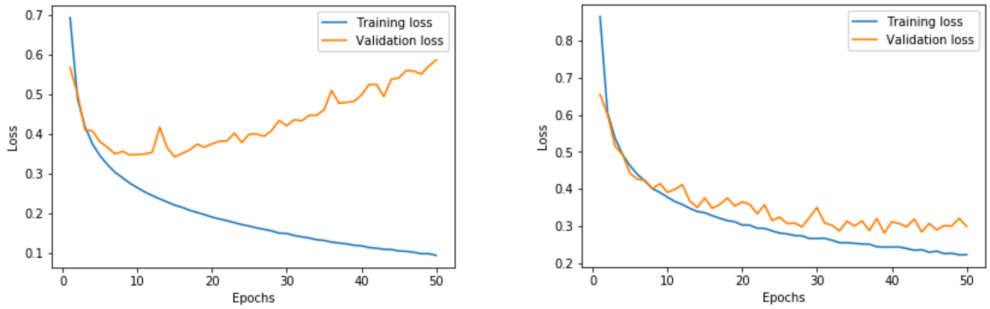
	id	1	2
0	0_0	5.525310e-22	1.0
1	0_1	5.525310e-22	1.0
2	0_2	5.525310e-22	1.0
3	0_3	5.525310e-22	1.0
4	0_4	5.525310e-22	1.0

Figur 3.26:Figuren til venstre viser hvordan bildet som kom ut etter å ha vært gjennom predict_generator, starter med laveste siffer til venstre. Til høyre kan en se hvordan bildene ble lagret i dataframen, sorteringen følger tallets verdi.

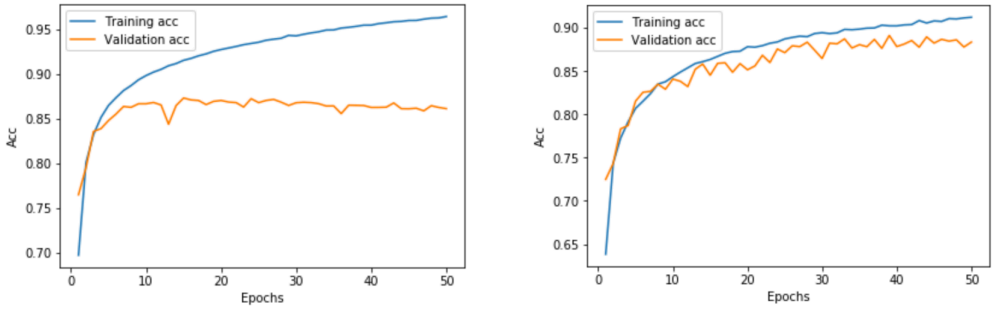
Løsningen ble å hente filstien direkte ut fra dataframe-iteratoren, og bruke denne listen med filnavn til å sette bildet sammen med de predikerte verdiene.

3.9.8 Validering

Under treningen ble valideringskurver benyttet for å se hvordan modellene presterte. Disse plottet accuracy og loss fra treningen i en graf, se figur 3.27 og 3.28. Stor forskjell mellom validering og trening tyder på at det eksisterte en bias eller at modellen er overtilpasset eller undertilpasset. Siden valideringssettet ble plukket ut av treningssettet før trening, sier dette bare noe om de 30% av dataene fra treningssettet. En må være oppmerksom på ubalanse i valideringss- og treningssett, siden accuracy ikke kan stoles på ved ubalanse i datasettet. Til vanlig gir accuracy en pekepinn på hvordan modellen har prestert.



Figur 3.27: Over kan en se to ulike plot over losset, der den blå linjen representerer treningslosset, mens den oransje linjen representere valideringslosset. I plottet til venstre starter modellen å lære seg mye unyttig informasjon rundt epoke 10, det ville nok vært lurt å ha stoppet den der. Men i stedet fortsetter den å lære mye tull som fører til overtilpasning. Mens plottet til høyre viser at begge linjene gradvis synker før det ser ut som valideringen flater ut. Dette viser en langt bedre modell.

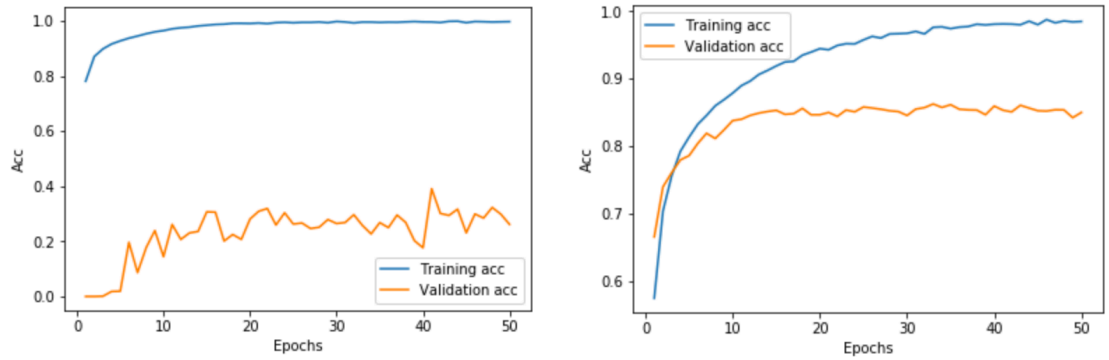


Figur 3.28: Over viser plottet accuracy (acc) til to ulike modeller. Den til venstre har trent for lenge, mens den til høyre stoppet i tide.

Etter at modellene var ferdig trent, ble testsettet som bestod av 20% av de utvalgte dataene brukt til å lage en forvirringsmatrise. Det er veldig vanlig å bruke forvirringsmatriser for å få et overblikk over resultatene. Forvirringsmatrisen sier mye om hvordan klassifiseringen har prestert generelt og innad i klassene. Denne matrisen kan brukes til å beregne parametere som blir brukt til å sammenligne modellen innad og i forhold til hverandre.

3.9.9 Problem med valideringssett

Under treningen av de ulike modellene ble det oppdaget en feil ved validerings- og treningssettet. Det hadde blitt antatt at bildegeneratoren skulle ta et tilfeldig utvalg fra treningssettet da valideringssettet ble laget, men dette var ikke tilfelle. De første dataene i rekken ble plassert i valideringssettet, og siden dataene hadde blitt lagt til klassevis, ble fordelingen mellom trening og validering veldig skjev. Under trening av fire klasser mistet treningssettet en hel klasse. Derfor ble ikke alle klassene trent, og modellen ble ikke i stand til å klassifisere klasse 1. Samtidig inneholdt valideringssettet nesten bare klasse 1. Dette førte til en veldig dårlig score på accuracy, siden det var så stor ubalanse i valideringssettet, se figur 3.29. Den dårlige scoren var årsaken til at dette ble oppdaget. Løsningen ble å randomisere treningssettet før valideringssettet ble laget.



Figur 3.29: I bildet over kan en se to plott som viser trenings-accuruay og validerings-accuruay. Disse plottene hører til samme treningsmodell. Forskjellen er at plottet til høyre har en jevn fordeling innad i klassene, mens det til venstre mangler en klasse i treningssettet.

3.10 LR og SVM

For algoritmene LR og SVM ble det generert CSV-filer fra de hyperspektrale bildene og fasit, disse CSV-filene ble så hentet inn i python for videre behandling og klassifisering med maskinlæring. Scriptet det refereres til i de neste avsnittene ligger i Vedlegg som vedlegg B.

Trenings- og testsettet
All trening og testing ble gjort på Valle Hovin-settene, både radians- og reflektansversjonene med 466 bånd. Tak- og vannflater ble masket ut med FKB-data og de ble begge koblet til den samme fasiten med 7 klasser. Settene ble så splittet i to, der den ene delen ble brukt til trening og den andre til validering. Valideringen skal gi en indikator på hvor bra treningen har fungert. Om treningsscorene avviker mye fra dette resultatet er dette gjerne en indikasjon på at modellen er overtilpasset eller undertilpasset.

Testsettet
Testsettet ble brukt for å teste ut ytelsen på de modellene som gjorde det best på Valle Hovin. Dette settet ble brukt da modellen var ferdig trent og testet, og var en endelig test for å se hvor godt modellen predikerer. Det er viktig at dette settet ikke har vært i kontakt med modellen under treningen. Testdatasettet ble generert fra Tøyensettet, som var et uavhengig område adskilt fra treningsområdet. Dette settet ble koblet med en fasit, som også var uavhengig fra Valle Hovin. Denne fasiten inneholdt 6 klasser.

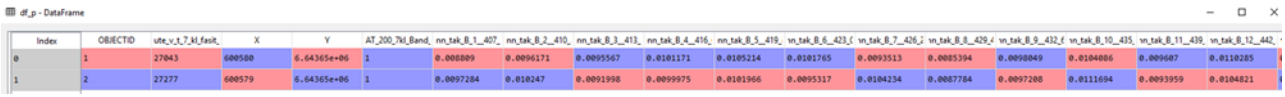
3.10.1 Uthenting av spektralinformasjon fra punkter

Det ble bestemt å bruke enkeltpikslers spektralsignaturer koblet mot fasitverdier til maskinlæringen i denne oppgaven. For å kunne lese inn disse spektralsignaturene og ha kontroll på pikslenes georeferering i Python, måtte dataene preprosesseres

Det ble sett på forskjellige metoder for å hente ut spektralsignaturer fra enkeltpiksler. Den som viste seg å fungere best var en ren GIS-jobb i ArcGis PRO. Først ble det generert ett enkeltpunkt per piksel, som ble georeferert til midtpunktet på hver piksel i Valle Hovin-settet. Deretter ble fasitpolygonene generert om til raster, der hvert rasterområde fikk sin klasse tilegnet som et tall fra 1-7. Det ble så gjort en romlig kobling mellom disse tre datasettene; hyperspektralt pikselraster, punkter og fasitraster. Ut av dette ble det generert en tabell der hver rad inneholdt blant annet en unik ID, X- og Y-koordinater til pikselets midtpunkt, klassetall fra fasit og 466 intensitetsverdier (radians) eller prosentverdier (reflektans), ett fra hvert bånd. Denne tabellen ble til slutt generert ut til en CSV-fil. Det ble først gjort et forsøk med en kobling på alle de 6 millioner pikslene fra Valle Hovin, og denne CSV-filen ble på rundt 30 GB. Denne filen ble uhåndterlig for Python. Løsningen ble derfor å bruke datasettet uten vann- og tak-piksler, og i tillegg kun inkludere piksler med tildelt fasitverdi (750.000 piksler). Disse endelige CSV-filene ble på henholdsvis 3,4GB (radians) og 4,4 GB (reflektans). Dette er en litt tungvint prosess, det er mange steg, og må noe forandres er det mye jobb å fikse opp i det.

3.10.2 Preprossesering av CSV i python

De ferdige CSV-filene ble så lest inn i Python og generert om til panda dataframes for videre behandling og preprossesering før klassifisering. Denne dataframen hadde nå form som en matrise på 466x 750.000 = 353.000.000 elementer, figur 3.30. Etter denne prosessen lå nå all data i maskinens arbeidsminne (RAM), og prosessen gjorde at Python beslagla ytelse ca 3 ganger filstørrelsen fra maskinens RAM. Dette er en viktig begrensning å være klar over. Hver piksel hadde nå fått tilegnet en klasse i form av et tall fra 1-7. Fordelen med denne metoden var at det nå var enkelt å slå sammen klasser igjen, for eksempel naturlig busk og plantet busk til klassen busk. Det samme når det skulle testes med to klasser, grønt mot urbant. Eller bare de tre grønne klassene mot hverandre.



Figur 3.30 Utsnitt av dataframe generert fra CSV-fil

3.10.3 Gjøre data klar til trening

.

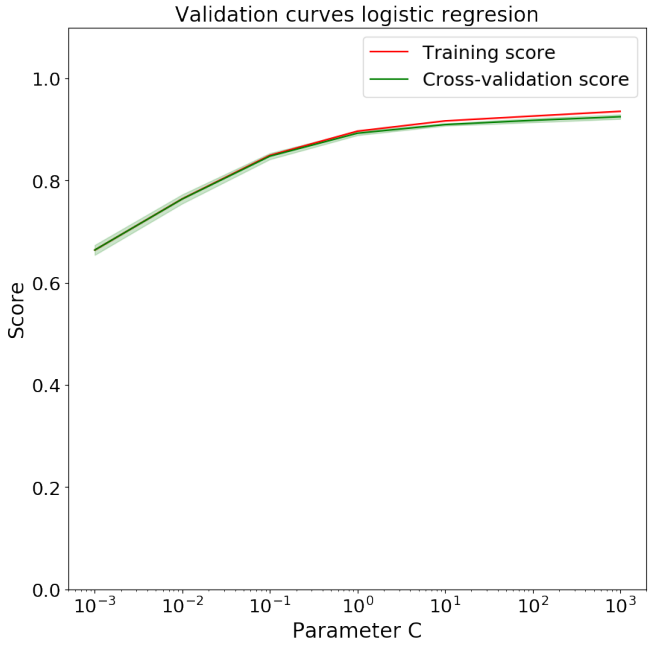
Når data var lest inn riktig, måtte disse behandles Det viktigste var å splitte de opp i mindre deler for trening og testing. Det ble gjort en splitt på 70% av dataene til trening, og 30% til validering. For denne oppgaven var det bare interessant å trene på de feltene som inneholder båndenes spektralverdier. Her måtte det og velges om det skulle trenes på hele spekteret eller bare VNIR. ID, X- og Y-posisjon var ikke interessant treningsmateriell, men måtte spares for å kunne generere en output-fil. Samtidig var det kun fasit-verdiene som var mål for hva klassifiseringsalgoritmene skal finne. Spektralverdiene måtte også skaleres, StandardScaler fra scikitlearn-modulen ble benyttet til dette.

3.10.4 Første forsøk

Da alt var klargjort ble det trent opp et par modeller for å få ut noen resultater. Disse ble så kjørt på test-settet for å få ut statistikk på nøyaktigheten. På dette tidspunktet var fasiten for Valle Hovin allerede ferdig utarbeidet, og det ble brukt en med 7 klasser. Dette er den samme som CNN-modellene het «smal fasit» med 6 klasser. Forskjellen var at klassen busk var atskilt som naturlig busk og plantet busk, for deretter å kunne slås sammen igjen til én klasse.
Det ble gjort noen tester på 7 klasser med både LR og SVM, både på radians og reflektans. Disse ble først kjørt ubalansert, altså med ulikt antall piksler fra hver klasse. Testing viste fort at SVM virket å gi best resultater, men brukte veldig mye lenger tid på å komme til et resultat enn LR, som var rask i forhold. Med lang tid menes her dager, ofte en hel helg for å få ut noe av resultater på et datasett som Valle Hovin
Modellene som ble trent ble lagret i to Python-formater, Pickle og Joblib.

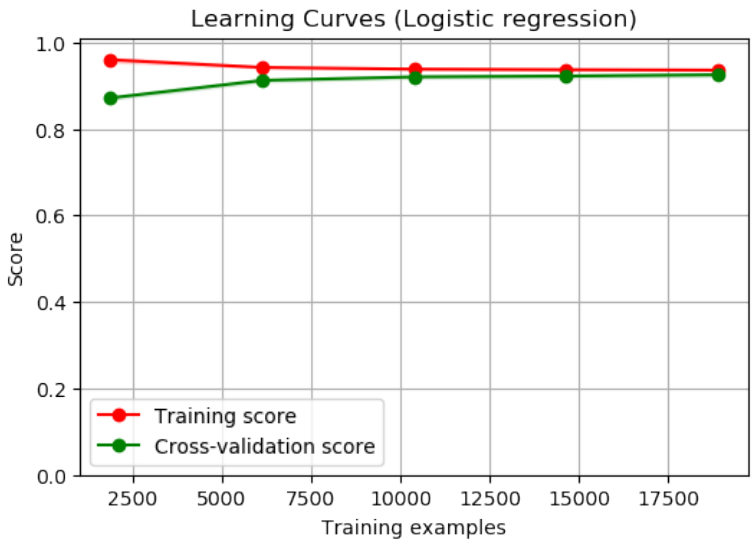
3.10.5 Parameterjustering

Etter å ha gjort de første testene, ble det kjørt kode for parameteroptimalisering for SVM og LR. Scriptet som ble brukt er hentet fra vedlegg D i masteroppgaven til Erik Røstad (Røstad, 2018), med noen forandringer. Det ble også kjørt kryssvalidering for å teste parameterne, men det virket som parameterjusteringen kun ga ubetydelige utslag på klassifiseringene (se figur 3.31, 3.32, 3.33 og 3.34). Det er derfor ikke lagt videre vekt på dette i kapittel om resultater og diskusjon. Funksjonen GridsearhCV fra Scikitlearn-pakken ble brukt til prosessen.
For LR ble det testet forskjellige C-verdier, og resultatet ble at C = 10 ga best utslag på valideringsresultat, se figur 3.31. Denne parameteren ble valgt for all videre klassifisering.



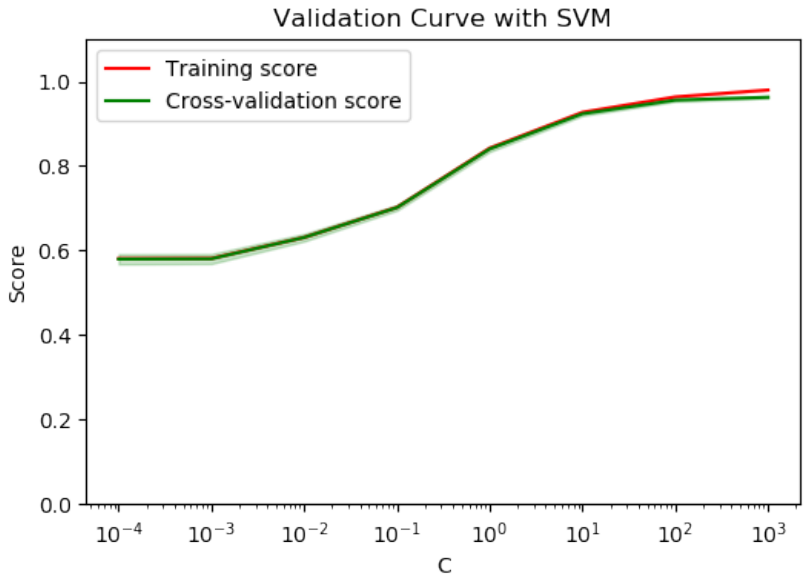
Figur 3.31 Valideringskurve for C-verdier med LR. Kurvene skiller seg ved C=10, så denne verdien ble valgt

Det ble og kjørt en kryssvalidering for å se hvordan modellen oppførte seg, og kurven som er generert tilsier ikke overtilpasning.

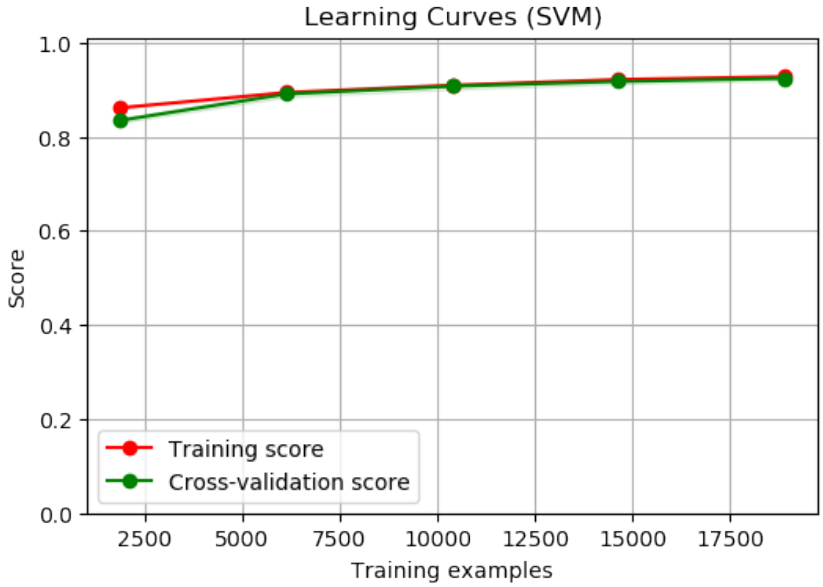


Figur 3.32 Læringskurve for LR generert fra kryssvalidering

Samme prosess ble gjort for SVM. Her ble både C, gamma og forskjellige kjerner testet mot hverandre.
De parameterne koden fant at presterte best ble valgt for videre læring, testing og validering



Figur 3.33 Valideringskurve for C-verdier med LR. Kurvene skiller seg ved C=100, så denne verdien ble valgt



Figur 3.34Læringskurve for SVM generert fra kryssvalidering

Parameterne som ble estimert og valgt var C = 100, gamma = 0.001 og beste kernel = rbf.

3.10.6 Trening

Det ble prioritert å bruke CNN til eksperimentering og produksjon av fasit. Da denne var fasiten ferdig definert ble den brukt på all trening og testing av modeller på Valle Hovin-dataene.

Fasit med 7 klasser hadde følgende innhold:

1. Gress 321166 piksler
2. Tre 222352 piksler
3. Busk naturlig 98447 piksler
4. Busk 17870 piksler
5. Betong/stein 116910
6. Grus 19767 piksler
7. Asfalt 65081 piksler

Denne fasiten brukt som utgangspunkt, og klasser slått sammen, utvidet eller fjernet etter behov. Klassene 3 og 4 ble alltid slått sammen til fellesklassen busk i trening på Valle Hovin.

Etter at parameterne var finjustert ble det foretatt over 20 forskjellige klassifiseringer med LR og SVM samtidig.
Av de 20 blir et lite utvalg av disse presentert i kapittel 4, resultater og diskusjon:

Reflektans mot radians i avsnitt 4.2.2 og tabell 4.12
Steinbasert 3 klasser i avsnitt 4.2.4 og tabell 4.12
Grønt 3 klasser i avsnitt 4.2.5 og tabell 4.12
4 tester Tøyen i avsnitt 4.2.8 og tabell 4.12
6 klasser balansert i avsnitt 4.2.9 og tabell 4.12

3.10.7 Validering og forvirringsmatriser

Det ble for alle klassifiseringer som ble gjort med både LR og SVM plottet forvirringsmatriser og generert tabeller med resultater fra valideringsparameterne accuracy, kappa, precision, recall og f1-score.

3.10.8 Validering mot Tøyen for LR og SVM

Tøyen-fasit ble ikke produsert før all trening og validering på Valle Hovin var ferdig. Området hadde altså ikke vært rørt i det hele tatt under treningsperioden. Denne fasiten inneholdt seks klasser: gress, tre, busk, betong, grus og asfalt.
Etter at fasit var produsert ble det generert et testsett på samme måte som beskrevet i avsnitt 3.10.1. Det ble nå testet for å få ut endelig statistikk på hvor bra modellene trent på Valle Hovin egentlig presterer. Det var ikke tid til å generere testbilde og CSV-fil for reflektanssettet over Tøyen. For både SVM og LR ble det gjort 4 tester, alle på radiansdataene med alle 466 bånd inkludert:

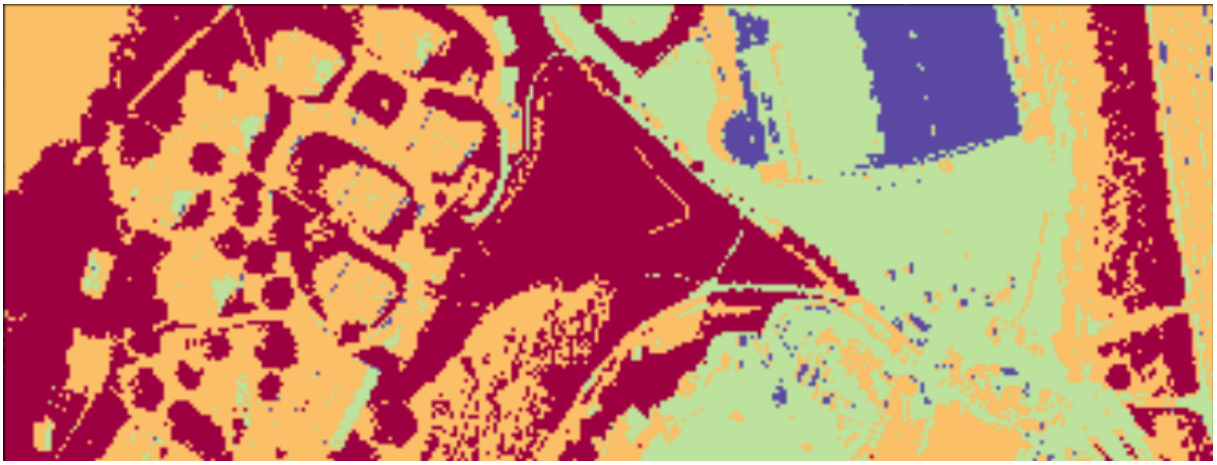
- Ubalansert trening 6 klasser
- Balansert trening 6 klasser
- Balansert trening 3 klasser grønt
- Balansert trening 3 klasser steinbasert

Resultater vises og diskuteres i kap 4.2.8 og 4.2.9.

4 Resultater og diskusjon

4.1 Forarbeid

Før klassifiseringen startet ble det gjort en eksplorativ analyse av datasettene, de ble først undersøkt med PCA og K-means clustering. Vi fikk ikke mye ut av disse analysene, siden klassene skulle deles inn i forhold til de utvalgte Mannings tall-klassene. Men det vi så av resultatene var det ikke ville bli lett å klassifisere asfalt og betong. I figur 4.1 kan en se resultatet til K-means med fire klasser, ulike typer asfalt har havnet i ulike klasser. Betong og parkeringsasfalt blir plassert i samme klasse, mens privat lagt asfalt, tak, og kommunal asfalt havner i en annen klasse.



Figur 4.1: K-means med fire klasser. Her har gress og trær havnet i samme klasse. Algoritmen greier å skille ut vegetasjon, men har noen problemer i barskog, og deler av denne har havnet i samme klasse som asfalt, vann og tak. Betong og parkeringsplass-asfalt har havnet i samme klasse: Dette kan antyde det at de oppfattes veldig lik spektralmessig

Mye av den videre eksplorative analysen handlet om å velge ut variasjoner av piksler fra ulike klasser og undersøke hvordan spektralsignaturene så ut. Dette for å se hvor store likheter og forskjeller det var mellom ulike materialer, både innad i klassene og mellom de forskjellige klassene. Det ble og oppdaget enkelte feil og mangler ved datasettet. Det er flere steder bølgelignde mønster i datasettet, og ved et flystripe overlapp ble var det opp til fire meter forskyvning. Den eksplorative analysen gikk så videre til å sammenlikne de ulike datasett mot hverandre. For å undersøke hvor lett det var å skille ulike vegetasjon og steinbaserte klasser fra hverandre ble dette gjort med CNN.

4.2Resultater av klassifisering

Denne delen vil ta for seg resultatene fra hovedklassifiseringene med CNN, SVM og LR. Først ble de ulike datasettene radians og reflektans sammenliknet, og det datasettet som presterte best vil blir brukt videre i oppgaven. Deretter ble algoritmene brukt til å klassifisere grønt mot ikke-grønt, steinbaserte overflater og vegetasjonsoverflater, de to siste ble undersøkt for å finne de mest hensiktsmessige klassefordelingene. Denne delen vil være mer eksplorativ, med fokus på å undersøke klassefordelinger. Til slutt ble de steinbaserte og vegetasjonsklassene kombinert så de kunne bli testet på det endelige valideringssettet Tøyen.

4.2.1 Reflektans mot radians

Det ble valgt å gjøre en sammenligning av klassifiseringer på radians- og reflektanssettet, for å se hvilket som presterte best av de to. Det ble testet både med og uten SWIR-båndene, mens VNIR-båndene alltid var inkludert. Datasettet som presterte best ble så brukt videre i resten av oppgaven for hver klassifisering, og til validering på Tøyen.

For å teste radians mot reflektans ble det for CNN utført klassifiseringer med følgende klasseinndeling:

2 klasser: Grønt – ikke-grønt
Grønt 3 klasser: Gress, tre og busk
Grønt 4 klasser: Gress, tre, hagebusk og kratt/eng
Steinbasert 3 klasser: Asfalt, betong og grus
Steinbasert 4 klasser: Asfalt, betong, grus, sand

Siden SVM og LR ble brukt etter at klassefordelingen ble endelig etablert ble sammenlikning mellom radians og reflektans kun gjort for:

6 klasser: Asfalt, betong, busk, gress, grus og tre.

CNN

Resultatene etter klassifiseringene som ble benyttet til sammenligningen av radians og reflektan kan ses i tabellen 4.1. Valideringsparametere som ble benyttet for å sammenligne resultatene mellom de ulike datasettene var accuracy. Resultatene viser at det settet som gjør det jevnt over best, er radians med både VNIR- og SWIR-båndene.

Tabell 4.1: I tabellen viser resultatene etter en sammenligning av de ulike datasettene radians og reflektans for ulike klassifiseringer. Kolonnene representerer hvilket datasett som er brukt og radene hvilken klassifisering som er blitt gjort. De to første kolonene er for radianssettet, der første kolonne bare er VNIR-båndene, mens andre kolonne inkluderer både VNIR- og SWIR-båndene. Kolonne tre og fire er tilsvarende bare for reflektanssettet. Felter uten verdi betyr at det ikke er blitt gjort noen klassifisering for det aktuelle datasettet.

	Accuracy			
	Radians		Reflektans	
	VNIR	VNIR + SWIR	VNIR	VNIR + SWIR
Grønn 3kl	0.8426	0.8594	0.6356	0.8225
Grønn 4kl		0.6825	0.4707	0.6624
Steinbasert 3kl	0.6890	0.8381	0.6377	0.7797
Steinbasert 4kl		0.7517		
Grønt vs. ikke-grønt	0.9978	0.9996	0.9972	0.9990

LR og SVM

Resultatene for LR og SVM kan ses i tabell 4.2. Det er valideringsparameteret accuracy som ble brukt til å sammenligne datasettenes ytelse.

Tabell 4.2: Tabellen viser klassifiseringer som er gjort for å sammenligne radians og reflektans, og samtidig VNIR mot alle bånd

	Accuracy 6kl			
	Radians		Reflektans	
	VNIR	VNIR + SWIR	VNIR	VNIR + SWIR
LR	0.9821	0.9827	0.9248	0.9550
SVM	0.9843	0.9953	0.9633	0.9622

For LR og SVM ble det et veldig jevnt resultat i sammenlikningen mellom radians- og reflektans-settene for Valle Hovin-området.

Resultatene viser at datasettene presterer ganske likt, men radians gjør det best. Derfor ble radiansbildet ble valgt til videre jobbing. Reflektans skal ikke avskrives, men på grunn av tidspress ble det besluttet å ikke jobbe videre med dette på Tøyen-settet.

Felles diskusjon

Det var radianssettet som presterte best for CNN. Nevrale nettverk er kjent for å være utmerkede variabelutvelgere, og dette gjør den kanskje mer egnet til å skille ut støy, som andre maskinlæringsalgoritmer ikke hadde klart. Grunnen til at reflektanssettet presterer dårligere for CNN kan komme av at settet har mistet nyttig informasjon da det ble atmosfærekorrigert. Dette kan tyde på at noe av informasjonen som ligger i atmosfæren, er nyttig for å kjenne igjen de ulike klassene. Det kan også ha vært avgjørende at store deler av det blå området i det elektromagnetiske spekteret var borte i reflektanssettet. Det er allikevel rart at det var stor forskjell mellom radians og reflektans for CNN, når det ikke var tilsvarende for SVM og LR.

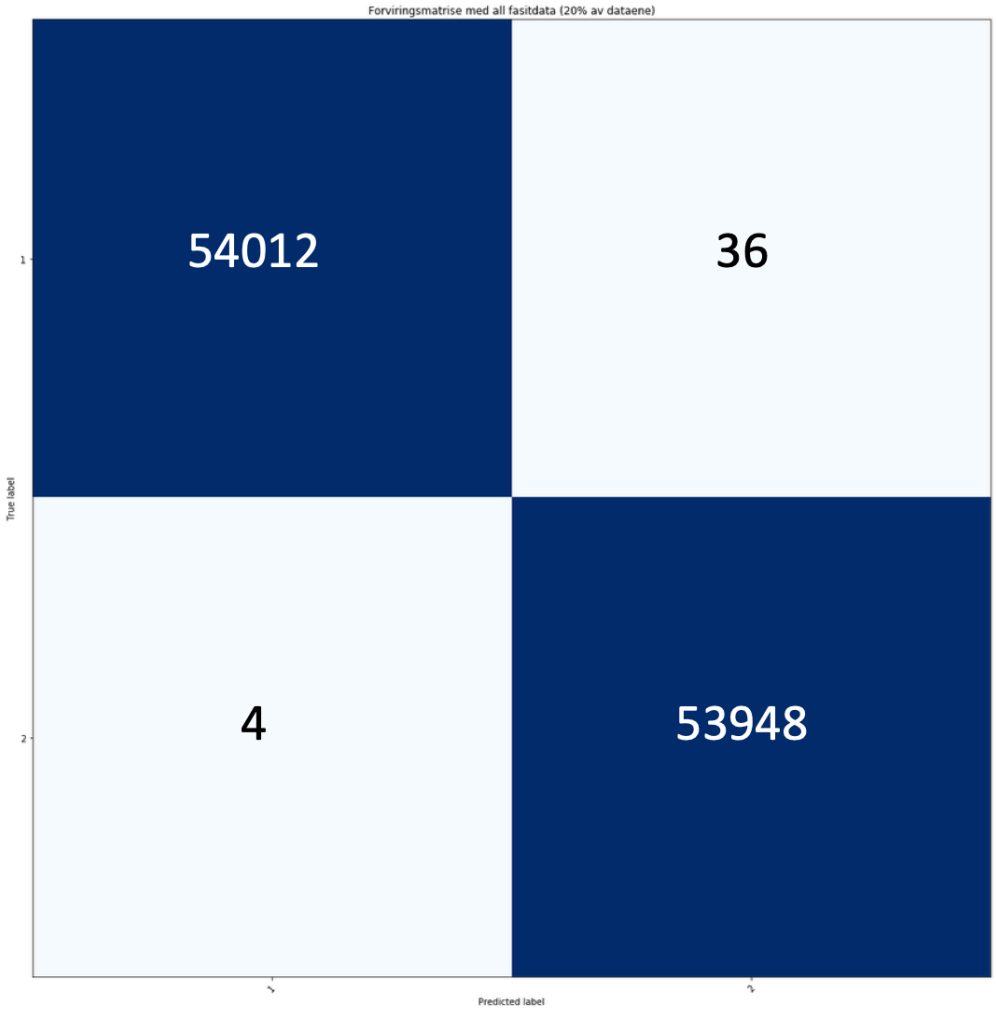
Det blir benyttet ulike matematiske formler for å produsere reflektanssettet, i denne prosessen forsvinner noe informasjon fra dataene, siden hver bearbeiding av de aktuelle datasett fører med seg noe usikkerhet. I akkurat dette reflektanssettet er det fjernet ganske mye informasjon.

Det ble bestemt å jobbe videre med radians for alle algoritmene, selv om det bare var ved CNN dette settet viste seg å være tydelig best. Ved SVM og LR presterte radians bare litt bedre. En avgjørende faktor for å velge radians, ble derfor at vi bare forholder oss til utvalgte datasett over Oslo, der mesteparten av bildene er tatt på samme tid. Det vil si at fasitene som ble laget i denne oppgaven bare gjelder for dette området. Skulle det blitt laget en robust modell for andre områder utenfor Oslo, måtte en sannsynligvis ha bruke reflektanssettet. Siden varierende radians oppstår mellom bilder som er tatt på ulike tidspunkt, selv om det er fra samme området (Howard et al., 2009).

4.2.2 To klasser, skille grønt og ikke-grønt

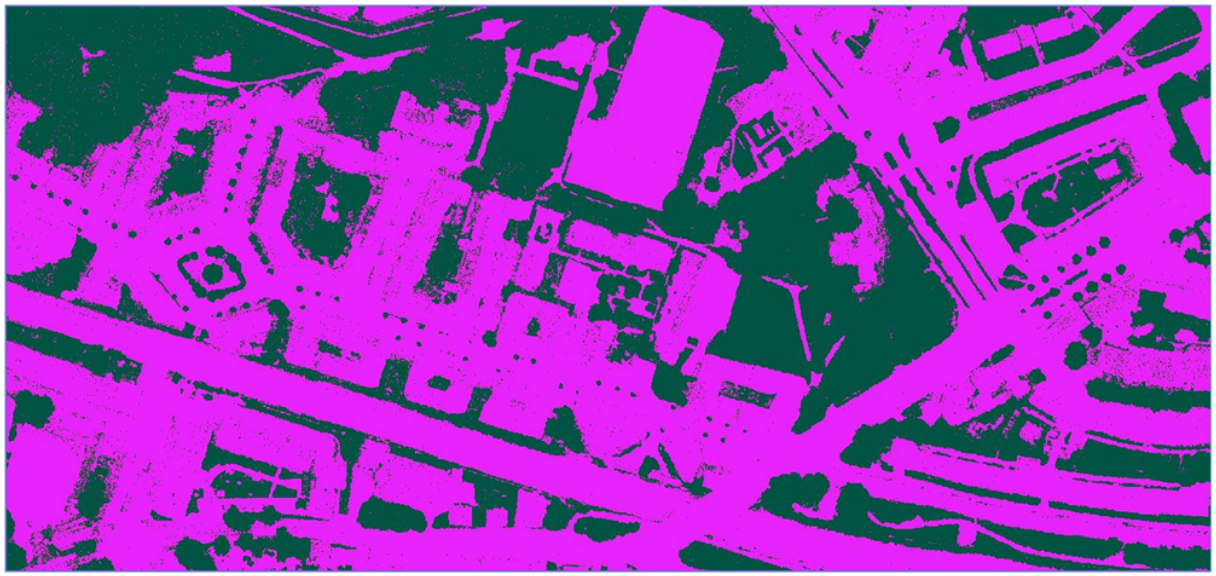
Den første klassifiseringen som ble utført var å skille vegetasjon fra ikke-vegetasjon. Dette ble testet siden denne type klassifisering i utgangspunktet skal være veldig enkel å utføre. Det er for eksempel veldig vanlig å bruke vegetasjonsindekser til å skille ut grønt eller vegetasjon. En av de mest brukte, NDVI, klarer ikke nødvendigvis å inkludere vegetasjon som ikke er frisk, eller er nesten dødt. Det var derfor spennenende å se hvordan de ulike algoritmene presterte på denne forholdvis enkle klassifiseringen.

Som forventet ble CNN-klassifiseringen grønt/ikke-grønt veldig bra. Forvirringsmatrisen viser resultatet etter klassifiseringen med dataene fra Valle Hovin, se figur 4.2. Det var bare 40 piksler som ble klassifisert feil av totalt 108 000. Resultatet for LR og SVM ble like bra, henholdsvis bare 18 og 9 av 120.000 piksler var feilklassifisert, resultatene for algoritmene LR og SVM blir derfor ikke presentert diskutert videre i.



Figur 4.2: Forvirringsmatrise der 20% av dataene fra Valle Hovin er brukt til klassifisering. De resterende 80% ble brukt til treningen av modellen. Dette er forvirringsmatrisen til CNN med datasettet radians, der både VNIR og SWIR er brukt.

Siden klassifiseringen gav gode resultatene, ble det bestemt å bruke CNN-modellen til å produsere et raster for å visualisere klassifiseringen. Her ble visualiseringssettet Gamle Oslo brukt, se figur 4.3.



Figur 4.3: Resultatbilde for grønn ikke-grønn klassifisering med radianssettet der modell 1 er brukt. Lilla symboliserer ikke-grønt, mens det grågrønne viser alt som er klassifisert som grønt. Bildet er produsert ved bruk av radianssettet med VNIR og SWIR båndene.

Resultatbildet fra klassifiseringen ble også lagt over originalbildet av Gamle Oslo, se figur 4.4. Ved første øyekast ser resultatet veldig bra ut, men ved å studere bildet nøye blir det tydelig at klassifiseringen har slitt med skyggeområder, og gressområder som enten er veldig tørre eller der jorden kommer til syne. Utenom dette har ikke modellen noen problemer å skille ut vegetasjon fra byområdet.

Figur 4.4: Bildet over Gamle Oslo og klassifiseringsresultatet overlappet, ved at resultatbildet er gjort gjennomsiktig.

4.2.3 Fastsetting av fasit

Det å produsere fasit er mye arbeid, men en vel så stor oppgave er å bestemme hvor grensene går, og hvor mye som skal inkluderes innenfor hver klasse. Det var ofte stor variasjon i spektralverdiene mellom to nabopiksler av samme materiell. Det vil derfor i denne delen nevnes noen av utfordringene som oppstod, og hvordan det har blitt valgt å løse disse.

Smale eller brede klasser
Det var noen viktige spørsmål som ble stilt under arbeidet med fasit. Hvordan skal klassene bli delt opp? Hvor mye variasjon skal være tillat innenfor hver klasse? Det som er ønskelig er en modell som gjør det jevnt over bra. Under produksjonen av fasit ble det derfor eksperimentert med å lage fasiter der hver av klassene kun inneholdt piksler som var veldig like hverandre, og fasiter der klassene hadde stor variasjon innad. De smale klassene vil i teorien gitt veldig gode resultater, men det er også en svært høy risiko for at modellen blir overtilpasset. Da vil den trolig prestere dårlig på ukjente områder. Brede klasser vil enten forvirre modellen eller gjøre modellen mer robust, slik at den presterer bedre på ukjente områder.

Gress
Man skal i utgangspunktet tro det er lett vite hva gress er, og vi mente det derfor var forholdvis enkelt å bestemme hva som skulle inkluderes i denne klassen. Det største problemet med gress er helsen på gresset. Det vil si om det er friskt, og ikke uttørket slik det ofte blir på sommeren. Lengden på gresset vil også ha innflytelse på spektralinformasjonen. På gressplener blir gresset klippet jevnlig, men i grøftekanter vil gresset gjerne ha en annen lengde og også inneholde mer av andre vekster. Dette kan gi en utfordring i klassifiseringen siden spektralverdier fra blomster og andre vekster inkluderes. Det ble derfor besluttet å bare inkludere gress fra store, homogene gressområder som gressplener.

Tre
Ulike arter innenfor trær har ulik spektralsignatur. Det kan derfor være vanskelig å presse alle trær inn i en klasse, spesielt med tanke på forskjellene innen løv- og bartrær (Røstad, 2018). Til tross for dette ble det allikevel besluttet å plassere alle trær innen en klasse. Grunnen til at dette ble gjort var at spektralverdiene skilte seg så mye fra gress, at det ikke ble sett på som nødvendig å skille ut grupper av trearter i første omgang.

Busk
Det var en stor utfordring å fastsette busk, for hva er det egentlig? Dersom en lar en busk stå uforstyrra over en lengre periode, så vil busken kunne vokser til noe som ligner mer på et tre. Det måtte bestemmes når er det en busk og når blir det et tre. En vanlig grense å sette er 5 meters høyde for tre (Skarpaas, 2017). Med bruk av laserdata ville høyder lett kunne settes, men siden denne oppgaven bare omhandlet piksler fra hyperspektrale bilder, var ikke dette et alternativ.

En annet utfordring med busk er at det finnes utallige sorter med buskplanter av ulik art og fasong. I et forsøk på å prøve å skille plantet busk fra vilt voksene krattbusker, ble det besluttet å prøve å dele opp klassen busk i to nye klasser. Disse ble kalt hagebusk og naturlig busk. Siden disse klassene viste seg å bare forvirre modellen enda mer, ble disse senere slått sammen til klassen busk igjen. Definisjonen på busk endte til slutt på vegetasjon over 40 cm og under 5 meter. Altså mer en samleklasse for vegetasjon som ikke var gress eller trær.

Sand
Sand er knuste mineraler som er mindre i diameter en grus (Spjeldnæs, 2015, Store norske leksikon, 2018). Dette gjør at grus og sand har veldig lik spektralsignatur. I utgangspunktet skulle sand og grus være to ulike klasser, men siden treningsområdet på Valle Hovin ikke hadde store nok areal med sand til å plukke 15.000 piksler, ble sand fjernet som egen klasse.

Grus
Akkurat som sand var det også utfordrende å finne nok gruspiksler på treningsområdet. Det var ingen grusbaner på området, bare tynne grusstier. Det ble inkludert grus/sand fra en bane på Ekeberg, men dette verken forbedret eller gjorde modellen dårligere. Det ble derfor besluttet å også inkludere sand i denne klassen for fasiten med brede klasser.

Betong
Det var vanskelig å finne betong eller støpt stein som i steinheller, men et stort, homogent betongområde på en kunstisbane på Valle Hovin ble brukt. Derfor er mesteparten av betongpikslene som ble brukt under treningen fra dette området. Optimalt skulle disse pikslene vært fordelt jevnt utover hele treningsområdet. Dette kan ha påvirket resultatet, og er kanskje en av grunnene til problemet med blanding mellom asfalt og betong under klassifisering.

Asfalt
Det ble fort oppdaget på spektralsignaturer at asfalt ikke bare er asfalt. Det ble funnet flere typer asfalt fordelt utover treningsområdet. Da alle de ulike asfaltene ble brukt oppstod det et problem med at asfalt og betong blandet seg. Det ble derfor forsøkt å velge bare en asfalttype, og valget falt på asfalt fra Ring 3 uten veimerking. Denne delen av Ring 3 er også europavei E6, så det kan antas at denne asfalten følger Vegvesenets standarder. Det ble i tillegg også brukt en mer generalisert asfaltklasse, som inneholdt flere typer asfalt.

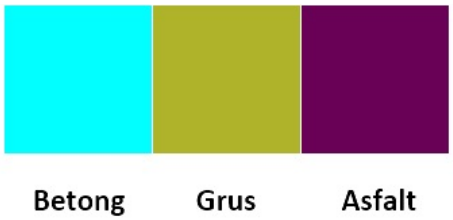
4.2.4 Klassifisering av steinbaserte materialer

CNN ble brukt til å utføre klassifisering av den steinbaserte delen av datasettet for å undersøke hvor utfordrende det var å skille disse klassene og deretter definere en endelig fasit. Denne fasiten ble så testet på LR og SVM, resultatene for disse to algoritmene kan ses i tabell 4.12. Resultatene etter de ulike klassifiseringene med CNN kan ses i tabell 4.3. Metrikkene som ble brukt til å sammenligne resultatene var accuracy og kappa, som kunne beregnes ut fra forvirringsmatrisen til klassifiseringen.

Tabell 4.3: Tabellen viser alle det ulike klassifiseringene for CNN, som er blitt gjort innenfor de steinbaserte klassene med radianssettet, der VNI-R og SWIR-bølgelengdene er inkludert. Accuracy- og kappaverdiene vises i kolonnen. De ulike klassifiseringen avhenger av hvordan klasseinndelingen er gjort (ulike fasiter) og hvilken modell som ble benyttet. Modell 1 er benyttet på alle radene der modell ikke er spesifisert.

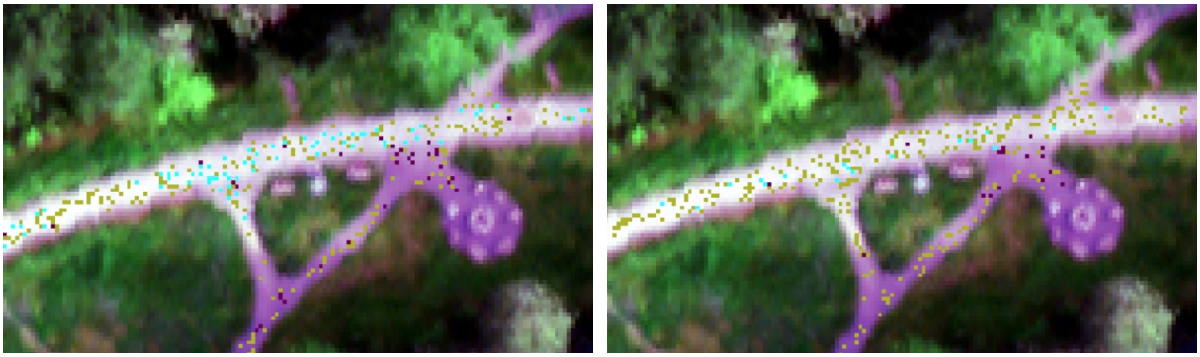
Steinbasert CNN Radians VNIR + SWIR		
	Accuracy	Kappa
4kl Generell	0.7517	0.6689
3kl Generell	0.8314	0.7472
3kl Med grusområde fra Ekeberg	0.8381	0.7549
3kl Smal asfalt klasse	0.9359	0.9000
3kl Smal asfalt klasse (modell 2)	0.9617	0.9402

Den steinbaserte klassifiseringen ble utført flere ganger med ulikt antall klasser og ulikt innhold i klassene. Det ble også testet med ulike arkitekturer i modellen. Dette siden det viste seg at klassene asfalt, betong og grus var vanskeligere å skille en forventet. Figur 4.5 viser fargen som er brukt for visualisering hver klasse i denne klassifiseringen.



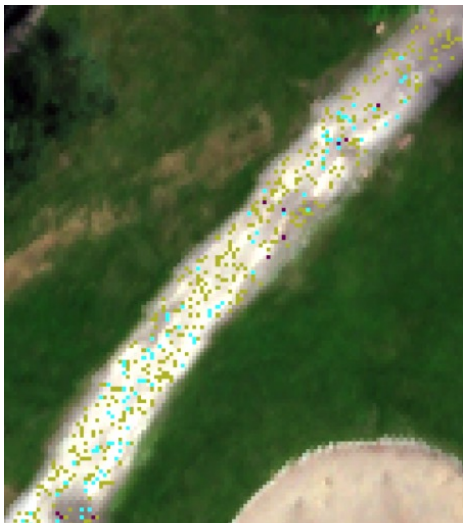
Figur 4.5: Fargene for visualisering av de tre klassene innenfor den steinbaserte klassifiseringen.

Den første klassifiseringen som ble utført hadde de fire klassene asfalt, betong, grus og sand, men som nevnt i kapitel 4.2.4 ble det besluttet å fjerne sand. På grunn av mangel på nok sandarealer i treningsområdet. Det ble derfor etter dette bare brukt tre klasser. Grus hadde også problem med lite treningsdata. I figur 4.6 vises to bilderesultater etter steinbasert klassifisering. Disse bildene viser hva grusveiene har blitt klassifisert som.



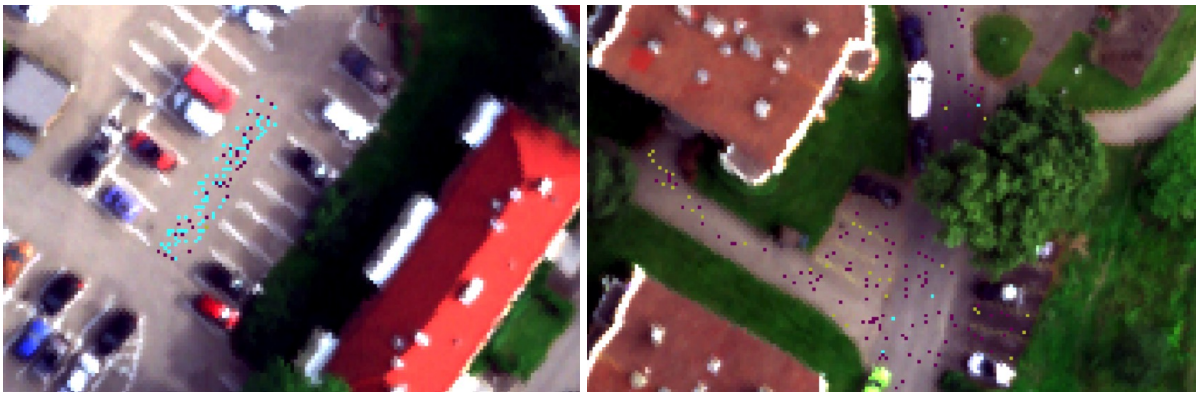
Figur 4.6: Bildet til venstre illustrerer den første steinbaserte klassifiseringen som ble utført. Denne er kalt 3kl generell siden det var stor spredning innenfor klassene (modell 1). Bildet til høyre viser den siste klassifiseringen som ble utørt med smal asfaltklasse og grus der klassen inneholdt treningsdata fra en grus/sandbane på Ekeberg (Modell 2.)

I det første bildet er det stor feilklassifisering av grus og betong i området med sollys, mens området som er i skyggen gir en blanding av asfalt og grus. Dette kommer av at spektralverdiene i hele det oppfangede spektere endrer seg i skyggeområder, noe som kan være årsaken til resulatet i bildet til venstre. Det var ønskelig å forbedre dette resultatet. Det ble derfor prøvd å bruke treningsdata fra et område på Ekeberg for å forberdre grusklassen, men resultatene ble ikke mye bedre da den nye modellen ble testet på Valle Hovin-området. Men siden resultatet ikke ble dårligere, ble det besluttet å fortsette å bruke dette området videre. I bildet til høyre vises resultatet etter den siste klassifiseringen for steinbasert. Det er fremdeles ikke perfekt, men klassifiseringen har blitt bedre enn den var i starten. Det kan også være nødvendig å dobbeltsjekke områder om en er usikker på klassene, se figur 4.7.



Figur 4.7: Denne vegen ble tolket som en grusvei under feltarbeidet. Men siden klassifiseringen av grus presterte veldig dårlig i starten, ble det bestemt å ta en ny tur for å se på denne veien. Da viste det seg at dette ikke var en grusvei, men en sølevei med grus, og jord og vann. Om veien var i samme stand da flybildene ble tatt sommeren 2017 får en aldri vite. Dette er en type usikkerhet knyttet til fasitene og datasettet.

Under den steinbaserte klassifiseringen var det store utfordringer med å skille asfalt fra de andre klassene. Det største problemet var blanding mellom asfalt og betong, og problemet med at det finnes mange typer asfalt. I følge Vegvesenet benyttes ulike typer asfalt på veien i forhold til trafikkbelastning, kostnader og tilgang på materialer (Vegvesenet, 2018). Det betyr at det blir benyttet ulik asfalt på kommunale veier, private veier og parkeringsplasser. Se eksempel i figur 4.8 og 4.9.



Figur 4.8: Illustrerer ulike typer asfalt og hva det har blitt klassifisert som. Bildet til venstre er over en privat parkeringsplass i et boligfelt. Bildet til høyre er over et boligfelt med parkeringsplasser. Når disse to bildene sammenliknes så er det tydelig at det er blitt brukt ulike typer asfalt på disse områdene, selv om begge er parkeringsplasser.

Figur 4.9: Bildene er fra samme parkeringsplass, men ulike steder på denne. I tillegg kommunal vei, sykkelvei og gangfelt. En ser her at det er veldig mange forskjellige typer asfalt bare innenfor dette lille området.

For å fikse problemet med ulike asfalttyper ble to løsninger prøvd ut. Den første var å inkludere alle de ulike typene asfalt samlet i en klasse i håp om at modellen lærer seg de mest generelle asfaltegenskapene. Den andre var å bare inkludere en type asfalt i klassen, og håpe at andre typer asfalt blir gjenkjent på et nytt område. Det var løsning to som ga best resultat på treningsområdet (figur 4.10).

Figur 4.10: Visualisering av resultatet etter klassifisering med smal asfaltklasse (hentet fra Ring 3/E6), og bruk av arkitekturen til modell 2. Mesteparten av asfalten i dette området er klassifisert rett, men noen av skyggeområdene forvirrer modellen, og det blir dermed klassifisert feil.

Det ble valgt å ta med en smal fordeling og en bred fordeling videre til klassifiseringen med seks klasser. Den smale ble valgt siden den presterte best. Årsaken til at den brede fordelingen også ble tatt med, var fordi vi trodde den brede ville prestere bedre på et ukjent område enn den med smal klassefordeling.

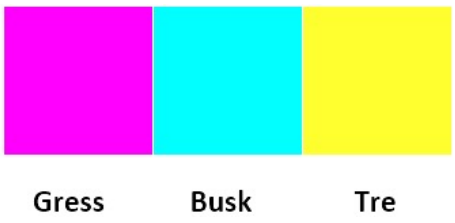
4.2.5 Klassifisering av grøntarealer

CNN ble brukt til å utføre ulike klassifiseringer på de grønne klassene, der ulikt antall klasser og modeller ble brukt. Målet var å se på vanskelighetene innad i de grønne klassene, og definere en endelig fasit. Denne fasiten ble deretter testet på LR og SVM, resultatet for disse kan ses i tabell 4.12. Resultatene etter de ulike klassifiseringene med CNN kan ses i tabell 4.4. Metrikkene som ble brukt til å sammenligne resultatene var accuracy og kappa.

Tabell 4.4: Tabellen under viser alle det ulike klassifiseringene som er blitt gjort innenfor de grønne klassene med radianssettet, der VNIR- og SWIR-bølgelengdene er inkludert. Accuracy- og kappaverdiene vises i kolonnene. De ulike klassifiseringen avhenger av klasseinndelingen (ulike fasiter) og hvilken modell som er blitt benyttet.

Grønn CNN Radians VNIR + SWIR		
	Accuracy	Kappa
4kl Model 1	0.6825	0.5766
3kl Model 1	0.8596	0.7891
3kl Model 2	0.8984	0.8476
3kl Model 3	0.9012	0.8519

Klassifiseringene for grønt ble utført med fire og tre klasser, tillegg ble det brukt tre ulike modeller under klassifiseringene med tre klasser. Fire klasser ble bare testet med modell 1 på grunn av dårlige resultater. Fargene som er blitt brukt for de ulike klassene i grønnklassifiseringene kan ses i figur 4.11.

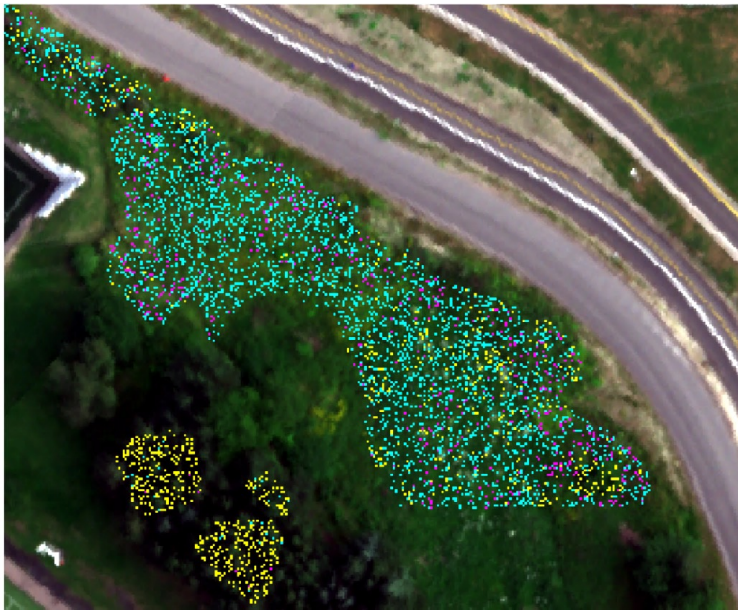


Figur 4.11: Fargene for visualisering av de tre klassene innenfor grøntklassifiseringen.

Det ble ikke gjort mye endringer innenfor klassefordeling ved grøntklassifisering. Grunnen til dette var at klassene gress og tre var veldig lette å plassere, busk derimot var en større utfordring. Som nevnt i fasitproblematikk var det vanskelig å definere denne klassen. Etter at klassegrensene var bestemt var det ikke så mye å endre, men busk var fremdeles et problem. Det ble fokusert på å teste tre ulike modeller med ulik arkitektur. I figur 4.12 kan en se en klassifisering der modell 1 er brukt.

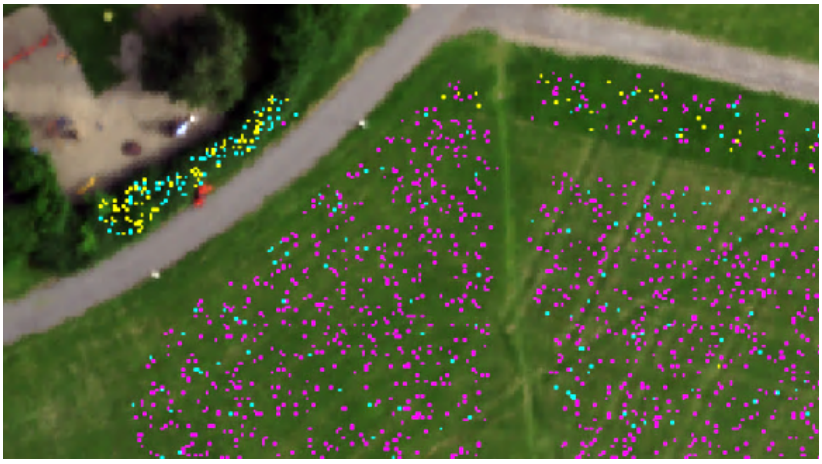
Figur 4.12: Resultatet etter klassifisering med tre klasser ved bruk av modell 1.

Når en ser på klassifiseringen som er utført i figur 4.12, ser det ikke ut til å være veldig store problemer med buskklassen. Det er noe blanding mellom busk og gress der det egentlig er gress, men ellers ser klassifiseringen helt grei ut. Buskområdet til høyre i bildet er veldig spesiell art som kun finnes der på hele Valle Hovin, det er derfor mulig å skille denne ut. Slik var det ikke med de andre buskeområdene i treningssettet, se figur 4.13.

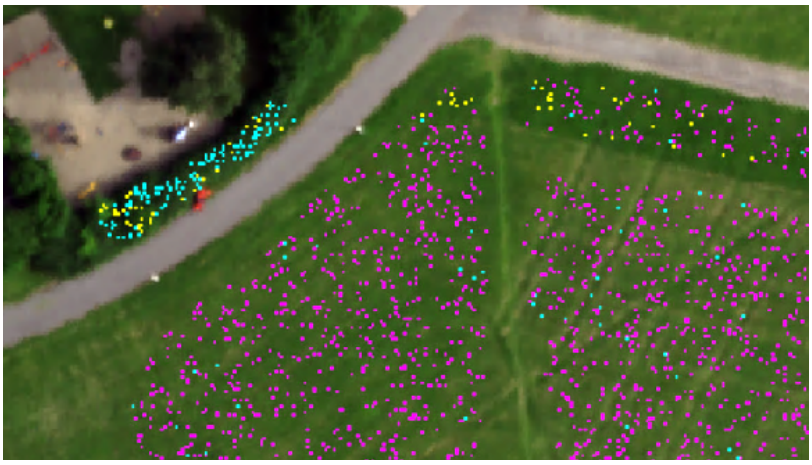


Figur 4.13: Dette er et utklipp fra treningssettet, bildet er over et busk/krattområde. Det er en blanding av høyt gress, små busker og små trær. På grunn av mye kaos i området ble det også en god blanding av alle klassene.

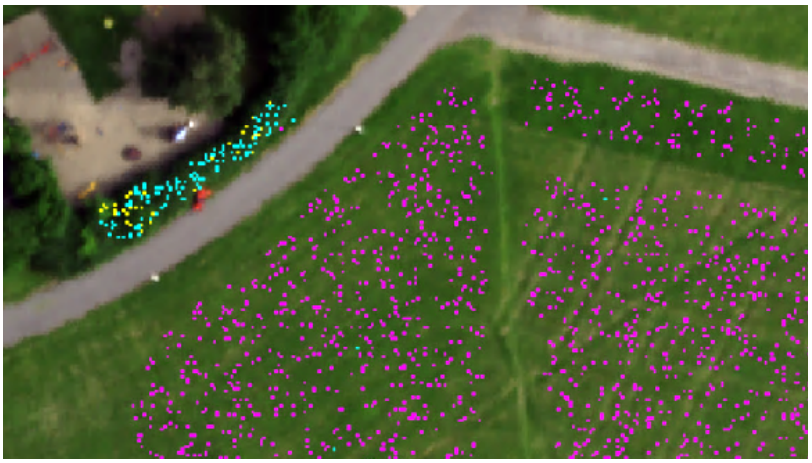
Det ble testet tre ulike modeller for klassifiseringen av grøntarealer. Modell 2 og 3 presterte bedre enn modell 1 (tabell 4.4). I de tre figurene 4.14, 4.15 og 4.16 kan en se resultatet til de tre modellene. I utklippet har en busk og tre gressflekker blitt klassifisert.



Figur 4.14: Grøntklassifisering med modell 1. Det er mye busk blandet inn i det to første gressflekkene, mens det siste gressområdet også har en del trepiksler i tillegg til noe busk. Det er cirka halvparten busk og tre i buskområdet.



Figur 4.15: Grøntklassifisering med modell 2. Denne modellen ser ut til å prestere noe bedre enn modell 1, men det har havnet mer trepiksler i gresset. Til gjengjeld er det mye færre trepiksler i buskområdet.



Figur 4.16: Grøntklassifisering med modell 3. I denne modellen er det nesten ingen andretyper piksler i gressområdene, og buskområdet ser ut til å prestere like bra som i modell 2.

4.2.6 6 klasser med CNN

Etter at mulighetene for å skille klassene innen grønt og steinbasert individuelt var undersøkt, ble den klassefordelingen som hadde gitt de beste resultatene innenfor hver av disse kombinert til seks klasser. Disse ble så undersøkt med både smale og brede klassefordelinger. I denne klassifiseringen ble modell 1, 2 og 3 brukt. Resultatene etter CNN-klassifiseringene kan ses i tabell 4.5, der accuracy og kappa er brukt som valideringsparameter.

Tabell 4.5: Tabellen inneholder alle de ulike klassifiseringene som er gjort med CNN for seks klasser, radianssettet med VNIR- og SWIR-bølgelengdene er brukt til dette. Accuracy- og kappaverdiene vises i kolonner. De ulike klassifiseringen avhenger av klasseinndelingen (ulike fasiter) og hvilken modell som er blitt benyttet.

Klassifisering av 6 klasser CNN		
	Accuracy	Kappa
Model 1, smal fasit	0.8247	0.7897
Model 2, smal fasit	0.8867	0.8641
Model 3, smal fasit	0.9118	0.8941
Model 3, bred fasit	0.8735	0.8482

Resultatet klassevis for de tre modellene finnes i tabellene 4.6, 4.7, og 4.8. I disse tabellene er valideringsparameterne predcition, recall, f1-score brukt. Disse er beregnet ut fra forvirringsmatrisen til hver klassifisering. Support angir antallet piksler brukt i validering i hver klasse.

Tabell 4.6: Tabellen viser resultatene fra klassifisering med seks klasser, der oppsett for modell 1 er brukt. Modellen har blitt trent og testet på smale klasser. Metrikkene precition, recall, f1-score, support er oppgitt i kolonnene.

Modell 1 Smale klasser				
Klasse	Precision	Recall	F1-score	Support
Asfalt	0.94	0.93	0.94	3860
Busk	0.62	0.67	0.64	3901
Grass	0.80	0.74	0.77	4011
Grus	0.94	0.89	0.91	3916
Betong	0.88	0.93	0.90	4031
Tre	0.79	0.78	0.79	3925

Tabell 4.7: Tabellen viser resultatene fra klassifisering med seks klasser, der oppsett for modell 2 er brukt. Modellen har blitt trent og testet på smale klasser. Metrikkene precition, recall, f1-score, support er oppgitt i kolonnene.

Modell 2 Smale klasser				
Klasse	Precision	Recall	F1-score	Support
Asfalt	0.95	0.98	0.96	3860
Busk	0.73	0.79	0.76	3901
Grass	0.88	0.80	0.84	4011
Grus	0.94	0.96	0.95	3916
Betong	0.95	0.90	0.93	4031
Tre	0.88	0.89	0.89	3925

Tabell 4.8: Tabellen viser resultatene fra klassifisering med seks klasser, der oppsett for modell 3 er brukt. Modellen har blitt trent og testet på smale klasser. Metrikkene precition, recall, f1-score, support er oppgitt i kolonnene.

Modell 3 Smale klasser				
Klasse	Precision	Recall	F1-score	Support
Asfalt	0.99	0.97	0.98	3860
Busk	0.82	0.77	0.79	3901
Grass	0.87	0.89	0.88	4011
Grus	0.95	0.97	0.96	3916
Betong	0.96	0.95	0.95	4031
Tre	0.89	0.92	0.91	3925

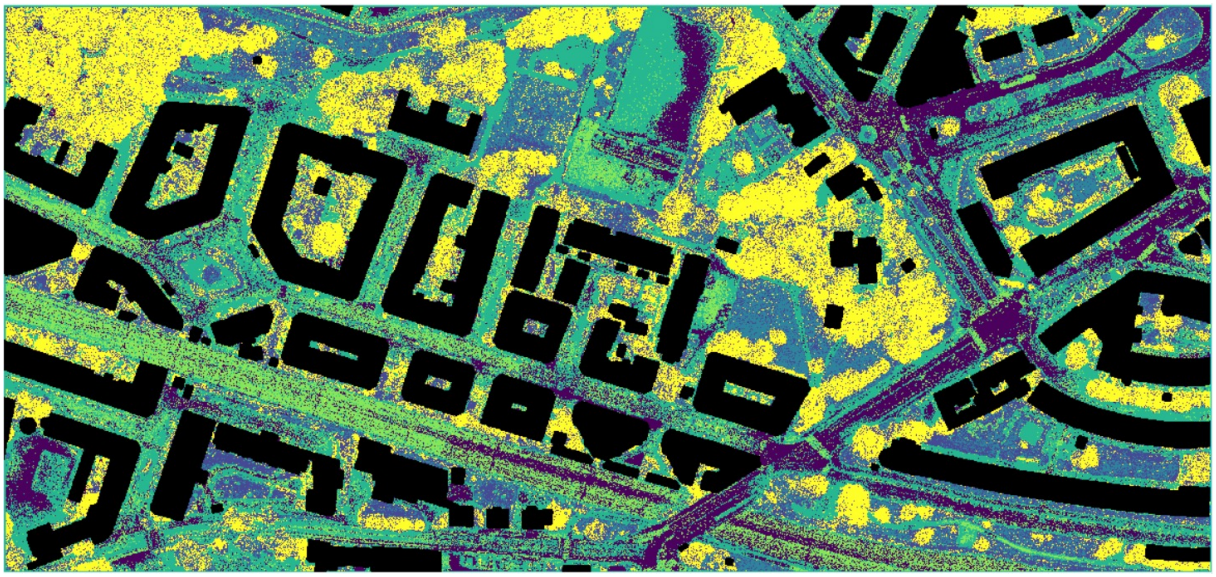
Når en studerer resultatene i tabellene for klassifiseringene med smale klasser, kan en se at det modellene har slitt mest med er de grønne klassene. Busk har vært den mest utfordrende klassen. Årsaken til dette kan komme av at hyperspektrale bilder kanskje ikke er den mest hensiktsmessige måten å skille ut en veldig bred og udefinert klasse som busk.

Av de tre modellene var det den mest kompliserte, modell 3, som presterte best, med modell 2 rett bak. En av årsakene til dette kan være at et nettverk som er mer komplisert og dermed har flere skjulte lag som fører til flere parametere, kan lagre mer informasjon i nettverket sitt. Slik kan nettverket være i stand til å finne mønstre i dataene som en enklere og mindre komplisert modell ikke er i stand til. Minuset med dette er at en mer komplisert modell også har større risiko for å bli overtilpasset i forhold til treningsdataene. De lærer mye unyttig informasjon som ikke blir oppdaget før modellen blir testet på et område som er adskilt fra treningsområdet.

Derfor ble modellen teste på visualiseringssettet Gamle Oslo (se figur 4.18). Siden det ikke var utarbeidet en fasit over dette området, ble det heller generert rasterbilder for visualisering av modellenes prestasjoner. Bildene kan ses i figur 4.19, 4.20 og 4.21. Datasettet Gamle Oslo er kun brukt til å produsere rasterbilder, og ingen fasit er utarbeidet for dette området. Derfor er det ikke mulig å generere noen statistikk på hvor bra klassifiseringene har prestert, men en får en god indikasjon på hvordan ved visuell sjekk. Fargene som er tilegnet hver klasse kan ses i figur 4.17.

Figur 4.17: Bildet viser fargene for de ulike klassen. Disse gjelder for alle videre bilder der modellen skal klassifisere seks klasser. Tak er tatt med selv om modellene ikke hadde klassen tak med i klassifiseringen. Takklassen er lagt på i ettertid ved maskering ved hjelp av FKB-data for tak..

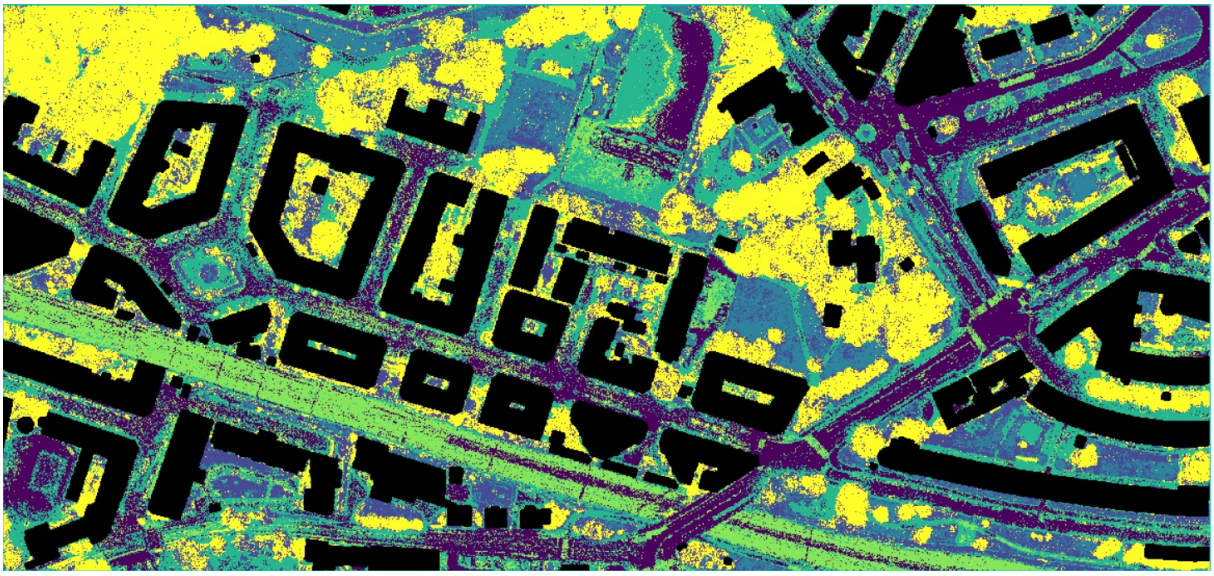
Figur 4.18: Bildet som ble brukt til grunnlag for å produsere rasterbildene. Utsnitt over deler av bydel Gamle Oslo.



Figur 4.19: Over er resultatbildet til modell 1 med smal klassefordeling.

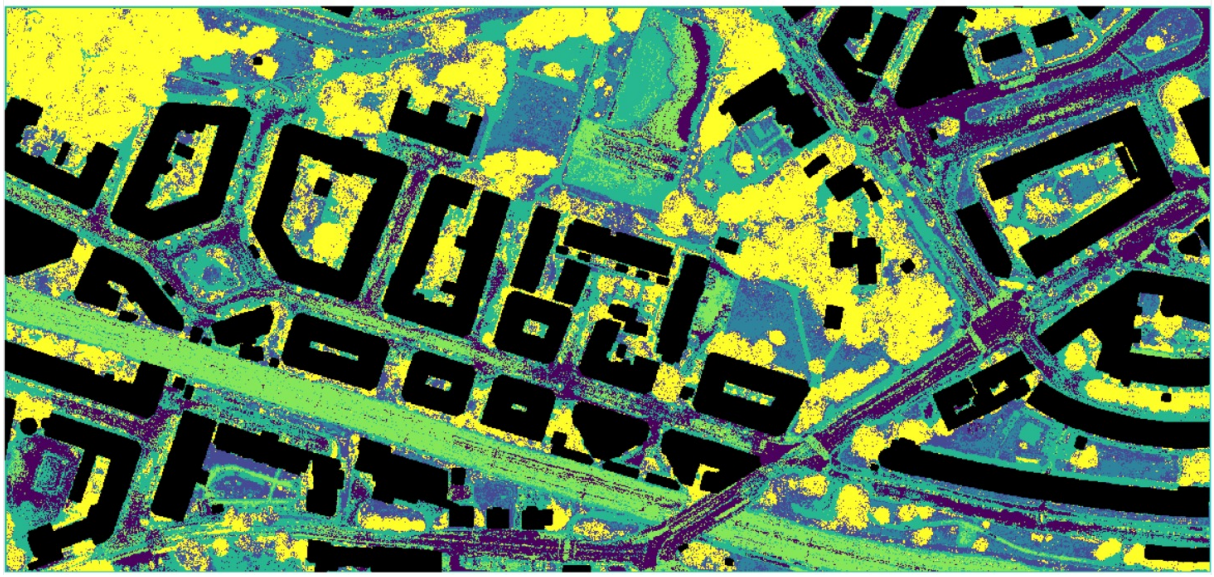
Bildet i figur 4.19 har blitt veldig kornete, siden modellen ikke har greid å bestemme seg for hvilket material signaturen tilhører. Derfor har den blandet mellom to eller tre ulike klasser. Det har vært spesielt mye forvirring mellom asfalt, betong og grus i områder som egentlig skal være klassifisert som asfalt. Siden det er en veldig smal asfaltklasse som er blitt brukt, var dette ikke veldig overaskende. Dette gjelder spesielt asfaltområder som havner i skyggen og områder med annen farge på asfalten. Det er også en grusbane som har blitt klassifisert

som grus og asfalt. Årsaken til dette virker å være skyggen som ligger over deler av banen. Noe annet som er verdt å merke seg er at buskklassen har havnet mellom gress og tre. Det vil si at buskklassen ikke opptrer alene, men gjerne sammen med gress- eller treklassen.



Figur 4.20: Over er resultatbildet til modell 2 med smal klassefordeling.

Akkurat som modell 1 sliter også denne klassifiseringen (figur 4.20) med blanding mellom asfalt, betong og grus, men denne modellen ser likevel ut til å ha prestert noe bedre på disse. Det er også mye mindre støy i bildet i forhold til modell 1. De grønne klassene har samme problem som i modell 1, men det ser ut til å ha forbedret seg noe, siden det nå er lettere å se klyngedannelse i disse klassene. Noe som er litt pussig er grusbanen i øvre del av bildet, akkurat som i modell 1 er skyggen blitt oppfattet som asfalt, men en stripe som passerer rett gjennom banen er plutselig blitt klassifisert som tre. Kan dette indikere at noe er galt med modellen?



Figur 4.21: Over er resultatbildet til modell 3 med smal klasserfordeling.

Når en sammenligner bildet i figur 4.21 med bilde fra modell 2 er det vanskelig å se hvilken modell av de to som presterte best. Modell 3 presterte kanskje litt bedre på asfalt klassen en

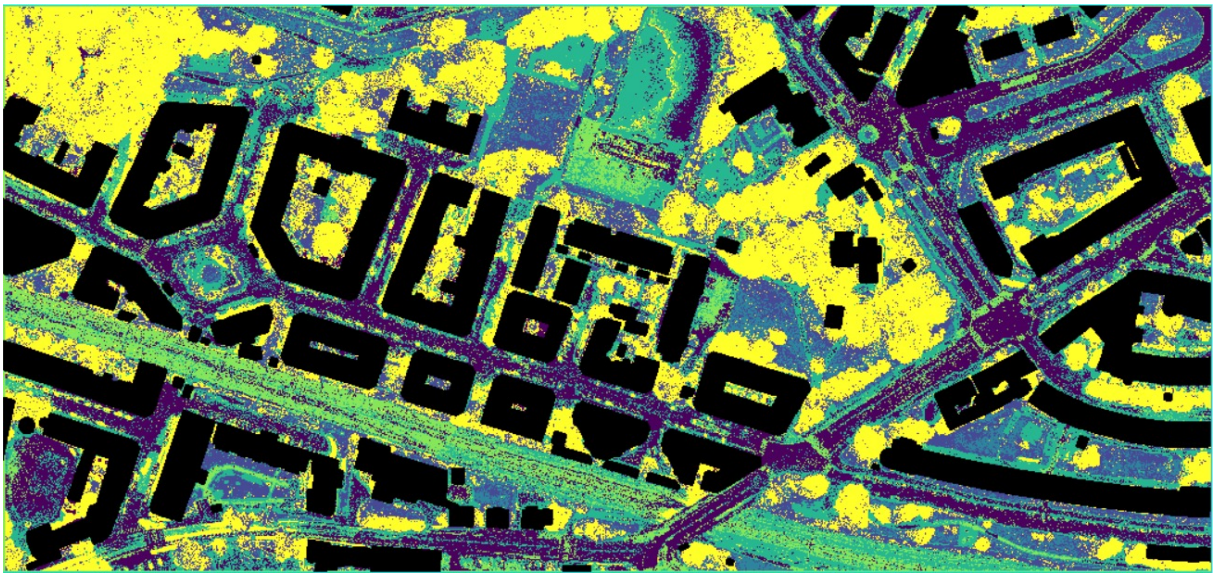
modell 3. Noe som er litt spesielt ved modell 3 er utviklingen på grusbanen. Nå består skyggeområdet av striper som følger skyggen. Disse klassifiseres som grus, asfalt, betong og til slutt en veldig tynn stripe som busk. Dette tyder på at modellen har lært seg noe unyttig.

Modell 3 ble valgt å jobbe videre med, til tross for noen snodige resultater i rasterbildet. Grunnen var at den hadde prestert best på klassifiseringen ifølge valideringsparameterne. Det ble derfor kjørt en ny klassifisering på denne modellen, den var trent på en mye bredere klassefordeling som inkluderte flere typer asfalt, og sand lagt til under grusklassen. Resultatet for denne klassifiseringen finnes i tabell 4.5 og 4.9, mens rasterbildet kan ses i figur 4.22.

Tabell 4.9: Tabellen viser resultatene fra klassifisering med seks klasser, der oppsett for modell 3 er brukt. Modellen har blitt trent og testet på brede klasser. Metrikkene precition, recall, f1-score og support er oppgitt i kolonnene.

Model 3 Brede klasser				
Klasse	Precision	Recall	F1-score	Support
Asfalt	0.95	0.94	0.95	21318
Busk	0.67	0.91	0.77	21216
Grass	0.94	0.69	0.79	21348
Grus	0.99	0.86	0.92	21366
Betong	0.86	0.97	0.91	21321
Tre	0.94	0.88	0.91	21396

Klassifiseringen presterer dårligere enn samme modell med smal klassefordeling i forhold til valideringsparameterne fra treningsområdet på Valle Hovin, men når en studerer rasterbildet fra klassifiseringen, virker ikke resultatet å være dårligere enn bildene som ble produsert av modell 2 og 3 med smal klassefordeling.



Figur 4.22: Resultatbildet etter bruk av modell 3 med en bred klassefordeling.

4.2.7 Endelig validering på Tøyenområdet med CNN

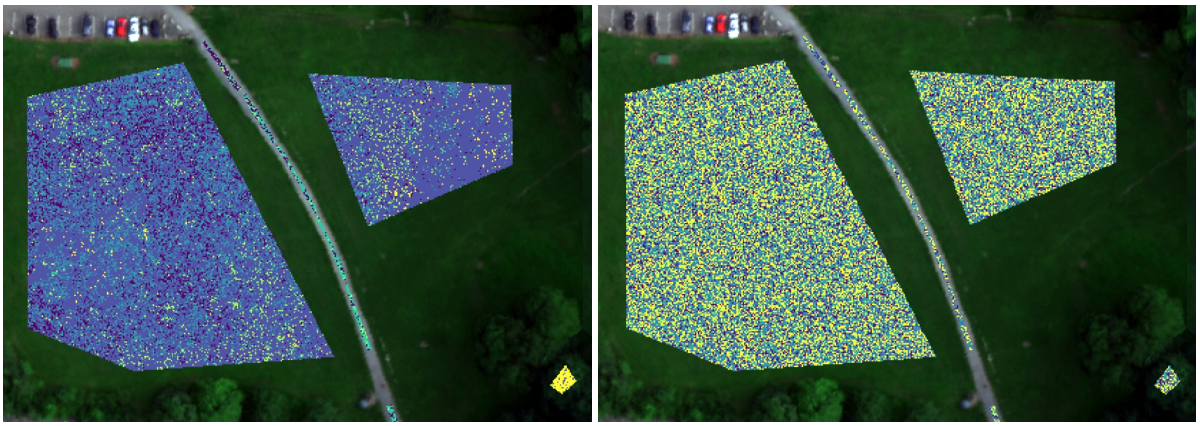
Siden Tøyen skulle brukes som en endelig validering, ble ingen modeller ikke testet på dette området før all trening på Valle Hovin var ferdig. Dette ble gjort for å se hvordan modellene presterte på totalt ukjente områder, og kan derfor bli sett på som den ultimate testen. Fire modeller ble kjørt for klassifisering på testsettet Tøyen. Disse var de samme modellen som det ble produsert raster over Gamle Oslo for. Resultatene etter klassifisering på Tøyen-området ble ikke bra, se tabell 4.10.

Tabell 4.10: Tabellen inneholder alle de ulike klassifiseringene som ble gjort med CNN på Tøyenområdet, radianssettet med VNIR- og SWIR-bølgelengdene ble brukt til disse klassifiseringene. Accuracy- og kappaverdiene vises i kolonnen. De ulike klassifiseringeen avhenger av klasseinndelingen (ulike fasiter) og hvilken modell som er blitt benyttet.

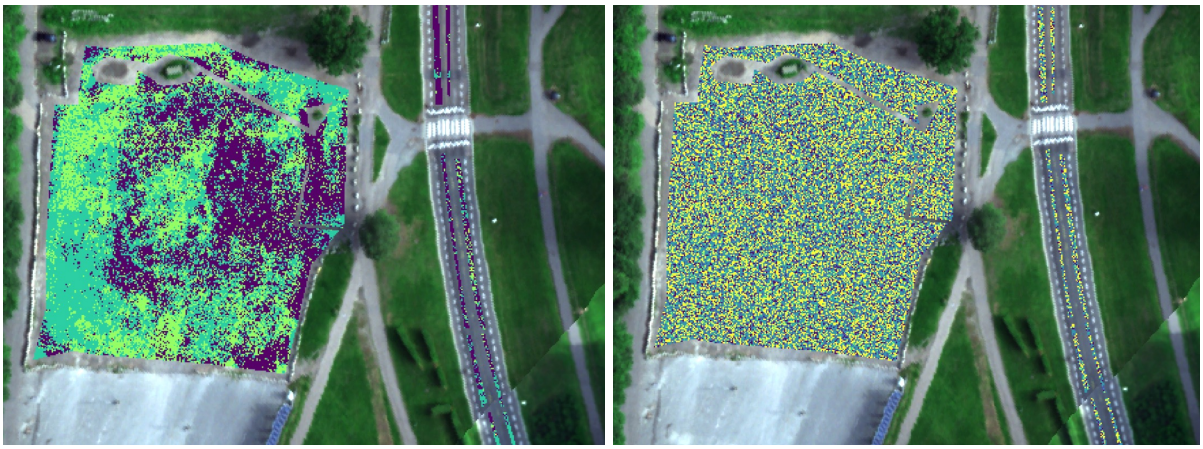
Tøyen CNN		
	Accuracy	Kappa
Model 1, smal fasit	0.5121	0.3839
Model 2, smal fasit	0.2337	-0.0005
Model 3, smal fasit	0.2515	0.0002
Model 3, bred fasit	0.1914	-0.0002

Modellen som presterte best var modell 1 med smale klasser, den fikk 0.51 accuracy og 0.38 kappa-koeffisient. Dette resultatet er ikke veldig bra, siden modellen da bommer annenhver gang, og en kappa-koffisient på under 40 pleier å indikere at modellen er dårlig, ifølge tabell 2.1. Resultatene fra de tre andre modellene indikerte at resultat fra disse var helt tilfeldige, siden kappaverdiene var 0. Urbane områder er veldig utfordrende å klassifisere, på grunn av inhomogene overflater. Det var derfor forventet noe reduksjon i resultatet i forhold til trening, men ikke en verdi nede på 0.51.

Sharma et al. (2017) som har jobbet med få og generaliserte klasser ved pikselbasert klassifisering på urbane områder har fått noe bedre resultat på urbane områder med pikselbasert. Dette kan antydet at urbane områder med stor variasjon innad i datasettet er utfordrende for CNN. Siden alle modellene fikk veldig mye bedre resultater da de ble testet på Valle Hovin-området, tyder dette på at alle modellene har blitt overtilpasset der. I figur 4.23 og 4.24 blir resultatbilder fra to klassifiseringer med modell 1 og modell 3 for Tøyen sammenliknet, begge har smal klassefordeling. Se figur 4.17 for fargevalg per klasse.



Figur 4.23: Til venstre er resultatbilde fra modell 1, til høyre resultatbilde fra modell 3. Utklippet er fra Tøyenområdet, og viser to gressplener med en grusvei mellom. Bildet til venstre klassifiserre flertallet av pikslene på plenene som busk, men det er stor variasjon og alle klassene er representert. Grusvegen blir klassifisert for det meste som asfalt eller grus, med noen piksler fra andre klasser. Dette er ikke en veldig bra klassifisering, men den presterer mye bedre enn bildet til høyre. Dette bildet har en tilfeldig plassering av klasser uten noen form for mønster.



Figur 4.24: Til venstre er resultatbilde fra modell 1 og til høyre resultatbilde far modell 3. Utklippet er fra Tøyenområdet, og viser en grusparkeringsplass (Sirkustomten) med en asfaltert veg til høyre. I bildet til venstre er grusbanen blitt klassifisert til en blanding av grus, betong og asfalt, også asfaltvegen til høyre består av tilsvarende. Akkurat som forrige bilde (figur 4.23) viser bildet til høyre ikke noe annet enn tilfeldig plasserte klasser..

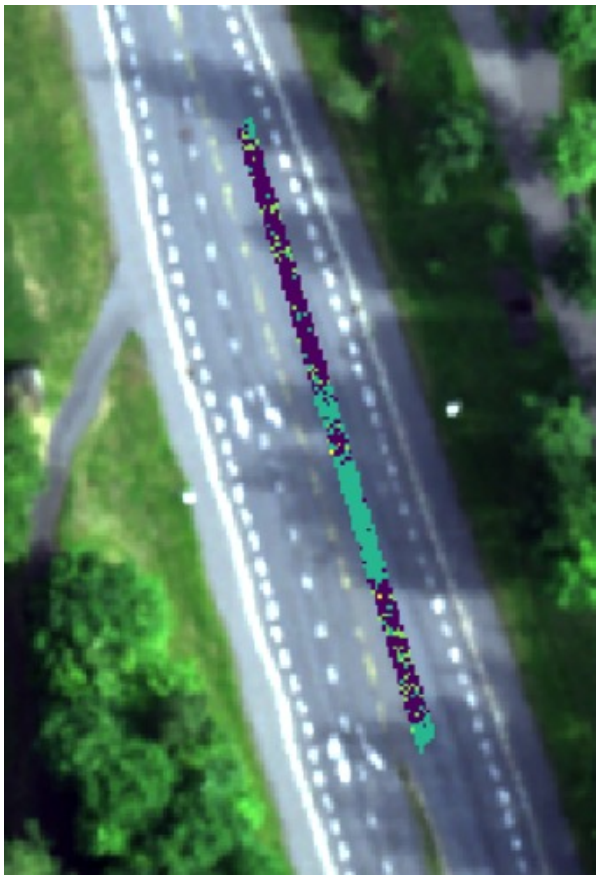
Siden modell 1 ifølge kappaverdien var den eneste modellen som ikke ga tilfeldig genererte klasser, vil bare resultatene til denne modellen bli brukt videre. Resultatene for hvordan hver av klassene har prestert kan ses i tabell 4.11.

Tabell 4.11: Tabellen viser resultatene fra klassifisering med seks klasser, der oppsett for modell 1 er brukt. Klassifiseringen er gjort på Tøyenområdet. Metrikkene precition, recall, f1-score, support er oppgitt i kolonnene.

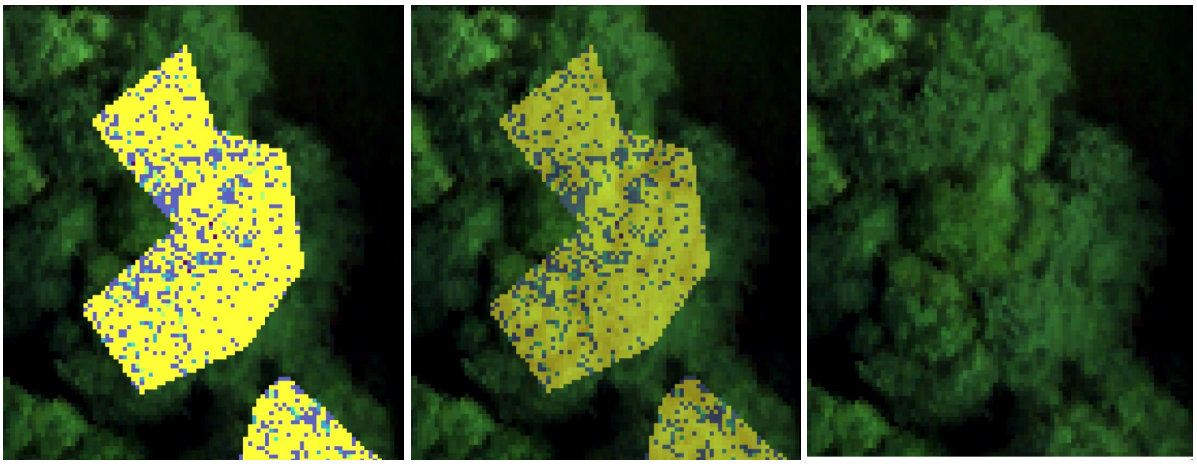
Model 1 Tøyen				
Klasse	Precision	Recall	F1-score	Support
Asfalt	0.20	0.26	0.23	22287
Busk	0.04	0.35	0.07	11462
Grass	0.95	0.45	0.61	181463
Grus	0.49	0.38	0.43	45050
Betong	0.19	0.40	0.25	10826
Tre	0.82	0.77	0.80	97415

Tre er den klassen som har prestert best, og det kan være flere årsaker til dette. Kanskje dette var den klassen som skilte seg mest fra det andre klassene, siden trær har mye høyere verdier enn gress og busk i den nær-infrarøde delen. En annen årsak kan være at oppsettet til modell 1 opprinnelig var laget for å skille ulike tresorter fra hverandre (Trier et al., 2018), og derfor var det lettere å finne trær?. Det kan også hende at trærne som ble brukt i treningssettet og valideringssetter var samme tresort eller at de lignet veldig mye på hverandre.

Det ble oppdaget at alle resultatene på Tøyenområdet var veldig sårbare for skygger og ulik lysinnstråling, figur 4.25 og figur 4.26 illustrerer dette. Det er mange som jobber med å finne en løsning på skyggeproblematikken (Sharma et al., 2017), og figurene viser hvordan hyperspektrale bilder blir påvirket av skygger. Også trær har denne problematikken på grunn av store variasjoner mellom trepikslene, se figur 4.26.



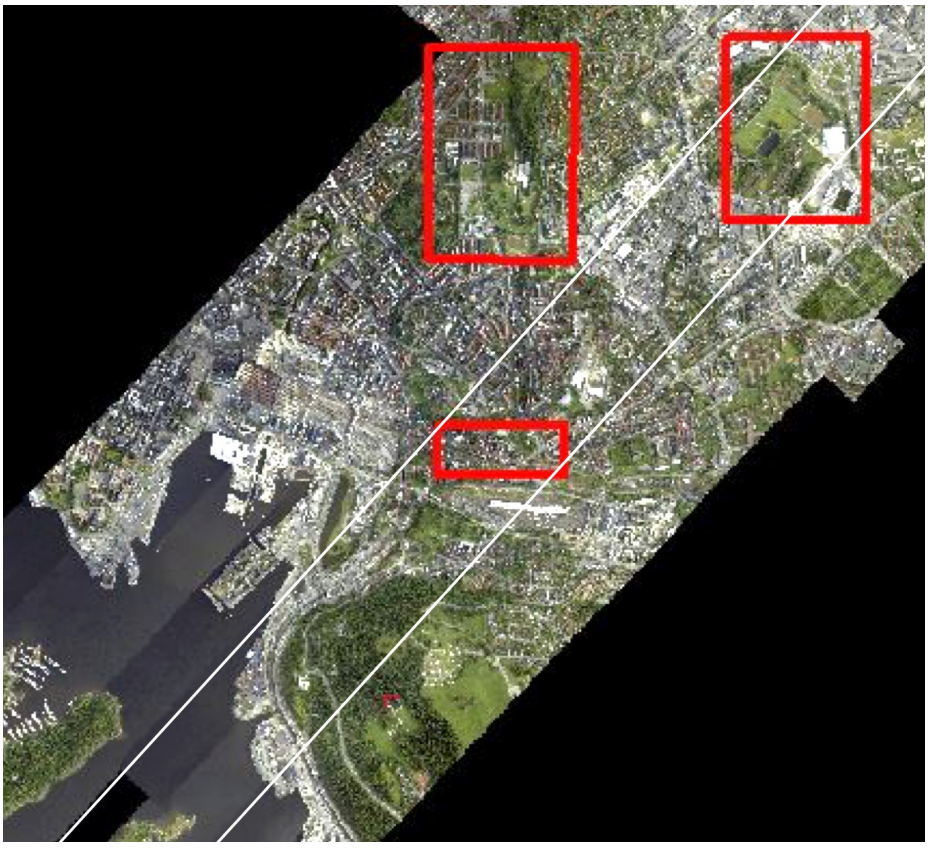
Figur 4.25: Dette bildet illustrerer hvordan asfaltpiksler blir påvirket av skygge. I skyggen blir pikslene klassifisert som grus. Dette illustrerer at modellen er veldig sårbar overfor skygger.



Figur 4.26: Til venstre utklipp fra resultatbilde fra Tøyen, i midten er resultatet gjort gjennomsiktig. Bildet helt til høyre er bare utklippet fra Tøyen uten resultatet lagt over. Mesteparten av trærne er blitt klassifisert som klassen tre, men noe som busk og også litt piksler fra de andre klassene. De største skyggeområdene ser ut til å gi utslag i resultatbildet, ved at akkurat disse blir klassifisert som noe annet enn tre.

Det at visualiseringssettet Gamle Oslo og testsettet Tøyen skulle prestere så ulikt var svært overraskende. Begge var holdt helt adskilt fra treningssettet Valle Hovin. Det er en mistanke om at flystripene kan være involvert forskjellen på resultatene. Som nevnt i kapittel 3.1 databestillingen ble det ventetid mellom noen av flystripene under billedtagningen, siden området lå i innflygningssonen til Gardemoen lufthavn. Dette førte til ulik solinnstråling på flystripene. Det kan observeres tydelige forskjeller mellom flystripene på figur 4.27. Siden de

ulike flystripene varierer mye seg imellom, så kan dette ha ført til at modellen ikke er i stand til å klassifisere et annet område med andre flystripe. Kanskje hadde vi unngått dette problemet hvis vi hadde forholdt oss til reflektanssettet isteden, siden dette i teorien ikke skal bli påvirket av forskjellig lysinnstråling.



Figur 4.27:Viser et utsnitt av mosaikken til flystripene. De røde rektanglene viser treningsområdet Valle Hovin i øvre høyre hjørne og testområdet Tøyen til venstre for treningsområdet. Det nederste rektanglet er visualiseringssettet Gamle Oslo. Det er lagt på to hvite linjer som illustrerer hvilke flystriper som er inkludert i både treningsområdet og visualiseringssettet.

At det var modell 1 som presterte best på treningsområdet, er overraskende siden dette var den modellen som var minst komplisert. Det overrasker siden den presterte dårligst i forhold til de andre klassifiseringene med seks klasser på treningsområdet og visualiseringssettet. Dette resultatet kan tyde på at det ikke er nødvendig med en komplisert modell når det er usette områder som skal klassifiseres. Om det hadde vært mer tid kunne det vært interessant å teste hvordan modell 1 hadde prestert med brede klasser.

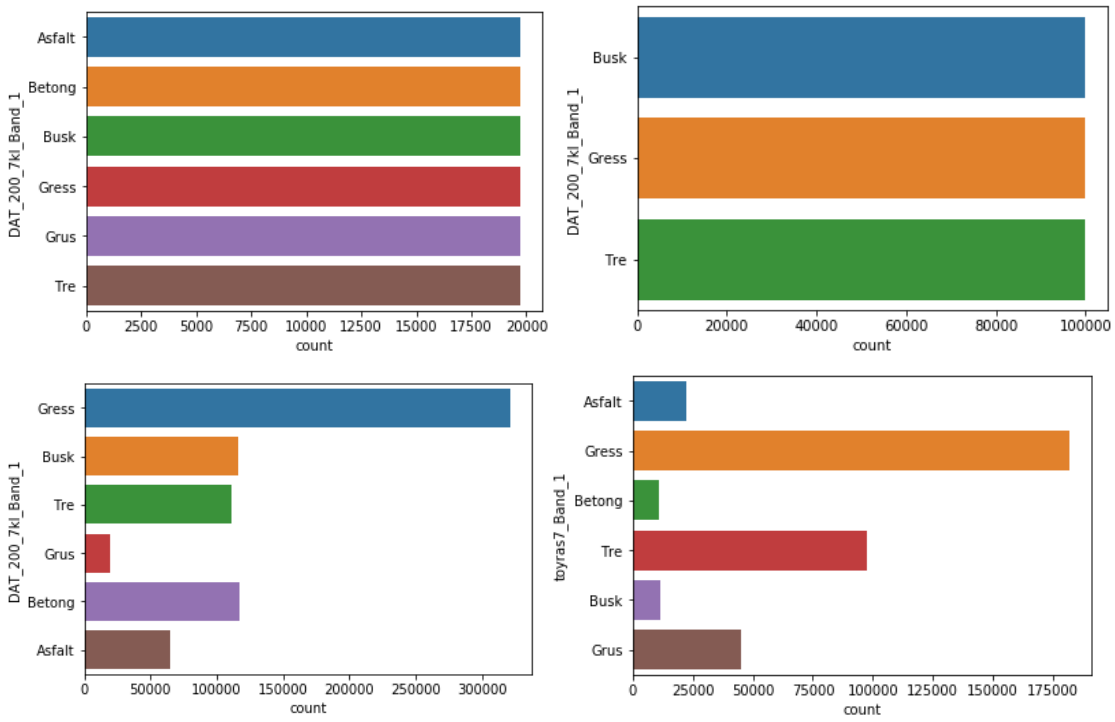
De ulike modellene hadde mest trolig prestert bedre om de hadde blitt trent med data fra flere områder fordelt over hele Oslo-prosjektområdet, og slikt fått data fra alle flystripene. Et av poengene med å bruke nevrale nettverk er å trene modellen flere ganger, etterhvert som en får mer fasitdata mates disse til modellen slik at modellen på sikt kan bli bedre og bedre. Grunnen til at dette ikke ble gjort i denne oppgaven var mangelen på fasitdata, og at dette ville krevd mye mer omfattende feltarbeid. Hadde det vært mer tid tilgjengelig ville dette blitt gjort.

4.2.8 Fire tester på Tøyen med LR og SVM

Fire modeller fra hver av algoritmene SVM og LR ble gitt en validering på Tøyen. Alle ble trent og validert på radiansdataene med alle bånd inkludert.
Disse fire modellene var følgende:

- 6 klasser med balansert treningsdata, trent på 19.700 piksler fra hver klasse
- 6 klasser med ubalansert treningsdata
- 3 klasser grønt med balansert treningsdata, trent på 100.000 piksler fra hver klasse
- 3 klasser steinbasert med balansert treningsdata, trent på 19.700 piksler fra hver klasse

Se figur 4.28 for antall piksler i treningssituasjon og valideringssituasjon.



Figur 4.28 Figur viser pikselmengden i de forskjellige klassifiseringene i kap 4.2.8:Øverst til venstre 6kl ubalansert/3kl steinbasert til trening, øverst til høyre 3 kl grønt balansert til trening, nederst til venstre 6kl ubalansert til trening, nederst til venstre, antall i valideringsfasiten

Det var resultatet fra 6 klasser balansert som egentlig var målet for validering på Tøyen, disse resultatene blir diskutert inngående i avsnitt 4.2.9 De andre ble testet kun for eksperimenteringens skyld, og for å gi svar på hvor bra LR og SVM presterte på det som ble testet i på CNN i avsnitt 4.2.4 og 4.2.5.
Se tabell 4.12 for alle valideringsresultater fra Tøyen og respektive resultater fra testing på Valle Hovin

Tabell 4.12: Resultater for alle valideringer på Tøyen sammmenlignet med testresultater for Valle Hovin

Tøyen LR og SVM			Valle Hovin LR og SVM	
	Accuracy	Kappa	Accuracy	Kappa
LR 6 kl balansert	0.8593	0.7965	0.9875	0.9830
SVM 6 kl balansert	0.7398	0.6482	0.9953	0.9986
LR 6 kl ubalansert	0.8808	0.8261	0.9353	0.9224
SVM 6 kl ubalansert	0.7544	0.6634	0.9681	0.9616
LR 3 kl vegetasjon	0.8857	0.7879	0.9157	0.8735
SVM 3 kl vegetasjon	0.7275	0.5678	0.9528	0.9292
LR 3 kl steinbasert	0.7744	0.5715	0.9925	0.9887
SVM 3 kl steinbasert	0.6473	0.3549	0.9989	0.9983

Gjennomgående kan man se fra tabellen at alle modellene presterer mye dårligere med validering på ukjent område enn testing på Valle Hovin etter trening. Dette tyder på overtilpasning i treningen. Samtidig kan man se at den enkleste metoden LR hele veien har prestert mye bedre enn den mer kompliserte algoritmen SVM. Det viser seg at modellen for LR fra eksperimentet med 6 klasser og ubalansert trening (det vil si at det ble trent på alle tilgjengelige piklser fra Valle Hovin-fasiten), var den som scoret aller høyest. Resultatet fra valideringsparameterne gir en accuracy på 0.88 og kappa på 0.82. Noe som ifølge tabell 2.1 for kappa-koeffisienter tilsier et nesten perfekt resultat. Da det er veldig stor skjevhet i antallet piksler i treningsmateriellet kan det være knyttet stor usikkerhet til dette resultatet. Siden det i denne oppgaven ble fokusert på balansert trening, skal ikke resultatet diskuteres i her. Men det er verdt å nevne at resultatene for denne klassifiseringen speiler de som kommer fram for balansert trening i avsnitt 4.2.9, med tilsvarende prestasjoner som dette resultatet hele veien.

4.2.9 Tøyen 6 klasser balansert LR og SVM

Av LR- og SVM-modellene som ble validert på Tøyen var de som ble trent balansert på Valle Hovin med 6 smale klasser og 19700 piksler fra hver klasse til trening mest interessant. Tabell 4.13 viser hvordan modellene har prestert, vurdert med valideringsparameterne accuracy og kappa. Både testresultatene fra Valle Hovin og valideringsresultatene fra Tøyen er med.

Tabell 4.13: Resultater fra trening på Valle Hovin til høyre og endelig validering på Tøyen til venstre.

Tøyen LR og SVM			Valle Hovin LR og SVM	
	Accuracy	Kappa	Accuracy	Kappa
LR 6 kl balansert	0.8593	0.7965	0.9875	0.9830
SVM 6 kl balansert	0.7398	0.6482	0.9953	0.9986

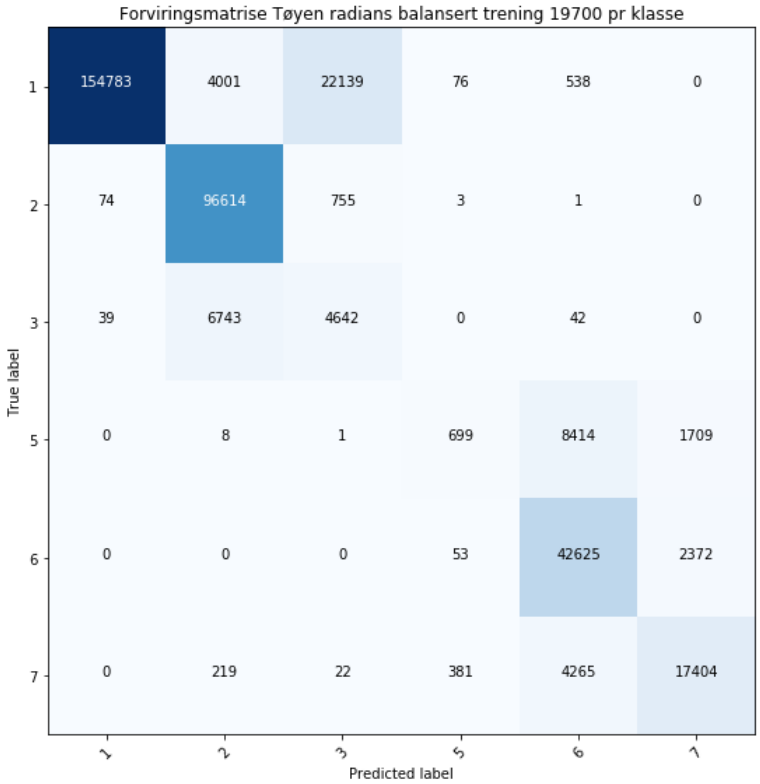
SVM gjorde det best på Valle Hovin med både en accuracy og kappa på 1. LR lå rett bak med accuracy på 0.99 og kappa på 0.98. Disse resultatene var så bra at det mistenkes overtilpasning. Det var derfor svært spennende å se hva algoritmene presenterte på et helt nytt område, spesielt siden vi allerede hadde testet CNN og denne ga til dels dårlige resultater. Det viste seg at LR presterte best på Tøyen, med en accuracy på 0.86 og en kappa 0.80. SVM presterte en del dårligere med accuracy på 0.74 og en kappa på 0.65. Ifølge tabell 2.1 vil kappa-verdien fra SVM tilsi et bra resultat, mens kappa fra LR tilsier en meget god prestasjon. Det viste seg altså at den simpleste metoden var den som håndterte overgangen til et nytt område best.

LR har klart å generalisere klassene, og overføre denne lærdommen til helt nye data. Det er derfor interessant å se hvordan metoden presterte innad i de seks forskjellige klassene, hvilke den klarte bra og hvilke klasser den slet med.

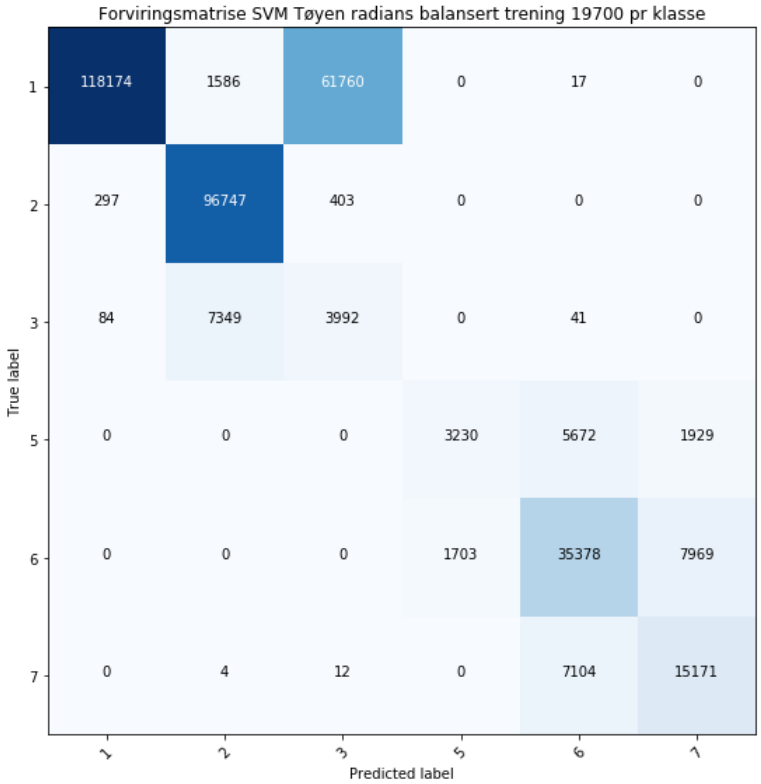
LR klassifiserte elendig på betong og busk, ok på asfalt, passe bra på grus, veldig bra på gress og best på trær. Dette sammenlignes med tilsvarende resultater fra SVM i tabell 4.14 som viser resultater for de individuelle klassene fra Tøyen for begge modellene. Resultatene er presentert med valideringsparameterne precision, recall og F1-score og forvirringsmatriser (se figur 4.29 og 4.30). Support angir hvor mange samples det ble validert på for hver klasse, alle pikslene i Tøyen-fasiten ble brukt i valideringen

Tabell 4.14: Resultater for de forskjellige klassene fra endelig validering på Tøyen: LR til venstre og SVM til høyre.

LR Smale klasser, balansert						SVM Smale klasser, balansert			
Klasse	Precision	Recall	F1-score	Support		Precision	Recall	F1-score	Support
Gress	1.00	0.85	0.92	181537		1.00	0.65	0.79	181537
Tre	0.90	0.99	0.94	97447		0.92	0.99	0.95	97447
Busk	0.17	0.40	0.24	11466		0.06	0.35	0.10	11466
Betong	0.58	0.06	0.12	10831		0.65	0.30	0.41	10831
Grus	0.76	0.95	0.84	45050		0.73	0.79	0.76	45050
Asfalt	0.81	0.78	0.80	22291		0.61	0.68	0.64	22291



Figur 4.29: Forvirringsmatriser for endelig validering av Tøyen med LR. Tallene 1-7 representerer klassene i rekkefølgen gress, tre, busk, betong, grus, asfalt.



Figur 4.30. Forvirringsmatriser for endelig validering av Tøyen med SVM. Tallene 1-7 representerer klassene i rekkefølgen gress, tre, busk, betong, grus, asfalt.

Generelt kan man med en rask titt på forvirringsmatrisene se at begge modellene traff tilnærmet perfekt metodene innad i de to hovedgruppene grønt og steinbasert. Dette speiler resultatet fra testen med trening på kun to klasser, at grønt og steinbasert i utgangspunktet har så forskjellige egenskaper at de ikke blander seg ved feilklassifisering. Det var veldig få samples som ble klassifisert feil ut fra sin hovedgruppe. Dette ser man ved at det klynger seg i de ni feltene oppe i venstre hjørne (vegetasjonsgruppen) og i de ni feltene ned i høyre hjørne på begge forvirringsmatrisene (steinbasert gruppe). Det grønne som ble feilklassifisert havnet innad i vegetasjonsgruppen, tilsvarende med steinbasert.

Klassen tre er den begge metodene presterte best på. De gjorde det tilnærmet likt, men faktisk SVM ørlite grann bedre. LR-modellen hadde en recall på 0.99 og en precision på 0.9. Det betyr at modellen gjenkjente 99% av verdiene i klassen tre, og av disse tilhørte 90% faktisk klassen. Dette gir en f1-score på 0.94 for LR og 0.95 for SVM, altså tilnærmet perfekt. De den bommer på havner, ikke overraskende, hovedsakelig i klassen busk.

Ser man på busk-klassen for LR-modellen gjenkjenner den 40% av verdiene, men bare 17% tilhører faktisk klassen. Dette gir en f1-score på 0.24, noe som må sies å være elendig. SVM gjør det enda dårligere på busk, med en recall på på 35% og precision på bare 6 %. Begge metodene predikerte nesten alle buskpikslene til å være trær.

På klassen gress treffer LR generelt bra, men det er også veldig stor overvekt av gresspiksler i fasiten, ca 181.500 mot 11.500 i busk. LR får en precision på 1, recall på 0.85 og f1-score på 0.92. SVM tilsvarende 1, 0.65 og 0.79, altså en del dårligere. For både LR og SVM havner størstedelen av de feilklassifiserte gresspikslene i busk-klassen. Det kan være verdt å merke seg at selv med bra tall fra valideringsparameterne bommer fortsatt LR på ca 27.000 av 181.500 piksler. Dette tilsvarer et areal på ca 2400m^2 av totalt ca 16000m^2 gress fra fasiten laget rundt Tøyenparken. SVM bommer tilsvarende med hele 63.000 piklser, tilsvarende 5600m^2.

Asfalt-klassen treffer LR ganske bra på, med precision på 0.81, recall på 0.78 og f1-score på 0.8. De fleste den bommer på plasseres i grusklassen isteden. Det interessante er at for SVM, med en f1-score på bare 0.64, som ikke kan sies å være veldig bra, havner alle de feilklassifiserte i grus, og ingen i betong. Det er altså grus og asfalt den blander om hverandre, modellene ser ut til å holde seg litt unna betong. Men når modellene skal klassifisere betong går det ordentlig galt for LR, det er den klassen modellen sliter aller mest med. Precision på 0.58, men recall på lave 0.06, og f1 på 0.12. Dette vil si at modellen kjenner igjen bare 6% av verdiene, og av disse hører kun 58% til i betongklassen. For SVM havner betong nest lavest, foran busk. Valideringsparameterne sier en precision på 0.65, recall på 0.3 og f1-score på 0.41.

4.2.10 Tøyenfasit – erfaringer og problematikk

Da vi var på feltarbeid og ble kjent med områdene, og senere utarbeidet fasiter, tenkte vi hele tiden at klasseinndelingene vi hadde gjort kom til å fungere bra på både Valle Hovin og Tøyen. Det viste seg å nødvendigvis ikke stemme. I ettertid er det lett å se alle feilvalgene som ble tatt, og dette er nyttige erfaringer.

Busk
Noe som etter hvert ble tydelig for Valle Hovin, er at busk-klassen var problematisk. Og den var enda mer problematisk på Tøyen enn på Valle Hovin, der det var store busk-områder å velge av til fasit. Det ble brukt mye tid til å definere busk til lærings-fasiten. Busk på Valle Hovin var mye mer variert og lettere å definere enn busk på Tøyen, der mesteparten nok mest sannsynlig er småtrær og treskudd. Modellene putter tilnærmet alle feilklassifiserte busk i treklassen. Samtidig putter modellene mesteparten av feilklassifisert gress i busk-klassen. Konklusjonen er at busk er en veldig dårlig klasse å bruke til klassifisering

Gress
Spesielt LR har klart å generalisere og treffe bra på gress på Tøyen. SVM presterte litt dårligere. Det kan være verdt å nevne (med tanke på CNN som ser ut til å ikke klare generalisering), at selv om gresslettene på de to områdene kan se ganske like ut, gis gresset på Tøyen årlig hard medfart ved at 80.000 mennesker en uke hver sommer tramper ned gresset i Tøyenparken i forbindelse med Øyafestivalen. Vi kan ikke si om dette har noen innvirkning, men det kan være en mulighet.

Tre
Trefasiten fra Valle Hovin inneholder et stort område plantet furu som man ikke finner lignende av på Tøyen. Om dette hadde noen innvirkning på resultatene vites ikke, men det er kanskje ikke så lurt å innlemme en stor, homogen og spesiell gruppe i en fasit som skal vøre generell.

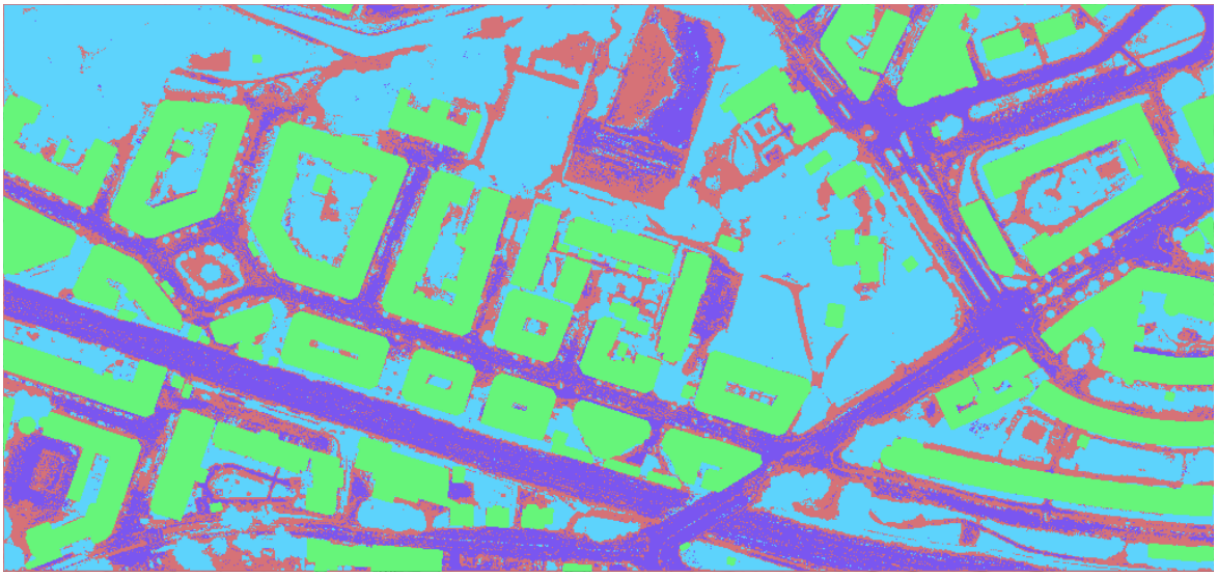
Grus
Etter å ha studert de dårlige Tøyen-resultatene fra CNN-modellene, tok vi en rask inspeksjon for å sjekke området som inneholdt grusen som ble brukt i fasiten på Tøyen. Området er en stor plass kalt Sirkustomta. Den viste seg å ikke være bare grus, men også store deler jord, leire og annet. Området lignet veldig på søleveien fra figur 4.7. Grusfasitene på Tøyen og Valle Hovin er altså basert på forskjellige typer materialer, og grus viste seg også å være en problematisk klasse.

Asfalt
LR- og SVM-modellen er trent kun på Ring 3-asfalt, men for det meste validert på Ring 2-asfalt. Dette kan ikke ha ført til de største problemene når man ser på resultatene, men samtidig er det ikke en heldig situasjon. Et større utvalg asfalt-typer burde vært valgt for trening for å lære opp robuste modeller.

Betong
Betongfasiten på Valle Hovin var for det meste hentet fra en betongoverflate som har funksjon som kunstisbane på vinteren, og parkeringsplass og konsertarena på sommeren. Denne type betong finnes ikke på Tøyen. Det er derfor ikke rart modellene sliter veldig med å klassifisere denne klassen, og dette er en svært uheldig situasjon. Resten av fasiten fra Valle Hovin består av små mengder piklser fra steinheller og støpt stein. Disse er nok ganske lik det som er representert i fasiten for Tøyen.

4.2.11 Raster med klasser for Mannings tall

Som en bonus ble resultatbildet til CNN modell 3 brukt som utgangspunkt for å lage et Mannings tall-raster. Resultatbildet ble overført til ArcGIS og konvertert til et vektordatasett. Der ble det tilegnet Mannings tall etter samme kategorier som figur 3.8 fra avsnitt 3.4.1 Det endelige resultatet kan ses i figur 4.31, dette er ment for å visualisere hvordan et slikt raster kan se ut og for å vise at et slikt raster er mulig å produsere ut fra blant annet klassifisering av hyperspektrale flybilder.



Figur 4.31: Viser et manningsraster, der manningstall er gitt til alle klassene. Dette bildet er laget ved bruk av resultatbildet til modell 3 med smale klasser. Oppløsningen er på 0,3 meter. De unike klassene er tak, asfalt/betong, grus/permiabelt og vegetasjon. Vann vart ikke inkludert, siden det ikke var vann i bildet.

4.3 Sammenligne CNN, LR og SVM

For våre 3 utvalgte algoritmer og klassifisering med trening på 6 balanserte klasser har vi fått følgende resultater, tabell 4.14.

Tabell 4.14: Tabellen inneholder de ulike algoritmenes score i form av accuracy og kappa for endelig validering på Tøyen.

Klassifisering av 6 klasser, Tøyen		
	Accuracy	Kappa
CNN	0.5121	0.3839
SVM	0.7398	0.6482
LR	0.8593	0.7965

En sammenlikning av resultatene fra Tøyen mot maskinlæringsalgoritmene viser at CNN presterte dårlig, SVM bra og LR prestere meget godt, basert på kappa-koeffisienten (tabell 2.1). Dette var meget overraskende, siden det var forventet at en mer avansert modell ville vært mer i stand til å plukke opp mønstrene i dataene.

CNN, den mest avanserte algoritmen som ble testet, presterte dårligst. Det ble trodd at denne ville prestere best, men dette resultatet kan tyde på at det ikke nødvendigvis er slik at en avansert modell gir et fortrinn under klassifisering. Denne påstanden styrkes av at det var CNN-modell 1 som ga de beste resultatene, selv om den scoret dårligere enn de mer avanserte modellene under trening. CNN fungerer ikke på data som ikke har en likhet med treningsdataene som er brukt. Den er derfor svært utsatt dersom det blir innført noe nytt den ikke har trent på innenfor en klasse. CNN er svært flink til å finne mønstre og detaljer, og en veldig komplisert CNN-modell vil derfor kanskje ikke være det mest optimale valget for så vide klasser.

Den minst kompliserte algoritmen, LR, var den som fikk høyest score på valideringsresultatene. Generalisering er antagelig nøkkelen til en bra klassifisering i dette tilfellet, noe det viser seg at LR er bra til. Sannsynligvis fordi den med sigmoid-funksjonen presser en sample mot 0 eller 1, og siden det er et distinkt skille mellom flere av klassene er det relativt enkelt for modellen å gjøre dette. Derfor sliter den da med busk, da dette skillet ikke nødvendigvis finnes, siden flere av samplene fra fasiten teknisk sett er tre eller eng.

SVM derimot prøver å finne den største separeringsmarginen, noe den gjorde ganske enkelt med Valle Hovin-pikslene siden klassene her var mer strengt definert, og det mest sannsynlig oppsto en overtilpasning her. Da den brukte treningen på Tøyen var det bare gress og tre den fikk bra resultater på. Disse er nok de tydeligst definerte klassene i begge fasitene sammenlignet.

Tabell 4.15: Sammenligning mellom det re algoritmene CNN, SVM, LR. Der F1-score er brukt til sammenligningen.

F1-score for 6 klasser alle tre algoritmer			
	CNN	SVM	LR
Klasse	F1-score	F1-score	F1-score
Asfalt	0.23	0.64	0.80
Betong	0.25	0.41	0.12
Busk	0.07	0.10	0.24
Grass	0.61	0.79	0.92
Grus	0.43	0.76	0.84
Tre	0.80	0.95	0.94

Alle tre algoritmene ser ut til å slite med klassene betong og busk. Dette er de samme klassene vi hadde problemer med underveis, og er nevnt tidligere beskrevet utførlig diskusjonen i avsnitt 4.2.3 og 4.2.10. Betong-klassen bestod for det meste av én type betong. Samtidig presterte alle algoritmene best på de samme klassene, altså tre og gress. Dette var de klassene som var lettest å definere.

Hyperspektrale data av det kaliber vi har jobbet med inneholder så mye informasjon at det kan påvirke bruksområdet til forskjellige algoritmer mer enn vi var klar over på forhånd.
Det kan se ut som LR fungerer bra til å filtrere denne informasjonen og finne de relativt generelle klassene vi har vært ute etter. CNN skal være utmerket til å skille unyttig fra nyttig informasjon, og derfor er det veldig rart at den har presterte så dårlig som den har gjort. Det er mulig den har valgt å fokusere på feil bånd og gått seg vill i detaljene, dette siden klassene er for generelle. Det virker som det ble for mye informasjon for CNN, denne algoritmen egner seg kanskje best til å lete etter spesifikke arter eller materialer.

Alt dette understreker det som det ble sagt i avsnitt 3.8.1 angående valg av algoritme:
At ingen klassifiseringsmetode passer best for alle mulige scenarier. Det finnes mange ulike metoder, ingen passer bra til alt, og noen er bedre egnet til spesielle oppgaver enn andre. Det er fordeler og ulemper med alle. Derfor er det viktig å kjenne til et utvalg forskjellige klassifiseringsalgoritmer, teste flere og ikke bare gå for èn.

5 Konklusjon

I denne oppgaven har det blitt jobbet med hyperspektrale flybilder over Oslo, og det har blitt undersøkt hvordan man kan gjøre en pikselbasert arealklassifisering ved bruk av maskinlæring.
Vi har jobbet med to områder, der Valle Hovin ble brukt for trening og testing av modellene, og den endelige valideringen ble gjort på Tøyen, et område helt uavhengig av treningsområdet. For våre 3 utvalgte algoritmer og klassifisering med trening på 6 balanserte klasser har vi fått følgende resultater:

- **LR**
  - Accuracy 0.86
  - Kappa 0.80

- **SVM**
  - Accuracy 0.74
  - Kappa 0.65

- **CNN**
  - Accuracy 0.51
  - Kappa 0.38

Hvis man tar utgangspunkt i tabellen for kappa-koeffisienten (tabell 2.1) kan prestasjonene til modellene tolkes til at LR presterte meget godt, SVM bra og CNN dårlig. Det virker som det er et system i at jo enklere metoden er, jo bedre resultat gir det på generelle klasser.
Datasettene vi har jobbet med har ekstremt mye informasjon, derfor kan det være svært utfordrende å hente ut den informasjonen som er nyttig for å spesifisere en klasse.

Den minst kompliserte metode, LR, viste seg å være den som fikk høyest score i denne oppgaven. For å prestere bra er det nødvendig for modellen å klare og generalisere de ulike klassene. Det kan derfor virke som generalisering var lettest for en enkel modell. LR er ikke flink på detaljer, men god på de store linjene og god til grovklassifisering (Røstad, 2018).

SVM presterte hele veien bedre enn LR på treningen, men heller ikke den har klart å generalisere alle klassene. Den har prestert veldig bra på noen klasser, men også veldig dårlig på andre, og disse trekker resultatet ned. Det den treffer dårligst på er klassene som har vært vanskelig å definere.

CNN, den mest avanserte modellene, er veldig rettet mot å fange små nyanser. Det å lete etter relativt generelle klasser med stor indre variasjon er derfor mer utfordrende. Det kan derfor virke som modellen har gått seg bort i for mye detaljer. Dette kan antyde at det ikke er nødvendig med for kompliserte modeller under arealklassifisering, når dette skal brukes på områder med generelle klasser. Dersom klassene hadde vært smale, og klassifiseringsjobben gikk ut lete etter en spesifikk art eller materiale ville trolig resultatet blitt motsatt.

Konklusjonen etter dette arbeidet er at siden våre resultater ikke er de beste, kan det virke som man bør kombinere pikselbasert klassifisering med andre metoder for å lette arbeidsprosessen og heve det ferdige klassifiseringsresultatet. De endelige resultatene gir inntrykk av at ved arealklassifisering med hyperspektrale bilder presterer den simpleste algoritmen, logistisk regresjon, best.

5.1 Forslag til videre arbeid

- Test andre maskinlæringsalgoritmer på oppsett som er brukt i denne oppgaven.
- Jobbe objektbasert på hyperspektrale data fra urbane områder.
- Benytte datasett som er korrigert for skygge til klassifisering.
- Trene CNN-modeller flere ganger med reflektansdata fra ulike urbane områder.
- Utvide de klassene som er brukt i denne oppgaven til flere underklasser
- Gjøre en sammenligning av radians og reflektans med en generell bakkefasit der det testes på et ukjent område.

Litteraturliste

BARANE, S. 2018. Estimering av biomasse for trær i urbane områder ved å bruke hyperspektrale flyfoto.

BENEDIKTSSON, J. A. & GHAMISI, P. 2015. *Spectral-Spatial Classification of Hyperspectral Remote Sensing Images*, Artech House.

BURGER, W. & BURGE, M. J. 2016. *Digital Image Processing,* London, Springer-Verlag.

CHOLLET, F. O. 2017. *Deep Learning with Python,* Shelter Island, Manning Publications Co.

DALPONTE, M., ØRKA, H. O., GOBAKKEN, T., GIANELLE, D. & NÆSSET, E. 2013. Tree Species Classification in Boreal Forests With Hyperspectral Data. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING,* 51.

DICK, Ø. B. 2009. *vegetasjonsindeks* [Online]. Store norske leksikon. Available: https://snl.no/vegetasjonsindeks [Accessed 01.05.2019].

ESRI 1998. *ESRI Shapefile Technical Description,* Environmental Systems Research Institute, Inc., Environmental Systems Research Institute, Inc.

FERGUS, T., HOSETH, K. A. & SÆTERBØ, E. 2010. *Vassdragshåndboka,* Trondheim, Tapir Akademiske Forlag.

GEONORGE. u.å. *Felles KartdataBase (FKB)* [Online]. Available: https://kartkatalog.geonorge.no/metadata/geovekst/felles-kartdatabase-fkb/0e90ca71-6a02-4036-bd94-f219fe64645f [Accessed 10.05.2019].

GHAMISI, P., MAGGIORI, E., LI, S., SOUZA, R. & TARABALKA, Y. 2018. Frontiers in Spectral-Spatial Classification of Hyperspectral Images. *IEEE geoscience and remote sensing magazine*.

GHAMISI, P., YOKOYA, N., LI, J., LIAO, W., LIU, S., PLAZA, J., RASTI, B. & PLAZA, A. 2017. Advances in Hyperspectral Image and Signal Processing: A Comprehensive Overview of the State of the Art. *IEEE Geoscience and Remote Sensing Magazine,* 5**,** 37-78.

GRAHAM, S. 2000. *The Earth-Sensing Legacy* [Online]. NASA. Available: https://earthobservatory.nasa.gov/features/EO1/eo1_2.php [Accessed 11.04 2019].

HAO, Z. 2017. *Activation Functions in Neural Networks* [Online]. isaacchanghau.github.io. Available: https://isaacchanghau.github.io/post/activation_functions/ [Accessed 02.04.2019].

HARRIS GEOSPATIAL SOLUTIONS. 2018. *Manual Band Selection* [Online]. Available: https://www.harrisgeospatial.com/docs/THORManualBandSelection.html

[Accessed 21.01.2019].

HOWARD, T. E., MENDENHALL, M. J. & PETERSON, G. L. 2009. Abstracting Gis Layers from Hyperspectral Imagery. *2009 First Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing***,** 295-298.

KERAS DOCUMENTATION. u.å. *Keras: The Python Deep Learning library* [Online]. Available: https://keras.io/ - why-this-name-keras [Accessed 05.05 2019].

KUBAT, M. & MATWIN, S. 1997. Addressing the curse of imbalanced training sets: One-sided selection. *ICML,* 97**,** 179-186.

LIED, F. 2018. *fjernmåling* [Online]. Available: https://snl.no/fjernm%C3%A5ling [Accessed 11.04.2019].

MAXWELL, A. E., WARNER, T. A. & FANG, F. 2018. Implementation of machine-learning classification in remote sensing: an applied review. *International Journal of Remote Sensing,* 39**,** 2784-2817.
MICROSOFT. 2009. *How NTFS Works* [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc781134(v=ws.10) - file-naming [Accessed 31/03/2019].
ML CHEATSHEET. 2017. *Gradient Descent* [Online]. ML Cheatsheet: ML Cheatsheet. Available: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html [Accessed 04.04.2019].
ML CHEATSHEET. 2018. *Activation Functions* [Online]. ML Cheatsheet. Available: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html [Accessed 02.04.2019].
MUELLER, A. 2019. *Introduction to ML with python* [Online]. Github. Available: https://github.com/amueller/introduction_to_ml_with_python/ [Accessed 25.04.2019 2019].
NASA. 2013. *The Electromagnetic Spectrum* [Online]. Available: https://imagine.gsfc.nasa.gov/science/toolbox/emspectrum2.html [Accessed 10.04.2019].
NORSK ELEKTRO OPTIKK AS u.å. HySpex Main Spesifications: Hyspex.
P, S. 2018. *[Keras] A thing you should know about Keras if you plan to train a deep learning model on a large dataset* [Online]. Medium. Available: https://medium.com/difference-engine-ai/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2 [Accessed 15.03.2019].
PU, R. 2017. *Hyperspectral remote sensing- fundamentals and practices,* USA, CRC Press.
RASCHKA, S. & MIRJALILI, V. 2017. *Python machine learning,* Birmingham, Packt Publishing.
RICHARDS, J. A. 2013. *Remote Sensing Digital Image Analysis,* New York, Springer.
ROHDE, R. A. 2007. *File:Atmospheric Transmission.png* [Online]. Wikimedia Commons. Available: https://commons.wikimedia.org/wiki/File:Atmospheric_Transmission.png [Accessed 29.04 2019].
RONAN, P. 2007. *File:EM spectrum.svg* [Online]. Wikimedia Commons. Available: https://commons.wikimedia.org/wiki/File:EM_spectrum.svg [Accessed 10.04 2019].
RØSTAD, E. 2018. Klassifisering av trær i urbane områder ved analyser på hyperspektrale flybilder.
SCHIEWE. 2006. *Grundlagen der Fernerkundung* [Online]. florianhillen: florianhillen. Available: http://www.florianhillen.de/studium/projekt/index.php?id=grundlagen&uid=sensoren [Accessed 11.04 2019].
SCIKIT LEARN. 2018. *sklearn.metrics.cohen_kappa_score* [Online]. Scikit learn. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html - sklearn.metrics.cohen_kappa_score [Accessed 01.04.2019].
SHARMA, A., LIU, X. W., YANG, X. J. & SHI, D. 2017. A patch-based convolutional neural network for remote sensing image classification. *Neural Networks,* 95**,** 19-28.
SHIPPERT, P. 2013a. *Digital Number, Radiance, and Reflectance* [Online]. Harris Geospatial Solutions: Harris Geospatial Solutions. Available: https://www.harrisgeospatial.com/Support/Maintenance-

Detail/ArtMID/13350/ArticleID/16278/Digital-Number-Radiance-and-Reflectance?fbclid=IwAR3_eo5m96tEcWZMNx6JmhnWcT9HNLmS2PFflOZryusPGGgGLiEiV4FrsH0 [Accessed 11.02.2019].
SHIPPERT, P. 2013b. *Push Broom and Whisk Broom Sensors* [Online]. Available: https://www.harrisgeospatial.com/Support/Self-Help-Tools/Help-Articles/Help-Articles-Detail/ArtMID/10220/ArticleID/16262/Push-Broom-and-Whisk-Broom-Sensors [Accessed 11.04.2019].
SKARPAAS, O. 2017. URBAN EEA VEGETATION SURVEY SAMPLE Summer 2017.
SMITH, R. B. 2012. *Introduction to Hyperspectral Imaging* [Online]. MicroImages. Available: http://www.microimages.com/documentation/Tutorials/hyprspec.pdf [Accessed 10.04.2019].
SPJELDNÆS, N. 2015. *Sand* [Online]. Store norske leksikon: Store norske leksikon. Available: https://snl.no/sand [Accessed 13.05 2019].
STORE NORSKE LEKSIKON. 2018. *Grus* [Online]. Store norske leksikon: Store norske leksikon. Available: https://snl.no/grus [Accessed 13.05 2019].
SUPERDATASCIENCE TEAM. 2018. *The Ultimate Guide to Convolutional Neural Networks (CNN)* [Online]. SuperDataScience. Available: https://www.superdatascience.com/blogs/the-ultimate-guide-to-convolutional-neural-networks-cnn [Accessed 01.04.2019].
TERRATEC 2017a. Rapport for hyperspektral datainnsamling 40822L HySpex Prosjektnr 8057. Oslo.
TERRATEC 2017b. Rapport for hyperspektral tillegsleveranse HySpex Prosjektnr 8439. Oslo.
THIIS, T. K., GAITANI, N., BURUD, I. & ENGAN, J. A. 2018. Classification of urban blue green structures with aerial measurements. *Dev. Plann. V,* 13**,** 506–515.
THORÉN, A.-K. H. & NYHUUS, S. 1994. *Planlegging av grønnstruktur i byer og tettsteder,* Trondheim, Direktoratet for naturforvaltning.
TRIER, Ø. D., SALBERG, A.-B., KERMIT, M., RUDJORD, Ø., GOBAKKEN, T., NÆSSET, E. & AARSTEN, D. 2018. Tree species classification in Norway from airborne hyperspectral and airborne laser scanning data. *European Journal of Remote Sensing*.
UTSAV B. GEWALI, S. T. M., AND ELI SABER 2018. Machine learning based hyperspectral image analysis: A survey.
VEGVESENET. 2018. *Asfalt* [Online]. Vegvesenet. Available: https://www.vegvesen.no/fag/teknologi/vegteknologi/Vegbyggingsmaterialer/Asfalt [Accessed 28.04.2019].

Vedlegg

Vedlegg A: Python script som ble brukt for CNN

```
__author__ = 'Annette Primstad'
__email__ = 'anpr@nmbu.no'

#=========================================================================
# Importerer moduler
#=========================================================================
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import spectral.io.envi as envi
import pandas as pd
import itertools
import random
import warnings
from sklearn.metrics import accuracy_score, classification_report, \
    confusion_matrix, cohen_kappa_score
from keras.models import Sequential, load_model
from keras.layers import Dense, Conv2D, Flatten, MaxPool2D
from keras_preprocessing.image import ImageDataGenerator

#=========================================================================
# Leser inn data
#=========================================================================
img = envi.open('testomrader/valle_hovin.hdr', 'testomrader/valle_hovin.dat')
img_gt = envi.open('Fasit/fasit_6kl.hdr', 'Fasit/fasit_6kl.dat')
arr = np.array(img.asarray())
arr_gt = np.array(img_gt.asarray())

#=========================================================================
# Deler opp bildene i piksler og generere nye bilder
#=========================================================================
data = []
coor = []
name = []
gt_label = []
for i,j in itertools.product(range(len(arr)), range(len(arr[0]))):
    if sum(arr[i][j]) != 0:
        #if arr_gt[i][j] != 0: # Hopper over nodata
        data.append(arr[i][j].reshape(466, 1))
        coor.append('{},{}'.format(i, j))
        gt_label.append(int(arr_gt[i][j])) # Alle er ukjent
        if i <= 798:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_00\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_00/pix{}_{}.tif'.format(i, j))
        elif i > 798 and i <= 1098:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_01\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_01/pix{}_{}.tif'.format(i, j))
        elif i > 1098 and i <= 1398:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_02\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_02/pix{}_{}.tif'.format(i, j))
        elif i > 1398 and i <= 1698:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_03\\pix{}_{}.tif'.format(i, j))
```

```
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_03/pix{}_{}.tif'.format(i, j))
        elif i > 1698 and i <= 1998:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_04\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_04/pix{}_{}.tif'.format(i, j))
        elif i > 1998 and i <= 2298:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_05\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_05/pix{}_{}.tif'.format(i, j))
        elif i > 2298 and i <= 2598:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_06\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_06/pix{}_{}.tif'.format(i, j))
        elif i > 2598 and i <= 2898:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_07\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_07/pix{}_{}.tif'.format(i, j))
        elif i > 2898 and i <= 3198:
            name.append('C:\\Users\\anpr\\Documents\\Master_arbeidsfiler_2019_C'
                        '\\pix_valle_hovin_08\\pix{}_{}.tif'.format(i, j))
            img = Image.fromarray(arr[i][j].reshape(466, 1), 'L')
            img.save(r'pix_valle_hovin_08/pix{}_{}.tif'.format(i, j))

# ============================================================================
# Lage dataframe med posidjon, filsti og label
# ============================================================================

dat = {'i,j': coor, 'Name': name, 'Label': gt_label}
df_pre = pd.DataFrame(dat, columns=['i,j', 'Name', 'Label'])

# ============================================================================
# Velger ut 19703 piksler fra hver klasse
# ============================================================================

df1 = df_pre.loc[df_pre['Label'] == 1]  # Gras
df2 = df_pre.loc[df_pre['Label'] == 2]  # Tre
df3 = df_pre.loc[df_pre['Label'] == 3]  # Busk
df4 = df_pre.loc[df_pre['Label'] == 4]  # Asfalt
df5 = df_pre.loc[df_pre['Label'] == 5]  # Betong
df6 = df_pre.loc[df_pre['Label'] == 6]  # Grus

random.seed(1234)
df = df1.sample(19703)
df = df.append(df2.sample(19703))
df = df.append(df3.sample(19703))
df = df.append(df4.sample(19703))
df = df.append(df5.sample(19703))
df = df.append(df6.sample(19703))

df = df.reset_index(drop=True)
df = df.reset_index()

# ============================================================================
# Del opp datasettete i treningssett og testsett
# ============================================================================

df_test = df.sample(frac=0.2)
df_train = df.drop(df_test['index'].values)
df_test = df_test.reset_index(drop=True)
df_test = df_test.drop(["index"], axis=1)
df_train = df_train.reset_index(drop=True)
```

```
df_train = df_train.drop(["index"], axis=1)

# ============================================================================
# Lagre data til CSV-filer
# ============================================================================

df_train[['i,j', 'Name', 'Label']].to_csv(r'csv/endelig/df_train_'
                                          r'valle_hovin_6kl.csv')
df_test[['i,j', 'Name', 'Label']].to_csv(r'csv/endelig/df_test_'
                                         r'valle_hovin_6kl.csv')
# ============================================================================
# Les CSV-filer
# ============================================================================

test = pd.read_csv("csv/endelig/df_test_valle_hovin_6kl.csv"
                   ).drop(['Unnamed: 0'], axis=1)
train = pd.read_csv("csv/endelig/df_train_valle_hovin_6kl.csv"
                    ).drop(['Unnamed: 0'], axis=1)

train['Label'][train['Label'] == 1] = "Gress"
train['Label'][train['Label'] == 2] = "Tre"
train['Label'][train['Label'] == 3] = "Busk"
train['Label'][train['Label'] == 4] = "Asfalt"
train['Label'][train['Label'] == 5] = "Betong"
train['Label'][train['Label'] == 6] = "Grus"
train = train.sample(frac=1).reset_index(drop=True)
# ============================================================================
# Lag bildegeneratorer
# ============================================================================

datagen=ImageDataGenerator(rescale=1./255, validation_split=0.3)

train_iter=datagen.flow_from_dataframe(dataframe=train, directory=None,
           target_size= (466, 1), batch_size= 64, x_col="Name", y_col="Label",
           subset="training", class_mode="categorical", color_mode= "grayscale",
           class_indices={'Gress': 0, 'Tre': 1, 'Busk': 2, 'Asfalt': 3,
                          'Betong': 4, 'Grus': 5})

val_iter=datagen.flow_from_dataframe(dataframe=train, directory=None,
         target_size= (466, 1), batch_size= 64, x_col="Name", y_col="Label",
         subset="validation", class_mode="categorical", color_mode= "grayscale",
         class_indices={'Gress': 0, 'Tre': 1, 'Busk': 2, 'Asfalt': 3,
                        'Betong': 4, 'Grus': 5})

# ============================================================================
# Lag modell
# ============================================================================

def model_1():
    model = Sequential()
    model.add(Conv2D(20, 17, 1, input_shape=(466, 1, 1), activation="relu"))
    model.add(MaxPool2D(pool_size=(4,1), strides=1))
    model.add(Flatten())
    model.add(Dense(output_dim=100, activation= "relu"))
    model.add(Dense(activation="softmax", units=6))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()
    return model

# ============================================================================
# Trene og lagre modell
# ============================================================================

model = model_1()
```

```
warnings.filterwarnings("ignore")
history1 = model.fit_generator(train_iter,
                      steps_per_epoch=train_iter.n//64 + 1,
                      epochs=50,
                      validation_data=val_iter,
                      validation_steps= val_iter.n//64 + 1, verbose=1)
model.save('model_6kl_rad_vnir_swir_v1.h5')

# ============================================================================
# Plott valideringskurver for loss og accuracy
# ============================================================================

train_loss = history1.history['loss']
val_loss = history1.history['val_loss']
plt.plot(range(1, len(train_loss) + 1), train_loss)
plt.plot(range(1, len(val_loss) + 1), val_loss)
plt.legend(['Training loss', 'Validation loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

train_acc = history1.history['acc']
val_acc = history1.history['val_acc']
plt.plot(range(1, len(train_acc) + 1), train_acc)
plt.plot(range(1, len(val_acc) + 1), val_acc)
plt.legend(['Training acc', 'Validation acc'])
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.show()

# ============================================================================
# Test en modell
# ============================================================================

model = load_model('model_6kl_rad_vnir_swir_v1.h5')

test['Label'][test['Label'] == 1] = "Gress"
test['Label'][test['Label'] == 2] = "Tre"
test['Label'][test['Label'] == 3] = "Busk"
test['Label'][test['Label'] == 4] = "Asfalt"
test['Label'][test['Label'] == 5] = "Stein/Betong"
test['Label'][test['Label'] == 6] = "Grus"

datagen_test=ImageDataGenerator(rescale=1./255)
test_valle_hovin = datagen_test.flow_from_dataframe(dataframe=test, directory=None,
                     target_size= (466, 1), batch_size= 64, x_col="Name",
                     y_col="Label", class_mode=None, shuffle=False,
                     color_mode= "grayscale", class_indices={'Gress': 0,
                     'Tre': 1, 'Busk': 2, 'Asfalt': 3, 'Stein/Betong': 4,
                                                             'Grus': 5})
y_pred = model.predict_generator(test_valle_hovin,
                                 steps = test_valle_hovin.n//64+1, verbose=1)


filenames = test_valle_hovin.filenames
ids = []
for f in filenames:
    ids.append(f[73:-4])
del filenames

# ============================================================================
# Produser raster med resultat
# ============================================================================

valle_hovin_6kl = np.zeros((2980, 2296))
bilde = [np.argmax(y_pred, axis=-1)+1][0]
```

```
for i in range(len(ids)):
    a, b =(ids[i].split('_'))
    a, b =(int(a), int(b))
    valle_hovin_6kl[a, b] = bilde[i]
valle_hovin_6kl[valle_hovin_6kl == 0] = np.nan

plt.imshow(valle_hovin_6kl, interpolation='none')
plt.imsave('valle_hovin_6kl_radians_VNIR_SWIR_v1.tif', valle_hovin_6kl)
plt.show()

# ============================================================================
# Lag forvirringsmatrise
# ============================================================================

predictions = np.argmax(y_pred, axis=-1)
label_map = train_iter.class_indices
label_map =dict((v,k) for k,v in label_map.items())
predictions = [label_map[k] for k in predictions]

labels = train_iter.class_indices
cnf_matrix = confusion_matrix(test["Label"], predictions)

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


# Compute confusion matrix
class_names = labels.keys()
# Plot non-normalized confusion matrix
plt.figure(figsize=[20,16])
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Forviringsmatrise med all fasitdata (20% av dataene)')
plt.show()

# ============================================================================
```

```
# Beregn validerings parametre
# ==============================================================================

# Accuracy_score
acc_score = accuracy_score(test["Label"], predictions)
print(f'Accuracy score: \n {acc_score} \n')


# Raport
label_report = classification_report(test["Label"], predictions,
                                     target_names=['Gress','Tre','Busk',
                                           'Asfalt','Stein/Betong','Grus'])
print(f'Classification report: \n {label_report} \n')


# Cohens kappa
coh_kappa = cohen_kappa_score(test["Label"], predictions)
print(f'Cohens kappa: \n {coh_kappa} \n')
```

Vedlegg B: Python script som ble brukt for SVM og LR

```
"""
Utarbeidet med utgangspunkt i kode fra faget DAT200 ved NMBU.
Utarbeidet i samarbeid med Erik Røstad
"""

__author__ = 'Åsmund Stemme'
__email__ = 'asst@nmbu.no'

"""
Klassifisering med logistisk regresjon og SVM

"""

#=============================================================================
# Import av nødvendige moduler
#=============================================================================
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import random
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import StandardScaler
import itertools
import pickle
from sklearn.externals import joblib
#=============================================================================
#  Lese inn data fra CSV og behandler
#=============================================================================
df = pd.read_csv('Valle_Hovin_m_fasit.csv',sep=';',header=0, encoding='utf-8',
                 decimal='.')
#gjør om komma til punktum og genererer ny csv
df_b.to_csv('Valle_Hovin_m_fasit_2.csv', sep=';', encoding='utf-8', decimal = '.')
df_c = df.iloc[[0, 1]]
df_d = df_b.iloc[[0, 1]]
df_e = df.fillna(0)
df_f = df_e.iloc[[0, 1]]
# Antall piksler i hver klasse i fasit
antall = df['fasit'].value_counts()
#=============================================================================
#  Velger ut klasser og gir de riktige navn
#=============================================================================
# data = df[df['fasit'] > 0]  # Fjerner alle piksler uten fasit-klasse
data_b = data.iloc[[0, 1]]
data_c = data.replace(4, 3) # EVT bytte om klassetall
data_f = data
antall2_g = data_f.value_counts()
antall2_h = data_f['fasit'].value_counts() # teller opp på nytt

data_f['fasit'][data_f['fasit'] == 1] = "Gress" #Gir klasser navn
data_f['fasit'][data_f['fasit'] == 2] = "Tre"
data_f['fasit'][data_f['fasit'] == 3] = "Busk"
data_f['fasit'][data_f['fasit'] == 5] = "Betong"
data_f['fasit'][data_f['fasit'] == 6] = "Grus"
data_f['fasit'][data_f['fasit'] == 7] = "Asfalt"
```

```
antall3 = data_f['fasit'].value_counts()
#=============================================================================
#  Henter ut piksler til balansert læring
#=============================================================================
random.seed(1234)
def sampling_k_elements(group, k=19700):
    if len(group) < k:
        return group
    return group.sample(k)

balanced =
data_f.groupby('fasit').apply(sampling_k_elements).reset_index(drop=True)
data_g = balanced

antall2_f = data_f['fasit'].value_counts()
sns.countplot(y="fasit", data=data_f)
# printer plot med antall piksler per klassse etter utvelgelse
plt.show()
stat_f = data_g.groupby(["fasit"]).size().reset_index(name="Antall2")
antall2_f = data_g['fasit'].value_counts()
# Plotter fordeling
sns.countplot(y="fasit", data=data_g)
plt.show()
#=============================================================================
#  Skiller ut spektralverdier (X) og respons (y)
#=============================================================================
X = data_g.iloc[:,6:183] # VNIR+SWIR Veger hvilke av 466 bånd som skal med
y = data_g.iloc[:,5] #skiller ut fasit
y_2 = y.iloc[[0, 1]]
y_3 = X.iloc[[0, 1, -1]]
antall2_g = y.value_counts()
#=============================================================================
# Funksjon for plotting av læringskurve
#=============================================================================
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,

    n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure(dpi=100) #forandret fra 300
    plt.title(title)

    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(

    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")

    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,

    color="g")

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
```

```
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")

    return plt
#=============================================================================
# Deler datasettet i treningssett og testsett, fordeling 70 - 30 %
#=============================================================================
for i in range(1):

    X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.3, random_state=5,stratify=y) #random_state 22

    #=============================================================================
    # Skaler med StandardScaler eller MinMaxScaler
    #=============================================================================
    # Initialise standard scaler and compute mean and STD from training data
    sc = StandardScaler()
    sc.fit(X_train)
    #mms = MinMaxScaler() #evt bruke MinMax
    #X_mms = mms.fit_transform(X)
    #Z_mms = mms.fit_transform(Z)
    # Transform (standardisere) both X_train and X_test with mean and STD from
    X_train_std = sc.transform(X_train)
    X_test_std = sc.transform(X_test)

    # logistisk regresjon-algoritmen

    lr = LogisticRegression(C=10) #Sett parameter valgt i gridsearch
    lr.fit(X_train_std, y_train)

    # svm-algoritmen
    sv = svm.SVC(kernel='rbf', gamma=10.0, C=100, ) #sett parametere valgt
    sv.fit(X_train_std,y_train)

#=============================================================================
# Plotter lærings- og valideringskurver
#=============================================================================
    ## læringskurve
#title = "Learning Curves (Logistic regression)"
#estimator = LogisticRegression( C=900)
#plot_learning_curve(estimator, title, X_train, y_train, ylim=(0, 1.01), cv=10)
#title = "Learning Curves (SVM)"
#estimator = svm.SVC(kernel='rbf', gamma=0.2, C=100)
#plot_learning_curve(estimator, title, X_train, y_train, (0, 1.01), cv=10)
#plt.show()
#=============================================================================
# Print resultater LR generert far valideringsparametere
#=============================================================================
print("resultat logistisk regresjon:")
print()
print("Detaljert klassifiseringsrapport:")
print()
y_true, y_pred = y_test, lr.predict(X_test_std)
print(classification_report(y_true, y_pred))
print()
print("F1 micro: %1.4f\n" % f1_score(y_test, y_pred, average='micro'))
print("F1 macro: %1.4f\n" % f1_score(y_test, y_pred, average='macro'))
print("F1 weighted: %1.4f\n" % f1_score(y_test, y_pred, average='weighted'))
print("Accuracy: %1.4f" % (accuracy_score(y_test, y_pred)))
# Regner ut kappa
kappalr = cohen_kappa_score(y_true, y_pred)
print("Kappa lr: %1.6f\n" % kappalr)
# Forvirringsmatrise LR
```

```
cmlr = confusion_matrix(y_test, y_pred)
def plot_confusion_matrix(cmlr, classes,
                          normalize=False,
                          title='Confusion matrix Logistic regression',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    #if normalize:
    #    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    #    print("Normalized confusion matrix")
   # else:
     #   print('Confusion matrix, without normalization')
    #print(cm)
    plt.imshow(cmlr, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cmlr.max() / 2.
    for i, j in itertools.product(range(cmlr.shape[0]), range(cmlr.shape[1])):
        plt.text(j, i, format(cmlr[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cmlr[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Compute confusion matrix
#np.set_printoptions(precision=2)
class_names = ['Gress', 'Tre','Busk','Betong','Grus','Asfalt']#gi rett navn
# Plot non-normalized confusion matrix
plt.figure(figsize=[10,8])
plot_confusion_matrix(cmlr, classes=class_names, # gi rett tittel
                      title='Forviringsmatrise radians VNIR 19700 samples pr
klasse')

plt.show()

#==============================================================================
# Print resultater SVM generert far valideringsparametere
#==============================================================================
print("resultat SVM:")
print()
print("Detaljert klassifiseringsrapport:")
print()

y_truesv, y_predsv = y_test, sv.predict(X_test_std)
print(classification_report(y_truesv, y_predsv))
print()
print("F1 micro: %1.4f\n" % f1_score(y_test, y_pred,
                                     average='micro'))#obs, sjekk at riktig pred
print("F1 macro: %1.4f\n" % f1_score(y_test, y_pred,
                                     average='macro'))#obs, sjekk at riktig pred
print("F1 weighted: %1.4f\n" % f1_score(y_test, y_pred,
                                        average='weighted'))#obs, sjekk at riktig
pred
print("Accuracy: %1.4f" % (accuracy_score(y_test, y_pred))) #obs, sjekk at riktig
pred
# Regner ut kappa
kappasv = cohen_kappa_score(y_truesv, y_predsv)#obs, sjekk at riktig pred
```

```
print("Kappa SVM: %1.6f\n" % kappasv)
# Forvirringsmatrise
cmsv = confusion_matrix(y_test, y_predsv)
#  plotter confusion matrix i seaborn plt.figure(dpi=300)
def plot_confusion_matrix(cmsv, classes,
                          normalize=False,
                          title='Confusion matrix SVM', #gi riktig tittel
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    #if normalize:
    #    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    #    print("Normalized confusion matrix")
   # else:
     #   print('Confusion matrix, without normalization')

    #print(cm)
    plt.imshow(cmsv, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cmsv.max() / 2.
    for i, j in itertools.product(range(cmsv.shape[0]), range(cmsv.shape[1])):
        plt.text(j, i, format(cmsv[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cmsv[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Compute confusion matrix
#np.set_printoptions(precision=2)
class_names = ['Gress', 'Tre','Busk','Betong','Grus','Asfalt'] #gi rett navn

# Plot non-normalized confusion matrix
plt.figure(figsize=[10,8])
plot_confusion_matrix(cmsv, classes=class_names,  # gi rett tittel
                      title='Forviringsmatrise radians VNIR 19700 samples pr klasse')
plt.show()

#==============================================================================
# Lagring av modellen i to formater
#==============================================================================
pickle.dump(lr, open("lr_6kl_rad_VNIR.pkl","wb"))
pickle.dump(sv, open("SVM_6kl_rad_VNIR.pkl","wb"))
#loading a model from a file called model.pkl
#model = pickle.load(open("lr1.pkl","r"))
model = lr
model.fit(X_train_std, y_train)
# Save to file in the current working directory
joblib_file = "lr_6kl_rad_VNIR.pkl"
joblib.dump(model, joblib_file)
# Load from file
#joblib_model = joblib.load(joblib_file)
# Calculate the accuracy and predictions
#score = joblib_model.score(X_test_std, y_test)
#print("Test score: {0:.2f} %".format(100 * score))
#Ypredict = pickle_model.predict(X_test_std)
```

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway