**Masteroppgave 2020 30 stp**
Fakultet for realfag og teknologi

# Predikering av råte i granskog med hyperspektrale flybilder og maskinlæring

Prediction of rot in spruce forest with hyperspectral aerial imagery and machine learning

Kristoffer Hönl Hagen
Geomatikk

# Innholdsfortegnelse

Sammendrag ..... IV
Abstract ..... V
Forord ..... VI
Ordliste ..... VII
1 Bakgrunn og problemstilling ..... 1
1.1 Koronapandemien ..... 1
2 Teori ..... 2
2.1 Granskog ..... 2
2.2 Flybårne sensorer ..... 2
2.3 Maskinlæring ..... 2
2.4 Kunstige nevrale nettverk (ANN) ..... 4
2.5 Ulike typer lag ..... 6
2.6 Konvulerende nevrale nettverk - CNN ..... 8
2.7 Recurrent neural network (RNN) ..... 8
2.8 U-nett ..... 9
2.9 Sammenligning med grunnlinje ..... 9
2.10 Forhåndstrente nettverk ..... 10
2.11 Programvare ..... 10
2.12 Maskinvare ..... 10
3 Data og metode ..... 12
3.1 Datagrunnlag ..... 12
3.2 Områdebeskrivelse ..... 12
3.3 Klargjøring av data og valg av metoder ..... 14
3.4 Felles for begge metoder ..... 14
3.5 Metode 1: Utsnitt av trekroner ..... 15
3.6 Metode 2: Kobling av trepunkter med trekroner ..... 18
3.7 Felles etterbehandling i Python for begge metoder ..... 21
3.8 U-nett ..... 22
3.9 Sensitivitetstest ..... 23
4 Resultat ..... 24
4.1 Grunnlinje ..... 24
4.2 Resultat fra metode 1 ..... 25
4.3 Resultat fra metode 2 ..... 27
4.4 Sensitivitetstest ..... 28
5 Diskusjon ..... 29
5.1 Sensitivitetstest ..... 29
5.2 Augmentering ..... 29

5.3 Mulige feilkilder ... 29
5.4 Feil under datainnsamling ... 34
5.5 Påvirkning fra ulike opptaksforhold og grunnforhold ... 34
5.6 Sammenligning med andre studier ... 34
5.7 Behov for datakraft ... 34
5.8 Videre arbeid ... 34
5.9 Konklusjon ... 35
6 Referanser ... 36
Figurliste ... 38
Tabellliste ... 39
Vedlegg ... 40
Vedlegg 1: Forbehandling metode 2 ... 41
Vedlegg 2: U-nett for metode 2 ... 43
Vedlegg 3: Funksjoner metode 2 ... 49

# Sammendrag

Råte i granskog (picea abies) fører i Norge til store økonomiske tap for skogbruket. Årlig går verdier på rundt 100 millioner tapt. Det er tidligere gjort studier på bruk av maskinlæring for predikering av råte i granskog, men få av disse har fokusert på forbehandling. I denne oppgaven har det derfor blitt undersøkt hvor stor påvirkning valg av forbehandling har på resultatet.

Et annet aspekt som ble undersøkt, var om bærbare datamaskiner kunne brukes til trening av dype nevrale nettverk for predikering av råte. Dette henger sammen med de siste årenes raske utvikling av teknikker for maskinlæring og kapasitet på grafikkort.

For å undersøke påvirkningen fra forbehandling før det nevrale nettverket, ble det laget to ulike metoder med ulike innfallsvinkler som deretter ble trent på et U-nett. Det ble testet med flere ulike hyperparametere.

Resultatene viser at forbehandling kan ha stor påvirkning på resultatet, da det var svært stor forskjell på resultatet fra de to metodene. Metoden med et lite manuelt korrigert datasett kunne avdekke 76 % av piksler merket med råte over 1 m, mens den andre metoden med et vesentlig større automatisert datasett klarte kun 13 %. Årsaken til det svært ulike resultatet er usikkert, men trolig handler det om en rekke feilkilder som hver og en påfører støy i forbehandlingen.

Resultatene viser og at man kan trene dype nevrale nettverk for predikering av råte på en bærbar datamaskin. For en av metodene ble det oppnådd en vektet F1-score på 0,8, noe som var betydelig bedre enn grunnlinjen.

# Abstract

Rot in spruce forests (picea abies) in Norway causes huge losses for forestry. Every year, values of around 100 million NOK are lost. Earlier studies have examined the use of machine learning to predict rot in spruce forests, but few has focused on pre-processing. This thesis has therefore examined how selection of pre-processing contributes to the result.

Another examined aspect was if laptops could be used to train deep neural networks for prediction of rot. This is linked to the rapid development of machine learning techniques and graphics cards.

To examine the influence of pre-processing before the neural network, two different methods for pre-processing, with different approaches, was made and trained in a U-net. The U-net was tested with multiple hyperparameters.

The results show that pre-processing can have a big contribution in the result, as they were a noticeably big difference between the methods. The method with a small manually corrected dataset could discover 76 % of the pixels labelled as rot, but the method with a significantly larger automatic dataset could only discover 13 %. The reason to this huge difference is unclear, but it is probably a combination of different factors.

The results also show that laptops could be used to train deep neural networks for prediction of rot. With one of the methods it was achieved a weighted F1-score at 0.8, which was far better than the baseline.

# Forord

Denne oppgaven markerer slutten på min 5-årige mastergrad i Geomatikk, ved NMBU. En mastergrad som har tatt en spennende vending underveis, grunnet utviklingen inne maskinlæring. Maskinlæring er et fagfelt i rask utvikling. Utviklingen er faktisk så rask at deler av den sentrale programvaren som er brukt i denne oppgaven ikke var utviklet for 5 år siden.

Jeg vil takke min veileder Ivar Maalen-Johansen og biveileder Terje Gobakken for god hjelp og støtte under arbeidet med oppgaven. Jeg vil også takke mine foreldre og medstudenter som har hjulpet meg under arbeidet.

# Ordliste

| Engelsk | Norsk |
|---|---|
| Artificial Neural Network | Kunstige nevrale nettverk |
| Backpropagation | Tilbakeforplantning |
| Bias | Skjevhet |
| Dropout | Frafall |
| Feature hierarchy | Egenskapshieraki |
| Forward propagation | Foroverforplantning |
| Fully connected layer | Helsammenkoblet lag |
| Hidden layer | Skjult lag |
| Loss | Tap |
| Overfitting | Overtilpasning |
| Recurrent neural network | Tilbakevendende nevralt nettverk |
| Sparse-connectivity | Glissenkoblinger |

# 1 Bakgrunn og problemstilling

De siste årene har det vært en rask utvikling av teknikker for maskinlæring og samtidig kapasitet på grafikkort. I samme periode har NMBU, i forbindelse med et forskningsprosjekt om råte i granskog, Precision(NMBU, 2018-2021), samlet inn hyperspektrale flybilder og hogstdata over et skogsområde i Etnedal.

Dette åpnet opp for å undersøke muligheten av å benytte maskinlæring og hyperspektrale flybilder til å oppdage råte i granskog automatisk. Det er tidligere blitt gjort flere studier der man har benyttet maskinlæring og hyperspektrale flybilder for å analysere skog(Fricker et al., 2019), men få av disse har fokusert på forbehandling av datasettet. Forbehandlingen og kvalitetssikringen av datasettet er et sentralt trinn i maskinlæring. I noen tilfeller det viktigste.(Chicco, 2017) «Søppel inn, søppel ut» er et uttrykk som beskriver dette.

For å undersøke påvirkningen fra forbehandling før det nevrale nettverket, ble det laget to ulike metoder med ulike innfallsvinkler som så har blitt trent på et nevralt nettverk. Det ble i tillegg fokusert på å skrive koden slik at den kan gjenbrukes.

En parallell utvikling innen grafikkort for bærbare datamaskiner, åpnet opp muligheten for å ikke lengre være avhengige av å utføre treningen av dype nevrale nettverk på datasentre eller stasjonære datamaskiner. Dette er spesielt interessant med tanke på fremtidig bruk av overføringstrening og sanntidstrening i felt. I forbindelse med tidligere emner ved NMBU har jeg benyttet U-nett(Ronneberger et al., 2015) med gode erfaringer, og dette blir derfor benyttet i denne oppgaven.

I denne oppgaven er det derfor jobbet med følgende problemstilling:

- Hvor stor påvirkning har valg av forbehandling på resultatet?

Samtidig er det jobbet med følgende underordnete problemstilling:

- Kan trening av dype nevrale nettverk for predikeringen av råte utføres på bærbare datamaskiner?

## 1.1 Koronapandemien

Under arbeidet med denne oppgaven rammet Koronaviruspandemien Norge. Dette førte til utfordringer for arbeidet. Blant annet ble masterlesesalen stengt på dagen, uten varsel. Jeg var på lesesalen når den ble stengt og fikk derfor med det viktigste når arbeidet ble flyttet på hjemmekontor. Men arbeidsevnen har blitt redusert blant annet på grunn av at den eksterne skjermen måtte bli igjen på lesesalen. Noe annet som ble igjen på lesesalen var ladekabelen til den trådløse musen, noe som medførte noen mindre problemer.

Koronapandemien medfører også at denne masteroppgaven ikke blir levert på papir.

# 2 Teori

## 2.1 Granskog

Gran er Norges vanligste treslag og det økonomisk viktigste. Et grantre kan bli opptil 50 m høyt og flere hundre år gammelt. Hvert år hogges det omtrent 13 millioner grantrær i Norge.(NIBIO, 2017)

### 2.1.1 Råte i granskog

Råte fører til store økonomiske tap for det norske skogbruket.(Dalen, 2018) Årlig går verdier på rundt 100 millioner tapt.(Dalen, 2018) Gran er særlig utsatt for kjerneråte, noe som danner skader i midten av trestammen.(Ryvarden, 2019) Råten kan spre seg opptil 10 meter opp i stammen og ødelegger cellulosen.(Dalen, 2018) Spredningen av kjerneråte mellom trær foregår i stor grad gjennom sammenkoblede rotnettverk.(Dalen, 2018) Dette fører til at de nærmeste trærne er mer utsatt for råte. Det er derfor sannsynlig at det finnes en klustring av råtne trær.(Gobakken, 2020)

## 2.2 Flybårne sensorer

### 2.2.1 Hyperspektrale bilder

Hyperspektrale bilder er bilder fotografert med sensorer som kan skille mellom svært mange bølgelengder. Dette fører til at man kan analysere forskjeller i spektralsignatur fra ulike objekter.

*Aktuelle sensorer*

For de hyperspektrale bildene i denne oppgaven er det brukt to ulike sensorer.

HySpex SWIR-384 registrerer 288 bånd i spekteret 930-2500 nm.(Hyspex)

HySpex VNIR-1800 registrerer 186 bånd i spekteret 400 – 1000 nm.(Hyspex)

### 2.2.2 Laserskanning og høydemodell

Ved å skanne terrenget med laser, kan man utforme en høydemodell som vil være nødvendig for å ortorektifisere de hyperspektrale bildene og danne trekroner.

Ved hjelp av laserskanning, kan det lages en høydemodell ved å danne en overflate mellom punktene. Som oftest benytter man enten første retur og lager en overflatemodell, eller siste retur og lager en terrengmodell. Man kan også lage en vegetasjon- eller trekronemodell ved å hente ut differansen mellom en overflatemodell og en høydemodell.

*Aktuelle sensorer*

For laserskanningen i denne oppgaven, ble det brukt Leica ALS70-HP laserskanner. (Leica, 2008)

*Trekronesegmentering*

Det er flere metoder for å danne trekroner. En modifisert versjon av Dalponte og Coomes (2016) som er benyttet i denne oppgaven, finner lokale maksima fra en trekronemodell og utvider disse med regiongroing. I tillegg blir hyperspektrale flybilder brukt til å klassifisere treslag.

## 2.3 Maskinlæring

Maskinlæring går i korte trekk ut på at dataprogrammer finner kompliserte mønstre i data ved hjelp av komplekse statistiske metoder for deretter å kunne gjenkjenne ny data.

### 2.3.1 Dyp læring

Dyp læring går i korte trekk ut på å benytte dype kunstige nevrale nettverk til å lære mønstre i treningsdata.

### 2.3.2 Ulike typer maskinlæring

Maskinlæring kan deles inn tre ulike typer:

#### *Styrt læring*
Styrt læring går ut på at man benytter kjente data til å trene opp modellen. Den opptrente modellen kan deretter brukes til å predikere på ukjente data.

#### *Ikke-styrt læring*
Ikke-styrt læring går ut på at modellen på egen hånd oppdager forskjeller og mønstre i datasettet.

#### *Forsterkende læring*
Forsterkende læring skiller seg fra styrt og ikke-styrt ved at det er en belønning/straff. F.eks. en modell som skal trenes opp til å spille et spill, vil få belønning når det vinner og straff når det taper.

### 2.3.3 Trening / validering / testdata
Det opprinnelige datasettet må deles i tre deler. En del som modellene skal trenes på, en del som brukes til fortløpende validering og en del som brukes for å teste den ferdig opptrente modellen.

#### *Valideringsdata*
For å ha kontroll på om modellen gir fornuftige resultater på data som den ikke trener på, benytter man valideringsdata som evalueres jevnlig. Ofte setter man opp treningen, slik at man evaluerer både treningsdataene og valideringsdataen etter hver runde. Nøyaktigheten etter hver runde plottes da som to grafer i et diagram som gir en indikasjon på hvordan modellen utvikler seg. Dette er en svært effektiv metode for å avsløre når modellen overtilpasser.

Det er viktig å nevne at valideringsdata ikke er ukjente data for modellen. Hver gang man bruker valideringsdataene til å justere modellen, vil informasjon i valideringsdataene påvirke modellen. Dette gjelder f.eks. når man bruker modellens nøyaktighet på valideringsdata til å velge mellom ulike oppsett av modellen.

#### *Testdata*
For å undersøke hvordan en modell fungerer på nye og usette data, må en del av datasettet holdes adskilt fra treningen. Testdataene evalueres først når modellen er ferdig, og nøyaktigheten skal tilsvare hvordan modellen håndterer ny data.

Det er viktig at det for det meste unngås å teste modellen på testdata. Jo flere ganger man bruker testdata når man justerer modellen, dess mer informasjon vil «lekke» fra testdata til modellen. Valideringsdata skal brukes til dette formålet. En tommelfingerregel er at man maksimalt kan teste et par ganger før testdataene ikke lengre tilsvarer nye og usette data. Nøyaktigheten fra evaluering av testdata vil dermed ikke gi et korrekt bilde på hvordan modellen håndterer nye data.

#### *Oppdeling av data*
For å dele opp datasettet i et trenings- og testsett, må man avgjøre hvor stor del av dataene som skal inngå i treningen. Det er en rekke faktorer som må tas i betraktning. Et større testsett medfører at modellen vil ha mindre treningsdata og derfor få dårligere evne til å fange opp mønstre i dataene. Men et mindre testsett legger stor vekt på noen få verdier, og dermed et dårlig bilde av virkeligheten. Normalt varierer fordelingen mellom test/trening fra 40:60 til 1:99.(Raschka & Mirjalili, 2017)

Det er hovedsakelig to ulike metoder for å splitte opp et geografisk datasett. Enten kan man holde områdene adskilt eller hente ut deler fra hele datasettet. Ved sistnevnte metode er det spesielt viktig å unngå at treningsdataene overlapper med testdataene.

*Treningsdata*

Treningsdataene er den delen av datasettet som modellen ser og lærer av. Modellen kjører treningsdataene gjennom og beregner nye gradienter og vekter i hver runde med trening. Dette fører til at modellen etter hver runde som regel blir bedre og bedre tilpasset trengingsdataene.

*Augmentering av treningsdata*

Ved å påføre tilfeldige endringer som bl.a. strekk og rotasjoner på deler av treningsdata vil man i mange tilfeller gjøre det nevrale nettverket mer robust, noe som kan gjøre det bedre egnet til å predikere nye data. Dette utføres ved å påføre på tilfeldige bildeutsnitt. Denne prosessen kalles augmentering.

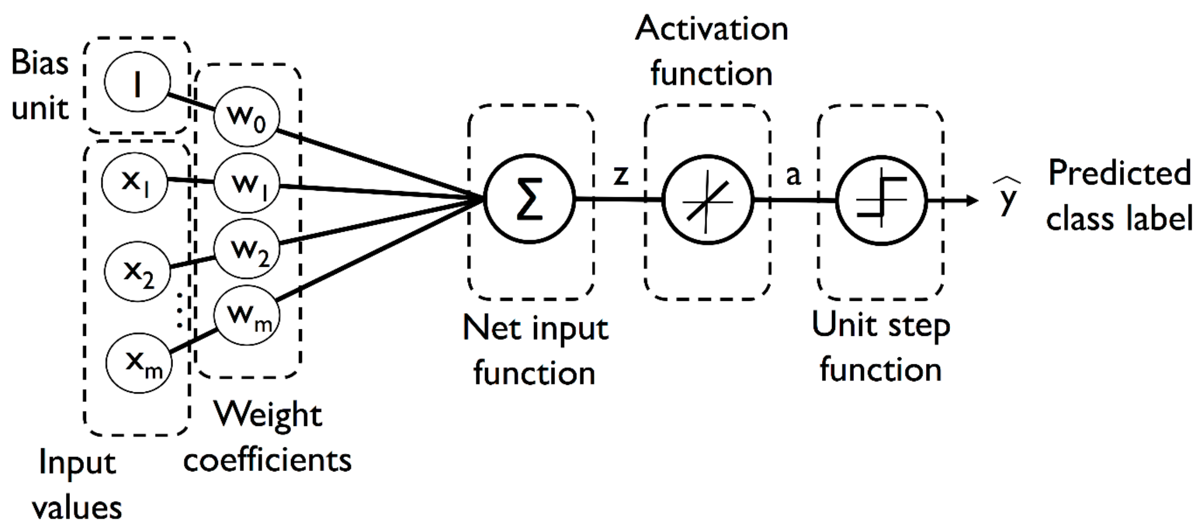
Augmentering kan også brukes til å utvide treningsdatasettet ved å mate inn det samme bildeutsnittet flere ganger med ulik augmentering.

## 2.4 Kunstige nevrale nettverk (ANN)

Et kunstig nevralt nettverk består av et eller flere lag med kunstige nevroner. Disse nettverkene er inspirert av hvordan biologiske nevrale nettverk fungerer.

### 2.4.1 Kunstig nevron

Et kunstig nevron er en matematisk funksjon som tar en avgjørelse basert på en eller flere vektede kilder, en skjevhet og en aktiveringsfunksjon. Vektene og skjevheten (eng: bias) er den delen av modellen som trenes. Figur 1 viser oppbyggingen av et kunstig nevron.




*Figur 1: Oppbyggingen av et kunstig nevron i et nettverk med klassifisering. Merk at Unit step function ikke er relevant her. Figur: Raschka og Mirjalili (2017)*

Det finnes en rekke ulike aktiveringsfunksjoner, men følgende er spesielt aktuelle for denne oppgaven:

#### *2.4.1.1 Sigmoid*

Sigmoid er en forkortelse for logistisk sigmoid funksjon og gir et resultat mellom 0 og 1. Sigmoid brukes ofte for å predikere et kontinuerlig resultat ved hjelp av en skalering.

#### *2.4.1.2 ReLU*

Avkortet lineær enhet (ReLU) sender kun positive resultater videre og setter et negativt resultat til 0. Ved å bryte kurven, blir funksjonen ikke-lineær, noe som gjør nettverket i stand til å oppdage ikke-lineære egenskaper.

### 2.4.2 Lag

Et lag i et kunstig nevralt nettverk, består av et eller flere kunstige nevroner som jobber sammen for å bearbeide hele dataområdet. Ofte setter man flere lag etter hverandre som da mater resultatet fra det ene til det andre. Størrelsen og utformingen av lagene kan variere.

Det er tre typer lag klassifisert etter plassering:

#### *2.4.2.1 Inndatalag*

Inndatalaget mottar ekstern data fra kilden og sender det videre. De kunstige nevronene i inndatalaget kan omtales som passive da de kun sender data videre.

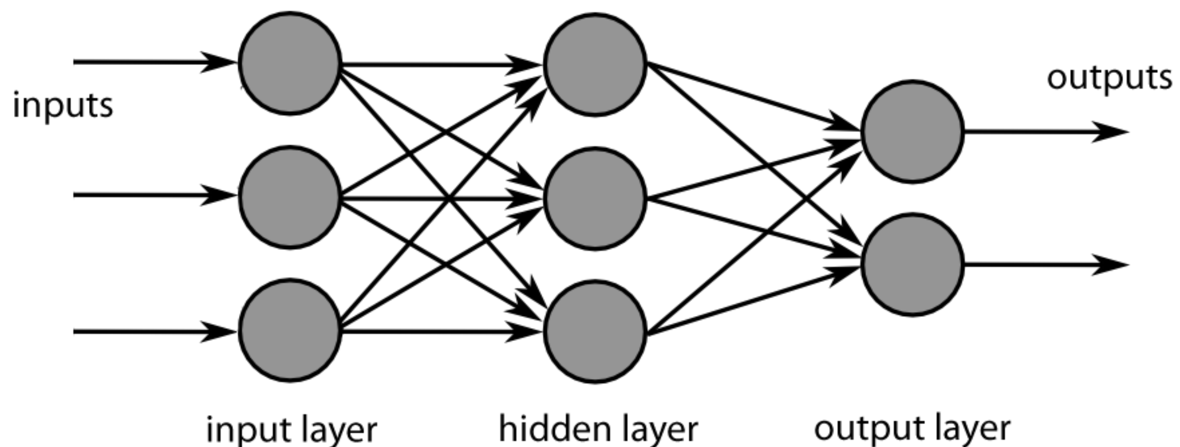
#### *2.4.2.2 Utdatalag*

Utdatalaget er det siste laget i nettverket og gjør resultatet klart. Formen på dette avgjør hvordan dataene kommer ut.

#### *2.4.2.3 Skjulte lag*

Skjulte lag er et eller flere lag som er plassert mellom inndata- og utdatalag. Til forskjell fra inndata- og utdatalag, kan man ikke se innholdet i skjulte lag. Nettverk som inneholder flere skjulte lag omtales som dype nettverk.

Figur 2 viser lagenes plassering i det nevrale nettverket.




*Figur 2: Strukturen på det nevrale nettverket.*
*Figur: MultiLayerNeuralNetwork_english.png: Chrislb derivative work: — HELLKNOWZ ▎TALK ▎enWP TALK (https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetworkBigger_english.png), „MultiLayerNeuralNetworkBigger english“, https://creativecommons.org/licenses/by-sa/3.0/legalcode*

### 2.4.3 Optimalisering og tap

Optimaliseringsfunksjonen er de delene av modellen som oppdaterer vektene og skjevheten. Det finnes en rekke optimaliseringsfunksjoner og mange er innebygd i Keras.

Optimaliseringsfunksjonen vil som regel prøve å redusere en verdi til et minimum, som beregnes med en tapsfunksjon.

### 2.4.4 Batchstørrelse

Dersom datasettet er for stort til å lastes inn i datamaskinens hurtigminne, må det deles opp i mindre grupper, batcher. Hver batch kjøres gjennom nettverket og trener på disse. Batchstørrelsen må reguleres etter mengde hurtigminne tilgjengelig og datasettets form.

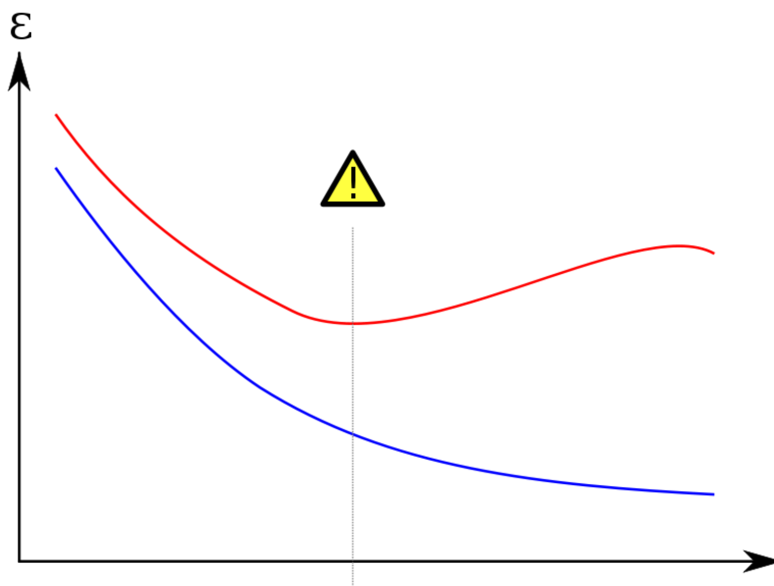
For å unngå at det blir for stor påvirkning fra de første batchene som mates inn i nettverket, brukes batchnormalisering.(Ioffe & Szegedy, 2015)

### 2.4.5 Læringskurver

Ved å bruke valideringskurver, kan man få visualisert hvordan modellens nøyaktighet utvikler seg for hver runde. Dette er blant annet svært nyttig for å finne ut når overtilpasning inntreffer.(Raschka & Mirjalili, 2017)

### 2.4.6 Overtilpasning

En stor utfordring ved maskinlæring, er at modellene etter gjentatte epoker begynner å trene seg opp til å gjenkjenne treningsdataene fremfor de underliggende egenskapene. Dette fenomenet kalles overtilpasning. Dette er illustrert i Figur 3.




*Figur 3: Illustrasjon av læringskurver, med punktet der overtilpasning inntreffer markert. Rød kurve viser utviklingen av feilen for valideringsdata over flere epoker. Etter at overtilpasning inntreffer, begynner stiger feilen for valideringsdata, men feilen for treningsdata fortsetter å synke.*
*Figur: Gringer (https://commons.wikimedia.org/wiki/File:Overfitting_svg.svg), „Overfitting svg“, https://creativecommons.org/licenses/by/3.0/legalcode*

### 2.4.7 Frafall

Ved å ignorere en tilfeldig del av nevronene under trening, vil man trene nettverket til å bli mer robust, og dermed forhindre overtilpasning.(Srivastava et al., 2014) Dette kalles frafall (eng. dropout)

## 2.5 Ulike typer lag

### 2.5.1 Helsammenkoblede lag

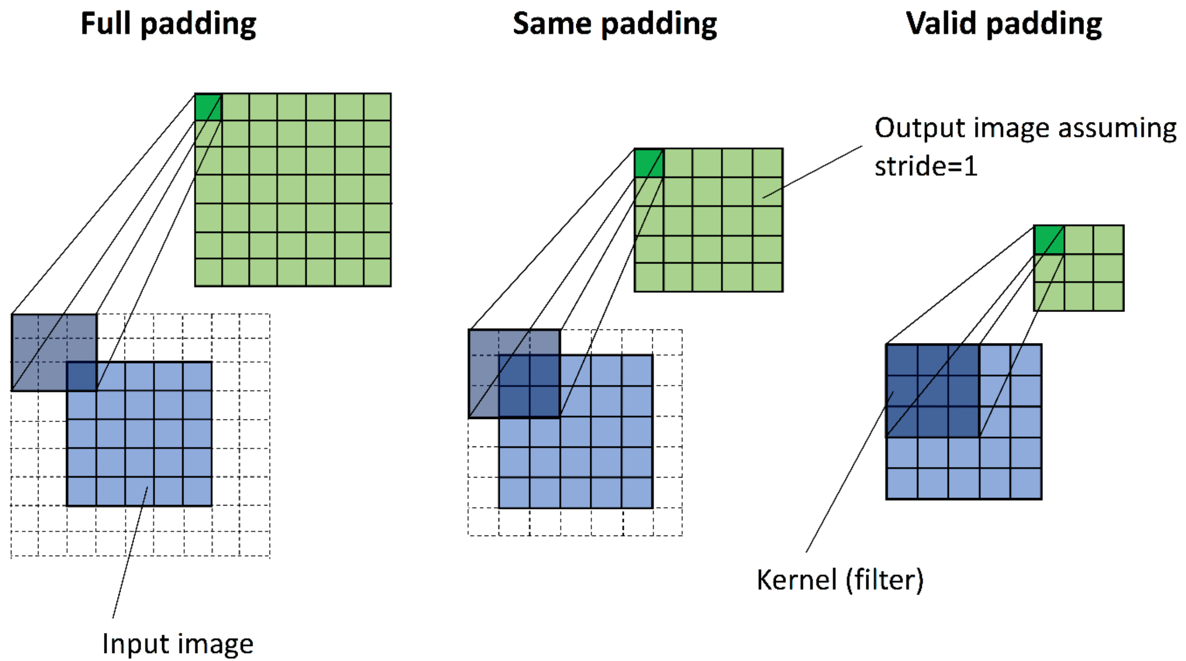
I et helsammenkoblet lag vil hvert kunstige nevron benytte hele det forrige laget som inndata. Dette fører til svært mange koblinger.

### 2.5.2 Konvulerende lag

I et konvulerende lag fungerer hvert nevron som et konvulsjonsfilter, som behandler hvert sitt område av bildet. Dette fører til at man lager et nytt bilde ut av laget som er et resultat av alle nevronene. Vektene i filteret er like for hele laget. Se omtale av konvulerende nettverk i kapittel 2.6.

#### 2.5.2.1 Padding

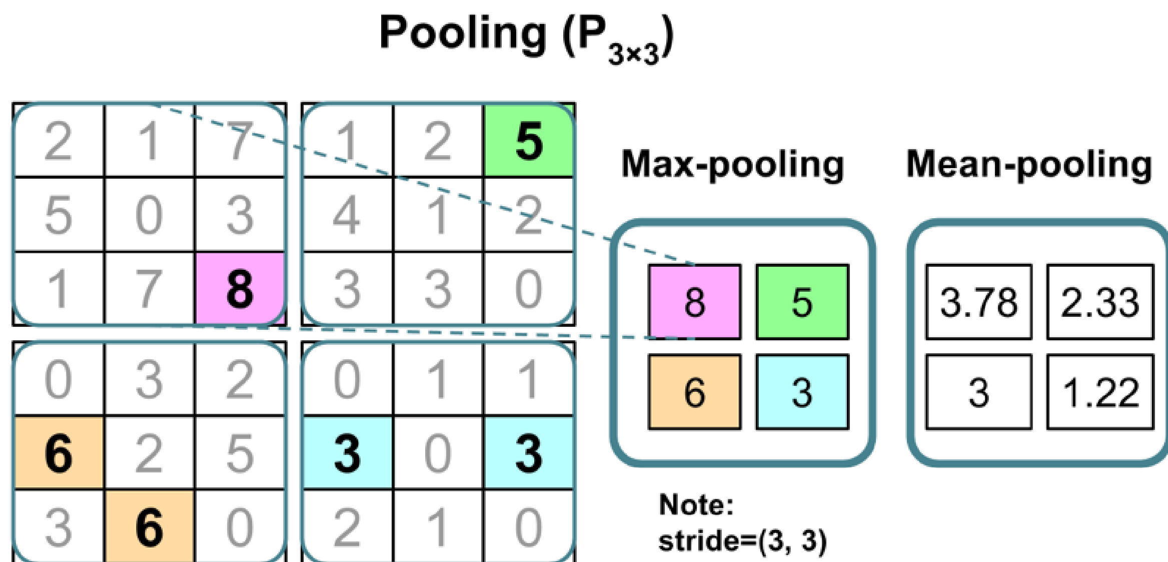
Et problem med konvulerende lag er kanteffekter. Dette løses ofte ved å la filtrene gå utenfor kanten. Det finnes flere ulike metoder, vist i Figur 4. Maskinlæring er ikke spesielt sårbar for kanteffekter, og man benytter derfor same padding som beholder dimensjonen inn og ut av laget.




*Figur 4: Ulike typer padding.*
*Figur: Raschka og Mirjalili (2017)*

### 2.5.3 Poolling lag

I et pooling lag henter hvert nevron en verdi ut av en matrise, som regel uten overlapp. Dette betyr at dataenes størrelse blir redusert. F.eks. reduserer en 2x2 pooling datastørrelsen med 75 %. Det er flere ulike metoder for å hente ut verdien. F.eks. Max-pooling henter ut den største og mean-pooling gir den gjennomsnittlige verdien. Max- og mean-pooling er illustrert i Figur 5.




*Figur 5: Illustrasjon av max- og mean-polling i størrelsen 3x3. Stride 3x3 betyr at filteret beveger seg i steg på 3 piksler.*
*Figur: Raschka og Mirjalili (2017)*

### 2.5.4 Upsampling lag

Upsampling lag ekspanderer dataene ved å fylle inn nye verdier mellom verdiene man har. De nye verdiene kan være null-data, gjennomsnitt eller komme fra andre kilder.

## 2.6 Konvulerende nevrale nettverk - CNN

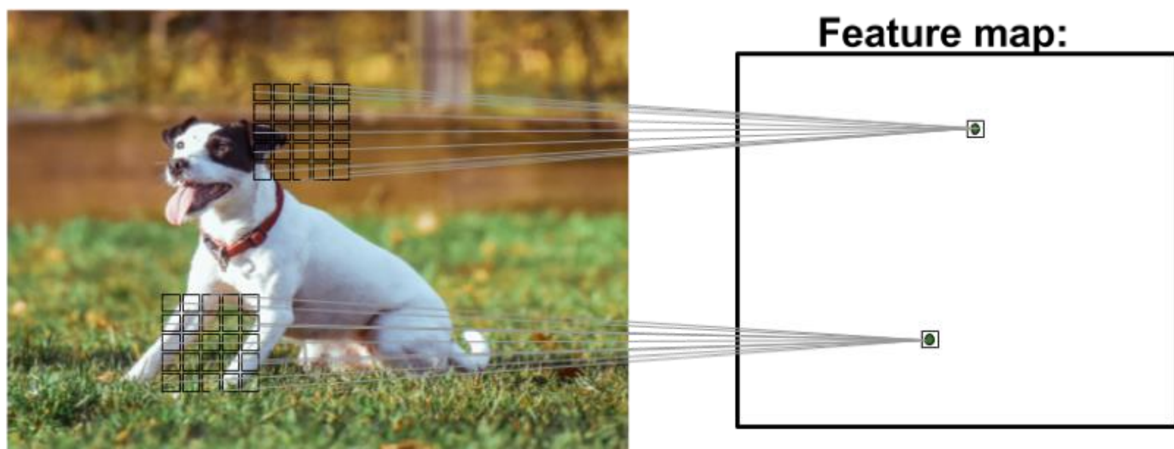
Konvulerende nevrale nettverk brukes for bildeklassifisering i maskinlæring.

CNN trekker i de fleste tilfeller ut relevante egenskaper/elementer fra bildet ved først å trekke ut lavnivåegenskaper i de første lagene for så å finne høynivåegenskaper ved å kombinere flere lavnivåegenskaper. Dette omtales som et egenskapshieraki.(Raschka & Mirjalili, 2017)

Det er spesielt to virkemåter som gjør CNN egnet for bildeklassifisering.

### 2.6.1 Glissenkoblinger

Som omtalt i kapittel 2.5.2 tar hvert nevron i et konvulerende nevralt nettverk utgangspunkt i en liten del av bildet for å lage et nytt bilde. Det nye bildet kalles et egenskapskart. Dette er illustrert i Figur 6. Metoden der man kun bruker en liten del av bildet, kalles glissenkoblinger (eng: Sparse-connectivity). Glissenkoblinger skiller CNN fra helsammenkoblede lag, der hvert nevron i stedet tar utgangspunkt i hele bildet.




*Figur 6: Glissenkoblinger. Filteret passerer over bildet og danner et nytt egenskapskart. Figur: Raschka og Mirjalili (2017)*

### 2.6.2 Fellesparametere

Vektene er like for hele bildet noe som medfører at det tar vesentlig mindre datakraft for å trene nettverket.

## 2.7 Recurrent neural network (RNN)

Sentrale elementer ved RNN er foroverforplantning og bakoverforplantning.

### 2.7.1 Foroverforplantning

Når data mates inn i nettverket, vil det passere gjennom nettverket fra start til slutt.

### 2.7.2 Bakoverforplantning

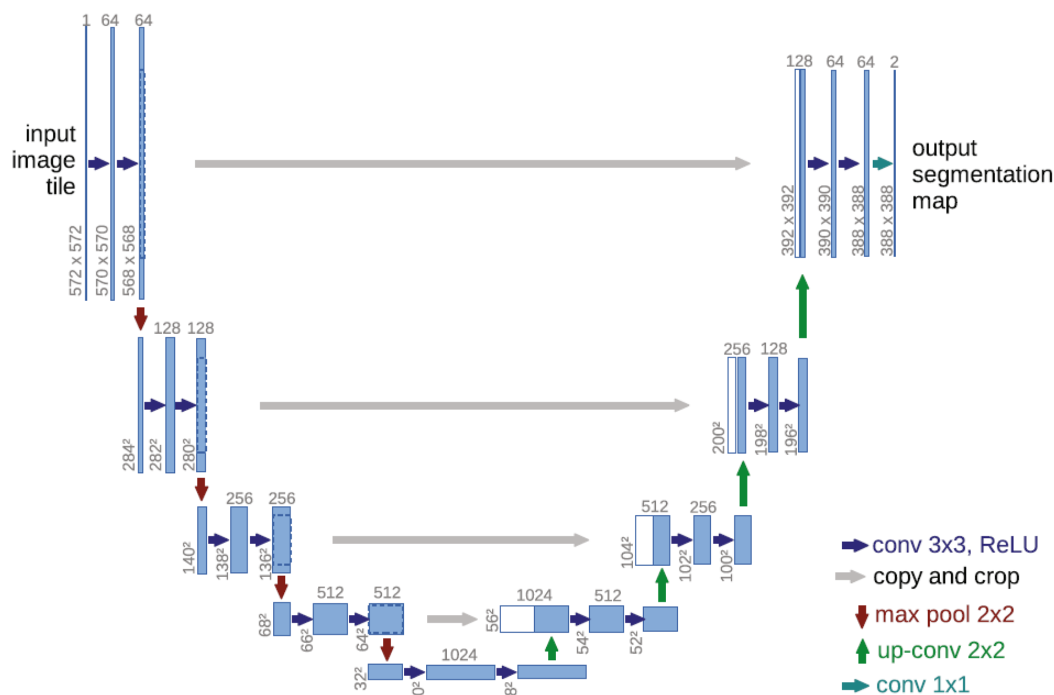
Etter at resultatet er beregnet, utføres bakoverforplantning som med utgangspunkt i tapsfunksjonen beregner gradient for hver vekt, med utgangspunkt i avvik. Dette gjøres fra høyre til venstre. Deretter oppdateres hver vekt av optimaliseringsfunksjonen.

## 2.8 U-nett

U-nett er en spesiell type konvulerende nevrale nettverk som har vist seg godt egnet til å trekke ut egenskaper mer effektivt enn tidligere nettverk.(Ronneberger et al., 2015)

U-nett er todelt. Først er det en sammentrekkende del som gjør det mulig å hente ut høynivåegenskaper. Den andre delen er ekspanderende, og lager et egenskapskart. Det som er spesielt ved U-nett, er at det overføres informasjon (kontekst) fra sammentrekningen til ekspansjonen. Dette fører til at egenskapskartet kan lages med et betydelig mindre antall koblinger i nettverket for å gi samme nøyaktighet på resultatet. Færre koblinger fører også til at behovet for treningsdata og datakraft som er nødvendig for å trene modellen, blir betydelig redusert. Sammentrekningen utføres av pooling lag, mens ekspansjonen utføres ved upsampling lag.(Ronneberger et al., 2015)

U-Nett krever at inndata passer inn i en 2x2 matrise for noe som fører til at det må være kvadratisk og toerpotens på hver side. Strukturen til U-nett er vist i Figur 7.




*Figur 7: Strukturen til U-nett.*
*Figur: Ronneberger et al. (2015)*

## 2.9 Sammenligning med grunnlinje

Grunnlinjen er det resultatet man oppnår ved en veldig enkel beregning. Ved en binær kategorisering vil man, ved et balansert datasett, enkelt oppnå 50 % nøyaktighet ved å kategorisere dataene tilfeldig i hver kategori. Ved et ubalansert datasett vil man ved å kategorisere alle observasjoner i en kategori, kunne oppnå den samme nøyaktigheten som den største gruppen. F.eks. vil et datasett med 100 punkt som består av to kategorier med fordeling 80/20 kunne oppnå en nøyaktighet på 80 % bare ved å klassifisere alle punkt som den første kategorien.

## 2.10 Forhåndstrente nettverk

Første gang man trener et nevralt nettverk, er vektene som regel satt tilfeldig, men i et ferdig trent nettverk er vektene optimalisert. Da vektene i et nevralt nettverk kan overføres, vil man kunne benytte et forhåndstrent nevralt nettverk, selv om man mangler et datasett å trene på.

Det finnes mange forhåndstrente nettverk med ulike arkitekturer og trent på ulike datasett. Et datasett som ofte brukes er ImageNet(Deng et al., 2009) I dag er det nesten utelukkende forhåndstrente nettverk trent på tekst eller vanlige fargebilder som er tilgjengelig. Det er ikke funnet forhåndstrente nettverk for hyperspektrale data som er relevante for denne oppgaven, men det er fullt mulig å lage slike nettverk.(Masarczyk et al., 2019)

### 2.10.1 Overføringstrening

Overføringstrening går ut på at man trener et forhåndstrent nettverk videre. Fordelen her er fremfor å begynne med tilfeldige vekter, er at nettverket allerede er optimalisert for å oppdage noe. Dette fører til at nettverket lettere kan tilpasse seg treningsdataene.

Som nevnt i 2.10 er det ikke noen relevante forhåndstrente nettverk tilgjengelig, men overføringstrening på dype nevrale nettverk med hyperspektrale data har blitt demonstrert av en polsk forskningsgruppe.(Masarczyk et al., 2019)

Gitt den raske utviklingen på dette feltet, vil overføringstrening for hyperspektrale data trolig være tilgjengelig i løpet av få år.

## 2.11 Programvare

### 2.11.1 QGIS

QGIS er et gratis og fritt GIS-program som kan utføre en rekke ulike operasjoner. QGIS utvikles av QGIS Development Team

### 2.11.2 Python

Python er et gratis og fritt programmeringsspråk som er svært mye brukt. Innen maskinlæring er det det mest brukte programmeringsspråket.(Voskoglou, 2017) Python utvikles av Python Software Foundation.

Det finnes en rekke programtillegg til Python. Vanlige pakker er Numpy, matplotlib og pandas. I denne oppgaven er bl.a. de nevnt over brukt.

#### *Scikit-learn*

Scikit-learn er et gratis og fritt programbibliotek for maskinlæring i Python.(Pedregosa et al., 2012)

#### *Xarray*

Xarray er et gratis og fritt programbibliotek for å håndtere geografiske flerbåndsbildedata i Python.(Hoyer & Hamman, 2017)

### 2.11.3 Tensorflow/Keras

Tensorflow er et gratis og fritt programbibliotek for en rekke bruksområder, blant annet nevrale nettverk. Siden 2017 er Keras innlemmet i Tensorflow. Keras er et grensesnitt som kan kjøre på flere ulike programbiblioteker, ikke bare Tensorflow. Tensorflow utvikles av Google og Keras av ulike utviklere.

## 2.12 Maskinvare

Bærbar datamaskin som ble benyttet var en ASUS ZenBook Pro 15 UX580GE
Prosessor/CPU: Intel® Core™ i7-8750H

Grafikkort/GPU: NVIDIA® GeForce® GTX 1050 Ti
Grafikkminne: 4GB GDDR5 VRAM
Hurtigminne: 16GB 2400MHz DDR4

Datamaskinen ble innkjøpt i desember 2018 og kostet ca. 18.000 kr, inkl. mva.
På det tidspunktet denne oppgaven ble skrevet, er det enda bedre løsninger tilgjengelig.

Det ble benyttet en ekstern harddisk på 2 TB for å lagre data underveis.

Det var planlagt å benytte ekstern GPU for å undersøke muligheter for å trene et nevralt nettverk via en internettforbindelse, men grunnet Koronapandemien og forsinket leveranse av maskinvare til NMBU måtte dette utgå.

# 3 Data og metode

## 3.1 Datagrunnlag

### 3.1.1 Hyperspektrale data, laserdata og terrengmodell

I august 2019 utførte Terratec på oppdrag av forskningsprosjektet Precision (NMBU, 2018-2021) flyfotografering med hyperspektrale sensorer og laserskanning, med etterfølgende arbeid.

Vedlagt er rapportene fra datainnsamlingen av de hyperspektrale bildene, laserskanningen og laging av terrengmodellen. Laserskann og terrengmodell ble ikke direkte brukt i denne oppgaven, men dannet grunnlag for trekronesegmenteringen.

### 3.1.2 Trekroner

Trekroner var utarbeidet med en modifisert versjon av Dalponte og Coomes (2016). Se nærmere omtale i kapittel 2.2.2.

### 3.1.3 Hogstdata

Etter at området ble laserskannet, ble området hogd med hogstmaskin utstyrt med GNSS og registrering av vinkel på arm. Dette ga en nøyaktighet på 1 m for posisjonering av hvert hogde tre. Råteverdiene for hvert tre ble så tastet inn manuelt av føreren av hogstmaskinen.(Gobakken, 2020)

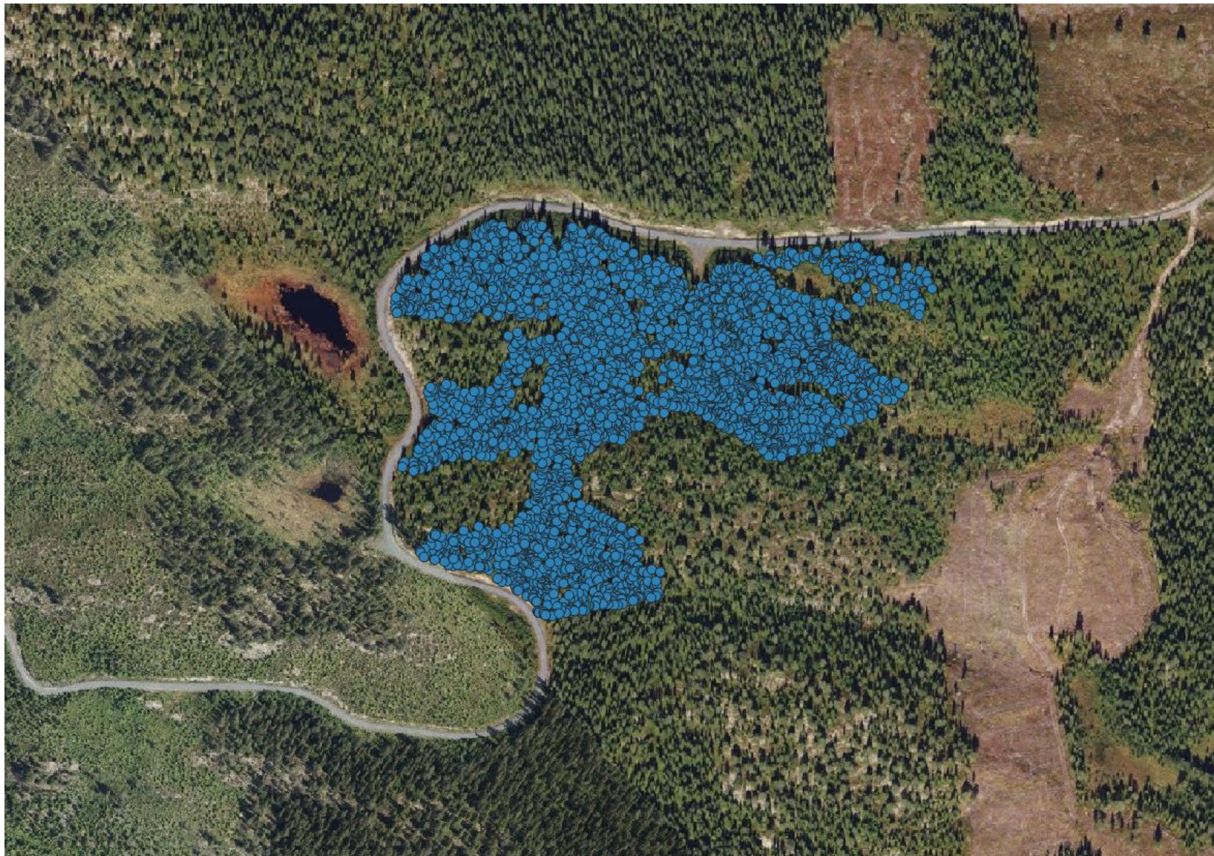
### 3.1.4 Datamottak

Dataene fra hogstmaskinen, hyperspektrale bilder og de segmenterte trekronene ble levert på ekstern harddisk og importert til QGIS.

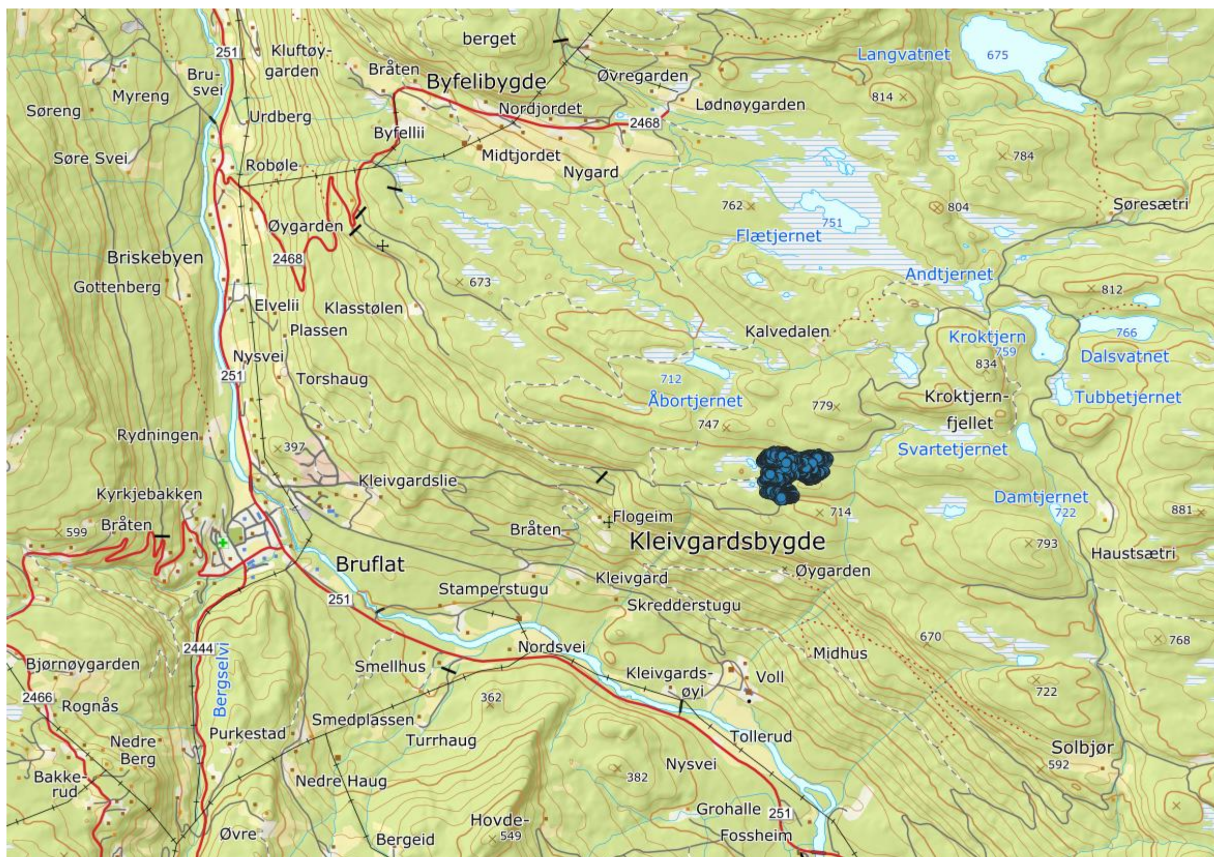
## 3.2 Områdebeskrivelse

Området som behandles i denne oppgaven ligger litt over 3 km øst for tettstedet Bruflat i Etnedal kommune i Innlandet fylke. Området tilhører Gnr/bnr. 124/10. Størrelsen på hogstområdet er rundt 50.000 $m^2$ og 4379 hogde trær.

Som man ser på Figur 8, er ikke området hogd i sin helhet, og det er øyer av gjenstående trær inne i området.



*Figur 8: Området vist på flyfoto fra Norge i Bilder. Blå prikker angir hogde trær. Kartdata: © Kartverket*




*Figur 9: Kart over det omkringliggende området.  Blå prikker markerer hogstområdet. Kartdata: © Kartverket*

## 3.3 Klargjøring av data og valg av metoder

Det ble tatt utgangspunkt i to ulike tilnærminger med utgangspunkt i U-nett:

1. Utsnitt som viser en trekrone med gitt råteverdi mates inn i nettverket.
2. Raster oppdelt i rutenett av konstruerte trekroner fra Lidar.

Grunnen til at det er valgt to ulike metoder, er for å eliminere feil ved de konstruerte trekronene og undersøke om innfallsvinklene ved de to metodene ga ulike resultater.

## 3.4 Felles for begge metoder

### 3.4.1 Utklipp av hyperspektrale data

De hyperspektrale dataene dekket et stort område og kun en liten del av dette, viste området som var hogd.
Det ble derfor tegnet et rektangulært utsnitt rundt området med hogstdata. SWIR og VNIR ble deretter klippet mot det samme utsnittet, noe som reduserte filstørrelsen fra 251 GB til 1,39 GB for VNIR og 71,5 GB til 0,41 GB for SWIR.

Pikselstørrelsen på bakken var 0,3 m for VNIR og 0,7 m for SWIR.

### 3.4.2 Valg mellom VNIR og SWIR

Underveis i prosessen ble det valgt å fokusere på VNIR, da pikselstørrelsen var vesentlig mindre enn for SWIR, og informasjonsmengden samtidig vesentlig større, noe man ser av størrelsen på datasettet. En masteroppgave ved NMBU(Sørhuus, 2019) som sammenlignet begge sensorer, kom frem til at VNIR-sensoren burde prioriteres.

Det ble også undersøkt flere metoder for å kombinere VNIR og SWIR, bla. pansharpening. En problemstilling var at pikslene i SWIR og VNIR var forskjøvet i forhold til hverandre, noe som måtte ha blitt korrigert. Denne forskyvningen var heller ikke stabil over bildet, men fulgte grensen mellom flystripene. Selv om både VNIR og SWIR var fotografert samtidig, var det ulike avgrensninger av flystripene. Dette vises i Figur 10. Videre arbeid med kombinasjonen måtte kuttes grunnet tidsutfordringer.

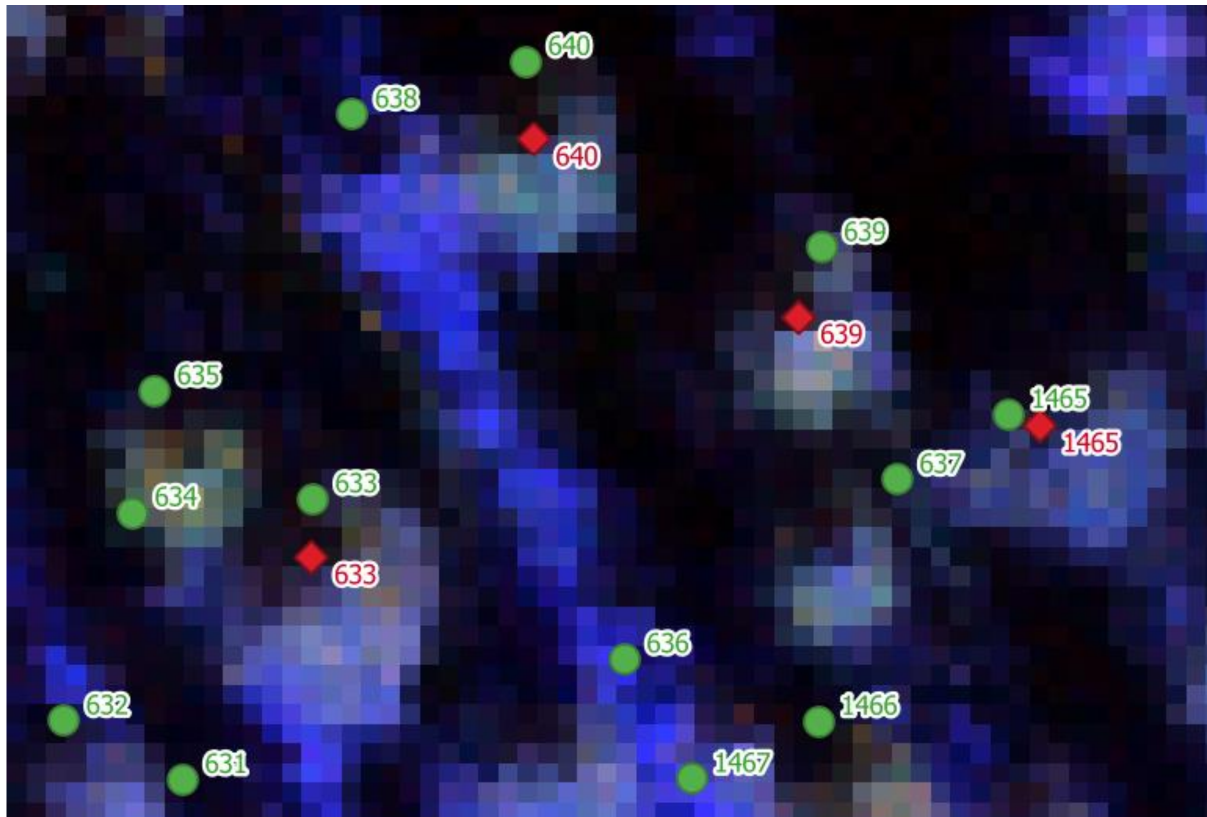
*Figur 10: Kombinasjon av SWIR og VNIR som dekker hogstområdet. Det er tydelige forstyrelser i kanten av flystripene.*

## 3.5 Metode 1: Utsnitt av trekroner

### 3.5.1 Rensing av data

Datasettet med trepunkt ble først lastet inn i QGIS og renset for andre treslag enn gran ved å filtrere attributttabellen.

Deretter ble 151 trær i innenfor hogstområdet manuelt avmerket i QGIS. Disse ble så koblet manuelt til trepunkt i datasettet, blant annet med hjelp av høyden på trepunktet. Som man ser av Figur 11, er det tydelig at punkt 639 kan kobles til det korrigerte trepunktet. På denne måten løser man problemet med usikkerhet både i posisjonering av treet ved hogst, og georefereringen av det hyperspektrale bildet.




*Figur 11: Manuelt korrigerte trepunkt vises som røde diamanter. Trepunkt fra datasettet vist med grønne sirkler.*

### 3.5.2 Bearbeiding av punktdatasettet

Deretter ble behandlingen flyttet over i Python.

Det første som ble utført, var å skille testdataene fra treningsdataene. For å få en jevn fordeling av testdataene ble det tatt et tilfeldig uttrekk av trær fra datasettet. Datasettet ble stokket før splitten, og etterpå ble det sortert etter index. Grunnen til at det ble sortert på nytt, var for å gjøre det mulig å koble trening med råteverdier. Det ble utført en 90/10 splitt mellom trening og testdata. Koden for dette er vist i Funksjon 1.

```
def trees_shuffle_split(trees, test_size = 0.1, random_state=0):

    from sklearn.model_selection import train_test_split

    trees.sample(frac=1, random_state=random_state)
    trees_train, trees_test = train_test_split(trees, test_size=test_size,
random_state=random_state+1)

    return trees_train.reset_index(drop=True), trees_test.reset_index(drop=True)
```

*Funksjon 1:Splitting av test- og treningsdata med tilfeldig rekkefølge*

### 3.5.3 Fjerne overlapp

Det var viktig at treningsdataene ikke overlapper med testdataene. Utsnittene i treningsdata som overlappet med utsnittene testdata, ble derfor fjernet. Dette ble gjort ved å kjøre en pythonfunksjon som automatisk fjernet trepunkt fra treningsdata som lå innenfor et rektangel med 90 % av radius på testdataene. Dette er vist i Funksjon 2. Det ble antatt at en overlapp på maks 10 % i enten høyde eller brede ikke ville ha noen vesentlig påvirkning. Dette er basert på en liknende tilnærming som Sylvain et al. (2019).

```
trees_train = trees_isolation(trees_train, trees_test, distance=window_radius*1.9, img=img)

def trees_isolation(trees_train=None, trees_test=None, distance=None, img=None):
    for i, tree in trees_test.iterrows():
        xcoord = int(tree['X'])
        ycoord = int(tree['Y'])
        imgcoordx = img.sel(x=xcoord,method='nearest').x.values
        imgcoordy = img.sel(y=ycoord,method='nearest').y.values
        imgcoordxmin = imgcoordx - distance
        imgcoordxmax = imgcoordx + distance
        imgcoordymin = imgcoordy - distance
        imgcoordymax = imgcoordy + distance

        for j, treeb in trees_train.iterrows():
            xcoordb = int(treeb['X'])
            ycoordb = int(treeb['Y'])

            if imgcoordxmin < xcoordb < imgcoordxmax and imgcoordymin < ycoordb < imgcoordymax:
                trees_train.drop(j, inplace = True)
            else:
                continue

    return trees_train
```

*Funksjon 2: Funksjon som fjerner trær fra treningsdata som er for nærme trær i testdata.*

### 3.5.4 Lager utsnitt

Det ble deretter laget et utsnitt som viste området rundt hvert trepunkt og en like stor maske som ble påført trepunktets råteverdi. Råteverdien ble skalert til å være mellom 0 og 1 ved å benytte MinMaxScaler fra Scikit-learn. Denne delen er inspirert av Sylvain et al. (2019). Se kode i Funksjon 3.

```
def window_label_patch(trees=None, img=None, patch_radius=None, radius=None, key=None, value=None, 
scaler=None):
    import numpy as np
    from sklearn.preprocessing import MinMaxScaler

    image_patches = []
    label_patches = []
    label_patches_cont = []
    tree_patches = []

    if scaler==None:
        scaler = MinMaxScaler()
        scaler.fit(trees[key].values.reshape(-1, 1))

    for i, tree in trees.iterrows():
        xcoord = int(tree['X'])
        ycoord = int(tree['Y'])
        imgcoordx = img.sel(x=xcoord,method='nearest').x.values
        imgcoordy = img.sel(y=ycoord,method='nearest').y.values

        imgcoordxmin = imgcoordx - patch_radius
        imgcoordxmax = imgcoordx + patch_radius
        imgcoordymin = imgcoordy - patch_radius
        imgcoordymax = imgcoordy + patch_radius

        image_patch = img.sel(x=slice(imgcoordxmin,imgcoordxmax),y=slice(imgcoordymax,imgcoordymin))

        imshape = image_patch.shape
        imwidth = imshape[0]
        imheight = imshape[1]

        if tree[key] >= value:
            label=2
        else:
            label=1

        labelcont = scaler.transform(np.array(tree[key]).reshape(1, -1))

        array = np.zeros((int(imwidth), int(imheight)))
        arraycont = np.zeros((int(imwidth), int(imheight)))
        array[:] = label
        arraycont[:] = labelcont[0][0]

        image_patches.append(image_patch)
        label_patches.append(array)
        label_patches_cont.append(arraycont)
        tree_patches.append(tree)

    return image_patches, label_patches, label_patches_cont, tree_patches, scaler
```

*Funksjon 3: Funksjon som lager bildeutsnitt og maske*

Videre behandling var i stor grad lik som for metode 2 og beskrives derfor senere.

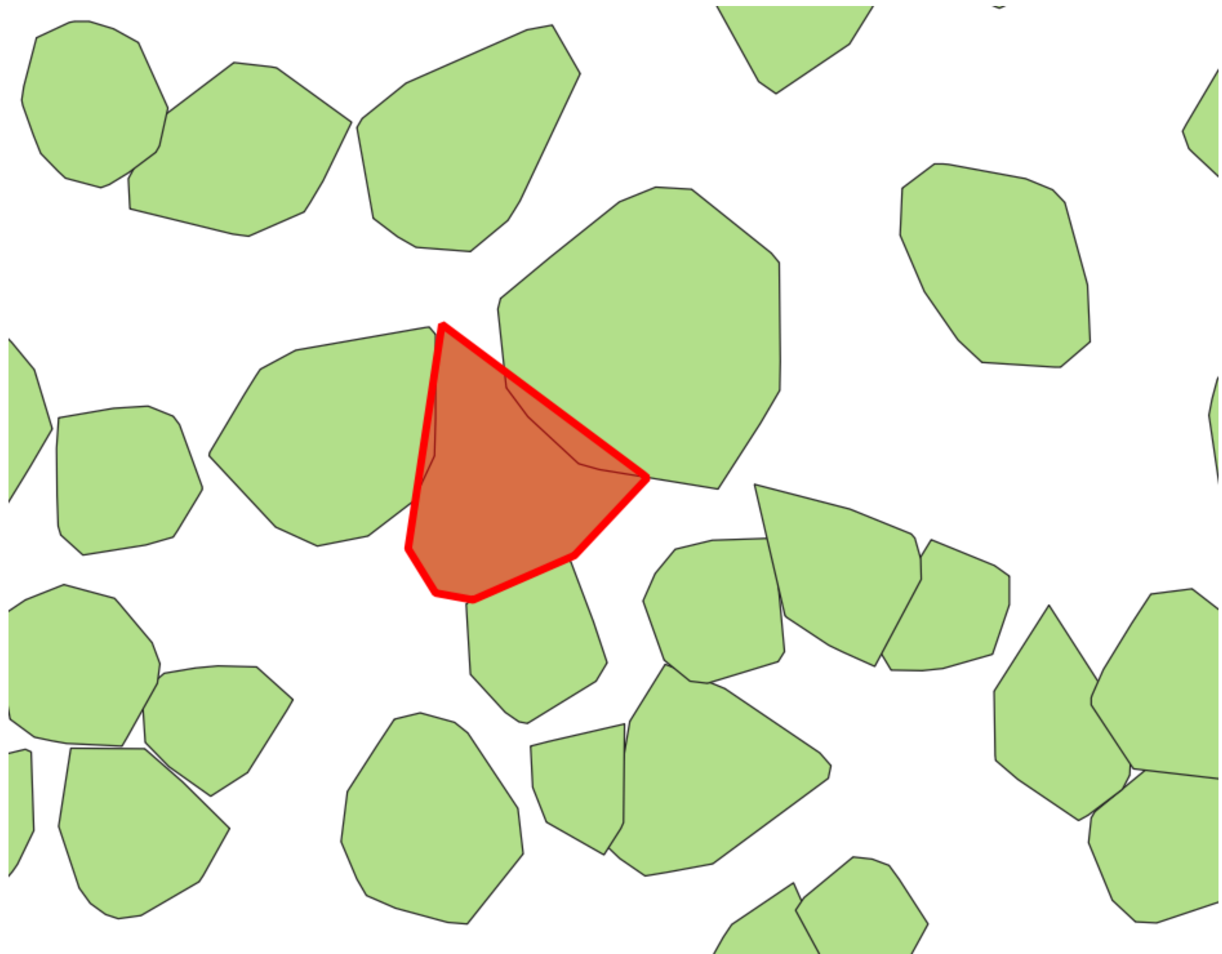
## 3.6 Metode 2: Kobling av trepunkter med trekroner

Metode 2 var en mer omfattende prosess enn metode 1, da trekronedatasettet måtte bearbeides og det måtte lages et rutenett. Den første delen av bearbeidingen ble utført i QGIS.

### 3.6.1 Fjerne overlapp mellom trekroner

Polygonene med trepunkt fra laserdata ble lastet inn i QGIS. Ved en rask kontroll, ble det oppdaget at polygonene hadde overlapp, noe som måtte fjernes. Dette er vist i Figur 12.

For å løse dette, ble programmet SAGA, som er innebygd i QGIS, benyttet. Nærmere bestemt verktøyene «Polygon self-intersection» og «Løs opp valgte flater» med hensyn på Lengst felles grense.



*Figur 12: Polygon i rødt overlapper med 2 nabopolygoner.*

### 3.6.2 Kobling av trepunkter mot trekroner

Trepunktene fra hogstmaskinen og trekronene fra Lidar kunne ikke kobles direkte, da disse ikke alltid sammenfalt grunnet posisjonsavvik i hogstdata. Et eksempel på dette er vist i Figur 13. For å korrigere for dette, ble det laget en buffer på 1 m rundt hvert punkt.

*Figur 13: Eksempel på at trepunkt og trekrone ikke overlapper.*

Deretter ble trekroner og bufrede trepunkt sammenslått, basert på plassering i en-til-mange sammenføyning. Ved å bruke en-til-mange sammenføyning ble det tatt hensyn til alle trepunkt som kunne være aktuelle for den enkelte trekrone. Et problem med denne fremgangsmåten, er at det i mange tilfeller ble flere trepunkter koblet til hver trekrone. Dette kommer av at det er flere trær i hogstdataene enn i trekronemodellen. To årsaker til dette er at trekronene er slått sammen i modellen og at mindre trær står under høyere trær.

For å løse dette ble det gjennomført to filtreringer:

#### Filtrering av treslag

Da kun gran er interessant for modellen, tas kun trekroner og trepunkter klassifisert som gran videre. Filteret som ble brukt mot attributttabellen var "SP"  = 'Picea_abies'  AND "Species"  = 'Gran'

#### Filtrering med hensyn på høyde

For å finne ut hvilket trepunkt som mest sannsynlig hører til trekronen, ble det beregnet høydedifferanse mellom høyden på trekronen og trepunktet fra hogstdata med funksjonen: («abs ( "treeH" - "Height_m" )»).
Deretter ble listen sortert på avviket før man kjørte funksjonen «Slett duplikater etter attributt» med hensyn på IDen til hver trekrone. Dette førte til at det kun ble en kobling mellom trepunktet og trekronen med lavest høydedifferanse.

Det ble vurdert å ta en ytterliggere opprydding for å finne trepunkt som var koblet til flere trekroner, men dette var for tidkrevende, da prosessen måtte gjøres manuelt.

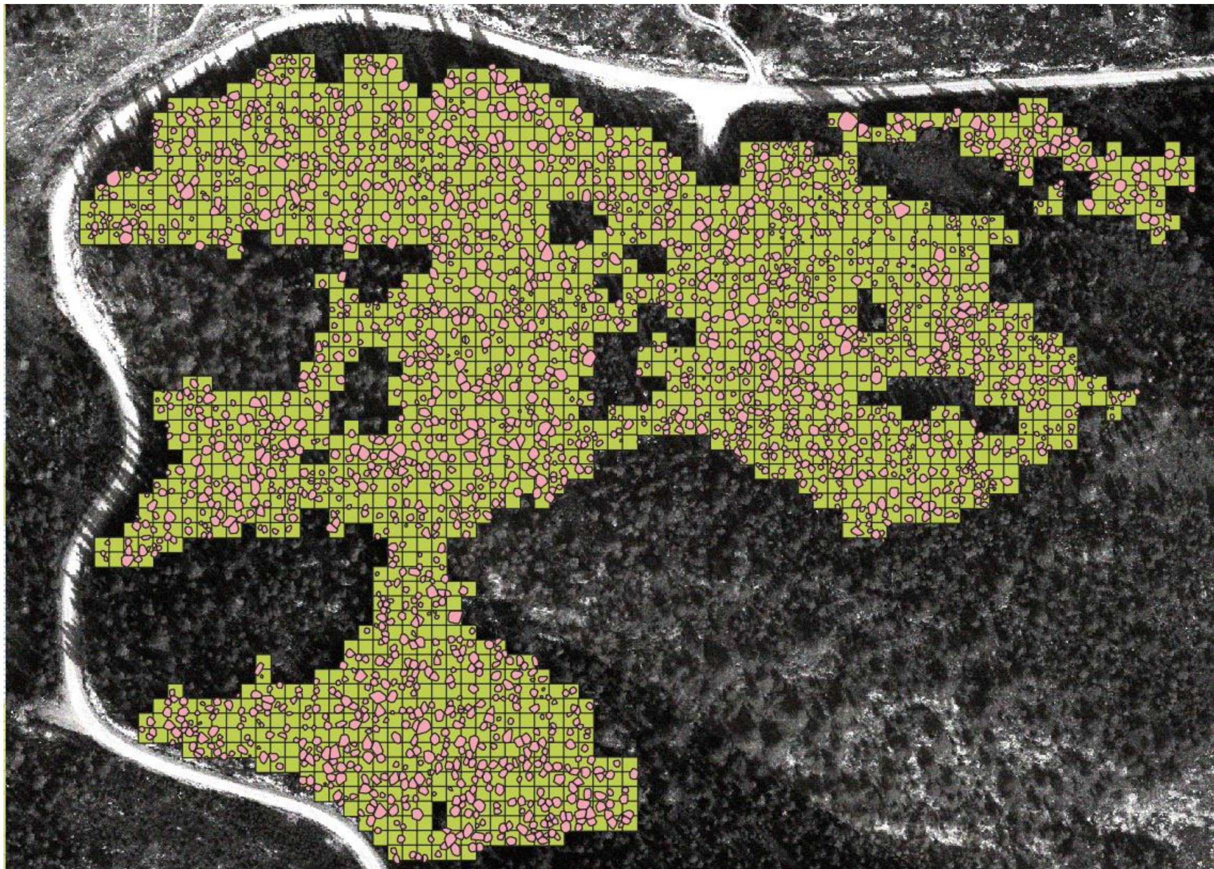
### 3.6.3 Trekronene kombineres med bakgrunnen i et raster

Trekronene ble deretter konvertert til raster. Dette ble gjort ved å rasterisere polygonene med hensyn på råte-verdien. (hegtRM). De to rasterlagene ble deretter slått sammen. Et problem som ble oppdaget, og som trolig skyldes en feil i programvaren, var at prosessen ikke tar vare på metadata. Denne måtte derfor kopieres inn manuelt.

### 3.6.4 Lage rutenett

Da rutenettet må tilpasses U-nett, må størrelsen settes slik at bildet blir kvadratisk og toerpotens på hver side. For å få hhv. 16 og 32 piksler i hvert bilde, må det lages rutenett med rutestørrelsen 4,8 m og 6,9 m. Utstrekning av rutenettet ble satt til rasterlag.

For å fjerne tomme ruter utenfor hogstområdet, ble rutenettet klippet til områdeavgrensing. Områdeavgrensing ble laget ved å ta en buffer på 5 m rundt hvert trekronepolygon. For å fjerne halve ruter, ble det til slutt beregnet areal av hver rute, og alle ruter med areal under maksverdien ble slettet.



*Figur 14: Visning i QGIS etter at rutenett er laget*

### 3.6.5 Splitte opp rutenett og dele opp raster

Til slutt måtte det kjøres en batchjobb, der rasteret klippes til hver rute i rutenettet.
Hver rute i rutenettet ble dermed splittet opp i egne vektorlag slik at de kunne behandles videre.

### 3.6.6 Videre behandling i Python

Dataene i metode 2 gjennomgikk en vesentlig enklere prosess i Python enn ved metode 1.

Ut av QGIS er råteverdien lagt inn som et eget bånd. Dette måtte skilles ut, slik at det ble et eget maskelag med råteverdier. Dette ble utført ved hjelp av Xarray som vist i Funksjon 4.

Det ble deretter utført en 90/10 splitt av datasettet til trening- og testdata. Her ble det brukt train_test_split fra Scikit-learn.

Det ble også utført en skalering på samme måte som ved metode 1, men dette ble gjort senere i prosessen. Videre behandling beskrives felles med metode 1. Hele koden er vist i vedlegg 1.

```
for name in filenames:
  if name[-3:] == 'tif':
    imgid = name[:-4]

    img = xr.open_rasterio(os.path.join(dirname + '\\' + imagefolder + '\\' +
name)).transpose('y','x','band')
    label = img.sel(band=[labellayer])
    img = img.drop_sel(band=[labellayer])

    imglist.append(img)
    labellist.append(label)
    imgidlist.append(imgid)
```

*Funksjon 4: Deling av datasett og maskelag*

## 3.7 Felles etterbehandling i Python for begge metoder

### 3.7.1 Konvertering til array

Bildedataene ble konvertert til array for å gjøre det mulig å behandle det i det nevrale nettverket. Se kode i Funksjon 5. Deler av koden er inspirert av Moneda (2017).

Metode 1 og 2 var litt ulike her. I metode 1 ble råtemaskene konvertert til array ved hjelp av numpy.asarray, mens i metode 2 ble koden vist i Funksjon 5 benyttet. En annen forskjell var at i metode 2 der råtemaskene var et bilde, ble skalering gjort etter at bildet ble konvertert til array. Grunnen til dette var at bildedata ikke kunne skaleres med MinMaxScaler. Hvor i prosessen skaleringen utføres, har ingen betydning for resultatet, så lenge min og maks- råteverdi ikke blir endret underveis.

### 3.7.2 Mellomlagring

Dataene ble mellomlagret på harddisk ved hjelp av pythonmodulen pickle. Dette ble gjort fordi datasettet var for stort for datamaskinens hurtigminne. Hadde ikke dette blitt gjort, ville beregningene av treningsarrayen måtte ha blitt kjørt parallelt med treningen av det nevrale nettverket. En annen fordel var at man kunne kjøre treningen av det nevrale nettverket med ulike konfigurasjoner uten å beregne treningsarrayen flere ganger.(Hetle & Hagen, 2019)

Også skaleringen måtte mellomlagres, da denne var nødvendig for å konvertere resultatet tilbake til råtehøyde.

```
def A_A(image_patches=None,override=None):
  import numpy as np
  from tensorflow.keras.preprocessing.image import img_to_array

  if override == None:
    imshape = image_patches[0].shape
  else:
    imshape = override
  imwidth = imshape[0]
  imheight = imshape[1]
  imbands = imshape[2]

  dataset = np.ndarray(shape=(len(image_patches), imwidth, imheight, imbands),
              dtype=np.float32)

  # Inspirert av https://www.kaggle.com/lgmoneda/from-image-files-to-numpy-arrays
  i = 0
  for file in image_patches:
    x = img_to_array(file,data_format=None)
    dataset[i] = x
    i += 1
    if i % 250 == 0:
      print("%d images to array" % i)
  print("All %d images to array!" % i)
  return dataset, imwidth, imheight, imbands
```

*Funksjon 5: Konvertering til array*

## 3.8 U-nett

Selve koden for U-nettet benytter Tensorflow/Keras og var i stor grad basert på tidligere arbeid (Hagen & Hetle, 2019), men det er gjort flere tilpasninger, bl.a. for å håndtere hyperspektrale bilder.

Det første som ble utført, var å definere antall epoker og batchstørrelse. Det ble valgt 100 epoker, da innledende tester viste at det ikke var noe mer å hente på å ha flere enn dette. Batchstørrelsen ble satt til 16 for metode 1, 32 for rutenett 4,8 og 64 for rutenett 9,6. Deretter ble dataene lastet inn fra mellomlageret. ImageDataGenerator fra Tensorflow/Keras ble videre benyttet for å kunne mate dataene inn i det nevrale nettverket i batcher og samtidig teste virkningen av en mindre augmentering, der en del av bildeutsnittet ble flippet horisontalt og vertikalt.

Det er noen små forskjeller i koden mellom metode 1 og metode 2, men disse kommer av at det i metode 1 er bilder av trekroner, mens i metode 2 er rutenett. Dette har ikke noen sentral betydning for hvordan koden fungerer.

I selve U-nettet ble aktiveringsfunksjonen i siste ledd valgt som sigmoid, da råteverdiene som skulle analyseres var kontinuerlige.

Som tapsfunksjon ble dice_coef(Jocić et al., 2016) valgt, da dice_coef fungerer svært godt på ubalanserte datasett.
Denne ubalansen førte til at vanlige tapsfunksjoner, slik som sparse_categorical_crossentropy, ikke ville gitt et fornuftig resultat. Det var heller ikke ønskelig å balansere datasettet, da dataene var naturlig skjevfordelt.

Modellens vekter ble lagret hver gang dice_score(Buda et al., 2016) for valideringen økte. Dette medfører at man kan hente inn de beste vektene på et senere tidspunkt. Det ble også lagt inn funksjoner som viste fremgang og summary report for beste vekter.

Ved å lagre vektene ble det mulig å analysere testdatasettet på de beste vektene, og dermed få et resultat.

Vedlegg 2 viser koden for U-nett i metode 2.

### 3.8.1 Test av hyperparametere

Det er en rekke ulike hyperparametere som kan justeres og det ble valgt å kjøre en test som testet alle kombinasjoner av frafall og optimaliseringsfunksjoner. For frafall ble verdiene 0,0, 0,1 og 0,2 testet og optimaliseringsfunksjonene Adam(Kingma & Ba, 2014) og RMSprop(Tieleman & Hinton) ble begge testet med læringsratene 0,001, 0,0001 og 0,00001.

## 3.9 Sensitivitetstest

Som nevnt i kapittel 2.10, blir vektene i det nevrale nettverket satt tilfeldig ved hver kjøring. Det er kjent at dette kan påvirke resultatet. For å undersøke hvor stor påvirkning de tilfeldige vektene hadde ble samme modell kjørt med samme hyperparameter tre ganger.

# 4 Resultat

## 4.1 Grunnlinje

### 4.1.1 Metode 1

Som nevnt i kapittel 2.9, må man sammenligne resultatet med grunnlinjen. Dette krever at man beregner grunnlinjen. Ut fra datasettet kan man finne at 69% av trærne er friske. Dvs. at i 69 % av trærne går råten mindre enn 1 m opp i stammen. Da hvert bildeutsnitt er gitt verdien til et tre, vil det tilsi at en algoritme som klassifiserer alle pikslene i samtlige bildeutsnitt som friske får en nøyaktighet på 69 %. Dette tilsvarer en vektet F1-score (weighted avg i classification_report) på 0,56 og uvektet (macro avg) på 0,41.

### 4.1.2 Metode 2

I metode 2 er trekronen til friske trær og bakgrunnen begge gitt råteverdien 0, da U-nett krever at alle pikslene gis en verdi. Ved hjelp av QGIS ble arealet av alle trekronene med råteverdi over 1 m beregnet. Dette tilsvarte 6 % av området. Dette vil tilsi at en algoritme som klassifiserer alle pikslene som friske får en nøyaktighet på 94 %. Dette tilsvarer en vektet F1-score (weighted avg i classification_report) på 0,91 og uvektet (macro avg) på 0,48.

## 4.2 Resultat fra metode 1

| Frafall | Optimiser | Læringsrate | Resultat med flipp | Resultat uten flipp |
|---|---|---|---|---|
| 0,0 | Adam | 0,001 | 0,32 | 0,25 |
| 0,0 | Adam | 0,0001 | 0,27 | 0,27 |
| 0,0 | Adam | 0,00001 | 0,21 | 0,22 |
| 0,0 | RMSprop | 0,001 | 0,25 | 0,36 |
| 0,0 | RMSprop | 0,0001 | 0,26 | 0,32 |
| 0,0 | RMSprop | 0,00001 | 0,24 | 0,26 |
| 0,1 | Adam | 0,001 | 0,27 | 0,26 |
| 0,1 | Adam | 0,0001 | 0,21 | 0,36 |
| 0,1 | Adam | 0,00001 | 0,23 | 0,21 |
| 0,1 | RMSprop | 0,001 | 0,28 | 0,41 |
| 0,1 | RMSprop | 0,0001 | 0,24 | 0,24 |
| 0,1 | RMSprop | 0,00001 | 0,21 | 0,21 |
| 0,2 | Adam | 0,001 | 0,26 | 0,26 |
| 0,2 | Adam | 0,0001 | 0,24 | 0,26 |
| 0,2 | Adam | 0,00001 | 0,22 | 0,20 |
| 0,2 | RMSprop | 0,001 | 0,24 | 0,47 |
| 0,2 | RMSprop | 0,0001 | 0,23 | 0,25 |
| 0,2 | RMSprop | 0,00001 | 0,20 | 0,20 |

*Tabell 1: Sammenligning av beste vekter ved ulike parametere ved metode 1. Resultatene er oppgitt fra tapsfunksjonen på valideringsdatasettet.*

| U-nett med horisontal og vertikal flipp: | U-nett uten horisontal og vertikal flipp: |
| --- | --- |
| best_score:<br>0.3190217<br>best_score_param:<br>'u_net-sigmoid_2-0.0-Adam-0.001' | best_score:<br>0.46764538<br>best_score_param<br>'u_net-sigmoid_2-0.2-RMSprop-0.001' |

*Tabell 2: Sammenstilling av beste resultat fra metode 1 med og uten flipp.*

Av Tabell 2 ser man at det beste resultatet ble oppnådd ved U-nett uten horisontal og vertikal flipp, frafall på 0,2, optimiser RMSprop og læringsrate 0.001.

### 4.2.1 Resultat fra testdatasett kjørt på beste parametere

```
                precision    recall  f1-score   support

Råte under 1 m       0.91      0.81      0.86      3468
 Råte over 1 m       0.57      0.76      0.65      1156

      accuracy                           0.80      4624
     macro avg       0.74      0.78      0.75      4624
  weighted avg       0.82      0.80      0.80      4624
```

*Tabell 3: Utskrift av model_summary for beste parametere i metode 1.*

Som man ser av Tabell 3 oppnådde metode 1 en vektet F1-score på 0,80 og uvektet på 0,75. Som nevnt i kapittel 4.1.1 var grunnlinjen på vektet 0,56 og uvektet 0,41. Metode 1 oppnådde altså bedre resultater enn grunnlinjen.

## 4.3 Resultat fra metode 2

| Rutenett | Frafall | Optimiser | Læringsrate | Resultat med flipp | Resultat uten flipp |
|---|---|---|---|---|---|
| 4,8 | 0,0 | Adam | 0,001 | 0,12 | 0,12 |
| 4,8 | 0,0 | Adam | 0,0001 | 0,07 | 0,07 |
| 4,8 | 0,0 | Adam | 0,00001 | 0,05 | 0,05 |
| 4,8 | 0,0 | RMSprop | 0,001 | 0,11 | 0,08 |
| 4,8 | 0,0 | RMSprop | 0,0001 | 0,06 | 0,05 |
| 4,8 | 0,0 | RMSprop | 0,00001 | 0,05 | 0,06 |
| 4,8 | 0,1 | Adam | 0,001 | 0,12 | 0,12 |
| 4,8 | 0,1 | Adam | 0,0001 | 0,11 | 0,10 |
| 4,8 | 0,1 | Adam | 0,00001 | 0,05 | 0,06 |
| 4,8 | 0,1 | RMSprop | 0,001 | 0,12 | 0,11 |
| 4,8 | 0,1 | RMSprop | 0,0001 | 0,10 | 0,11 |
| 4,8 | 0,1 | RMSprop | 0,00001 | 0,05 | 0,05 |
| 4,8 | 0,2 | Adam | 0,001 | 0,13 | 0,12 |
| 4,8 | 0,2 | Adam | 0,0001 | 0,10 | 0,10 |
| 4,8 | 0,2 | Adam | 0,00001 | 0,05 | 0,05 |
| 4,8 | 0,2 | RMSprop | 0,001 | 0,12 | 0,11 |
| 4,8 | 0,2 | RMSprop | 0,0001 | 0,09 | 0,10 |
| 4,8 | 0,2 | RMSprop | 0,00001 | 0,06 | 0,06 |
| 9,6 | 0,0 | Adam | 0,001 | 0,08 | 0,09 |
| 9,6 | 0,0 | Adam | 0,0001 | 0,05 | 0,05 |
| 9,6 | 0,0 | Adam | 0,00001 | 0,05 | 0,06 |
| 9,6 | 0,0 | RMSprop | 0,001 | 0,09 | 0,10 |
| 9,6 | 0,0 | RMSprop | 0,0001 | 0,05 | 0,07 |
| 9,6 | 0,0 | RMSprop | 0,00001 | 0,05 | 0,05 |
| 9,6 | 0,1 | Adam | 0,001 | 0,11 | 0,07 |
| 9,6 | 0,1 | Adam | 0,0001 | 0,07 | 0,08 |
| 9,6 | 0,1 | Adam | 0,00001 | 0,06 | 0,06 |
| 9,6 | 0,1 | RMSprop | 0,001 | 0,09 | 0,09 |
| 9,6 | 0,1 | RMSprop | 0,0001 | 0,06 | 0,08 |
| 9,6 | 0,1 | RMSprop | 0,00001 | 0,04 | 0,07 |
| 9,6 | 0,2 | Adam | 0,001 | 0,08 | 0,09 |
| 9,6 | 0,2 | Adam | 0,0001 | 0,08 | 0,08 |
| 9,6 | 0,2 | Adam | 0,00001 | 0,05 | 0,06 |
| 9,6 | 0,2 | RMSprop | 0,001 | 0,11 | 0,13 |
| 9,6 | 0,2 | RMSprop | 0,0001 | 0,08 | 0,08 |
| 9,6 | 0,2 | RMSprop | 0,00001 | 0,05 | 0,05 |

*Tabell 4: Sammenligning av beste vekter ved ulike parametere. Resultatene er oppgitt fra tapsfunksjonen på valideringsdatasettet.*

| U-nett med horisontal og vertikal flipp: | | U-nett uten horisontal og vertikal flipp: | |
|---|---|---|---|
| **Rutenett 4,8:** | **Rutenett 9,6:** | **Rutenett 4,8:** | **Rutenett 9,6:** |
| best_score: 0.12889221 best_score_param: 'u_net-sigmoid_2-0.2-Adam-0.001' | best_score: 0.113919884 Best score param: 'u_net-sigmoid_2-0.1-Adam-0.001' | best_score: 0.121114604 best_score_param: 'u_net-sigmoid_2-0.2-Adam-0.001' | best_score: 0.12820707 best_score_param: 'u_net-sigmoid_2-0.2-RMSprop-0.001' |

*Tabell 5: Sammenstilling av beste resultat med og uten flipp og ulike rutenett.*

Av Tabell 5 ser man at det beste resultatet ble oppnådd ved rutenett 4,8 og U-nett uten horisontal og vertikal flipp, frafall på 0,2, optimiser Adam og læringsrate 0.001.

### 4.3.1 Resultat fra testdatasett kjørt på beste parametere:

```
                precision    recall  f1-score   support

Råte under 1 m       0.95      0.96      0.96     64254
 Råte over 1 m       0.17      0.13      0.14      3842

      accuracy                           0.91     68096
     macro avg       0.56      0.54      0.55     68096
  weighted avg       0.90      0.91      0.91     68096
```

*Tabell 6: Utskrift av model_summary for beste parametere.*

Som man ser av Tabell 6 oppnådde metode 2 en vektet F1-score på 0,91 og uvektet på 0,55. Som nevnt i kapittel 4.1.2 var grunnlinjen på vektet 0,91 og uvektet 0,48. Metode 2 oppnår altså ikke bedre resultat enn grunnlinjen for vektet F1-score, men oppnår noe bedre på uvektet.

## 4.4 Sensitivitetstest

Resultatet varierte mellom 0.09-0.12, noe som gir en variasjon på 0,03.

# 5 Diskusjon

Resultatene viser at både metode 1 og metode 2 klarer å oppdage råtne trær med kjerneråte over 1 m bedre enn grunnlinjen, men det er sentrale forskjeller på hvor effektivt de klarer dette. Ser man på recall for piksler merket med råte over 1 m, er det spesielt metode 1 som utmerker seg ved å kunne avdekke 76 % av pikslene, mens metode 2 kun klarer 13 %.

Da selve det nevrale nettverket i praksis er likt mellom metodene, må forklaringen være i selve forbehandlingen.

## 5.1 Sensitivitetstest

Resultatet fra sensitivitetstesten, vist i kapittel 4.4, indikerer at resultatet kan variere med opptil 0,03 grunnet den tilfeldige tildelingen av vekter. Dette betyr at differansen mellom to ulike resultater fra ulike innstillinger, vist i Tabell 1 og Tabell 4, bør overstige 0,03, før man tillegger differansen noe vekt. Denne sensitiviteten gjør at man i praksis ikke kan tillegge enkeltverdier for stor vekt, men heller se på tendenser i resultatene.

## 5.2 Augmentering

Resultatene viser en klar tendens at nettverk trent uten horisontal og vertikal flipp gir bedre resultat enn nettverk der deler av bildeutsnittene blir flippet. Dette har trolig sammenheng med at datasettet dekker et mindre område, med liknende forhold. Trolig ville et nettverk trent eller validert på flere områder med ulike forhold gi et annet resultat.

## 5.3 Mulige feilkilder

I de følgende avsnittene er ulike årsaker til det forskjellige resultatet for metode 1 og 2 diskutert.

### 5.3.1 Ulike batchstørrelser

I de nevrale nettverkene opereres det med ulike batchstørrelser avhengig av bildeutsnittenes størrelse. Det ble utført noen tester for å kontrollere at dette ikke påvirket resultatene vesentlig. Det må påpekes at metode 1 ga tilsvarende resultater ved batchstørrelse 64 som ved 16, men fungerte dårlig ved 32. Det har ikke vært mulig å undersøke dette fenomenet nærmere, da det krever en betydelig dypere gjennomgang av hvordan de skjulte lagene oppfører seg.

### 5.3.2 Mindre overlapp og feil i metode 1

Det er en mindre overlapp mellom treningsdatasett og testdatasett i metode 1. Dette er beskrevet i kapittel 3.5.3. Det er også en mindre feil i metode 1 som introduseres av at man bruker pythonfunksjonen slice som kun støtter heltall. Dette fører til at det legges på en ekstra rad og kolonne med piksler, noe som må korrigeres i koden for U-nettet.

Da det trolig er den mest sentrale delen av bildeutsnittet som er av interesse, antas ikke en slik overlapp å ha noen betydning. Dette påvirker også kun en mindre del av treningsdataene, noe som medfører at den umulig kan ha noen stor virkning.

### 5.3.3 Størrelse og utvalg av datasett

Normalt vil man regne med at et større datasett vil gi bedre resultat. Metode 1 har kun 151 trepunkt, mens metode 2 har 518 trekroner i datasettet. Som man ser av resultatene har den metoden som har et betydelig mindre datasett, best resultat

En mulig forklaring på dette er at i metode 1 er det kun valgt ut trær der man lett kan finne senter i trekronen. Dette medfører trolig at det blir en større andel trær med tydelig krone i datasettet. Samtidig er det trolig de samme trærne som er klassifisert best av trekronesegmenteringen som

benyttes i metode 2. En metode for å undersøke forskjellen på dette, er ved å se på gjennomsnittlig trehøyde.

I metode 1 er gjennomsnittlig trehøyde 18,9 m, mens den i metode 2 er 17,6 m, noe som er en uvesentlig forskjell.

### 5.3.4 Posisjonsnøyaktighet og kobling

I metode 1 er hvert tre manuelt korrigert, slik at senter av treet plasseres nær midten av hvert bildeutsnitt. Koblingen mellom hogstdata og hyperspektrale flyfoto er også gjort manuelt som omtalt i kapittel 3.5.2. Ved at trærne er manuelt korrigert for de hyperspektrale dataene bør det ikke være noen vesentlig usikkerhet i denne koblingen. Uten en slik korreksjon ville det vært en unøyaktighet på 1 m fra posisjoneringen av treet ved hogst og i tillegg usikkerhet i koblingen mellom tre og hyperspektrale data.

I metode 2 er trepunktene koblet mot segmenterte trekroner, og bildeutsnittet er bestemt av rutenett. Da trekronen er laget ved hjelp av hyperspektrale data og laserskanning(Dalponte & Coomes, 2016), burde det heller ikke ved denne metoden være noen usikkerhet i koblingen mellom trepunkt og hyperspektrale data. Dette må sees i sammenheng med kapittel 5.3.7.

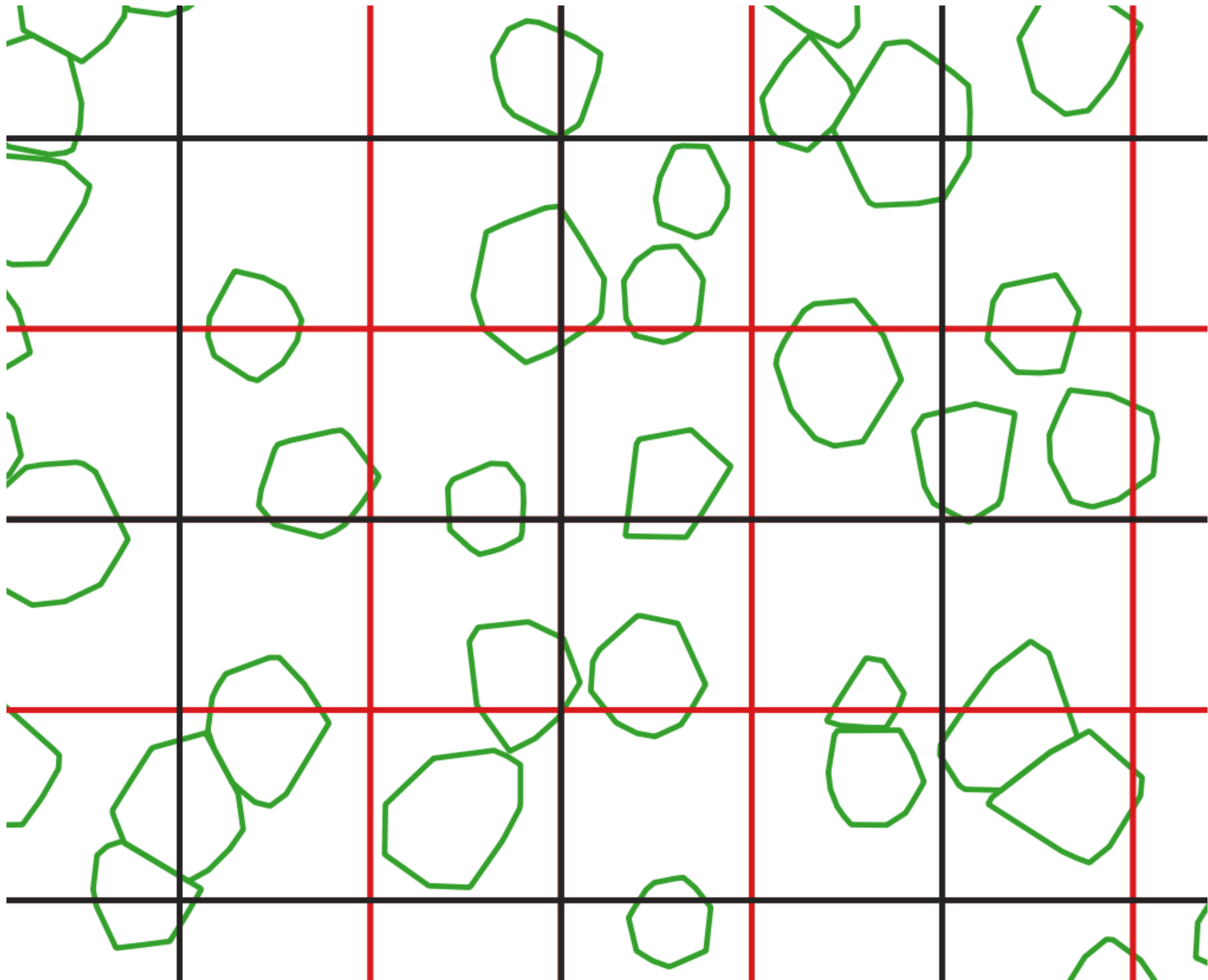
### 5.3.5 Gjenstående trær

Trær som blir stående igjen i hogstfeltet, er synlige på de hyperspektrale dataene, laserskanningen og de segmenterte trekronene, men kommer ikke frem i hogstdata. Dette vil kunne medføre at trekroner kobles til feil tre, dersom det korrekte treet ikke er hogd. Dette vil kunne medføre støy i datasettet og redusere det nevrale nettverkets evne til å trene. En oversikt over trærne som står igjen, vil kunne føre til at man kan filtrere bort trekroner som ellers ville blitt koblet feil.

### 5.3.6 Kanteffekt

Når trekronepolygonene havner på kanten mellom to ruter, blir de derfor splittet. Dette er vist i Figur 15. Når ikke hele trekronen er synlig på et bilde, vil nettverkets evne til å kjenne igjen trekronen trolig bli svekket.

I rutenett 4,8 er det fire ganger så mange ruter på samme område som rutenett 9,6. Som vist i Figur 15 er et betydelig høyere antall trekroner som delt rutenett 4,8 enn i 9,6, noe som skulle tilsi at rutenett 4,8 skulle få et dårligere resultat enn 9,6. Av Tabell 4 ser man derimot at det er en tendens til det motsatte, at rutenett 4,8 får et bedre resultat enn rutenett 9,6.

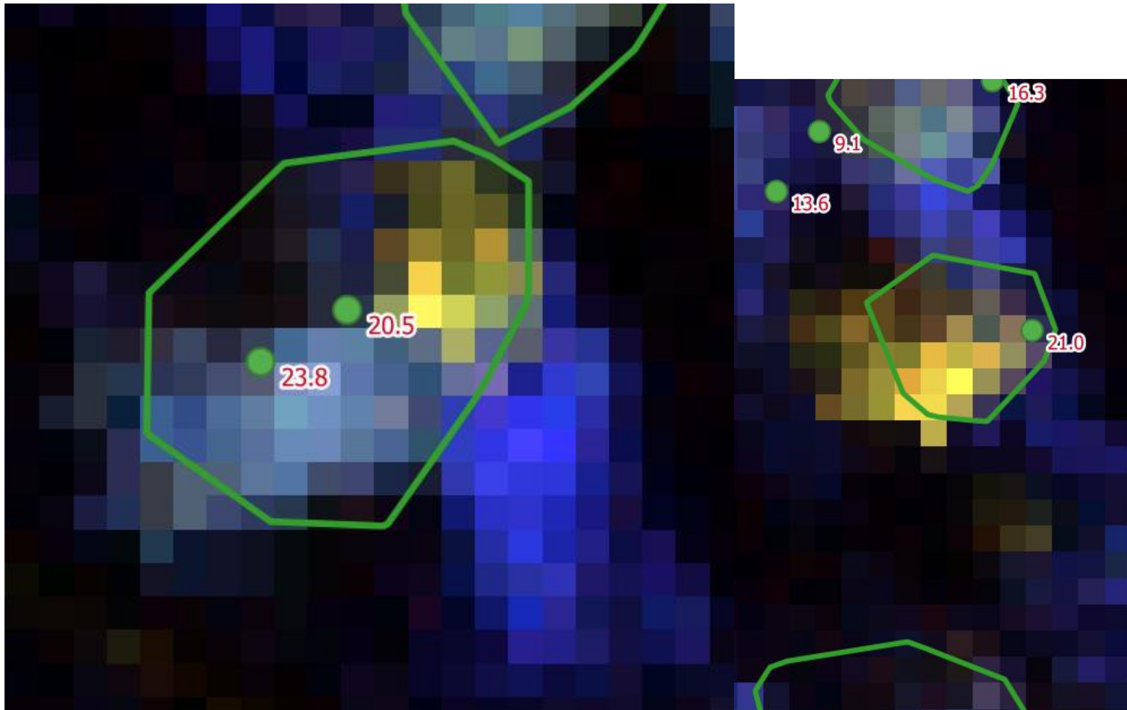


*Figur 15: Grønne trekronepolygoner blir splittet av rutenett. Sorte og røde streker viser hvordan mindre rutestørrelser fører til at flere polygnoner blir delt.*

### 5.3.7 Avvik i trekronesegmentering og kobling mellom trepunkt og trekrone.

Som beskrevet i kapittel 3.6.2, er det i metode 2 utført en automatisk kobling mellom trepunkt og trekrone. Det er mulig det er oppstått feil i denne koblingen. En nærmere undersøkelse av datasettet viser at 21 trepunkt er koblet til 2 trekroner og 1 trepunkt er koblet til 3 trekroner. Dette betyr at 23 trekroner trolig er koblet til feil trepunkt. Dette utgjør 0,9 % av samtlige 2474 trekroner av gran.

Det er også observert en del feil i trekronesegmenteringen. Til venstre i Figur 16 ser man et eksempel på at to trær er slått sammen av trekronesegmenteringen. Dette er to trær som er nesten like høye og står inntil hverandre. Til høyre ser man et eksempel på at et den segmenterte trekronen kun dekker deler av et tre.

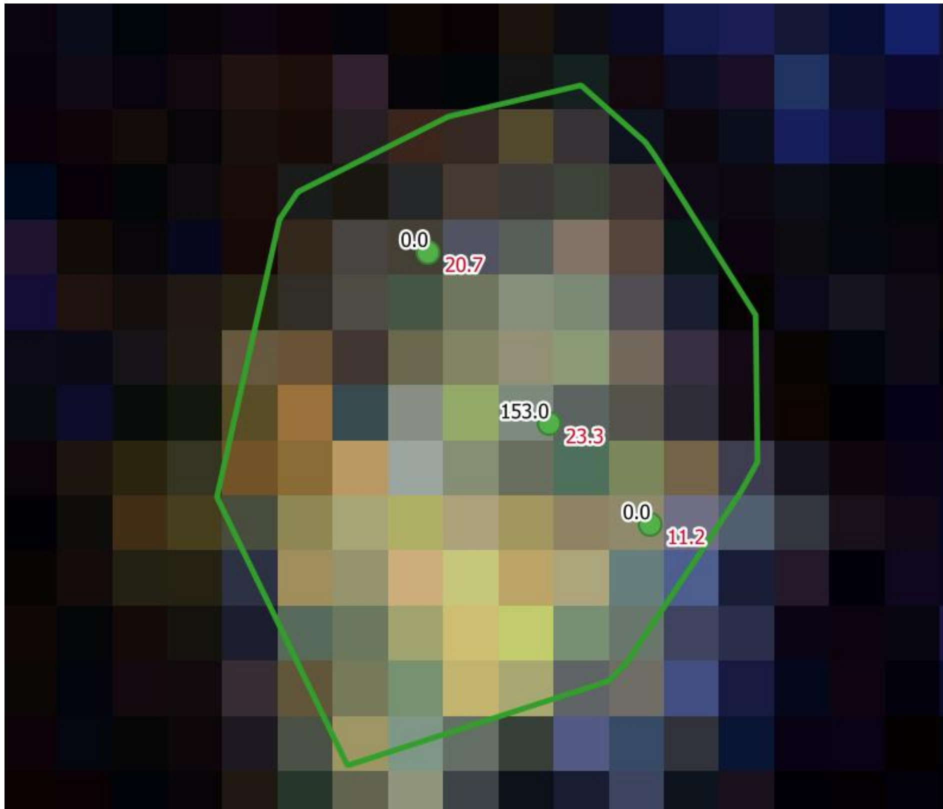



*Figur 16: Til venstre: To trær som er segmentert som en trekrone. Til høyre: Trekrone som kun dekker deler av treet. Grønt omriss viser segmentert trekrone. Grønne prikker viser hogstpunkt med angitt høyde.*

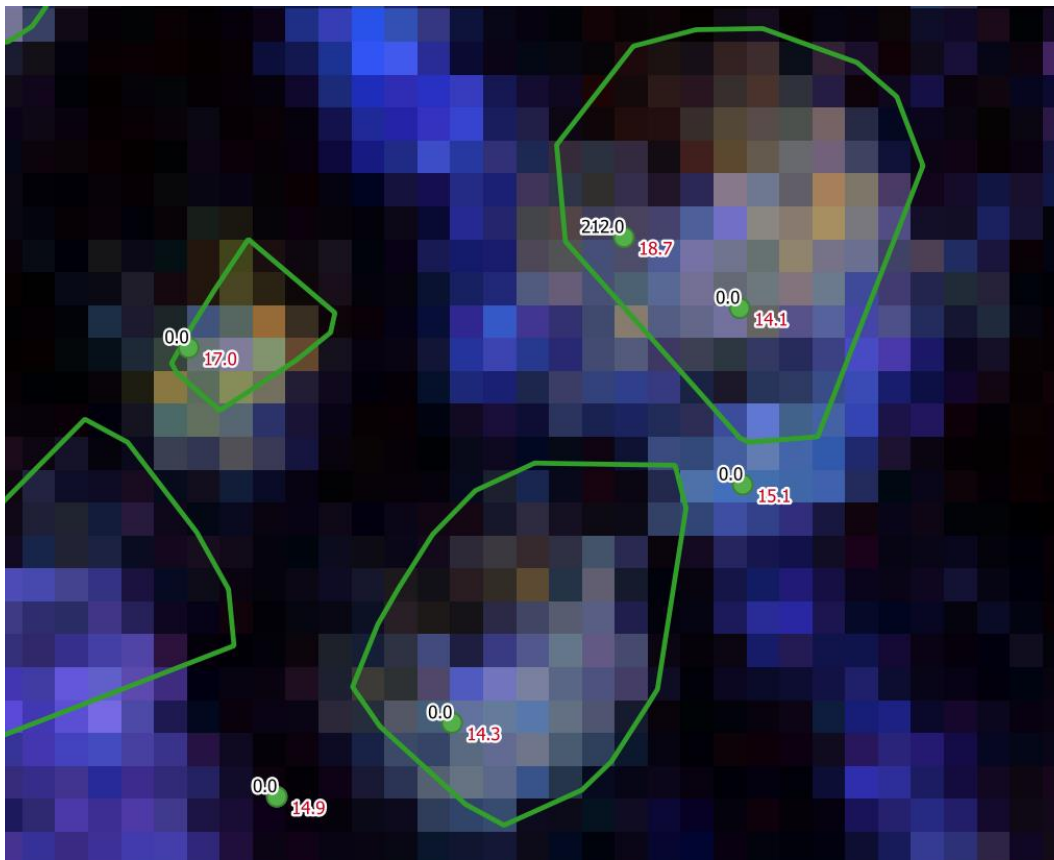
I Figur 17 ser man et tilsvarende eksempel med flere trær som er slått sammen til en trekrone, men i dette tilfellet er det også ulike råteverdier, vist som sorte tall.

Dersom trekronen kobles til et av de friske trærne, vil de hyperspektrale pikslene fra samtlige tre trær bli merket som friske. Dersom trekronen kobles til det råtne treet, vil pikslene fra samtlige tre trær bli merket som råtne. Dette fører til at enten det friske eller det råtne treet får feil merking av piksler under treningen. Ved at trærne blir merket feil, blir det tilført støy som gjør det vanskeligere for det nevrale nettverket å finne egenskaper som skiller råtne trær fra friske trær.

Et annet eksempel er vist i Figur 18, der det er registrert 6 hogde graner med høyde mellom 14,1 og 18,7 m, men kun 3 trekroner merket som gran ligger inne i datasettet.




*Figur 17: Flere trær segmentert som en trekrone. Røde tall viser trærnes høyde i meter. Sorte tall er råtehøyde i millimeter.*




*Figur 18: Utsnitt som viser flere feil som opptrer innenfor et mindre område.*

*Feil i trekronesegmentering*

Det er kjent at metoden som er brukt for å segmentere trekroner, ikke registrerer mindre trær som er skjult under større.(Dalponte & Coomes, 2016) Dette er trolig hovedårsaken til at det kun er 2474 trekroner klassifisert som gran, på tross av at det er registrert hogd 4208 grantrær. Ved at de mindre trærne ikke blir tatt med, blir også andelen med råte økt. Etter koblingen er 21 % av trekronene registrert som råtne, mens kun 18 % av hogstpunktene hadde råte over 1 m.

Metoden fokuserer også på å kartlegge karboninnholdet i skogen.(Dalponte & Coomes, 2016) Det er uklart hvor effektiv denne metoden er for å kartlegge råte. Råte i trær er ikke omtalt i artikkelen som behandler denne metoden. Det er mulig at egenskapene som er benyttet for å optimalisere for karbonintensitet, gjør det vanskeligere å skille råtne og ikke råtne trær fra hverandre.

## 5.4 Feil under datainnsamling

Det må antas at det har oppstått feil i datainnsamlingen ved hogst. Det må også påpekes at det er dårlig dokumentasjon for hvordan hogstdataene er innsamlet. Slik dokumentasjon bør fremstilles. Samtlige metadata har vært muntlige. Det er også personvernutfordringer ved datasettene som gjør videre bruk problematisk.

## 5.5 Påvirkning fra ulike opptaksforhold og grunnforhold

Forskjellige opptaksforhold under fotograferingen av de hyperspektrale bildene vil påvirke muligheten for å overføre en forhåndstrent modell fra et område til et annet. Det er en rekke ulike kilder til støy, bl.a. hvor mye vekst det er på trærne og hvilken retning sollyset kommer fra. En masteroppgave ved NMBU (Sørhuus, 2019) påpekte at grunnforholdene kan påvirke radiansen i de hyperspektrale bildene, og dermed påvirke resultatet ved maskinlæring.

Dette medfører at en modell trent på et område trolig ikke vil gi like gode resultater for et annet område med andre opptaksforhold og grunnforhold.

## 5.6 Sammenligning med andre studier

Tidligere studier har kommet frem til at man kan finne råte ved hjelp av hyperspektrale flyfoto.(Jørgensen et al., 2008; Sørhuus, 2019) Men resultatene fra disse kan ikke sammenlignes med denne oppgaven, da det er vesentlig forskjell i metode for datainnsamling. I datasettet benyttet i denne oppgaven, har råten blitt registrert under hogst og behandlet som en høydeverdi, mens i andre studier har man som regel nøyaktig registrert posisjon og registrert råteinnhold på stubber i etterkant av hogst. Sistnevnte metode er mer ressurskrevende, men gir høyere nøyaktighet.

## 5.7 Behov for datakraft

Samtlige resultater er utarbeidet på datamaskinen beskrevet i kapittel 2.12. Kjøring av U-nett på metode 1 med 100 epoker tar 28,4 sekunder.

## 5.8 Videre arbeid

### 5.8.1 Kombinasjon av metode 1 og 2

En tilnærming som bør utforskes videre, er en kombinasjon av metode 1 og 2. F.eks. ved å operere med et bildeutsnitt for hvert hogstpunkt, og deretter predikere på et større område. For store datasett er det viktig at man legger opp til en arbeidsflyt med størst mulig automatisering. Dette er en av de store forskjellene mellom metode 1 og 2.

### 5.8.2 Forbedring av dokumentasjon av datasett

Under arbeidet med oppgaven er det blitt klart at metadata om datasett, da spesielt hogstdata fra Etnedal ikke er tilgjengelig. Det er også uklart om dataene er standardiserte eller om man bruker nye

skalaer fra prosjekt til prosjekt. Dette er elementer som bør forbedres med tanke på fremtidig gjenbruk av datasett og kombinasjon av flere prosjekt til større datasett.

### 5.8.3 Muligheter ved åpne og standardiserte data

Mye av utviklingen innen maskinlæring de siste årene har vært drevet frem av konkurranser på åpne datasett, og med krav om åpen og fri kildekode. En arena for slike konkurranser er Kaggle, drevet av Google.

Det må påpekes at innenfor maskinlæring er det ikke uvanlig at programkode og datasett publiseres uten artikkel eller endres vesentlig etter at artikkel er publisert. Et eksempel på dette er den mye brukte optimaliseringsfunksjonen RMSprop,(Tieleman & Hinton, 2012) der referansen er til en presentasjon.

### 5.8.4 Forhåndstrente nettverk og overføringstrening

Som nevnt i kapittel 2.10 vil man trolig i løpet av få år kunne benytte overføringstrening på forhåndstrente nettverk. Dette vil trolig løse problemet nevnt i kapittel 5.5.

Den lokale fasiten kan f.eks. fremskaffes ved en mindre hogst som gir samme type data som modellen er trent på. Som vist i metode 1, er det ikke nødvendig med mange trær for å gi et tydelig utslag. En annen tilnærming er samle data i sanntid fra hogstmaskiner, ved at hogstdata overføres direkte fra hogstmaskinen og mates kontinuerlig inn i et nevralt nettverk. Denne treningen kan trolig utføres på en bærbar datamaskin i felt. Metode 2 er lagt opp slik at denne kan automatiseres.

## 5.9 Konklusjon

I denne oppgaven har det blitt undersøkt i hvor stor grad forbehandling påvirker resultatet og om trening av dype nevrale nettverk for predikeringen av råte kan utføres på bærbare datamaskiner.

For å undersøke virkningen av forbehandling har det blitt testet to ulike metoder med ulike innfallsvinkler som så har blitt trent på et nevralt nettverk. I tillegg har alt arbeid med denne masteroppgaven inkl. trening av dype nevrale nettverk blitt utført på en bærbar datamaskin.

Resultatene viser at valg av forbehandling kan ha stor påvirkning på resultatet. Det er usikkert hva årsaken til det svært ulike resultatet mellom metodene skyldes, og det er omtalt en rekke mulige feilkilder. Trolig handler det om en rekke ulike kilder som hver og en påfører støy i forbehandlingen.

Da alle resultatene er fremskaffet ved kjøring på bærbar datamaskin, er det avklart at man kan trene dype nevrale nettverk for predikering av råte på en bærbar datamaskin. I denne oppgaven ble det for en av metodene oppnådd en vektet F1-score på 0,8, noe som var betydelig bedre enn grunnlinjen. Oppgaven har benyttet U-nett for trening og predikering, men dette resultatet vil trolig også gjelde andre arkitekturer med tilsvarende behov for datakraft. Dette åpner opp for fremtidig bruk av overføringstrening og sanntidstrening i felt. Trolig vil fremtidig bruk av overføringstrening kunne oppnå enda bedre resultater enn vist her.

# 6 Referanser

Buda, M. J. M., Mary, H. & Clark, M. (2016). *Dice score function*: GitHub. Tilgjengelig fra: https://github.com/keras-team/keras/issues/3611 https://github.com/jocicmarko/ultrasound-nerve-segmentation/blob/master/train.py.

Chicco, D. (2017). Ten quick tips for machine learning in computational biology. *BioData mining*, 10 (1): 35.

Dalen, L. S. (2018). *Ny kunnskap skal redde skog fra råte*. NIBIO: NIBIO. Tilgjengelig fra: https://www.nibio.no/nyheter/ny-kunnskap-skal-redde-skog-fra-rate.

Dalponte, M. & Coomes, D. A. (2016). Tree-centric mapping of forest carbon density from airborne laser scanning and hyperspectral data. *Methods in Ecology and Evolution*, 7 (10): 1236-1245. doi: 10.1111/2041-210x.12575.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). *Imagenet: A large-scale hierarchical image database*. 2009 IEEE conference on computer vision and pattern recognition: Ieee.

Fricker, G. A., Ventura, J. D., Wolf, J. A., North, M. P., Davis, F. W. & Franklin, J. (2019). A convolutional neural network classifier identifies tree species in mixed-conifer forest from hyperspectral imagery. *Remote Sensing*, 11 (19): 2326.

Gobakken, T. (2020). *Ulike samtaler i forbindelse med master*.

Hagen, K. & Hetle, J. (2019). *Obligatoriske innlevering i DAT300*.

Hetle, J. & Hagen, K. (2019). *Samtaler i forbindelse med arbeid med obliger i faget DAT300 ved NMBU*.

Hoyer, S. & Hamman, J. (2017). xarray: ND labeled arrays and datasets in Python. *Journal of Open Research Software*, 5 (1).

Hyspex. *HySpex SWIR-384*. Tilgjengelig fra: https://www.hyspex.com/hyspex-products/hyspex-classic/hyspex-swir-384/.

Hyspex. *HySpex VNIR-1800*. Tilgjengelig fra: https://www.hyspex.com/hyspex-products/hyspex-classic/hyspex-vnir-1800/.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jocić, M., Buda, M., Mary, H. & Clark, M. (2016). *Dice score function*: GitHub. Tilgjengelig fra: https://github.com/keras-team/keras/issues/3611 https://github.com/jocicmarko/ultrasound-nerve-segmentation/blob/master/train.py.

Jørgensen, N. E., Solberg, S., Baarstad, I., Wangensteen, B., Bergsaker, E. & Solheim, H. (2008). *Registrering skogskader med hyperspektral skanner*: Norskog. Upublisert manuskript.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Leica. (2008). *Leica ALS70-HP*: Leica Geosystems AG. Tilgjengelig fra: http://w3.leica-geosystems.com/downloads123/zz/airborne/ALS70/product-specification/ALS70_HP_ProductSpecs_en.pdf.

Masarczyk, W., Głomb, P., Grabowski, B. & Ostaszewski, M. (2019). Effective transfer learning for hyperspectral image classification with deep convolutional neural networks. *arXiv preprint arXiv:1909.05507*.

Moneda, L. (2017). *From image files to numpy arrays!*: Kaggle. Tilgjengelig fra: https://www.kaggle.com/lgmoneda/from-image-files-to-numpy-arrays.

NIBIO. (2017). *Gran*: NIBIO. Tilgjengelig fra: https://www.nibio.no/tema/skog/skoggenetiske-ressurser/treslag-i-norge/gran.

NMBU. (2018-2021). *Presisjonsskogbruk for bedre ressursutnyttelse og redusert råte i norske skoger (PRECISION)*. Tilgjengelig fra: https://www.nmbu.no/prosjekter/node/38805.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Louppe, G., Prettenhofer, P. & Weiss, R. (2012). Scikit-learn: machine learning in python. arXiv. *arXiv preprint arXiv:1201.0490*.

Raschka, S. & Mirjalili, V. (2017). *Python Machine Learning*. 2. utg.: Packt Publishing Ltd.

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention, Pt Iii*, 9351: 234-241. doi: 10.1007/978-3-319-24574-4_28.

Ryvarden, L. (2019). *Råte*. Store norske leksikon. Tilgjengelig fra: https://snl.no/r%C3%A5te.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15 (1): 1929-1958.

Sylvain, J.-D., Drolet, G. & Brown, N. (2019). Mapping dead forest cover using a deep convolutional neural network and digital aerial photography. *ISPRS Journal of Photogrammetry and Remote Sensing*, 156: 14-26. doi: https://doi.org/10.1016/j.isprsjprs.2019.07.010.

Sørhuus, Ø. (2019). *Bruk av flybåren hyperspektral sensor for påvisning av råte i stående granskog (Picea abies)*. Masteroppgave. Ås: Norges miljø- og biovitenskapelige universitet. Tilgjengelig fra: http://hdl.handle.net/11250/2601373.

Tieleman, T. & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2): 26-31.

Voskoglou, C. (2017). *What is the best programming language for Machine Learning?* Towards Data Science. Tilgjengelig fra: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7.

# Figurliste

Figur 1: Oppbyggingen av et kunstig nevron i et nettverk med klassifisering. Merk at Unit step function ikke er relevant her. Figur: Raschka og Mirjalili (2017) ........ 4
Figur 2: Strukturen på det nevrale nettverket. Figur: MultiLayerNeuralNetwork_english.png: Chrislb derivative work: — HELLKNOWZ ▎TALK ▎enWP TALK (https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetworkBigger_english.png), „MultiLayerNeuralNetworkBigger english“, https://creativecommons.org/licenses/by-sa/3.0/legalcode ........ 5
Figur 3: Illustrasjon av læringskurver, med punktet der overtilpasning inntreffer markert. Rød kurve viser utviklingen av feilen for valideringsdata over flere epoker. Etter at overtilpasning inntreffer, begynner stiger feilen for valideringsdata, men feilen for treningsdata fortsetter å synke. Figur: Gringer (https://commons.wikimedia.org/wiki/File:Overfitting_svg.svg), „Overfitting svg“, https://creativecommons.org/licenses/by/3.0/legalcode ........ 6
Figur 4: Ulike typer padding. Figur: Raschka og Mirjalili (2017) ........ 7
Figur 5: Illustrasjon av max- og mean-polling i størrelsen 3x3. Stride 3x3 betyr at filteret beveger seg i steg på 3 piksler. Figur: Raschka og Mirjalili (2017) ........ 7
Figur 6: Glissenkoblinger. Filteret passerer over bildet og danner et nytt egenskapskart. Figur: Raschka og Mirjalili (2017) ........ 8
Figur 7: Strukturen til U-nett. Figur: Ronneberger et al. (2015) ........ 9
Figur 8: Området vist på flyfoto fra Norge i Bilder. Blå prikker angir hogde trær. Kartdata: © Kartverket ........ 13
Figur 9: Kart over det omkringliggende området. Blå prikker markerer hogstområdet. Kartdata: © Kartverket ........ 13
Figur 10: Kombinasjon av SWIR og VNIR som dekker hogstområdet. Det er tydelige forstyrelser i kanten av flystripene. ........ 14
Figur 11: Manuelt korrigerte trepunkt vises som røde diamanter. Trepunkt fra datasettet vist med grønne sirkler. ........ 15
Figur 12: Polygon i rødt overlapper med 2 nabopolygoner. ........ 18
Figur 13: Eksempel på at trepunkt og trekrone ikke overlapper. ........ 19
Figur 14: Visning i QGIS etter at rutenett er laget ........ 20
Figur 15: Grønne trekronepolygoner blir splittet av rutenett. Sorte og røde streker viser hvordan mindre rutestørrelser fører til at flere polygnoner blir delt. ........ 31
Figur 16: Til venstre: To trær som er segmentert som en trekrone. Til høyre: Trekrone som kun dekker deler av treet. Grønt omriss viser segmentert trekrone. Grønne prikker viser hogstpunkt med angitt høyde. ........ 32
Figur 17: Flere trær segmentert som en trekrone. Røde tall viser trærnes høyde i meter. Sorte tall er råtehøyde i millimeter. ........ 33
Figur 18: Utsnitt som viser flere feil som opptrer innenfor et mindre område. ........ 33

# Tabellliste

Tabell 1: Sammenligning av beste vekter ved ulike parametere ved metode 1. Resultatene er oppgitt fra tapsfunksjonen på valideringsdatasettet ........ 25
Tabell 2: Sammenstilling av beste resultat fra metode 1 med og uten flipp. ........ 26
Tabell 3: Utskrift av model_summary for beste parametere i metode 1. ........ 26
Tabell 4: Sammenligning av beste vekter ved ulike parametere. Resultatene er oppgitt fra tapsfunksjonen på valideringsdatasettet. ........ 27
Tabell 5: Sammenstilling av beste resultat med og uten flipp og ulike rutenett. ........ 28
Tabell 6: Utskrift av model_summary for beste parametere. ........ 28

# Vedlegg

Vedlegg 1: Forbehandling metode 2

Vedlegg 2: U-nett for metode 2

Vedlegg 3: Funksjoner metode 2

Vedlegg 4: Rapport for hyperspektral datainnsamling og prosessering

## Vedlegg 1: Forbehandling metode 2

```
# -*- coding: utf-8 -*-
"""
Vedlegg 1: Forbehandling metode 2
Created on Thu Feb 13 13:50:47 2020

@author: Kristoffer1
"""

import os
import xarray as xr
from funksjoner import A_A

#%% Setup
dirname = 'filepath'
Rutenett = '9-6'
imagefolder = 'Grid'+Rutenett
labellayer = 187

#%% Import files
imglist = []
labellist = []
imgidlist = []

filenames = os.listdir(dirname + '\\' + imagefolder)

for name in filenames:
    if name[-3:] == 'tif':
        imgid = name[:-4]

        img = xr.open_rasterio(os.path.join(dirname + '\\' + imagefolder +
'\\' + name)).transpose('y','x','band')

        label = img.sel(band=[labellayer])
        img = img.drop_sel(band=[labellayer])

        imglist.append(img)
        labellist.append(label)

#%% Calculate window
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

X_train, X_test, y_train, y_test = train_test_split(imglist, labellist,
test_size=0.1, random_state=1)
train_imgidlist, test_imgidlist = train_test_split(imgidlist,
test_size=0.1, random_state=1)

X_train_a, imwidth, imheight, imbands = A_A(X_train)
X_test_a, imwidth, imheight, imbands = A_A(X_test)
y_train_a, imwidth, imheight, imbands =A_A(y_train)
y_test_a, imwidth, imheight, imbands = A_A(y_test)

scaler = MinMaxScaler()
y_train_a_scaled = scaler.fit_transform(y_train_a.reshape(-1,
1)).reshape(y_train_a.shape)
y_test_a_scaled = scaler.transform(y_test_a.reshape(-1,
1)).reshape(y_test_a.shape)

#%% Pickle
```

```
import pickle

pickle.dump(X_train_a, open('X_train_a'+Rutenett+'-2.pkl', 'wb'))
pickle.dump(y_train_a_scaled, open('y_train_a'+Rutenett+'-2.pkl', 'wb'))
pickle.dump(scaler, open('scaler'+Rutenett+'-2.pkl', "wb"))
pickle.dump(train_imgidlist, open('train_imgidlist'+Rutenett+'-2.pkl',
"wb"))

pickle.dump(X_test_a, open('X_test_a'+Rutenett+'-2.pkl', "wb"))
pickle.dump(y_test_a, open('y_test_a'+Rutenett+'-2.pkl', "wb"))
pickle.dump(test_imgidlist, open('test_imgidlist'+Rutenett+'-2.pkl', "wb"))
```

## Vedlegg 2: U-nett for metode 2

```
# -*- coding: utf-8 -*-
"""
Vedlegg 2: U-nett for metode 2

Created on Sat Feb 15 13:30:01 2020

@author: Kristoffer Hagen
"""
from funksjoner import train_val_split

batch_size = 32 # 32 ved 9-6, 64 ved 4-8.
epochs = 100
Rutenett = '9-6'

#%% Load pickle
import pickle

with open('X_train_a'+Rutenett+'-2.pkl', 'rb') as f:
    dataset = pickle.load(f)

with open('y_train_a'+Rutenett+'-2.pkl', 'rb') as g:
    label_patches = pickle.load(g)

with open('scaler'+Rutenett+'-2.pkl', 'rb') as e:
    scaler = pickle.load(e)

imshape = dataset.shape
imwidth = imshape[1]
imheight = imshape[2]
imbands = imshape[3]

#%% Train val split

X_train, X_val, y_train, y_val = train_val_split(dataset, label_patches)

#%%
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import math

traindata = ImageDataGenerator(
    # vertical_flip=True,
    # horizontal_flip=True,
    data_format='channels_last')
traindata.fit(X_train)

steps_per_epoch = (math.ceil(len(X_train) / batch_size))
validation_steps = (math.ceil(len(X_val) / batch_size))

#%% Basert på Dat300 øving
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Activation, Conv2DTranspose
from tensorflow.keras import backend as K
import numpy as np

def dice_coef(y_true, y_pred, smooth=1):
```

```
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) +
K.sum(y_pred_f) + smooth)

def dice_coef_loss(y_true, y_pred):
    return 1-dice_coef(y_true, y_pred)


def conv2d_block(input_tensor, n_filters, kernel_size = None, batchnorm =
True):
    """Function to add 2 convolutional layers with the parameters passed to
it"""
    # first layer
    x = Conv2D(filters = n_filters, kernel_size = (kernel_size,
kernel_size),\
              kernel_initializer = 'he_normal', padding =
'same')(input_tensor)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # second layer
    x = Conv2D(filters = n_filters, kernel_size = (kernel_size,
kernel_size),\
              kernel_initializer = 'he_normal', padding = 'same')(x)
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x

def get_unet(input_img, n_filters = 16, dropout = None, batchnorm = True,
kernel_size=3):
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size = kernel_size,
batchnorm = batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size = kernel_size,
batchnorm = batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size = kernel_size,
batchnorm = batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size = kernel_size,
batchnorm = batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters = n_filters * 16, kernel_size =
kernel_size, batchnorm = batchnorm)

    # Expansive Path
```

```
    u6 = Conv2DTranspose(n_filters * 8, (kernel_size, kernel_size), strides
= (2, 2), padding = 'same')(c5)
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters * 8, kernel_size = kernel_size,
batchnorm = batchnorm)

    u7 = Conv2DTranspose(n_filters * 4, (kernel_size, kernel_size), strides
= (2, 2), padding = 'same')(c6)
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 4, kernel_size = kernel_size,
batchnorm = batchnorm)

    u8 = Conv2DTranspose(n_filters * 2, (kernel_size, kernel_size), strides
= (2, 2), padding = 'same')(c7)
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 2, kernel_size = kernel_size,
batchnorm = batchnorm)

    u9 = Conv2DTranspose(n_filters * 1, (kernel_size, kernel_size), strides
= (2, 2), padding = 'same')(c8)
    u9 = concatenate([u9, c1])
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 1, kernel_size = kernel_size,
batchnorm = batchnorm)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model

#%% Forløkke for testing
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt
plt.ioff()

dropoutlist = [0.0, 0.1, 0.2]
optimizerlist = ['RMSprop', 'Adam']
learning_ratelist = [0.001, 0.0001, 0.00001]

best_score = 0
best_score_param = ''

for dropout in dropoutlist:
    for optimizer in optimizerlist:
        for lr in learning_ratelist:
            inputs = Input(shape=(imwidth,imheight,imbands))
            model = get_unet(inputs, dropout=dropout)

            model.compile(optimizer=optimizers.get({"class_name":
optimizer,"config": {"learning_rate": lr}}),
                          loss=dice_coef_loss,
                          metrics=[dice_coef])

            filepath='filepath'+Rutenett+'/u-net_sigmoid_2-weights-
{epoch:02d}-{val_dice_coef:.2f}'-'+str(Rutenett)+'-'+str(dropout)+'-
'+str(optimizer)+'-'+str(lr)+'.hdf5'
            checkpoint = ModelCheckpoint(filepath, monitor='val_dice_coef',
verbose=0, save_best_only=True, mode='max', save_weights_only=True)
```

```
            callbacks_list = [checkpoint]

            history = model.fit_generator(traindata.flow(
                    X_train, y_train, batch_size=batch_size),
                epochs=epochs, steps_per_epoch=steps_per_epoch,
                validation_data=(np.array(X_val),np.array(y_val)),
                validation_steps=validation_steps,
callbacks=callbacks_list)

            if max(history.history['val_dice_coef']) > best_score:
                best_score = max(history.history['val_dice_coef'])
                best_score_param = str('u_net-sigmoid_2-'+str(dropout)+'-
'+str(optimizer)+'-'+str(lr))

            plt.plot(history.history['dice_coef'])
            plt.plot(history.history['val_dice_coef'])
            plt.plot(history.history['loss'])
            plt.plot(history.history['val_loss'])
            plt.title("model accuracy")
            plt.ylabel("Accuracy")
            plt.xlabel("Epoch")
            plt.legend(["Accuracy","Validation Accuracy","loss","Validation
Loss"])
            figfile= str('filepath'+Rutenett+'/u_net-sigmoid_2-
'+str(dropout)+'-'+str(optimizer)+'-'+str(lr)+'.png')
            plt.savefig(figfile)
            plt.clf()

            K.clear_session()

#%% Load checkpoint
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

dropout = 0.1
optimizer = optimizers.get({"class_name": 'RMSprop',"config":
{"learning_rate": 0.001}})
model.load_weights('filename)
cutoff = 100

inputs = Input(shape=(imwidth,imheight,imbands))
model = get_unet(inputs, dropout=dropout)

model.compile(optimizer=optimizer,
              loss=dice_coef_loss,
              metrics=[dice_coef])

cutoff_scaled = scaler.transform(np.array(cutoff).reshape(1, -1))

predicted = model.predict(X_val)
predicted = np.where(predicted > cutoff_scaled, 1, 0)
predicted = predicted[:, :, :,0]
predicted = np.reshape(predicted,[len(X_val)*imwidth*imheight,1])

truth = y_val
truth = np.where(truth > cutoff_scaled, 1, 0)
truth = np.reshape(truth,[len(y_val)*imwidth*imheight,1])

cm = confusion_matrix(truth,predicted)
# print(cm)
```

```
target_names = ['Råte under 1 m', 'Råte over 1 m']
print(classification_report(truth, predicted, target_names=target_names))

p = model.predict(X_val)[:, :, :,0]
t = y_val[:, :, :,0]
rows = 10
nrows = math.ceil(len(range(1,rows))/2)
Frows = math.floor(len(range(1,rows))/2)
fig, axes = plt.subplots(nrows=nrows, ncols=4, figsize=(7, 7))
i = 0

for plotnum in range(Frows):

    axes[i, 0].matshow(p[plotnum])
    axes[i, 0+2].matshow(p[plotnum+Frows])
    axes[i, 1].matshow(t[plotnum])
    axes[i, 1+2].matshow(t[plotnum+Frows])

    i += 1

if rows > Frows:
    axes[i, 0].matshow(p[plotnum+1])
    axes[i, 1].matshow(t[plotnum+1])

fig.tight_layout()
plt.show()

#%% Test
from tensorflow.keras import optimizers
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import pickle
import math
plt.ioff()

Rutenett = '9-6'

with open('X_test_a'+Rutenett+'-2.pkl', 'rb') as h:
    test_dataset = pickle.load(h)

with open('y_test_a'+Rutenett+'-2.pkl', 'rb') as i:
    test_label = pickle.load(i)

with open('scaler'+Rutenett+'-2.pkl', 'rb') as e:
    scaler = pickle.load(e)

dropout = 0.2
optimizer = optimizers.get({"class_name": 'RMSprop',"config":
{"learning_rate": 0.001}})
model.load_weights('filename')
cutoff = 100

imshape = test_dataset.shape
imwidth = imshape[1]
imheight = imshape[2]
imbands = imshape[3]

inputs = Input(shape=(imwidth,imheight,imbands))
model = get_unet(inputs, dropout=dropout)
```

```python
model.compile(optimizer=optimizer,
              loss=dice_coef_loss,
              metrics=[dice_coef])

cutoff_scaled = scaler.transform(np.array(cutoff).reshape(1, -1))

predicted = model.predict(test_dataset)
predicted = np.where(predicted > cutoff_scaled, 1, 0)
predicted = predicted[:, :, :,0]
predicted = np.reshape(predicted,[len(test_dataset)*imwidth*imheight,1])

truth = test_label
truth = np.where(truth > cutoff, 1, 0)
truth = np.reshape(truth,[len(test_label)*imwidth*imheight,1])

cm = confusion_matrix(truth,predicted)
# print(cm)

target_names = ['Råte under 1 m', 'Råte over 1 m']
print(classification_report(truth, predicted, target_names=target_names))

p = model.predict(test_dataset)[:, :, :,0]
t = test_label[:, :, :,0]

start = 0
rows = 16
nrows = math.ceil(len(range(0,rows))/2)
Frows = math.floor(len(range(0,rows))/2)
pmax = p.max()
pmin = p.min()
tmax = t.max()
tmin = t.min()
fig, axes = plt.subplots(nrows=nrows, ncols=4, figsize=(7, 7))
i = 0

for plotnum in range(Frows):

    axes[i, 0].matshow(p[plotnum+start], vmin=pmin ,vmax=pmax)
    # axes[i, 0].matshow(p[plotnum+start])
    axes[i, 0+2].matshow(p[plotnum+start+Frows], vmin=pmin ,vmax=pmax)
    # axes[i, 0+2].matshow(p[plotnum+start+Frows])
    axes[i, 1].matshow(t[plotnum+start], vmin=tmin ,vmax=tmax)
    # axes[i, 1].matshow(t[plotnum+start])
    axes[i, 1+2].matshow(t[plotnum+start+Frows], vmin=tmin ,vmax=tmax)
    # axes[i, 1+2].matshow(t[plotnum+start+Frows])

    i += 1

if nrows > Frows:
    axes[i, 0].matshow(p[plotnum+start+1], vmin=pmin ,vmax=pmax)
    # axes[i, 0].matshow(p[plotnum+start+1])
    axes[i, 1].matshow(t[plotnum+start+1], vmin=tmin ,vmax=tmax)
    # axes[i, 1].matshow(t[plotnum+start+1])

fig.tight_layout()
plt.show()
```

## Vedlegg 3: Funksjoner metode 2

```
# -*- coding: utf-8 -*-
"""
Vedlegg 3: Funksjoner metode 2
Created on Thu Feb 13 13:47:57 2020

@author: Kristoffer Hagen
"""

#%% Array and Augmentation

def A_A(image_patches=None,override=None):
    import numpy as np
    from tensorflow.keras.preprocessing.image import img_to_array

    if override == None:
        imshape = image_patches[0].shape
    else:
        imshape = override
    imwidth = imshape[0]
    imheight = imshape[1]
    imbands = imshape[2]

    dataset = np.ndarray(shape=(len(image_patches), imwidth, imheight, 
imbands),
                         dtype=np.float32)
    i = 0
    for file in image_patches:
        x = img_to_array(file,data_format=None)
        dataset[i] = x
        i += 1
        if i % 250 == 0:
            print("%d images to array" % i)
    print("All %d images to array!" % i)
    return dataset, imwidth, imheight, imbands


#%% Splitting

def train_val_split(dataset, label_patches, val_size = 0.1, 
random_state=33):
    from sklearn.model_selection import train_test_split
    import numpy as np

    X_train, X_val, y_train, y_val = train_test_split(dataset, 
label_patches, test_size=val_size, random_state=random_state)
    return X_train, X_val, y_train, y_val
```

# Rapport for hyperspektral datainnsamling og prosessering

## Terratec AS

| | |
|---|---|
| **Prosjektnavn:** | **Etnedal hyperspec. + lidar 2019** |
| **Prosjektnummer:** | **11163** |
| | |
| **Dekningsnavn:** | **Etnedal hyperspec. + lidar 2019** |
| **Dekningsnummer:** | **41154** |

# INNHOLDSFORTEGNELSE

1 GENERELLE OPPLYSNINGER I PROSJEKTET ..... 3
1.1 Oppdragsgiver ..... 3
1.2 Oppdraget ..... 3
1.3 Oppdragstaker ..... 3
1.4 Oppdragets innhold ..... 3
1.5 Kvalitetssikring ..... 3
2 DATAINNSAMLING ..... 4
2.1 Utførelse ..... 4
2.2 Operativ flyplan ..... 4
2.3 Måleutstyr ..... 6
2.4 Grunnlag/datum ..... 6
2.5 Transformasjoner ..... 6
2.6 Utfordringer i datainnsamlingen ..... 6
3 PROSESSERING/DATABEARBEIDING ..... 7
3.1 Beregnings- og editeringsarbeid ..... 7
3.2 GNSS/INS-beregninger ..... 7
3.3 Ortorektifisering av hyperspektrale data ..... 7
4 LEVERANSER ..... 8
4.1 Oppdeling, navngivning og lagring av filer ..... 8

Rapport utarbeidet,

Oslo, 27.08.2019

Vetle O. Jonassen

Produksjonsansvarlig for hyperspektrale data

# 1 GENERELLE OPPLYSNINGER I PROSJEKTET

## 1.1 Oppdragsgiver

| | |
|---|---|
| Navn: | Forskningsprosjekt Precision v/ NMBU |
| Besøksadresse: | Universitetstunet 3 |
| Postadresse: | 1430 Ås |
| Ansvarlig for arbeidspakke: | Terje Gobakken |

## 1.2 Oppdraget

| | |
|---|---|
| Navn: | Etnedal hyperspec. + lidar 2019 |
| Fylke: | Oppland |

## 1.3 Oppdragstaker

| | |
|---|---|
| Navn: | Terratec AS |
| Besøksadresse: | Vækerøveien 3 |
| Postadresse: | 0281 Oslo |
| Prosjektleder: | Vetle Jonassen |
| Prosjektreferanse: | 11163 |

## 1.4 Oppdragets innhold

Oppdraget innebærer georefererte og ortorektifiserte hyperspektrale bildekuber med radiansverdier med 0,3 meter oppløsning for HySpex VNIR-sensoren og 0,7 meter oppløsning for HySpex SWIR-sensoren. Bestilt områdedekning er på totalt 10,2 $km^2$.

## 1.5 Kvalitetssikring

I forbindelse med flyfotograferingen blir det fotograferte området kontrollert slik at det samsvarer med flyplanen. Operatøren i flyet kontrollerer at bilder har riktig eksponering, og at de ikke inneholder feil, som skygger og skyer, ut over krav. Etter fotografering kontrolleres bildedekning på nytt, eventuelle skyer og skygger registrert av operatør, generell bildekvalitet og at GNSS/INS-data er riktig logget. Det blir gjort kvalitetssjekk av GNSS-dataene brukt for georeferering av de hyperspektrale dataene, samt en avsluttende kvalitetssjekk av den totale georefereringen.

# 2 DATAINNSAMLING

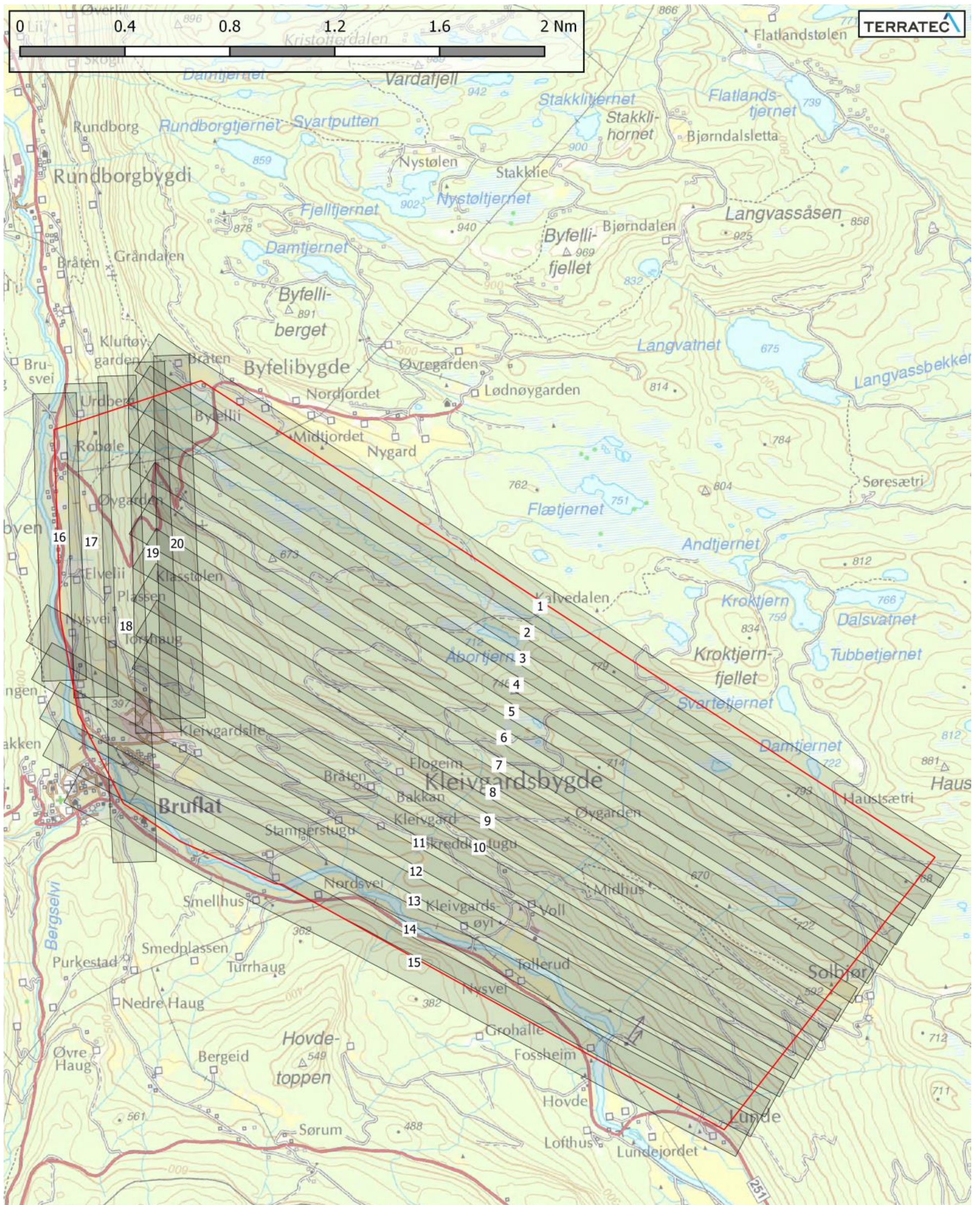
## 2.1 Utførelse

Terratec AS har gjennomført den hyperspektrale datainnsamlingen i følgende operasjoner:

| Stripenr | Operatør | Dato |
|---|---|---|
| **1-20** | Ainar Härm | 03.08.2019 |

## 2.2 Operativ flyplan

Det hyperspektrale systemet bestod av to sensorer. Begge sensorer ble montert i gyroramme, og flyplanen ble lagt for 16 grader etter maks åpningsvinkel for HySpex SWIR-sensoren.



Figur 1: Flyplan brukt for datainnsamling.

## 2.3 Måleutstyr

Kalibreringsparametre for de hyperspektrale sensorene er gitt av Norsk Elektro Optikk AS.

Spesifikasjoner

| | |
|---|---|
| Hyperspektral avb: | HySpex SWIR-384 og HySpex VNIR-1800 |
| IMU: | Micro IRS IE-IPAS-uIRS |
| GNSS-mottaker: | Topcon Legacy E |
| Plattform: | FW (LN-TTC) |
| Høyde over terreng: | 1150 m |
| Maks flyhastighet: | 130 kt |
| Åpningsvinkel: | ±8 grader brutto for SWIR |
| | ±8,5 grader brutto for VNIR |
| Sideoverlapp: | ca 23 % for SWIR |
| | ca 23 % for VNIR |

## 2.4 Grunnlag/datum

Horisontalt: WGS84 UTM sone 32

## 2.5 Transformasjoner

GNSS-beregning i TerraPOS er utført i WGS84. Transformasjon er utført for levering i de datum prosjektet skal leveres i:
Transformasjon WGS84 UTM32

## 2.6 Utfordringer i datainnsamlingen

Området var preget av begrensede værvindu for datainnsamling. Dette bød på utfordringer i datainnsamlingen, og flere innsamlingsforsøk ble gjort. Ettersom det var beregnet at hele datainnsamlingen var mulig å gjennomføre på én dag, ble dette tatt hensyn til slik at data fra avbrutte forsøk på innsamling ble forkastet. Dette sørget for et mest mulig homogent datasett.

# 3 PROSESSERING/DATABEARBEIDING

## 3.1 Beregnings- og editeringsarbeid

Programvarene som ble brukt for dataprosessering og deres funksjon er listet i tabellen under:

| Program/versjon | Produsent | Funksjon |
|---|---|---|
| MissionPro | Leica Geosystems | Flyplanlegging |
| TerraPOS | Terratec | GNSS/IMU-prosessering |
| IPAS Pro | Leica Geosystems | GNSS/INS-integrasjon |
| HySpex_RAD | Norsk Elektro Optikk | Filkonvertering til spektral intensitet |
| Parge 3.4 | ReSe | Georeferering og ortorektifisering |

## 3.2 GNSS/INS-beregninger

Blending av GPS og IMU-data er utført med kalmanfilter i Leicas programvare IPAS-PRO v 15.

## 3.3 Ortorektifisering av hyperspektrale data

De hyperspektrale dataene ble ortorektifisert i Parge 3.4 basert på en 30 cm DOM samlet inn fra en ALS70 laserskanner 03.08.2019. Georefereringen og ortorektifiseringen er gjort med nærmeste-nabo-interpolasjon.

I tillegg ble det påført en siste korreksjon av heading, roll og pitch ved manuelt utvalg av naturlige kontrollpunkter mellom to kryssende flystriper.

Korreksjonene er gitt i tabellen under:

| Sensor | Heading-korreksjon | Roll-korreksjon | Pitch-korreksjon |
|---|---|---|---|
| SWIR-384 | 0,529° | 0,404° | 0,316° |
| VNIR-1800 | 0,248° | -0,526° | 0,240° |

Korreksjonene var basert på kontrollpunkt på bakken med følgende restfeil:

| Sensor | X gjennomsnitt (m) | Y gjennomsnitt (m) | X standardavvik (m) | Y standardavvik (m) |
|---|---|---|---|---|
| SWIR | 0.00 | 0.00 | 0.24 | 0.41 |
| VNIR | 0.00 | 0.00 | 0.11 | 0.11 |

# 4 LEVERANSER

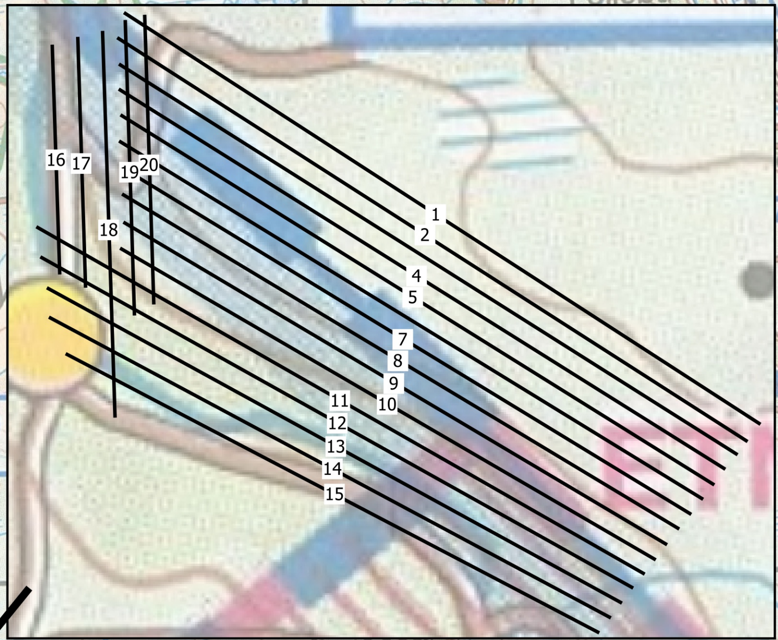
## 4.1 Oppdeling, navngivning og lagring av filer

Navn på de hyperspektrale filene følger oppsettet:
*Flylinjenummer*_*Sensornavn*_*Serienummer*_*Flisnummer*.dat

| Filformat | Beskrivelse |
|---|---|
| #.dat | Georefererte og ortorektifiserte hyperspektrale data lagret som BSQ-bildefiler. |
| #.hdr | Tilhørende tittelfiler for de respektive hyperspektrale filene |

Leveransefiler for hyperspektrale data er inndelt i.h.t. flystripe for hver av HySpex SWIR- og VNIR-sensorene og komprimert på .zip-format. Videre er flystripene delt inn i 15 fliser per flylinje for enklere datahåndtering.

Data levert på ekstern disk 27.8.2019. Dette inkluderer prosjektrapport, hyperspektrale bildekuber og laserdata.



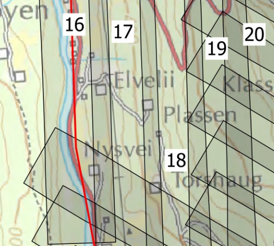

Planner/Date: MP/20.06.19

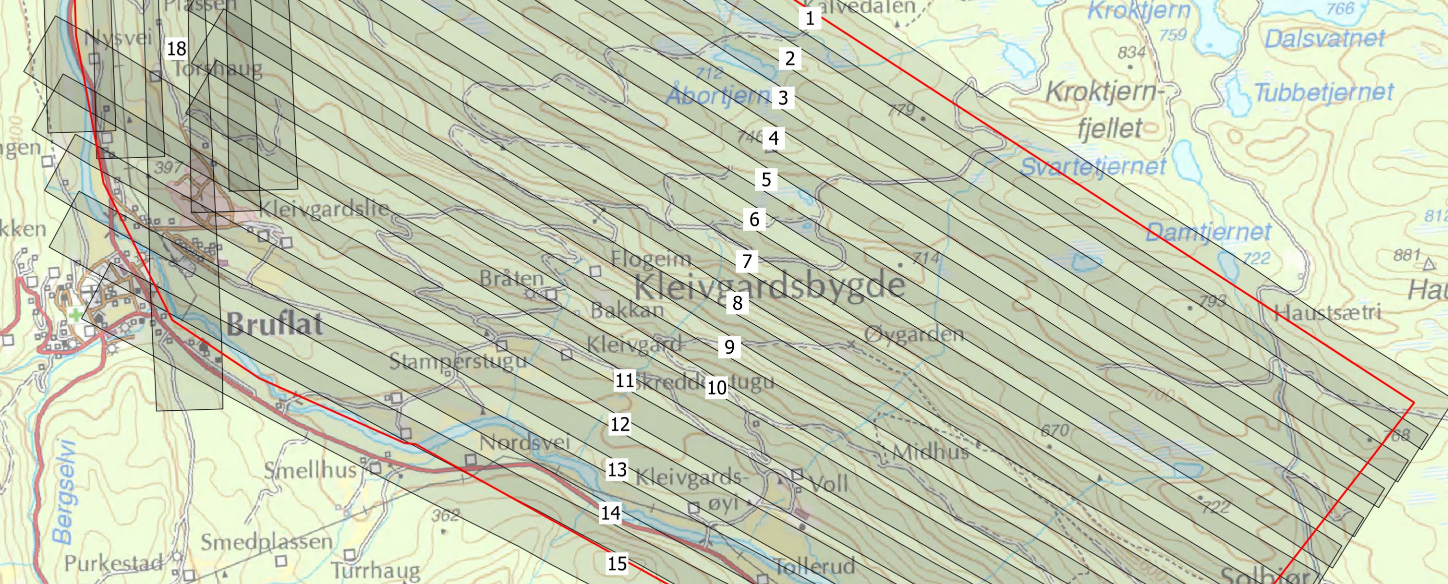
41154_01_01_Etnedal_hyperspec.+lidar_2019_FW_L204

Leica ALS70-HP - Hyspex

| | | | | | |
|---|---|---|---|---|---|
| Flying height MSL: | 4596- 5771 ft | FOV: | 16 ° | Camera: | Hyspex VNIR/SWIR |
| Flying altitude AGL: | 1150 m | Laser pulserate: | 495.2 kHz | Image GSD: | 35/80 cm |
| Total flightlines: | 20 | Scan rate: | 68.9 hz | Image lat. overlap: | 23-37 |
| Total length: | 55.9 nm | Min. pt density: | 17.6 pt/m2 | Max GND speed: | 130 kts |
| Total images: | n/a | Laser power: | 100 % | Min. Sun angle: | 30° |
| Time on block: | 1:26 | Strip width: | 323 m | | |
| Area: | 10.2 km2 | Laser lateral overlap: | 23-37 % | | |

TERRATEC




Rundborgbygdi
Byfelibygde
Byfelli-
berget
Byfelli-
fjellet
Langvassåsen
Langvatnet
Flætjernet
Kroktjern-
fjellet
Dalsvatnet
Svartetjernet
Damtjernet
Hovde-
toppen
Storstein
Skardtjernet
Beitesmyran
Karussputten
Nøkletjernet
Trondsætri
Rustbakk-
berget
Lundejordet
Lofthus
Hovde
Sørum
Bergeid
Purkestad
Smedplassen
Turrhaug
Nedre Haug
Øvre
Haug
Bergselvi
251
33

Run details: 41154_01_01_Etnedal_hyperspec.+lidar_2019_FW_L204

TERRATEC

| Flight Line Label | Min ref Height [m] | Alt MSL [feet] | Line Direction [deg] | Length [nm] |
|---|---|---|---|---|
| 1 | 609 | 5771 | 122.9 | 3.6 |
| 2 | 548 | 5571 | 122.7 | 3.6 |
| 3 | 532 | 5518 | 122.3 | 3.5 |
| 4 | 532 | 5518 | 122.1 | 3.4 |
| 5 | 530 | 5512 | 122 | 3.4 |
| 6 | 508 | 5440 | 121.5 | 3.3 |
| 7 | 524 | 5492 | 121.4 | 3.2 |
| 8 | 471 | 5318 | 121 | 3.1 |
| 9 | 387 | 5043 | 120.7 | 3 |
| 10 | 334 | 4869 | 120.2 | 3 |
| 11 | 307 | 4780 | 119.7 | 3.4 |
| 12 | 284 | 4705 | 119.2 | 3.3 |
| 13 | 277 | 4682 | 118.8 | 3.2 |
| 14 | 266 | 4646 | 118.2 | 3.1 |
| 15 | 251 | 4596 | 117.6 | 2.9 |
| 16 | 344 | 4902 | 178.1 | 1.1 |
| 17 | 341 | 4892 | 178.2 | 1.2 |
| 18 | 332 | 4862 | 178.1 | 1.9 |
| 19 | 400 | 5085 | 178.2 | 1.4 |
| 20 | 426 | 5171 | 178.2 | 1.4 |

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

Postboks 5003
NO-1432 Ås
Norway