Norges miljø- og biovitenskapelige universitet

Masteroppgave 2020 30 stp

Fakultet for realfag og teknologi
Håvard Tveite

Klassifisering av arealtyper i satellittbilder ved bruk av konvolusjonelle nevrale nettverk

Prediction of Land Cover Classes from Satellite Images using Convolutional Neural Networks

Rebecca Seim Brekke
Geomatikk
Fakultet for realfag og teknologi

Forord

Oppgaven er en avslutning på en femårig mastergrad i Geomatikk ved Norges miljø- og biovitenskapelige universitet (NMBU), som er gjennomført i perioden 2015-2020. Oppgaven ble skrevet våren 2020 og har et omfang på 30 studiepoeng. En stor takk til førsteamanuensis Håvard Tveite, som har vært veileder på oppgaven. Dine faglige innspill og råd har vært til stor hjelp.

En spesiell takk til min kjæreste, Håvard Bergheim, for støtte og tålmodighet under arbeidet med oppgaven og god hjelp til korrektur av oppgaven. Til slutt ønsker jeg å takke alle mine medstudenter for fem fantastiske år på Ås med uforglemmelige minner.

Ås, 31.05.2020.

Rebecca Seim Brekke

Sammendrag

I denne masteroppgaven har jeg undersøkt muligheten for nøyaktig segmentering av satellittbilder fra Sentinel-2 ved bruk av konvolusjonelle nevrale nettverk. Dette er for å utforske muligheten for å anvende maskinlæring til å predikere arealressurskart.

Ved å bruke offentlig tilgjengelig satellittbilder fra Sentinel-2 sammen med arealressurskart i målestokk 1:5000 foreslås det et datasett. Dette brukes for å trene to dype konvolusjonelle nevrale nettverk for å klassifisere fire arealtyper: Bebyggelse, jordbruk, skog og vann. Dette blir løst på to måter; ved flerklasseklassifisering og som ett binært datasett for hver av de fire arealklassene. Prediksjonen fra de binære datasettene fusjoneres sammen og sammenlignes med prediksjonen gjort med flerklassedatasettet. Resultatet viser at U-net-4 med flerklasseklassifisering gir det det beste resultatet.

Videre blir et arealressurskart generert basert på et satellittbilde over Nittedal, for å se hvordan nettverket presterer og kan brukes i praksis. Resultatet her viser at U-net-4 med fusjonert binær klassifisering gir det beste resultatet.

Abstract

In this masters thesis I have explored the possibility of accurately segmenting satellite images from Sentinel-2 using convolutional neural networks. This is done to explore the possibility of using machine learning to predict land cover classes.

By using publicly available satellite images from Sentinel-2 with land cover classes from AR5, a dataset is proposed. The dataset is used to train two deep convolutional neural networks to classify four land cover classes: Settlement, agriculture, forests and water. This is done in two ways; by multiclass classification and as one binary classifier for each of the four land cover classes. The predictions from the binary classifiers are merged together and compared with the prediction done with the multiclass dataset. The result shows that U-net-4 with multi-class classification gives the best result.

Furthermore, a land cover map is generated based on a satellite image of Nittedal municipality, to see how the network performs and can be used in practice. The result shows that U-net-4 with merged binary classification gives the best result.

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

I dette kapittelet presenteres bakgrunn og motivasjon for masteroppgaven. I denne delen av oppgaven settes undersøkelsen i kontekst. Videre beskrives forskningsspørsmålet for studien før relatert arbeid krediteres.

1.1 Bakgrunn og motivasjon

«Geomatikk, samlebetegnelse for virksomhet knyttet til innsamling, bearbeiding, analyse, lagring, distribusjon, presentasjon og anvendelse av romlig stedfestet informasjon.» [14].

Dette er en definisjon av fagfeltet Geomatikk, og som bringer med seg betydningen av databehandling i sammenheng med romlig stedfestet informasjon. Geomatikk er et felt i endring og hvor det har vært skjedd stor utvikling. Romlig datavitenskap (eng: Geospatial Data Science) er et felt innenfor geografiske informasjonssystemer (GIS) som har fått mye oppmerksomhet og er beskrevet slik: «Geospatial Data Science is «The art and craft of people leveraging technology to create value out of data using location and time.» » [15]. GIS har alltid handlet om å ta i bruk store datamengder (eng: big data), og anvendelse av mange romlige analysemetoder. Slike metoder er former for datavitenskap, men dette ble gjort lenge før begrepet datavitenskap ble tatt i bruk.

Moderne teknologi gjør det mulig å produsere arealressurskart ved hjelp av satellittbilder. Det er derfor aktuelt å se på i hvilken grad satellittbilder kan anvendes til å produsere nøyaktige arealressurskart ved hjelp av maskinlæring. Arealressurskart i målestokk 1:5000 er et viktig datagrunnlag for blant annet prosjektering, saksbehandling og geografiske analyser. I dag produseres arealresurskart manuelt fra flybilder, noe som er en tidkrevende jobb. Det er derfor interessant å undersøke om arealressurskart kan produsereser effektivt, men fortsatt nøyaktig, ved hjelp av maskinlæring.

Et problem når en skal lage datasett til bruk i maskinlæring er å samle inn nok data. Dette er i utgangspunktet en tidkrevende jobb, men blant annet mobilt utstyr, fjernmåling og tingenes internett har gjort det mulig å samle inn enorme mengder data. Slik data inneholder som regel både tid og sted og gjør at man kan få mer informasjon ut fra dataene ved å se det i sammenheng med disse to faktorene [15]. Ved bruk av maskinlæring er det mulig å bruke dataene på nye måter.

Innenfor fjernmåling er Sentinelsatellittene, overvåket av Den europeiske romfartsorganisasjon (ESA), den største datakilden [15]. Sentineloppdraget består av åtte satellitter fordelt på seks oppdrag [39]. Sentinel-2 består av Sentinel-2a og Sentinel-2b som kretser jorda med en omløpstid på fem dager. Dette gjør det mulig å laste ned gratis høyaktuelle bilder fra jorda, i teorien ett nytt bilde hver femte dag hvis det ikke er skydekke.

1.2 Forskningsspørsmål

Målet med oppgaven er å utforske i hvilken grad satellittbilder kan klassifiseres basert på detaljert arealressurskart i målestokk 1:5000 ved bruk av maskinlæ-

ring, herunder konvolusjonelle nevrale nettverk. Hensikten med dette er å bidra til forskningen på produksjon av kart som kan være nyttig ved for eksempel arealplanlegging, arealforvaltning, utbygging og kontrollering av store terrengendringer.

For å nå forskningsspørsmålet har jeg lastet ned satellittbilder og foreslått et treningsdatasatt basert på satellittbildet og et arealressurskart (AR5). Deretter har jeg ved hjelp av konvolusjonelle nevrale nettverk, trent modellen på det foreslått datasettet. Videre blir testdatasettet predikert og sammenlignet med fasitdata for å vurdere nøyaktigheten av modellen.

For å konkretisere forskningsspørsmålet har jeg utarbeidet to problemstillinger:

- Hvor nøyaktig kan satellittbilder klassifiseres etter arealtyper ved hjelp av konvolusjonelle nevrale nettverk.
- Hva kan det predikerte arealressurskartet brukes til.

1.3 Forskningsmetode

For å tilnærme meg forskningsspørsmålene har jeg startet med å lese relevant teori for få en bedre forståelse for forskningsspørsmålet og rammene rundt. Videre har jeg laget et datasett der relevant informasjon er valgt for å lage to nettverk.

Når de konvolusjonelle nevrale nettverkene er trent brukes de til å lage predikerte arealressurskart for å undersøke hva disse kan brukes til.

1.4 Relatert arbeid

Semantisk segmentering av bilder innebærer å klassifisere hver piksel i bildet inn i forhåndsdefinerte klasser. Dype konvolusjonelle nevrale nettverk har tidligere bitt anvendt med god suksess for semantisk segmentering, og er derfor populært å ta i bruk. Klassifisering av satellittbilder med tilhørende fasitdata er gjort i tidligere studier. Heryadi og Miranda [19] klassifiserte arealtyper i Sentinel-2 satellittbilder over et område i India og sammenlignet en konvolusjonell nevralt nettverk (CNN) modell med en «gradient boosting model» hvor CNN-modellen ga høy nøyaktighet.

Papadomanolakia, Vakalopouloua, Zagoruykob og Karantzalosa [31] sammenlignet utførelsen til «AlexNet», «AlexNet»-small og «VGG»-modeller ved klassifisering av høyoppløselige satellittbilder fra SAT-4 og SAT-6.

Iglovikov, Mushinskiy og Osin [20] kom på tredjeplass i Kaggle utfordringen; «DSTL Satellite Imagery Feature Detection challenge» i 2017. Nøyaktigheten er

sammenlignbar med første- og andreplass, men forfatterne foreslår en metode som ikke innebærer bruk av komplekse teknikker. Forfatterne foreslår en modifisert U-net. Datasettet består av 57 satellittbilder fra WorldView-3 med fasitdata som representerer ti arealklasser.

Arief, Indahl, Strand og Tveite [3] så i sin studie ved NMBU på klassifisering av LiDAR punktskydata med tilhørende fasitdata fra arealressurskart i målestokk 1:5000. De foreslår ett nettverk «SA-Net», som er en dyplæring arkitektur basert på «atrous» kjerner (eng: kernel) på en «ResNet-FCN» arkitektur med stokastisk dybde teknikk. Studien viser en forbedret klassifisering av LiDAR data sammenlignet med lignende nettverk.

Yang, Rottensteiner og Heipk [47] klassifiserte åtte arealtyper fra flyfoto med ulike CNN-modeller av «SegNet» og «LiteNet». Resultatet viser at CNN med ulike arkitekturer og bruk av ulik inputdata gir totalt best nøyaktighet.

Det er også en masteroppgave fra NTNU hvor Mällberg og Rolfsen [29] klassifiserte flyfoto med arealressurskart som fasitdata ved bruk av CNN.

På det tidspunkt denne masteroppgaven er utført, finnes det ingen publiserte forsøk på klassifisering av satellittbilder med tilhørende fasitdata fra arealressurskart i målestokk 1:5000.

1.5 Begrensinger

I prinsippet eksisterer det satellittbilder og arealressurskart for hele Norge som er tilgjengelige og kan brukes som treningsdata. Men, det er ønskelig å få et datasett som har god fordeling av arealtypene som skal klassifiseres. Det er også en fordel at satellittbildene som brukes til trening er tatt omtrent ved samme tid som arealressurskartet sist ble oppdatert. Dette gir noen begrensninger i valg av treningsområde, men hovedbegrensingen i studien er mangel på datakraft. NMBU har i løpet av arbeidet med denne masteroppgaven fått GPUer installert på serveren sin. Her kunne det vært interessant å se hvordan modellene presterer med mer data tilgjengelig.

Det er også begrenset hvor nøyaktig modellene kan klassifisere. Dette gjøres gjeldende fordi satellittbildene har en pikselstørrelse på 10 meter, mens arealressurskartet i målestokk 1:5000 er de mest nøyaktige kartdata i Norge. Noe informasjon går derfor tapt når arealressurskartet gjøres om til et raster med 10 meters pikselstørrelse.

Maskinlæring er et felt i rask endring, det blir gjort omfattende forsknings på fagfeltet og det kommer forbedrede algoritmer fortløpende. Dette gjør at nye forbedrede modeller kan komme på banen som raskt endrer aktualiteten til tidligere modeller.

1.6 Oversikt

Oppgaven er delt inn i følgende kapitler:

- Kapittel 1: Introduksjon
- Kapittel 2: Teori
- Kapittel 3: Metode
- Kapittel 4: Resultat
- Kapittel 5: Diskusjon
- Kapittel 6: Konklusjon

Kapittel 2

Teori

Formålet med dette kapittelet er å danne det teoretiske grunnlaget for studien. Det er lagt vekt på teori rundt maskinlæring med fokus på veiledet læring og konvolusjonelle nevrale nettverk.

2.1 Maskinlæring

Maskinlæring er en gren av kunstig intelligens som gjør det mulig å gjenkjenne komplekse mønstre i data[33]. Ved hjelp av maskinlæring kan tidkrevende arbeid, slik som analysering av store datamengder, gjøres effektiv og automatisk. Det kan øke både effektiviteten og forbedre resultater ved at modeller finner mønster som mennesker ikke kan se.

Spamfilter i e-post, bilde- og stemmegjenkjenning og pålitelige søketjenester er noen eksempler på produkter av maskinlæring som er nyttige i dagens samfunn [33]. I denne oppgaven brukes konvolusjonelle nevrale nettverk for å klassifisere piksler i satellittbilder ved bruk av veiledet læring.

Veiledet læring

Veiledet læring gjør det mulig å lage prediksjoner for fremtidige data ved å lære opp en modell ut ifra treningsdata og fasitdata. Datasettet består som regel av et treningssett; $X_{trening}$ og $y_{trening}$, valideringssett; $X_{validering}$ og $y_{validering}$ og et testsett; X_{test} og y_{test}. Basert på tilgjengelig datamengde er det vanlig å velge en kombinasjon hvor treningssettet består av 60-90 % av dataene, og validerings- og testsettet av 5-20 % hver. Treningssettet brukes til å trene klassifiseringsmodellen og valideringssettet brukes for å teste og predikere ytelsen til modellen. Dette er for å unngå at modellen over- eller undertilpasser seg til de usette testdataene. Når modellen er ferdig trent, brukes testsettet som en siste evaluering for resultatet. Testsettet skal ikke være vist til modellen under trening for å unngå at modellen tilpasser seg etter testdatasettet[33].

Klassifisering av data kan være binært eller ikke-binært. Når et datasett er binært består fasitdataen av verdiene 0 og 1. Mens for et flerklassedatasett består verdiene av koder for klassene som skal klassifiseres. Eksempelvis kan en undersøke sykdomssymptomer og klassifisere basert på verdier. Dette kan være binært, som å skille mellom syk/ikke syk, eller bestå av flere klasser, som å skille mellom bestemte sykdommer.

Regression er en annen form for veiledet læring, hvor det predikeres kontinuerlige verdier, for eksempel lønn basert på forklarende variabler.

Objektdeteksjon er en metode for å klassifisere objekter i et bilde og samtidig opptegne avgrensingsbokser rundt objektene [5].

Bildesegmentering er en metode for å skille de ulike objektene fra hverandre ved å gi objekter i samme klasse, ulike farger og merke de med en avgrensingsboks med for eksempel «person 1» og «person 2» .

Semantisk segmentering er metoden for å klassifisere hver piksel i et bilde inn i klasser [38]. Dette er metoden som anvendes i denne studien.

Ikke-veiledet læring

Ved ikke-veiledet læring er ikke dataene kategorisert eller strukturert. Ved bruk av ikke-veiledet læring kan maskinen finne meningsfulle strukturer i dataene, uten å vite fasiten.

Klustering gjør det mulig å organisere dataene ut ifra likheter mellom objekter for å gruppere eller organisere dataene [33].

2.1.1 Nevrale nettverk

Kunstige nevroner er byggeblokkene i *kunstige nevrale nettverk* (eng: artificial neural network, ANN), *konvolusjonelle nevrale nettverk* (eng: convolutional neural networks, CNN) og *tilbakevendende nevrale nettverk*(eng: recurrent neural networks, RNN). Kunstige nevroner er inspirert av hvordan den menneskelige hjernen løser komplekse oppgaver. Hvert lag i et nettverk består av ett eller flere kunstige nevroner [33].

Nevroner er dataelementer som kan motta og sende tallverdier til hverandre. *Perceptron* er et eksempel på et nevron som mottar nuller og enere i et binært nettverk, som skal symbolisere «ja/nei-verdier». På bakgrunn av disse verdiene skal en beslutning fattes. Ved bruk av «vekting» kan en egenskap ha mer å si enn en annen ved en beslutningen. De binære verdiene multipliseres med vekter avhengig av egenskapene de representerer. Videre blir alle verdiene summert og sammenlignet med en minimumsverdi. Dette utgjør nevronets *skjevhet* (eng: bias). Er summen høyere enn skjevheten vil svaret være «ja», hvis ikke vil svaret være «nei». Nevrale nettverk er tilpasningsdyktige og endrer vekter og skjevheter for å forbedre nettverket, basert på treningsdata. Hvis resultatet er lavere enn forrige beslutning, endres vektene og blir sammenlignet med fasitdata på nytt. Læringsprosessen består av å prøve og feile. Ved bruk av *skjulte lag* (eng: hidden layers), kan nettverket tilpasse seg dataene og fange opp flere nyanser i disse.

Kraftig maskinvare har gjort det mulig å ta i bruk nevroner med kontinuerlige inn- og utverdier som gjør det mulig å få nyanserte modeller og effektive læringsprosessser [9].

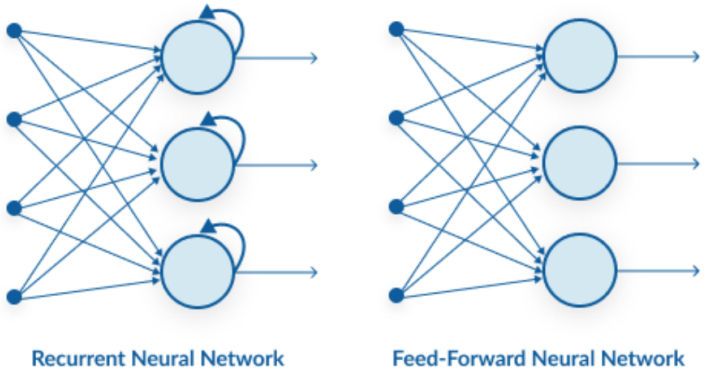
Dyp læring

Når nettverket består av flere lag kalles det «dype kunstige nevrale nettverk». Dyp læring er prosessen med å lære opp dype nevrale nettverk basert på treningsdata som inneholder fasitdata. Dyp læring blir brukt til å forstå blant annet bilder, tekst

og lyd [45]. Det er tre typer av nevrale nettverk i dyp læring; kunstige nevrale nettverk (ANN), konvolusjonelle nevrale nettverk (CNN) og tilbakevendende nevrale nettverk (RNN).

2.1.2 Kunstige nevrale nettverk -Artificial neural networks (ANN)

Kunstige nevrale nettverk (ANN) består av flere perceptroner i hvert lag hvor innputtverdiene blir prosessert fremover i det nevrale nettverket. Dette er kjent som mat-fremover nevrale nettverk. Aktiveringsfunksjonen introduserer ikke-lineære egenskaper til nettverket som gjør det mulig å lære komplekse mønstre mellom inn- og utverdier [33].



Figur 2.1: Forskjellen mellom en mat-fremovernevralt nettverk og et tilbakevendende nevralt nettverk.

2.1.3 Tilbakevendene nevrale nettverk - Recurrent neural networks (RNN)

Tilbakevendende nevrale nettverk (RNN) brukes når dataene er sekvensielle og rekkefølgen er viktig, ved for eksempel tidsdata, tale og tekst. Figur 2.1 viser forskjellen på en ANN og en RNN, hvor en ANN får informasjon fra innputtlaget til det skjulte laget, mens RNN får informasjon fra både innputtlaget og det forrige skjulte laget. Beregning av aktiveringsfunksjonen er veldig lik som for ANN [33].

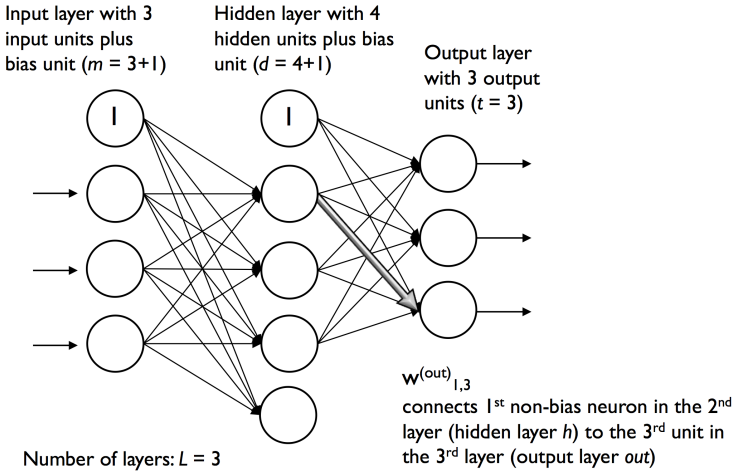
2.1.4 Mat-fremover nevralt nettverk - Feed forward neural network

Mat-frammover nevrale nettverk er bygd opp av et innputtlag, ett eller flere skjulte lag, og et utputtlag. Hvert lag i et mat-frammover nettverk fungerer som en innputt til det neste laget, som vist i figur 2.2.

Nettverket definerer en kartlegging og lærer de verdiene til parameterene som gir det beste resultatet. Gjennom trening vil utputtverdiene bli sammenlignet med

fasitdata for å beregne feilen som skal minimeres.

De skjulte lagene består av lagene mellom innputt- og utputtlagene. Disse kalles skjulte lag fordi trenings-dataene ikke sier noe om hvilke verdier lagene skal produsere.



Figur 2.2: Mat-frammover nevralt nettverk med et skjult lag. Figur hentet fra [33].

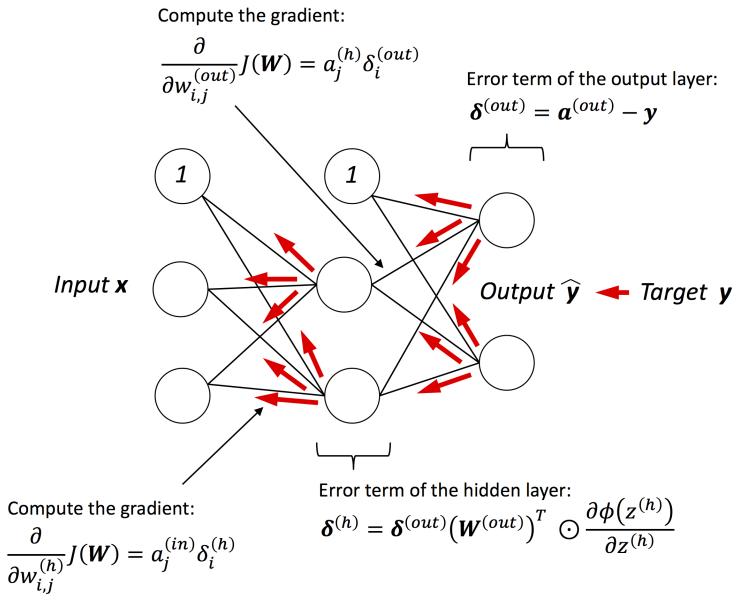
Via *tilbakepropagering* kan nettverket tilpasse seg og lære å utnytte de skjulte lagene for å få best tilnærming til ønsket utputt [16].

Tilbakepropagering

Tilbakepropagering er en mye brukt metode hvor utputtverdiene fra mat-frammover propageringen blir brukt for å propagere feilen fra høyre til venstre. Ved tilbakepropagering beregnes gradienten av tapsfunksjonen med respekt på vektene til nettverket bakover, et lag av gangen til en når innputtlaget. Ved å gjøre dette vil vektene som beregner feil svar bli vektet lavere, mens vektene som gir rett svar blir styrket [49]. Tilbakepropagering er vist i figur 2.3.

2.1.5 Konvolusjonelle nevrale nettverk - Convolutional neural networks (CNN)

Konvolusjonelle nevrale nettverk (CNN) er en spesiell form for nevrale nettverk for analyse av rutenett-lignende strukturer. Et eksempel på dette er bilder som kan ses på som et 2D-rutenett av piksler. Ved bruk av CNN kan en finne objekter i bilder eller klassifisere hvert piksel.



Figur 2.3: Tilbakepropagering med ett skjult lag. Figur hentet fra [33].

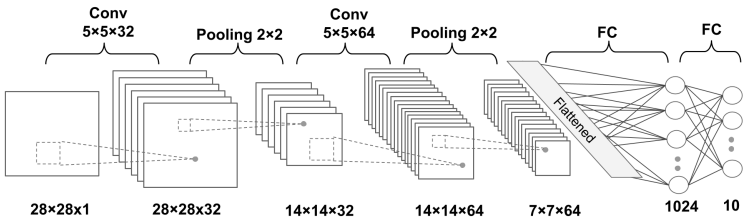
Navnet konvolusjon kommer av de generelle matriseoperasjonene som blir *byttet ut med konvolusjon* i minst ett lag. En konvolusjon er en operasjon mellom to funksjoner, en innputt og en kjerne (eng: kernel). Innputt er ofte en flerdimensjonal matrise, for eksempel et bilde, mens en kjerne ofte er en flerdimensjonal matrise med parametere som tilpasses av læringsalgoritmen. Disse matrisene er kalt for tensorer. Utputt kan i den sammenheng bli kalt for «feature map» [16].

En typisk CNN består av flere konvolusjonslag, poolinglag og fullstendig-tilkoblet-lag på slutten av nettverket. Poolinglag har ikke vekter og skjevheter (eng: bias), mens konvolusjonslag og «fullstendig-tilkoblet-lag» har vekter og skjevheter [33].

Krizhevsky, Sutskever og Hinton la i sin studie fundamentet for dype CNN med konvolusjonelle lag etterfulgt av en aktiveringsfunksjon igjen etterfulgt av et maks poolinglag med arkitekturen «Alexnet» i 2012 [24]. AlexNet består av 8 nevrale nettverkslag; 5 konvensjonellelag og 3 fullstendig-tilkoblet-lag.

Fordelen med å ha flere lag i nettverket er at hvert lag lærer komplekse egenskaper, slik som objekter, kanter og former. Figur 2.4 viser en flerklasse CNN-arkitektur med innputtbilder av størrelsen 28*28 piksler med ett bånd. Den består av to konvolusjonelle lag, to poolinglag og to fullstendig-tilkoblet-lag.

En kjerne er en matrise som blir ført over ett bilde og multiplisert med verdiene for å forbedre bildet, for eksempel ved å gjøre bildet skarpere eller forsterke



Figur 2.4: Arkitekturen til en flerklasse fullstendig tilkoblet CNN modell med et bånd. Figur hentet fra [33]

og glatte viktige strukturer. Verdiene til kjernen blir bestemt i modellen [46].

Padding

Padding er en måte å velge hvordan kantene skal behandles når kjernen beveges over bildet. Ruter mot midten tar del i flere beregninger enn cellene langs kanten, avhengig av p, vil kantene ta del i beregningene ulikt. De tre mest brukte paddingene er *full*, *same* og *valid*. Disse er vist i figur 2.5.

Full padding, parameteren p er satt til $p=m-1$, hvor m er kernel-størrelsen. Dette øker dimensjonen betraktelig og alle ruter langs kanten av bildet vil bidra like mye som rutene i midten.

Same padding blir beregnet ut ifra filterstørrelsen med et krav om at innputt- og utputtstørrelse skal være den samme. Same padding er den mest brukte formen, ettersom størrelsen på innputtbildet eller tensoren blir bevart.

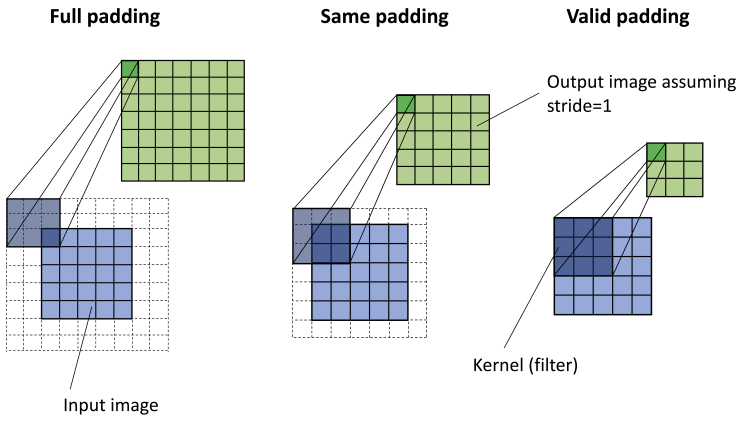
Valid padding er når $p=0$. Padding blir da ikke brukt. En ulempe ved bruk av valid padding er at volumet til tensoren vil minke for hvert lag.

Det er anbefalt å bruke same padding for de konvolusjonelle lagene og endre den romlige størrelsen med poolinglag [33].

Bassenglag - Pooling

Et typisk lag i et CNN består av tre steg. Det første er å produsere et sett med lineære aktiveringer ved hjelp av *konvolutter*. I steg to blir den lineære aktiveringen ført gjennom en ikke-lineær aktiverings-funksjon. Det tredje steget er pooling, også kalt subsampling, som brukes for å endre utputt [16].

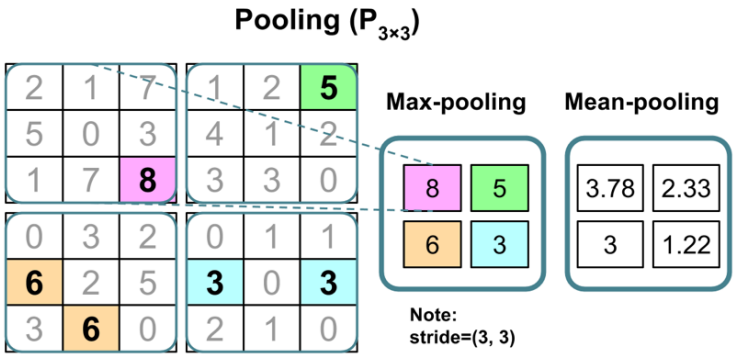
Funksjonen summerer opp statistikk fra nærliggende utputter og endrer netutputt. Det er flere former for pooling hvor max-pooling og mean-pooling er mest



Figur 2.5: Eksempel på padding for en 5x5 piksel innputt med kernel størrelse på 3x3, med ett skritt. Hentet fra [33].

brukt. Disse er vist i figur 2.6.

Mean pooling finner gjennomsnittet. *Max pooling* tar maksimumverdien fra rutene og finner lokal varians. Dette genererer robuste features som ikke blir påvirket av små endringer i naboruter eller støy i innputtdata. Pooling bidrar til høyere beregningseffektivitet og minker overtilpasning av data fordi størrelsen på featurene minker [33].

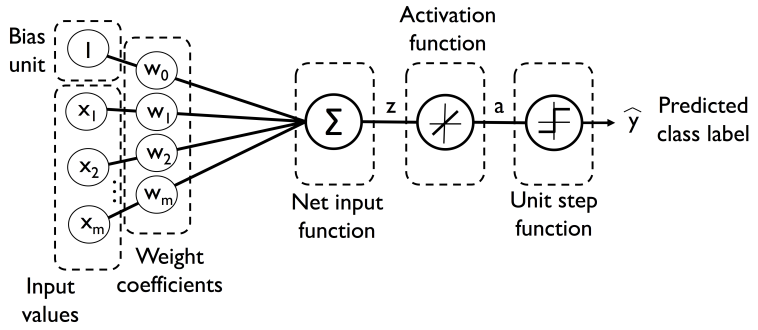


Figur 2.6: Eksempel på max- og mean-pooling. Hentet fra [33].

2.1.6 Aktiveringsfunksjon

Aktiveringsfunksjonen beskriver «utputtoppførselen» til et nevron som en funksjon av innputt, vekter og skjevhet. En aktiveringsfunksjon kan være både lineær og ikke-lineær, men den må være deriverbar. For å løse komplekse oppgaver, som for eksempel bildeklassifisering, må den være ikke-lineær for de skjulte lagene og for utputtlagene [33]. Figur 2.7 viser bruk av aktiveringsfunksjon i Adalinealgo-

ritmen.



Figur 2.7: Adalinealgoritmen, for å illustrere bruk av aktiveringsfunksjon. Figur hentet fra [33].

Sigmoid

Sigmoidfunksjonen er en logistisk aktiveringsfunksjon som kartlegger nettoinnputt z på en logistisk distribusjon i intervallet [0, 1]. Funksjonen gir en S-formet kurve og skjærer y-aksen i $z=0$.

$$\phi(z) = \frac{1}{1+e^{-z}} \tag{2.1}$$

Softmax

Softmax er en generalisering av den logistiske funksjonen. Softmax er en form for Argmaxfunksjon, men utputt gir sannsynligheten for hver klasse. Den kan derfor brukes når flerklassedatasett skal klassifiseres. Funksjonen beregner sannsynligheten for at nettoinnputt z tilhører de $i$te klasse [49];

$$\phi(z) = p(y = i \mid z) = \frac{e^{z_i}}{\Sigma_{i=1}^{M} e^{z_j}} \tag{2.2}$$

Hyperbolic tangent

Hyperbolic tangent er mest kjent som «tanh» og er mye brukt i skjulte lag. Funksjonen er en skalert versjon av den logistiske funksjonen Sigmoid. Funksjonen har et bredere utputtspekter og er distribuert i intervallet [-1, 1], som kan forbedre konvergens i tilbakepropagering [4].

$$\phi(z) = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.3}$$

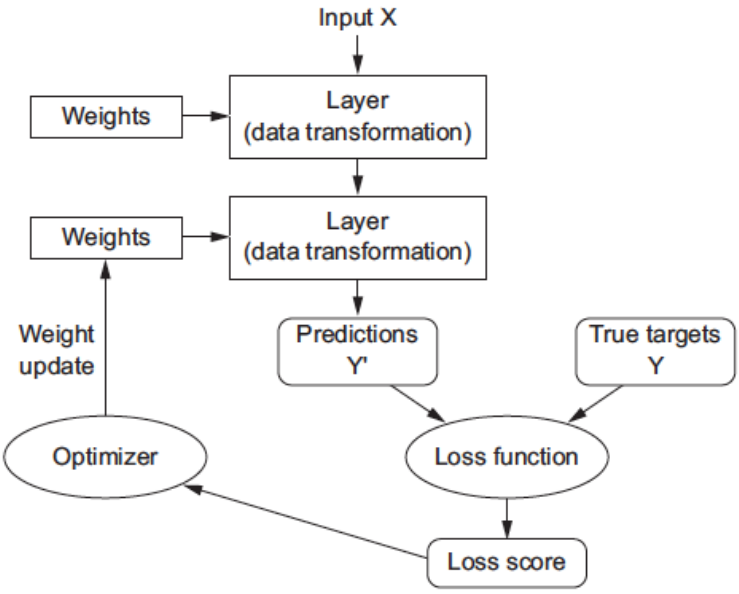
Rectified linear unit activation

Rectified linear unit activation, bedre kjent som ReLU, er en mye brukt aktiveringsfunksjon. Forsvinnende-gradient (eng: vanishing) er et problem for Sigmoid og Tanh. Når z-verdiene for nettoinnputt er store vil den deriverte verdien minske og nærme seg null. Det resulterer i at vektene læres tregt i treningsfasen. ReLU løser dette problemet ved at den deriverte av aktiveringsfunksjnen alltid er 1 for positive innputtverdier.

$$\phi(z) = max(0, z) \tag{2.4}$$

2.1.7 Tapsfunksjon - Loss function

Tapsfunksjonen måler hvor presist det nevrale nettverket er, ved at den sammenligner predikerte verdier med sanne verdier. Utputt er distansen mellom disse, og kalles taps-score. Denne brukes for å gi tilbakemelding til vektene som må justeres, med formål å få et lavt taps-score. En optimaliserer tilpasser vektene i en tilbakepropageringsalgoritme. I utgangspunktet er vekten tilfeldige, som gir en høy taps-score i første omgang, men som med trening skal bli lavere. Figur 2.8 viser hvordan tapsfunksjonen fungerer.



Figur 2.8: Eksempel å hvordan tapsfunksjonen fungerer. Figur 1.9 hentet fra [5]

Binær kryssentropi - Binary cross entropy

Denne tapsfunksjonen brukes for løsing av binære problemer. Utputt er da en skalar mellom 0 og 1 og gir sannsynligheten. Kryssentropi måler avstanden mellom

sann observasjon og modellens predikerte verdier. Siste lag burde ende med et «Denselag» med en *enhet*, og *aktiveringsfunksjonen* sigmoid [5].

Kategorisk kryssentropi - Categorical cross entropy

Kategorisk kryssentropi brukes ved flerklasse-klassifisering. Treningsdataene må være kodet med «one-hot-encoding», som gjør at datasettet består av 0 og 1 basert på hvilken klasse de tilhører. Instansene kan klassifiseres til en bestemt klasse, eller bestå av flere klasser. Avstanden mellom sann verdi og predikert verdi gir en sannsynlighetsdistribusjon. Ved minimering av avstanden mellom de to distribusjonene vil nettverket gi utputt som er nær «sann-verdi».

Det siste laget bør bruke softmax-aktivering fordi det gjør at utputtverdien gir en sannsynlighetsdistribuering for hver av klassene, hvor totalsummen blir 1.

2.1.8 Optimaliserer - Optimizer

Optimaliserer er en metode for å oppdatere nettverket basert på treningsdata og tapsfunksjoen i modellen. Optimalisereren bestemmer hvordan gradienten til tapsfunksjonen skal bli brukt for å oppdatere parameterene i læringsprosessen [5].

Stochastic gradient descent

Stochastic gradient descent (SGD) er en av de mest brukte optimaliseringsalgoritmene. Den skiller seg fra «*batch gradiant decent*» ved at vektene oppdateres gradvis for hvert treningseksempel i stedet for å oppdatere vektene basert på summen av feil for alle treningseksemplene. Dette gjør at SGD konvergerer raskere, men også inneholder mer støy. Det kan bidra til at den unngår å bli fanget i lokalt minimum[33]. Beregningstiden vokser ikke med treningseksemplene noe som gjør at modellen konvergerer selv om antall treningseksempler blir stort. Læringsraten er en viktig parameter i SGD, hvor den gradvis bør avta over tid [16]. Likning 2.5 beregner gradientens estimat.

$$g_{GD} = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta_t) \tag{2.5}$$

hvor L er tapsfunksjonen som skal minimeres, ∇_θ er gradientene. Parameterene oppdateres da med formelen;

$$\theta_{t+1} = \theta_t - \eta g_{GD} \tag{2.6}$$

hvor η er læringsraten [17].

Adaptive Gradient Algorithm - AdaGrad

Adagrad tilpasser læringsraten til alle parameterene i modellen ved å oppdatere mye brukte parametere ofte, og lite brukte parametere sjeldnere. Målet med de lite brukte parametrene er at algoritmen skal legge merke til disse parametrene når de først er til stede [8]. Adagrad modifiserer den generelle læringsraten η, for hver parameter og θ_i for hvert tids-skritt$t$. Den baserer seg på den forrige gradienten $g_{t,i}$, beregnet for θ_i.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{2.7}$$

Hvor $G_{t,ii}$ er en diagonal matrise med diagonale elementer i som er summen av de kvadrerte gradientene for θ_i. Variabelen ϵ bidrar til å unngå divisjon med null [37]. Fordelen med AdaGrad er at det ikke er behov for å manuelt stille inn læringsraten. En ulempe er at den akkumulerte summen øker gjennom trening, som følge av at nevneren alltid er positiv. Læringsraten vil dermed minke og bli uendelig liten, som gjør at den ikke lengre kan tilegne seg kunnskap gjennom læring. AdaGrad er designet for å konvergere raskt ved ikke-konvekse funksjoner for å finne ett lokalt konvekst bunnpunkt. Læringsraten synker raskt og kan føre til at læringsraten blir for liten før den havner i den konvekse bunnen. Root Mean Square propagation (RMSProp) er en modifisering av AdaGrad, som skal forhindre denne utfordringen. Dette beskrives senere.

Adadelta

Adadelta er en algoritme som bygger på Adagrad, men har som mål å redusere den minkende læringsraten. Dette gjøres ved å ikke samle opp tidligere gradienter, men i stedet bruke nylige gradienter. Zeiler forklarer i [48] hvordan AdaGrad sin algortime endres for å løse dette problemet. Formelen for beregning av adadelta er:

$$\nabla\theta_t = -\frac{RMS[\nabla\theta]_{t-1}}{RMS[\nabla\theta]_t} \tag{2.8}$$

Root Mean Square propagation - RMSprop

RMSProp er en form for AdaGrad, men modifisert for å fungere i en ikke-konveks modell ved å bruke eksponentielt vektet avtagende gjennomsnitt for å konvergere raskt etter at en konveks bunn er funnet [16]. Beregning av RMSprop er vist i likning 2.10, som er hentet fra [35].

$$rms_t = \rho \cdot rms_{t-1} + (1-\rho) \cdot g_t^2 \tag{2.9}$$

$$\theta_t = \theta_{t-1} - \frac{\eta \cdot g_t}{\sqrt{rms_t + \epsilon}} \tag{2.10}$$

ρ er diskonteringsfaktor for kommende gradienter. ϵ er en konstant for numerisk stabilitet.

Adaptive moment estimator - Adam

Adaptive moment estimator (Adam) [23] tilpasser læringsraten for å optimalisere. Adam kombinerer fordelene med RMSProp og AdaGrad. Adam er ulik ved at den bruker førsteordens moment med eksponensiell vekting av gradientent- og skjevhetskorreksjon av både første- og andreordens momentum. Dette er for å kompensere for initialiseringen i starten. Optimalisereren er beregningseffektiv og krever lite minne og den passer derfor bra for datasett med mye data eller modeller med mange parametere. Likningene for oppdatering av eksponentiell bevegelig gjennomsnitt av gradienten m_t og den kvadratiske gradienten v_t er vist under:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{2.11}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \tag{2.12}$$

β_1 og β_2 er hyperparametere som kontrollerer *forfallshastigheten*. m_t og v_t er forholdsvis estimat av første moment (gjennomsnittet) og usentrert varians av gradienten, som er initialisert ved null. Dette kan føre til at momentestimatet er *partisk* mot null. For å ta hensyn til dette beregnes skjevhetskorrigert estimat $\hat{m}_t$ gradienten $\hat{v}_t$.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.13}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.14}$$

Adam's oppdateringsregel blir da:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot m_t \tag{2.15}$$

Hvor ϵ er skrittlengde, og η er læringsraten.

2.1.9 Hyperparametere

Hyperparametere har fått navnet for å skille de fra parametrene i modellen som er nettverkets vekter. Ved bruk av valideringsdata kan man teste ulike hyperparametere for å se hvordan modellen reagerer på endringene. Dette er en måte å se om modellen er overtilpasset eller undertilpasset og gjøre justeringer deretter. Gjør man mye endringer kan valideringssettet også bli overtilpasset.

Dersom modellen ikke fungerer optimalt kan man endre aktiveringsfunksjon, legge til eller fjerne lag, endre antall enheter per lag til læringsratens optimalisator, eller prøve ulike arkitekturer. I det følgenden beskrives noen hyperparametere som er vanlig å endre for å optimalisere modellen [5].

Læringsrate

Å velge riktig læringsrate er viktig for å bestemme hvor mye modellen skal endre seg basert på estimert feil for hver gang vektene er oppdatert. En liten læringsrate vil gi en lang treningsprosess hvor modellen kan sette seg fast, mens en høy læringsrate kan gi en ustabil treningsprosess eller vekter som ikke er optimale.

Ved bruk av små *batch-størrelser* burde det brukes en liten læringsrate på grunn av den høye variansen i gradient-estimatet. [16].

Regularisering

Et problem i CNN og andre nevrale nettverk er under- og overtilpasning. Kapasiteten til nettverket avhenger av kompleksiteten til funksjonen og evnen til å lære. Små nettverk med lav kapasitet har få parametere, og har en tendens til å undertilpasse seg fordi datamengden er for liten til å se komplekse strukturer.

Store datasett med høy kapasitet kan bli overtilpasset fordi selve nettverket memoriserer treningsdataene, men gir et dårlig resultat for valideringsdataene.

Ønsket datamengde avhenger av problemet og algoritmen som velges, men det er som regel nyttig å ha mest mulig data og benytte seg av regularisering for å unngå overtilpasning.

L2 regularisering og *dropout* er to teknikker av regularisering som er mye brukt. Dropout gir gode resultater for dype nevrale nettverk. Dropout blir brukt i de skjulte lagene. For hver iterasjon i treningsfasen blir tilfeldige nevroner tilbakeholdt. Dette gjør at de gjenværende nevronene må skalere vektene sine for å ta hensyn til de tilbakeholdte nevronene. Videre i prediksjonsfasen bidrar alle nevronene til aktivering av neste lag [33]. I «ImageNet 2012» konkurransen brukte vinnerene Krizhevsky, Sutskever og Hinton [25] dropout som regulariseringsmetode og fikk gode resultater.

Det er vanlig å regularisere vektene ved å legge på en kostnad (eng: cost) til tapsfunksjonen når vektene er store. Ved L1 regularisering er kostnaden som legges til proporsjonal med absoluttverdien av vekt-koeffisienten. Ved L2 regularisering er kostnaden som legges til, proporsjonal med kvadratet av vekt-koeffisienten. Navnet vektforfall (eng: weight decay) blir brukt i sammenheng med nevrale nettverk [5].

Momentum

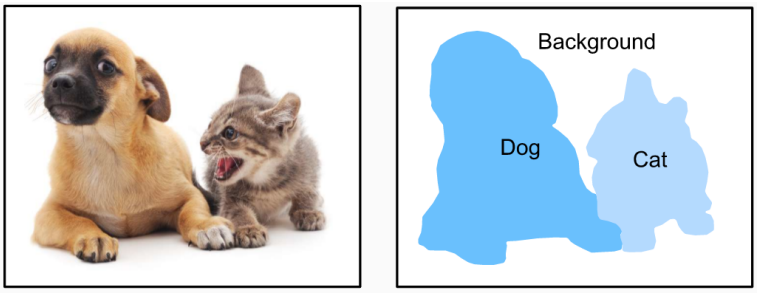
Momentum er en metode som blir brukt for å akselerere læring. Dette blir brukt ved liten gradient, støyete gradient eller ved mye krumming. Momentumalgoritmen samler opp gjennomsnittet av de tidligere eksponensielt forfallende gradientene, for så å bevege seg i den retningen. En hyperparameter bestemmer hvor raskt algoritmen skal tilpasse seg gjennomsnittet. Momentum kan derfor ses på som læringsratens læringsrate. [16].

Forfall - Decay

Forfall reduserer læringsraten over epoker. Avhengig av problemet som skal løses er det mulig å legge inn minkende læringsrate til den har nådd et bestemt minimum. Når valideringsfeilen flater ut kan forfall brukes ved å redusere læringsraten med en bestemt faktor mellom 2 og 10 [16].

2.1.10 Semantisk segmentering

Semantisk segmentering klassifiserer hver piksel i et bilde slik at alle pikslene tilhører en bestemt klasse. Figur 2.9 viser et segmentert bilde som har fått merkelapp hund, katt og bakgrunn [49].



Figur 2.9: Segmentering av bilde. Fig. 13.9.1 fra [49].

Semantisk segmentering består av to metoder; bildesegmentering og instanssegmentering.

Bildesegmentering deler et bilde inn i flere regioner og benytter seg av korrelasjonene mellom pikslene. For å få riktige segmenter må man benytte seg av fasitdata i prediksjonene.

Instanssegmentering fungerer på samme måte som bildesegmentering, ved at den klassifiserer hver piksel. I tillegg skiller den mellom de ulike objektene i bildet, for eksempel ved å skille to katter fra hverandre i et bilde. [49]

Et viktig datasett innenfor semantisk segmentering er «PASCAL Visual Object Classes» (VOC) [10], som består av et åpent datasett med bilder og bakkesannhet (eng: ground truth) hvor bildene består av mennesker, dyr, objekter og handlinger. Fra 2006 til 2012 ble det holdt årlige utfordringer innenfor fem felt hvor segmentering ble lagt til i utfordringen i 2009;

- Klassifisering; bildet inneholder bestemte objekter, for eksempel en sykkel.
- Deteksjon; hvor objektene er.
- Segmentering; hvilke piksler tilhører hvilke klasser.
- Handlingsklassifisering; hvilken handling personen gjør, for eksempel å sykle.
- Personklassifisering; hvor for eksempel hender og hode er i bildet.

Innenfor segmentering ga bruk av flere «bunn-opp segmentering» (eng: bottom-up segmentation) et godt resultat. Integrering av segmenteringsmodellen med deteksjon- og klassifiseringsmodellen forbedret også resultatet.

Bildeklassifiseringsdatasettet [27] med 101 objektklasser og 15-30 treningsbilder per klasse, var et av de første standardiserte datasettene for flerkategoribildeklassifisering. «ImageNet ILSVRC-2012» konkurransen gikk ut på å klassifisere bilder basert på 1000 ulike klasser, med et treningsdatasett på 1.2 millioner høyoppløselige bilder. Krizhevsky, Sutkever og Hinton [25] vant konkurransen i 2012. De tok i bruk CNN med fem konvolusjonelle lag, hvor noen var etterfulgt av bassenglag, etterfulgt av tre fullstendige tilkoblede lag med softmax til slutt. For regularisering brukte de «Dropout» som ga gode resultater.

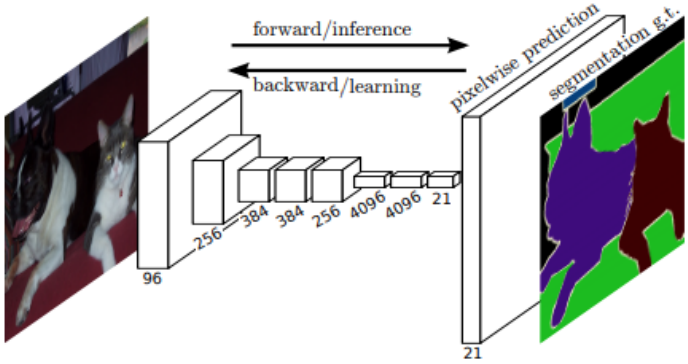
2.1.11 Fullstendige konvolusjonelle nettverk

Long, Shelhamer og Darell publiserte i 2015 «fully convolutional networks for semantic segmentation» [28], dette er et av de første forsøkene på å bruke CNN for pikselbasert veiledet klassifisering. Dette ble gjort ved å tilpasse de moderne klassifiseringsnettverkene Alexnet [24], VGGnet [43] og GoogLeNet [44] til fullstendige konvolusjonelle nettverk (FCN), og finjustere dem til å utføre segmenteringsoppgaver nøyaktig og detaljert. Figur 2.10 illustrerer hvordan et bilde kan bli segmentert basert på piksler med et fullstendig konvolusjonellt nettverk.

2.1.12 U-net

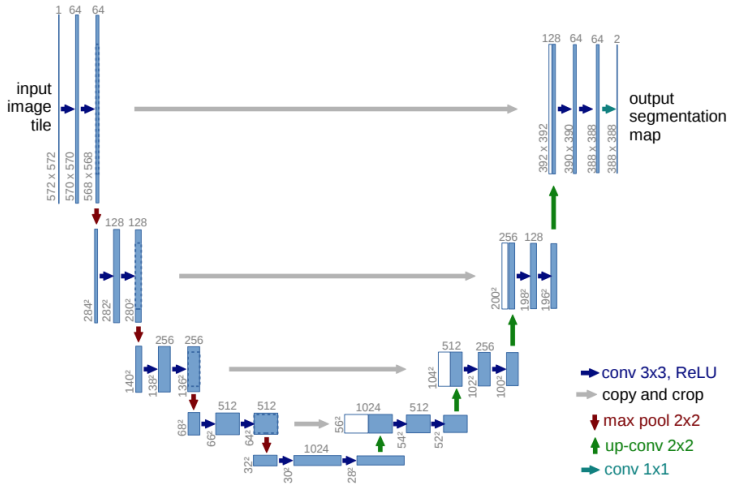
U-net ble laget i 2015 av Ronneberger, Fischer og Brox [36] for å utføre biomedisinsk bildesegmentering hvor det er få bilder tilgjengelig. I mangel på bilder tar de i bruk «dataaugmentaion» (eng: augmentation) for å bruke tilgjengelige bilder mer effektivt. Dataaugmentation innebærer blant annet å skalere, snu og tilføre støy i bilder.

Arkitekturen er basert på fullstendig konvolusjonelt nettverk [28] som utvides og modifiseres, for å fungere med få treningsbilder og gi mer presise segmente-



Figur 2.10: Fullstendig konvolusjonell nettverk. Figur 1, hentet fra [28]

ringer. En viktig endring i arkitekturen fra fullstendig konvolusjonelle nettverk til U-net, er at i oppsamlingsdelen er det mange «feature channels» som gjør at nettverket propagerer informasjon til lag med høyere oppløsning. Dette gir den tydelige U-formen som er illustrert i figur 2.11. Hver blå boks er et flerkanal (multi-channel) «featurekart» med antall kanaler på toppen av boksen og X-Y størrelsen i bunnen. Hvite bokser representerer kopierte «featurekart» og piler representerer de ulike opperasjonene. Arkitekturen inneholder ingen fullstendige konvolusjonelle lag.

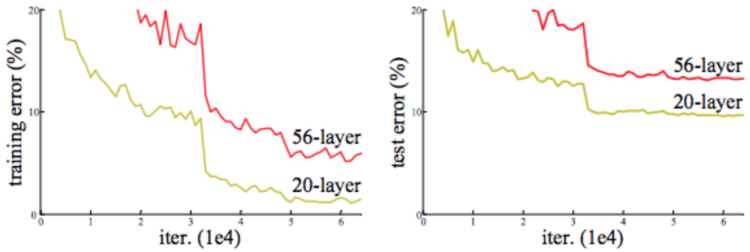


Figur 2.11: U-net arkitekturen. Figur 1, hentet fra [36]

2.1.13 Residuale nettverk

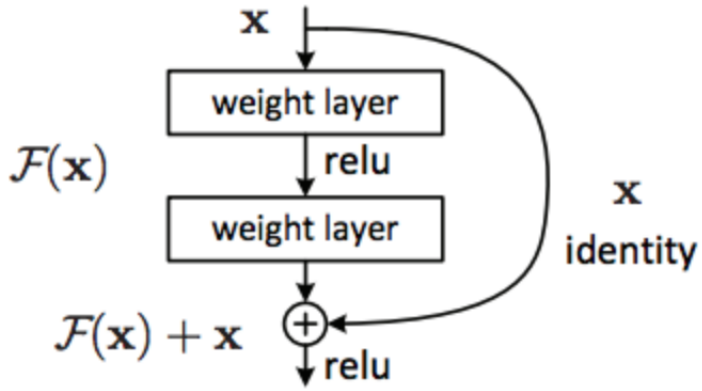
He, Zhang, Ren og Sun presenterer i artikkelen «Deep Residual Learning for Image Recognition» [18], residual læring for å adressere degraderingsproblemet. Proble-

met oppstår ved trening av dype nevrale nettverk, når det begynner å konvergere. Dette fører til at nøyaktigheten blir mettet og vil degraderes raskt. Problemet skjer ikke som følge av overtilpassing som man vil forvente, men som følge av høyere treningsfeil. Figur 2.12 viser at de dypeste nettverkene med 56 lag har høyere treningsfeil enn ved 20 lag.



Figur 2.12: Det dypeste nettverket har høyere treningsfeil, som følge av degraderingsproblemet. Figur 1. hentet fra [18]

Forfatterne introduserer et nytt nevralt nettverkslag; residualblokk (eng: The Residual Block) , som vist i figur 2.13. Identitetskartleggeren blir og kalt «Skip Connection». Den har ingen parametere, men i stedet legger den til utputtet fra det tidligere laget, til laget foran. Identitetskartleggeren blir multiplisert med en lineær projeksjon, W, for å få samme oppløsning som bildet for deretter å kombinere disse som innputt til neste lag.



Figur 2.13: Byggeklokken i residual læring. Figur 2. hentet fra [18]

2.1.14 Mål for score - Scoring metrics

Mål for score er en metode for å følge med på nøyaktigheten til nettverket ved trening og testing. Denne defineres basert på problemet som skal løses og hvordan suksess skal defineres. Valg av mål for score avhenger av type data, og mengde data for hver klasse.

Forvirringsmatrise - Confusion matrix

En forvirringsmatrise er en matrise som summerer antall sann positiv (TP), sann negativ (TN), falsk positiv (FP) og falsk negativ (FN) predikert av modellen. Disse verdiene brukes videre i de ulike scoreberegningene.

Figur 2.14: Forvirringsmatrise, bilde fra [33]

Nøyaktighet - Accuracy

Nøyaktighet gir en indikasjon på hvor mange tilfeller som er feilklassifisert ved å kalkulere summen av korrekte prediksjoner delt på det totale antallet prediksjoner. Hvis klassene er i ubalanse, kan nøyaktighet gi en feiloppfatning av resultatet fordi den forteller antall negative tilfeller når den positive klassen er underrepresentert [33].

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} \tag{2.16}$$

Presisjon - Precision

Presisjon predikerte positive tilfeller som er sant positive [32].

$$PRE = \frac{TP}{TP + FP} \tag{2.17}$$

Tilbakekalling - Recall

Tilbakekalling er antall positive prediksjoner i forhold til bakkesannheten. Dette gir et resultat som forteller hvor komplett de positive prediksjonene er [32].

$$REC = \frac{TP}{FN + TP} \tag{2.18}$$

F1-score

F1-score er en kombinasjon av presisjon og recall [33].

$$F1 = 2 \cdot \frac{PRE \times REC}{PRE + REC} \tag{2.19}$$

Mean intersection over union - mIOU

Mean intersection over union (mIOU) er en mye brukt scoreberegning i segmentering, for eksempel i «Pascal VOC» [10] og [28]. Ved å dele arealet som overlapper hverandre på arealet av unionen, blir resultatet hvor nøyaktig segmentet er i forhold til sannheten. Arealet i denne sammenheng kan for eksempel være et

skogsegment, hvor en legger sannsegment over predikert skogsegment. Deretter utgjør overlappen mIOU.

$$mIOU = \frac{TP}{TP + FP + FN} \tag{2.20}$$

Kapittel 3

Metode

I dette kapittelet presenteres fremgangsmåten i studien. Først presenteres innhenting av datagrunnlag og behandling av data, før valg av modell og optimalisering av nettverk er videre beskrevet.

3.1 Datagrunnlag

3.1.1 Sentinel-2

Sentinel-2 er fellesbetegnelsen for to identiske europeiske satellitter som går i polar solsynkron bane med 180 grader avstand. Sentinel-2a og Sentinel-2b ble skutt opp henholdsvis 23.06.15 og 07.03.17 og er beregnet til å operere i 7.25 år, men batteriet er ment å vare i 12 år [41]. Figur 3.1 viser hvordan de to satellittene står i forhold til hverandre.

Polar bane betyr at satellittene kretser nær polene og dekker jorda fra 58 grader sør til 84 grader nord. Solsynkron bane betyr at satellittene vil passere over en bestemt breddegrad til to faste lokale tider. Med to satellitter vil samme område ved ekvator bli besøkt hver femte dag. Satellitten er solsynkron fordi det optiske instrumentet er passivt, og fungerer ved at det samler inn reflekterte solstråler fra jorda. Med fem dagers besøkstid er det mulig å oppnå skyfrie bilder som kan brukes videre.

Sporbredden er 290 km, som gir et stort dekningsområde for hver passering. Hensikten med satellittene er å overvåke variasjoner på landoverflaten, som forurensing i innsjø, vann og kystområder, kartlegging av katastrofer, skogbruk og jordbruk.

Det optiske instrumentet (MSI - Multispectral Imager) registrerer data og fordeler det i 13 spektrale bånd. Dette innebærer fire bånd med 10 meters oppløsning; rødt, grønt, blått og nærinfrarødt og seks bånd med 20 meter og tre bånd med 60 meters oppløsning [41] som vist i tabell 3.1.

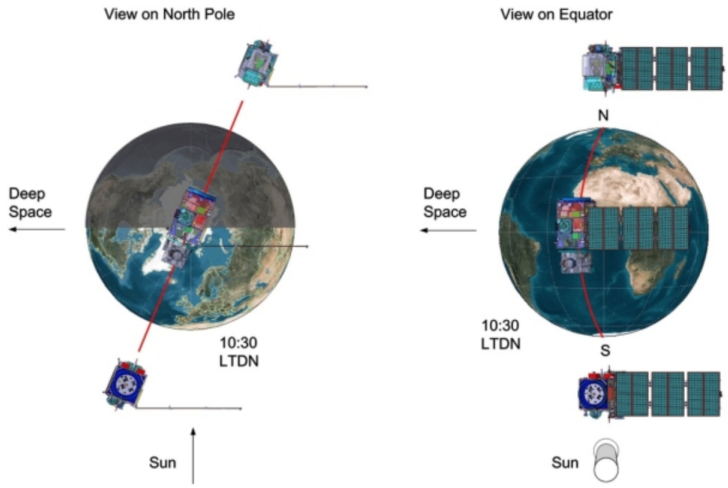
Forskjellige vegetasjonstyper absorberer og reflekterer sollys ulikt. Båndene B02; blå, B03; grønn og B04; rød fanger opp det synlige lyset, og kan settes sammen i ulike kombinasjoner for å vise bildet i ekte eller falske farger. Båndet B08; nærinfrarødt fanger opp reflektert lys menneskeøyner ikke kan se. Frisk vegetasjon gir høy refleksjon i dette spekteret fordi de lange bølgelengdene trenger gjennom det øverste laget av planten og reflekteres fra celler midt i bladet/bårnålen. Dette gjelder på samme måte som menneskets øyne ser det grønne i blader; ved at det grønne lyset fra klorofyllet reflekteres, mens det røde og blå lyset til motsetning blir absorbert [7]. For å visualisere det nærinfrarøde båndet kan man bytte det ut med det røde, grønne eller blå båndet.

Bilder fra Sentinel-2 er mulig å laste ned fra eos.com/landviewer. Der kan man velge prosentandel skyer og solvinkel det skal være på bildet som lastes ned. Man kan laste ned Sentinel-2 L1C-bilder som gir refleksjonen *over* atmosfæren. Etter 28.03.2017 kan man og laste ned Sentinel-2 L2A-bilder som gir refleksjonen *under* atmosfæren. Begge har radiometriske og geometriske korreksjoner, og er georefererte. Forskjellen er at sistnevnte gir et klarere bilde hvor disen av atmo-

Tabell 3.1: Båndene i Sentinel-2, med bruksområde. Hentet fra [40]

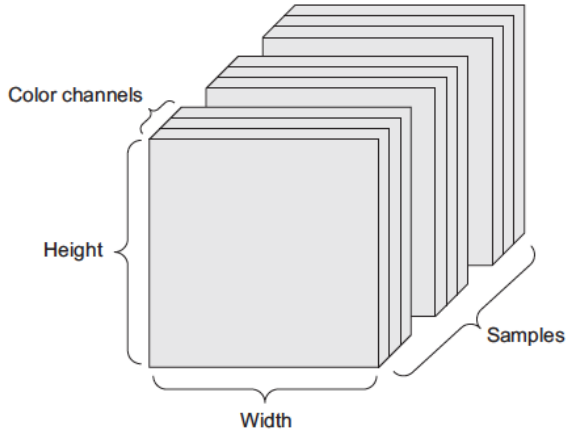
Båndnavn	Oppløsning (m)	Sentral bølgelengde (nm)	Bånd-bredde (nm)	Bruksområde
B01	60	443	20	Aerosoldeteksjon
B02	10	490	65	Blå
B03	10	560	35	Grønn
B04	10	665	30	Rød
B05	20	705	15	Vegetasjonklassifisering
B06	20	740	15	Vegetasjonklassifisering
B07	20	783	20	Vegetasjonklassifisering
B08	10	842	115	Nærinfrarød
B08A	20	865	20	Vegetasjonklassifisering
B09	60	945	20	Vanndamp
B10	60	1375	30	Cirrus
B11	20	1610	90	Skille mellom snø / is / skyer
B12	20	2190	180	Skille mellom snø / is / skyer

sfæren er fjernet [42].



Figur 3.1: Figuren illustrerer hvordan de to satellittene står i forhold til hverandre. Hentet fra [41], Figur 1.

Et satellittbilde har tre dimensjoner; høyde, bredde og dybde, som vist i figur 3.2. Avhengig av om bildet er i for eksempel gråtoner eller RGB-farger vil dybden variere. Et satellittbilde med 4 bånd; rødt, grønt, blått og nærinfrarødt som er 256*256 piksler, vil bli lagret i en tensor med størrelse (X, 256, 256, 4), hvor X er antall bilder[5].



Figur 3.2: Bildedimensjon ved tre bånd. Figur 2.4 i [5]

Gašparović og Jogun [13] testet ut to modeller; Maximum Likelihood Classifier (MLC) og artificial neural networks (ANNs) og undersøkte hvordan fusjonering av bånd endrer nøyaktigheten i modellene. Forfatterne fikk et noe bedre resultat på klassifisering av landareal ved å forbedre oppløsningen av lavoppløste bånd, og fusjonere de med høyoppløselige bånd på 10 piksler. Klassifiseringene ble mer detaljerte, men fusjoneringen førte også til økning av støy i klassifiseringene.

Heryadi og Miranda [19] undersøkte i sin stuide hvordan nøyaktigheten i en CNN-modell kunne forbedres ved å endre og legge til ulike bildeegenskaper. Resultatet ble at CNN-modellen viste litt forbedringer i nøyaktigheten av klassifiseringen av landareal ved å endre og legge følgende bildeegenskaper: NDVI (Normalized difference vegetation index), lysstyrke, GLCM (Gray-Level Co-occurrence Matrix) homogenitet og rektangulær tilpassing.

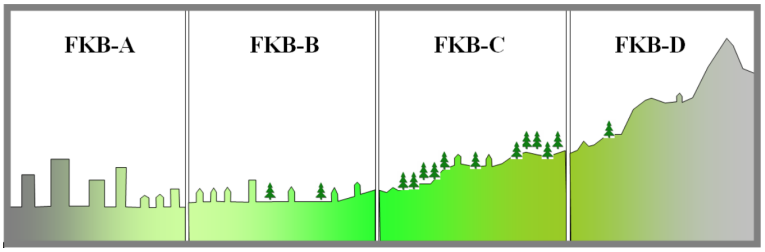
Vegetasjonsindeks (NDVI) gjør det mulig å skille mellom ulike vegetasjonstyper basert på hvor mye klorofyll og vann det er i vegetasjonen. Formelen for å beregne NDVI er:

$$NDVI = \frac{r_{NIR} - r_R}{r_{NIR} + r_R} \tag{3.1}$$

I formelen er r_{NIR} refleksjonsverdien av det nærinfrarøde lyset, og r_R er refleksjonsverdien av det røde lyset. Indeksen varierer fra -1.0 til 1.0, hvor frisk vegetasjon vil ha verdier mellom 0.7 til 1.0, og overflater som vann, is og snø vil ha negative verdier [6]. Dette innebærer i praksis at en kan skille mellom ulike arealtyper basert på refleksjonsverdier.

3.1.2 Felles kartdatabase (FKB)

Felles kartdatabase (FKB) består av de mest detaljerte kartdataene i Norge [12]. Etablering og vedlikehold av dataene foregår gjennom avtalen Geovekst, som omhandler deling og innsamling av kartdata. Sentrale parter i avtalen er Statens kartverk, Statens vegvesen, kommuner og landbruket. Sentral felles kartdatabase (SFKB) er et forvaltningssystem hvor brukere kan hente ut oppdatert og kvalitetssikrede data. FKB inneholder vektordata som brukes i kart, saksbehandling, prosjektering og geografiske analyser [12]. FKB er delt inn i fire standarder; FKB-A, FKB-B, FKB-C og FKB-D, vist i figur 3.3, basert på behovet for detaljer og stedfestingsnøyaktighet i det aktuelle område. FKB-A har høyest nøyaktighet og blir typisk brukt i byer, og nøyaktigheten er mellom 10 cm til 55 cm i grunnriss. FKB-D har lavest nøyaktighet, og brukes der det er mindre behov for detaljer, slik som i fjellområder. Der er nøyaktighet mellom 48 cm til 100 cm i grunnriss. [21]



Figur 3.3: FKB-data fordelt på ulike områdetyper. Bilde hentet fra [21] , figur 1

Arealressurskart, AR5

Arealressurskart med målestokk 1:5000 inngår i FKB, og kalles AR5. Norges institutt for bioøkonomi - NIBIO er fagansvarlig for datasettet, og kommunene har ansvar for kontinuerlig ajourføring av dette.

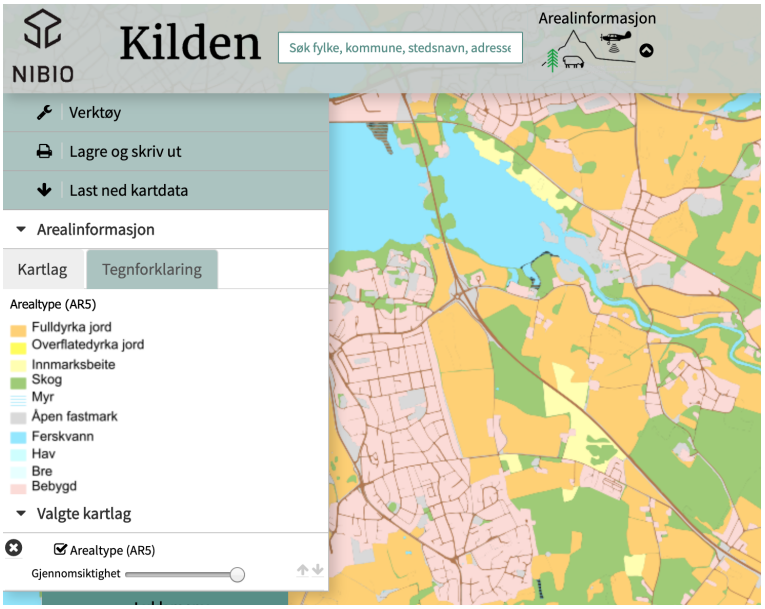
AR5 er et klassifikasjonssystem for arealressurser med vekt på jord- og skogbruk. Dette er den mest pålitelige kilden til informasjon om arealressursene i Norge. AR5 er felles for hele Norge, hvor hovedinndelingen er basert på arealtyper, der de har følgende tilleggsegenskaper: treslag, skogbonitet og grunnforhold. Disse fremstilles visuelt i kart som polygoner. Figur 3.4 viser et kartutsnitt fra nettsiden Kilden [22] med AR5 tegnforklaring.

Arealtype kan ha 12 verdier;

- Fulldyrka jord
- Overflatedyrka jord
- Innmarksbeite
- Skog
- Myr

- Åpen fastmark
- Ferskvann
- Hav
- Bre
- Samferdsel
- Bebygd
- Ikke kartlagt

Jordbruksareal brukes som en samlebetegnelse på fulldyrka jord, overflatedyrka jord og innmarksbeite [1]. For å laste ned data må man være medlem av *Norge digitalt-samarbeidet*. Som student ved NMBU kan man laste ned AR5-data gratis.



Figur 3.4: Eksempel på kartutsnitt med AR5 tegnforklaring, skjermbilde av Hamar, fra Kilden [22]

3.2 Maskinvare

- GPU: GeForce GTX 1650, 4 gb
- RAM: 16 gb

I denne studien er tilgjengelig maskinvare en begrensing på gjennomføringen. Det vil alltid være ønskelig med størst mulig datagrunnlag, og kapasitet for å prosessere dette, men tilgjengelig maskinvare vil være premissgivende for størrelsen på data som kan samles inn og behandles. På PC anvendt i denne studien, er det maskinvare som gjør det mulig å kjøre modellene med cirka 1000 bilder, ved batchstørrelse på 20 bilder. Det tar da 25 sekunder pr. epoke. Dette er maksgrensen på denne PCen uten at GPUen blir overbelastet.

3.3 Databehandling

3.3.1 Arealressurskart (AR5)

I denne studeien er det benyttet arealressurskart i målestokk 1:5 000, som ble lastet ned fra Geonorge av Ivar Maalen-Johansen, Amanuensis ved NMBU, 7.januar 20200, og lagret på min PC. Følgende datasett over fylker ble lastet ned; Troms og Finnmark, Vestland, Viken, Innlandet og Nordland. Arealressurskart består av 11 arealtyper og flere underkategorier. Ettersom satellittbildene har en pikselstørrelse på 10 meter er det nødvendig å samle sammen arealtypene i færre klasser fordi modellen kan slite med å skille ulike arealtyper. Dette kan for eksempel gjelde arealtypene «bebygd» og «samferdsel».

Tabell 3.2: Arealtypene i AR5 og fordelingen i nytt datasett.

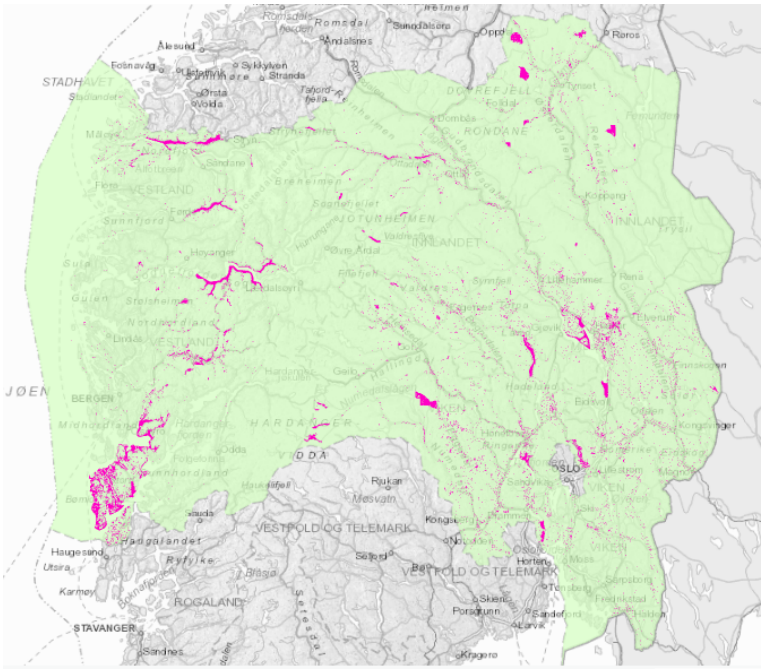
Arealtype	Verdi	Ny arealtype	Ny verdi
Bebygd	11	Bebygd	4
Samferdsel	12	Bebygd	4
Fulldyrket jord	21	Jordbruk	3
Overflatedyrket jord	22	Jordbruk	3
Innmarksbeite	23	Jordbruk	3
Åpen fastmark	50	Jordbruk	3
Skog	30	Skog	2
Myr	60	Skog	2
Hav	82	Vann	1
Ferskvann	81	Vann	1
Bre	70	Vann	1
Ikke kartlagt	99	Ikke kartlagt	0

Basert på hvordan arealtypene klassifiseres [1] er det valgt inndelingen som vist i tabell 3.2. De nye *verdiene* er fra 0 til 4 og fungerer både som en kode for

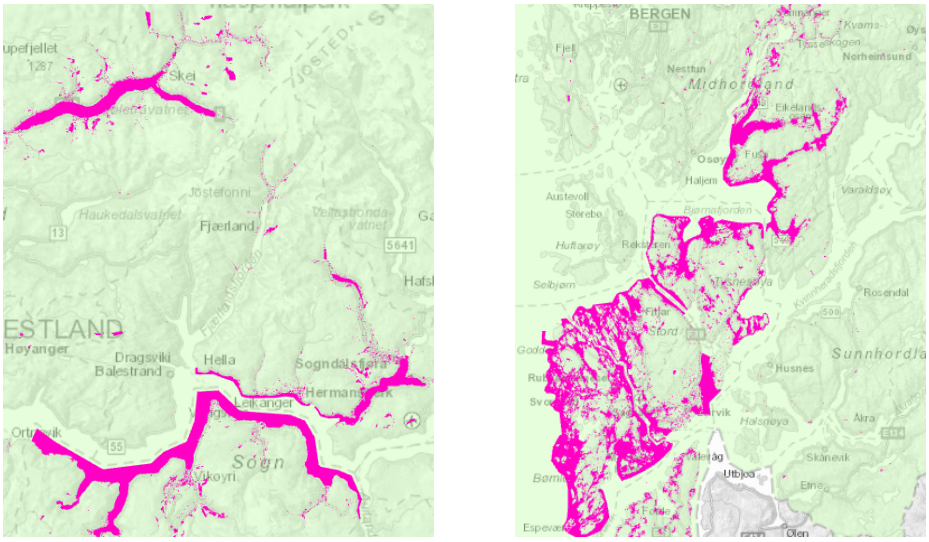
arealklassen den representerer, men også en rangering av klassene og prioritert rekkefølge. For eksempel har 4, bebygd, høyest prioritering.

- Arealtypene *bebygd* og *samferdsel* blir slått sammen til «bebygd».
- Arealtypene *fulldyrket jord*, *overflatedyrket jord*, *innmarksbeite* og *åpen fastmark* blir slått sammen til «jordbruk».
- Arealtypene *hav*, *ferskvann* og *bre* har blitt slått sammen til «vann». Arealtypen bre er ikke i datasettet, men ble likevel lagt til i denne klassen.
- Arealtypen «myr» er vanskelig å skille fra skog da arealet ofte er lite og omringet av skog. Studering av satellittbilder med menneskelige øyner er i seg selv vanskelig, da områdene ikke er homogene og varierer i utseende. Derfor blir «myr» kategorisert som arealtypen «skog».

Arealressurskartet er lagt inn i Arcgis Pro og analysert for å finne områder som egner seg for å bruke som trening-, test- og valideringsdata. Arealressurskartet inneholder også dato for datafangst av attributtene. I et området vil derfor noen attributter være oppdatert nylig, mens andre ikke har blitt oppdatert siden de ble opprettet. Typiske områder som ikke har blitt oppdatert på mange år er fjell-, skog- og jordbruksområder hvor det er lite bebygde områder og hvor det har skjedd få endringer. Ettersom Sentinel-2a ble skutt opp 23.06.2015 og Sentinel-2b skutt opp 07.03.2017, er det ønskelig å finne områder som er oppdatert etter 22.06.2015.



Figur 3.5: Det grønne området viser datasettet tilgjengelig øst i Norge, mens de rosa områdene viser oppdaterte områder etter 22.06.15.



(a) Sogn **(b)** Bergen og Stord

Figur 3.6: Oppdaterte områder, som er uegnet som treningsområde.

(a) Hamar og Elverum. **(b)** Drammen.

Figur 3.7: Oppdaterte områder, som egner seg som treningsområde.

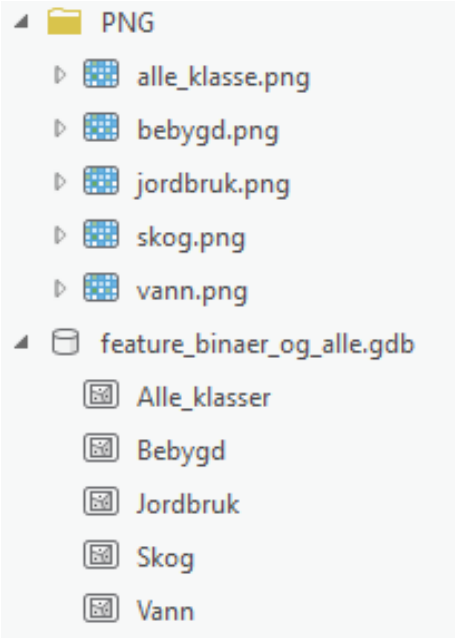
Figur 3.5 viser et utsnitt av datasettet som er analysert. Analysen viser at 367 143 områder av 4 637 666 områder er oppdatert etter 22.06.15. Det vil si 0.8 % av datasettet. Ettersom datasettet kun inneholder data for fem av elleve fylker er ikke dette representativt for hele Norge, men tatt i betraktning av at Norge består av 1.7 % bebygde områder og 3.5 % jordbruksområder [2], er ikke 0.8 % en lav andel oppdatert data. Arealene som er oppdatert befinner seg hovedsakelig rundt bebygde områder, da det er bebygde områder som oftestes endres.

For å finne egnede områder til bruk som treningsdatasett, er datasettet studert. I Vestland fylke kan man se at de oppdaterte områdene befinner seg langs fjord og kystlinje og er store oppdaterte områder, vist i figur 3.6. Men, fordi det kun er arealtypen «vann» som er oppdatert er ikke dette ikke av stor interesse

for studien. Det er også dårlig sammensetning av arealtypene, for eksempel at de bebygde områdene er for små. Dette gjelder også for områder i fylkene Nordland og Troms og Finnmark.

Figur 3.7 viser oppdaterte områder rundt Hamar og Drammen. Dette er mellomstore byer og har oppdaterte data fra 2016. Arealene er godt fordelt mellom bebygd, jordbruk, skog og vann. Hamar og Drammen er derfor naturlige steder å bruke som treningsdata. Av samme grunn tas de bebygde områdene i Elverum med i datasettet.

For å klargjøre arealklassedatasettet for å lage treningsdata, behandles det på to måter. Ett datasett med alle klassene samlet, og fire binære datasett hvor de fire klassene klassifiseres hver for seg. De blir lagret i en geodatabase som «feature class» og så eksportert som PNG-bilder til en mappe som vist i figur 3.8.



Figur 3.8: Geodatabase med vektordata og PNG raster bilder basert på vektordataene.

Flerklasse

I attributtabellen legges det til to kolonner; classvalue og classname. Disse er nødvendige for å lage treningsdatasettet med verktøyet «Export trainingdata for Deep learning» [11] i Arcgis. I kolonnen «classvalue» og «classname» legges det til ny attributtverdi og navn i forhold til tabell 3.2. «Classvalue» inneholder da verdier

fra 0-4 og «Classname» inneholder navn.

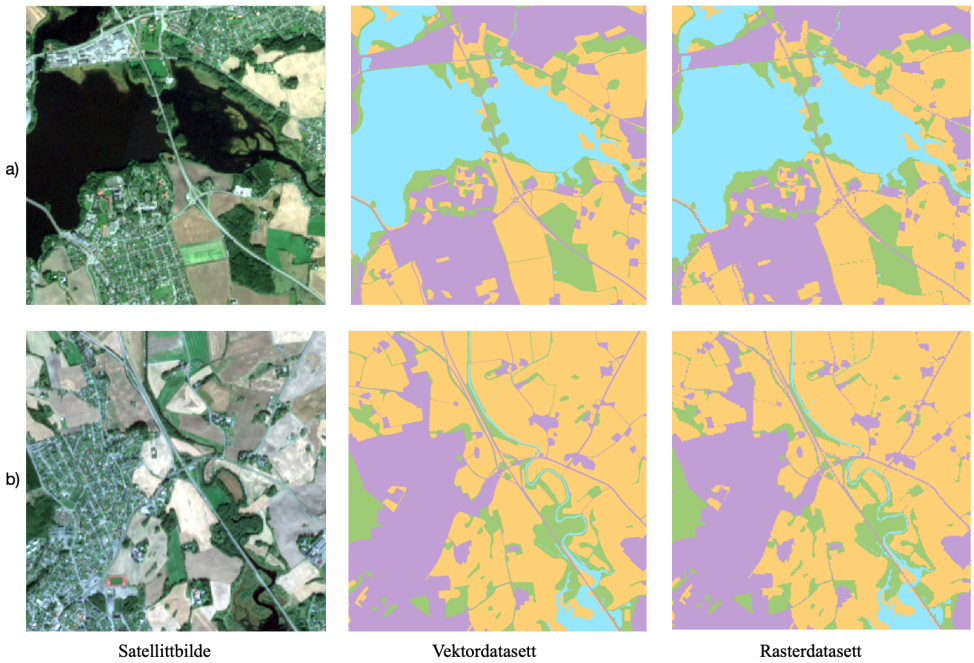
Binærklasse

Basert på flerklassedatasettet lages det fire nye datasett; bebygd, skog, jordbruk og vann. I attributtabellen endres «classvalue» og «classname» fra 0-4 til 0 og 1. Datasettet for bebygd vil for eksempel ha «classvalue» = 1 for områder som er bebygd og resterende areal vil ha «classvalue» = 0 med «classname», bakgrunn.

PNG

Videre gjøres de fem vektordatasettene om til rasterdatasett. De lagres som georefererte PNG-bilder med 10 meters pikselsoppløsning, slik som satellittbildene.

Figur 3.9 viser eksempler på hvordan oppløsningen i datasettet endrer seg fra vektordata til rasterdata. Noe informasjon går tapt, for eksempel kan smale veger eller skogstriper få hull i linjen i rasterdatasettet som man kan se i både a) og b). Midt i bilde a) er det et vegkryss som er tydelig å i vektordatasettet, men mindre tydelig i rasterdatasettet. I bilde a) kan man også se at noen øyer til høyre i bildet og øverst i venstre hjørne av bildet, ikke er registrert i arealressurskartet. Dette viser at arealressurskartet inneholder feilklassifiseringer.



Figur 3.9: Eksempel på hvordan vektordatasettet og rasterdatasettet ser ut.

3.3.2 Satellittdata

Nedlasting fra Landviewer

Satellittbilder lastes ned fra Landviewer [26] med fire bånd fusjonert sammen. På nettsiden kan man laste ned 10 satellittbilder per dag med gratis bruker. Ettersom bildene dekker store områder er det mulig å laste ned nok data for studien på kort tid. Satellittbildet over Hamar og Drammen blir lastet ned som L1C-bilder. Dette gir refleksjonen over atmosfæren med radiometriske og geometriske korreksjoner og er georeferert. Bildet over Hamar og Elverum er tatt 5.september 2016 med 0% skydekke. Bildet over Drammen er tatt 15.august 2015 med 2% skydekke. Disse to satellittbildene blir brukt til å lage trenings-, validerings- og testdatasett.

Det ble også lastet ned ett satellittbilde over et område i Nittedal kommune for å bruke det til å predikere et arealressurskart. Dette ble gjort for å se hvordan modellene presterer på et satellittbilde som ikke er brukt i trenings-, valideringsog testdatasett. Dette er også gjort for å vise hvordan et eventuelt arealressurskart vil se ut når det predikeres i modellene.

Av de 13 spektrale båndene som er tilgjengelig nedlastes båndene med 10 meters pikseloppløsning; rød, grønn, blå og nærinfrarødt.

Det eksisterer nyere satellittbilder, men for å få et mer nøyaktig datasett er satellittbildene som lastes ned så nær i tid som oppdateringsdatoen i arealressurskartet som mulig.

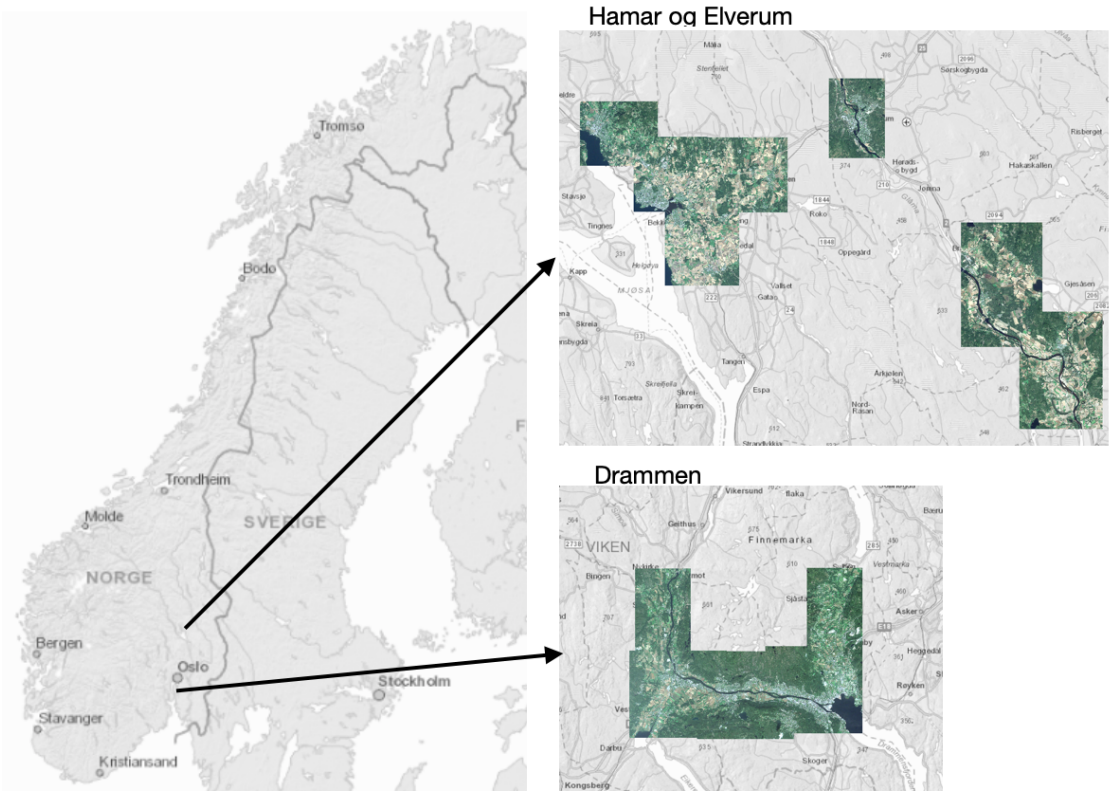
Bildebehandling i Arcgis Pro

I Arcgis klippes bildene for å begrense områder av skog og vann. Figur 3.10 viser de utklipte områdene som er brukt som treningsdatasett. Selv om datasettet ikke fysisk henger sammen, tilhører det samme datasett.

Satellittbildet er behandlet på tre ulike måter for å se hvilken kombinasjon av bånd som gir det beste resultatet i modellene. Det blir delt inn i:

- 3 bånd: med bånd 1, 2 og 3; rød, grønn og blå.
- 4 bånd: med bånd 1, 2, 3 og 4;rød, grønn, blå og nærinfrarødt.
- 5 bånd: med bånd 1, 2, 3, 4 og 5;rød, grønn, blå, nærinfrarødt og NDVI-vegetasjonsindeks.

Figur 3.11 viser et lite utsnitt av satellittdatasettet og hvordan de ulike båndene ser ut. Når de ulike kombinasjonene av bånd er fusjonert sammen, vil alle tre datasett se ut som a) visuelt i Arcgis, men de har ulikt antall bånd. Dette er fordi det vises i ekte farger. Vegetasjonsindeksen beregnes ved bruk av verktøyet «NDVI colorized function» [30].



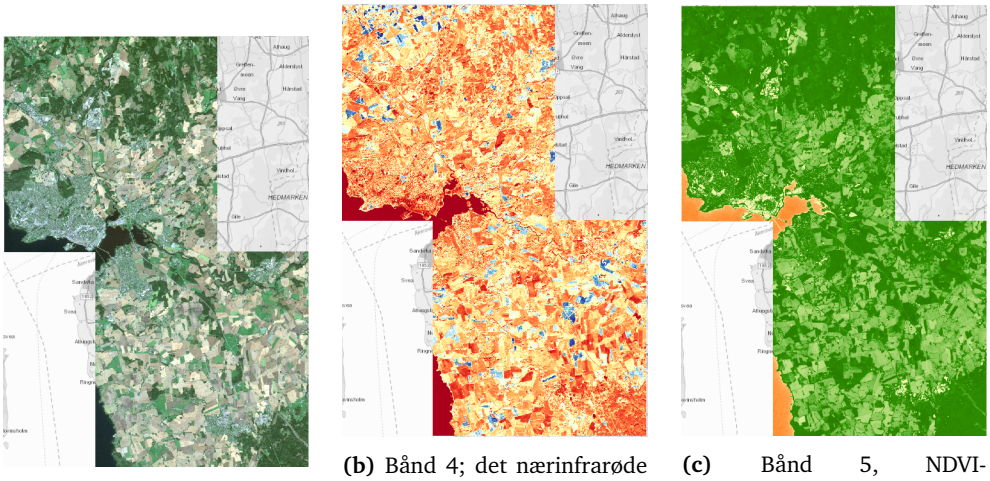
Figur 3.10: Datasettets utstrekning.

3.3.3 Produksjon av treningsdatasett

Ved bruk av prosesseringsverktøyet «Export trainingdata for deep learning» [11] i Arcgis Pro, deles satellittbildene i kvadrater på 256*256 piksler med en overlapp på 100 piksler. De tilgjengelige satellittbildene gir et treningsdatasett på 1044 bilder. Videre deles PNG-arealressurskartet opp i kvadrater basert på de 1044 satellittbildene. Dette tar totalt 5 timer for hvert datasett. Dataene blir lagret i to mapper; «Bilder» og «Labels». Disse brukes for å trene, teste og validere modellen. Alle de 1044 bildene og labelene er georefererte.

Prosessen som er beskrevet over gjøres tre ganger for flerklassedatasette; en gang for hvert av de ulike båndkombinasjonene i satellittbildene. U-net modellene optimaliseres på flerklassedatasettet. Den båndkombinasjonen i satellittbildet som gir best resultat i optimaliseringen brukes videre til å lage de binære treningsdatasettene.

For de binære datasettene gjøres dette en gang for hver av de fire arealklassene sammen med satellittbildet som gir best resultat i optimaliseringen for flerklasse-



(a) Bånd 1, 2 og 3; rødt, grønt og blått, vist med ekte farger.

(b) Bånd 4; det nærinfrarøde båndet, vist med fargeskala blå til rød, hvor blå har høyest verdi og rød lavest.

(c) Bånd 5, NDVI-vegetasjonsindeks, hvor mørk grønn har høyest verdi og rød er laveste verdi.

Figur 3.11: Et utsnitt av satellittdatasettet og kombinasjonene av bånd.

datasettet.

3.4 Nettverksarkitektur

Det er flere typer arkitekturer som er interessante å vurdere for studien, blant annet gir residuale nettverk av ulike lengder og U-net gode resultater på klassifisering av piksler i bilder. Dype residuale nettverk krever mer beregningskraft enn U-net, og er derfor ikke å foretrekke med begrenset datakraft.

I denne studiene er det valgt å ta i bruk U-net fordi den har tidligere gitt gode resultater for denne typen segmentering. Mällberg og Rolfsen [29] skrev i sin masteroppgave ved NTNU i 2018 om semantisk segmentering av flyfoto med CNN. Forfatterne tar i bruk ulike konfigurasjoner av DenseNet og U-net, og tester dette på både binære- og flerklassedatasett. Resultatet viste at U-net i hovedsak gir de beste resultatene.

Det er utarbeidet to modeller, en baserer seg på den originale arkitekturen, mens den andre modellen har en ekstra blokk. Scriptene brukt i denne studien ligger vedlagt i vedlegg B.

3.4.1 U-net-4: Original arkitektur, fire blokker

Arkitekturen som er valgt, bygger på forelesningsnotater fra forelesningen «CNN for images» i DAT300 ved NMBU. Arkitekturen er lik som i Ronnebergers [36] originale arkitektur, vist i figur 2.11.

Figur 3.12 viser oppbyggingen av blokkene. Aktiveringsfunksjonen som blir brukt i blokkene er «ReLu», mens for den siste blokken blir «Softmax» brukt for flerklasse og «Sigmoid» for binærklasser. Denne modellen har 3 372 502 trenbare parametere. Denne modellen vil videre i oppgaven bli kalt U-net-4. Figur 3.13a viser full oppbygging av U-net-4.

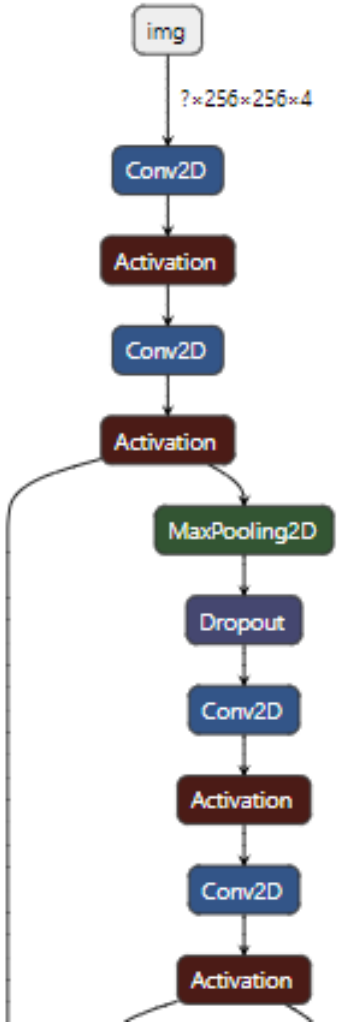
3.4.2 U-net-5; Utvidet arkitektur, fem blokker

Den utvidede arkitekturen er lik den originale arkitekturen, men har en blokk mer. Denne modellen har 13 512 342 trenbare parametere. Denne modellen vil videre i oppgaven bli kalt U-net-5. Figur 3.13b viser full oppbygging av U-net-5.

3.5 Trening av nettverket

Før nettverket kan trenes må det velges parametere som gir best resultat. Dette gjøres ved optimalisering av nettverket. Under optimaliseringen brukes 20 epoker, og for treningen brukes 40 epoker.

Datasettet deles inn i trenings-, validerings- og testdata, slik at satellittbildene som brukes i alle tilfeller er av samme utsnitt. Antall bånd i satellittbildene er



Figur 3.12: Blokkene i U-net modellen.

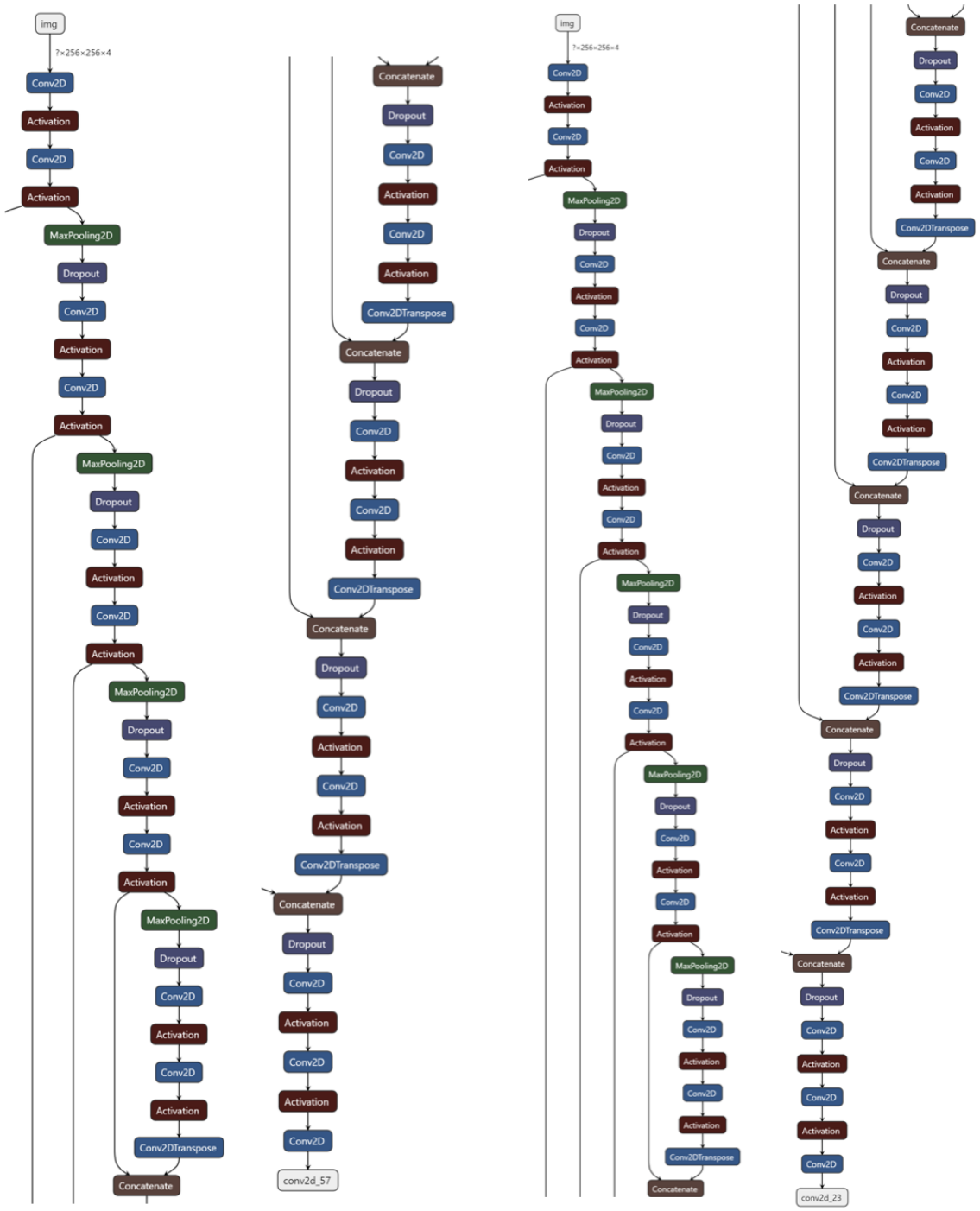
ulikt, men utsnittet er det samme. Fasitdataene består av arealressurskart med ulike verdier basert på om det er for flerklasse eller binært, med samme utsnitt. Fordelingen av totalt 1044 bilder:

- Trening: 750 bilder
- Validering: 193 bilder
- Test: 99 bilder

3.5.1 Fordeling av arealtyper i datasettene

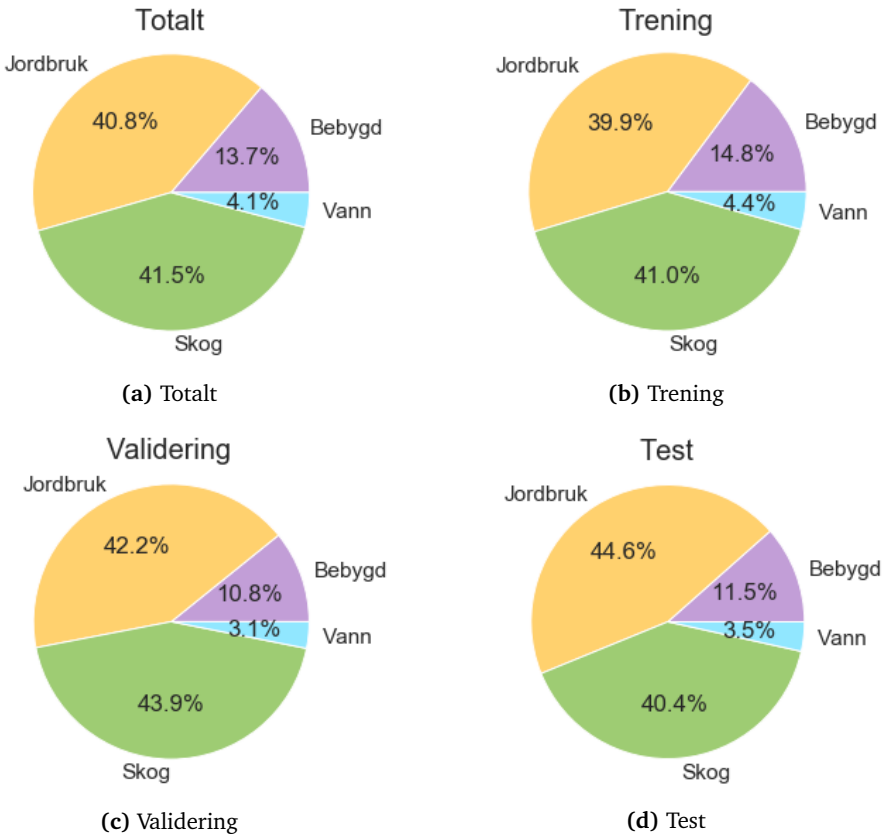
Fordelingen av areal i datasettet er:

- Bebygd: 88.1 km^2



(a) U-net med 4 blokker **(b)** U-net med 5 blokker

Figur 3.13: Modellenes oppbygging delt i to etter den kontraherende delen og den ekspanderende delen. «Concatenate» nederst og øverst er felles.



(a) Totalt

(b) Trening

(c) Validering

(d) Test

Figur 3.14: Fordeling av piksler for de ulike arealtypene i flerklassedatasettet.

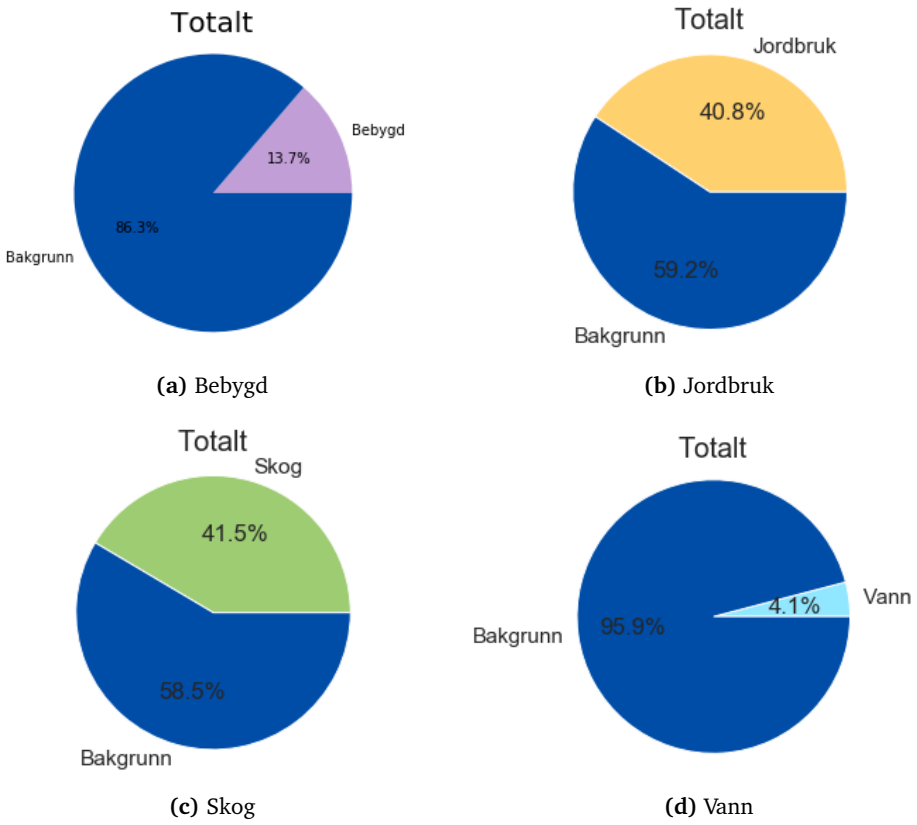
- Jordbruk: 777.6 km^2
- Skog: 805.1 km^2
- Vann: 7.7 km^2

Flerklassedatasettet

Fordelingen av pikslene i flerklassedatasettet er vist i figur 3.14. Her kan man se at datasettet består av mye jordbruk og skog, og lite vann. Man kan og se at trenings-, validerings- og testdatasettene har omtrent den samme fordelingen av arealtyper.

Binærdatasettet

Fordelingen av pikslene i de binære datasettene er vist i figur 3.15. Her kan man se at den samme prosentandelen fra figur 3.14a går igjen i de fire binære datasettene. Fordelingen av trenings-, validerings- og testdatasettet har derfor også lik prosentandel i de binære datasettene som i figur 3.14 ettersom den samme inndelingen av bilder brukes.



(a) Bebygd

(b) Jordbruk

(c) Skog

(d) Vann

Figur 3.15: Fordeling av piksler for de ulike arealtypene i binæredatasettet.

3.6 Optimalisering av nettverket

Ved optimalisering av nettverkene er det to faktorer som er interessant å se på, og som legges til grunn for endelig valg av modell og treningsdata før nettverkene testes mot testdatasettet:

- Hvilken kombinasjon av bånd i satellittbildet gir best resultat.
- Hvilke parametere for de to modellene gir best resultat.

Optimaliseringen er delt i to separate deler. En for datasettet med flerklasse og en for datasettet med binære klasser. De to U-netalgoritmene optimaliseres i hvert av tilfellene med en batchstørrelse på 20 bilder, med 20 epoker og 18 konfigurasjoner. Hver konfigurasjon tar omtrent 10 minutter å gjennomføre. Optimaliseringen av de to modellen med de tre flerklassedatasettene tar da 6 timer per datasett, totalt 18 timer.

Den båndkombinasjonen i satellittbildet som gir best resultat i optimaliseringen med flerklassedatasettet blir videre brukt for å lage binære datasett med de fire arealklassene. For hver av de to modellene blir de fire klassene optimalisert. Med 18 konfigurasjoner tar dette totalt 24 timer.

3.6.1 Optimaliserer

Halvparten av flerklassedatasettet blir brukt for å finne ut hvilke optimaliserere som er hensiktsmessig å teste med. Adam, Adadelta, AdaGrad og RMSprop testes for de tre satellittbildene, hvor Adam og RMSprop gir best resultat og brukes videre.

3.6.2 Hyperparametere

Ved å «redusere læringsraten på platåer» [34], vil læringsraten endre seg når modellen ikke forbedrer seg over et visst antall epoker. De valgte parametrene er:

- Monitor: mIOU
- Faktor: 0.2
- Tålmodighet: 4
- Minimum læringsrate: [0.0, 0.01, 0.001]

Hvor minimum læringsrate er den eneste parameteren som endres i optimaliseringen, som vist i klammeparantesen ovenfor.

For å *regularisere* nettverket brukes teknikken dropout. Her testes fire verdier vist i klammeparantesen under:

- Dropout: [0.2, 0.3, 0.5]

3.6.3 Tapsfunksjon

For flerklassedatasettet brukes «Categorical crossentropy» som tapsfunkjson. For det binære datasettet brukes «Binary crossentropy».

3.6.4 Vurdering av resultater

For å vurdere resultatet brukes det mIOU og F1 score, hvor mIOU er den som overvåkes ved redusering av læringsraten. Den gir realistiske resultater ved bildesegmenentering selv om det er ubalanse i klassene. Det er og interessant å se på F1-score selv om mIOU gir det mest realistiske resultatet. mIOU valideringsresultat er også interessant å ta med for å tilse at modellen ikke over- eller undertilpasser seg.

3.7 Resultat fra konfigureringen, flerklasse

3.7.1 U-net-4

Resultatet av optimaliseringen som er vedlagt i Vedlegg A, mens tabell 3.3 viser resultatet av optimaliseringen for de tjue beste konfigurasjonene. Tabellen viser at den beste konfigurasjonen er med 4-bånds satellittbilde, Adam som optimaliserer, 0.2 i dropout og 0.01 i minimum læringsrate. Generelt gjør Adam det bedre som optimaliserer enn RMSprop. Topp 5 beste resultater har relativt lik score.

3.7.2 U-net-5

Resultatet av optimaliseringen er vedlagt i Vedlegg A, mens tabell 3.4 viser resultatet av optimaliseringen for de tjue beste konfigurasjonene, sortert etter mIOU. Tabellen viser at den beste konfigurasjonen er med 4-bånds satellittbilde, Adam som optimaliserer, 0.2 i dropout og 0.001 i minimum læringsrate. Topp 4 beste resultat gir et relativt likt score.

Den beste konfigurasjonen for U-net-4 og for U-net-5 er relativt like, men skilles ved ulike minimum læringsrater. Satellittbildet med fire bånd gir best resultat i begge tilfeller selv om det er tett ned til både fem bånd og tre bånd.

3.8 Resultat fra konfigureringen, binære klasser

Satellittbildet med fire bånd blir brukt for de binære klassene, ettersom det ga best resultat i optimaliseringen for flerklassedatasettet i begge U-netmodellene.

Tabell 3.5 og tabell 3.6 viser de beste resultatene av optimaliseringen for U-net-4 og U-net-5. Tabellene er delt inn i seksjoner basert på hvilken arealtype det klassifiserer. For hver av de fire seksjonene er de sortert basert på mIOU. Tabellene

Tabell 3.3: Resultat fra optimaliseringen av U-net-4. Sortert etter mIOU.

Antall bånd	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
4	Adam	0.2	0.01	0.759446	0.766429	0.862121
5	Adam	0.2	0.01	0.751962	0.759732	0.857130
5	Adam	0.2	0.001	0.746486	0.765077	0.853524
4	Adam	0.3	0.01	0.746382	0.765088	0.853368
3	Adam	0.2	0.001	0.744570	0.762210	0.852343
4	Adam	0.2	0.001	0.739733	0.752157	0.849045
3	Adam	0.3	0.001	0.731759	0.753832	0.843629
5	Adam	0.3	0.001	0.731422	0.731307	0.843492
3	Adam	0.2	0.01	0.729402	0.750230	0.841980
4	Adam	0.3	0.001	0.727043	0.738456	0.840254
5	Adam	0.3	0.01	0.720464	0.734210	0.835833
4	Adam	0.5	0.01	0.720184	0.750911	0.835676
3	Adam	0.3	0.01	0.715132	0.738276	0.832189
5	Adam	0.5	0.01	0.708946	0.723420	0.827884
4	Adam	0.5	0.001	0.707912	0.717258	0.827313
5	Adam	0.5	0.001	0.705714	0.723261	0.825682
3	Adam	0.5	0.001	0.701411	0.739316	0.822743
4	RMSprop	0.2	0.01	0.673623	0.706290	0.802213
4	RMSprop	0.3	0.01	0.668936	0.711854	0.798942
4	RMSprop	0.2	0.001	0.648093	0.712022	0.783622

Tabell 3.4: Resultat fra optimaliseringen av U-net-5. Sortert etter mIOU.

Antall bånd	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
4	Adam	0.2	0.001	0.756624	0.768917	0.860227
3	Adam	0.2	0.001	0.745640	0.760351	0.852865
5	Adam	0.2	0.001	0.744010	0.757408	0.851859
4	Adam	0.2	0.01	0.742358	0.747034	0.850771
4	Adam	0.3	0.001	0.735193	0.758160	0.845904
5	Adam	0.2	0.01	0.735025	0.742998	0.845845
4	Adam	0.3	0.01	0.732183	0.748859	0.843864
5	Adam	0.3	0.01	0.731175	0.734747	0.843175
5	Adam	0.3	0.001	0.726384	0.744395	0.839787
3	Adam	0.3	0.001	0.724440	0.741323	0.838533
3	Adam	0.3	0.01	0.724199	0.729081	0.838402
3	Adam	0.2	0.01	0.722935	0.752568	0.837524
4	Adam	0.5	0.001	0.710587	0.739315	0.829053
4	Adam	0.5	0.01	0.708333	0.732572	0.827399
3	Adam	0.5	0.01	0.703576	0.733852	0.824159
5	Adam	0.5	0.01	0.701702	0.714210	0.822799
5	Adam	0.5	0.001	0.699165	0.701370	0.821093
3	Adam	0.5	0.001	0.694027	0.709536	0.817500
5	Adam	0.2	0.0	0.676153	0.686503	0.804638
3	Adam	0.2	0.0	0.673607	0.673801	0.802864

viser åtte av atten konfigurasjoner for hvert datasett. Fullstendig tabell for de 72 konfigurasjonene er vedlagt i Vedlegg A.

3.8.1 U-net-4

Resultatet av optimaliseringen er vist i tabell 3.5. Resultatet viser at Adam og en dropout på 0.2 gir best resultat i alle tilfeller. Læringsraten varierer mer, hvor 0.01 gir best resultat for datasettene med bebygd, jordbruk og vann, mens datasettet for vann gir best resultat med en læringsrate på 0.001. Ettersom topp tre resultat for vann-datasettet er relativt likt, kan en læringsrate på 0.01 også fungere bra.

3.8.2 U-net-5

Resultatet av optimaliseringen er vist i tabell 3.6. Resultatet viser at Adam gir best resultat i alle tilfeller. En dropout på 0.2 gir best resultat for datasettene bebygd, jordbruk og vann, mens jordbruk får best resultat med 0.3 som dropout. Likevel er det nest beste resultatet relativt likt, hvor en dropout på 0.2 er brukt. En læringsrate på 0.001 gir best resultat for alle datasett unntatt bebygd, hvor 0.01 gir best resultat.

Tabell 3.5: Restultat av optimalisering med U-net-4

Datasett	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
Bebygd	Adam	0.2	0.01	0.870537	0.877914	0.930438
Bebygd	Adam	0.2	0.001	0.860956	0.872806	0.924798
Bebygd	Adam	0.3	0.01	0.856992	0.877173	0.922435
Bebygd	Adam	0.3	0.001	0.854111	0.880114	0.920736
Bebygd	Adam	0.5	0.01	0.852531	0.869290	0.919717
Bebygd	Adam	0.5	0.001	0.850476	0.867477	0.918585
Bebygd	Adam	0.2	0.0	0.838017	0.850852	0.911156
Bebygd	RMSprop	0.3	0.0	0.824725	0.843500	0.902919
Jordbruk	Adam	0.2	0.01	0.827075	0.827268	0.904082
Jordbruk	Adam	0.3	0.001	0.818063	0.816944	0.898560
Jordbruk	Adam	0.2	0.001	0.817455	0.815711	0.898252
Jordbruk	Adam	0.3	0.01	0.803479	0.807692	0.889532
Jordbruk	Adam	0.5	0.001	0.798546	0.804593	0.886440
Jordbruk	Adam	0.5	0.01	0.796497	0.801938	0.885252
Jordbruk	Adam	0.2	0.0	0.777951	0.765828	0.873092
Jordbruk	RMSprop	0.3	0.001	0.758828	0.781886	0.860686
Skog	Adam	0.2	0.001	0.827252	0.843089	0.904744
Skog	Adam	0.2	0.001	0.824831	0.844510	0.903138
Skog	Adam	0.3	0.01	0.820212	0.840929	0.900373
Skog	Adam	0.2	0.01	0.819212	0.837601	0.899711
Skog	Adam	0.3	0.001	0.817581	0.834102	0.898674
Skog	Adam	0.5	0.01	0.808203	0.831959	0.892936
Skog	Adam	0.3	0.0	0.769308	0.797751	0.868349
Skog	Adam	0.2	0.0	0.768483	0.793892	0.867778
Vann	Adam	0.2	0.01	0.973283	0.973137	0.986401
Vann	Adam	0.2	0.001	0.972386	0.972978	0.985933
Vann	Adam	0.3	0.001	0.972034	0.972458	0.985759
Vann	RMSprop	0.2	0.001	0.970080	0.976972	0.984743
Vann	RMSprop	0.2	0.01	0.970032	0.975123	0.984706
Vann	Adam	0.3	0.01	0.969668	0.974294	0.984517
Vann	Adam	0.5	0.001	0.969420	0.962708	0.984386
Vann	Adam	0.5	0.01	0.967107	0.968929	0.983164

Tabell 3.6: Restultat av optimalisering med U-net-5

Datasett	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
Bebygd	Adam	0.2	0.01	0.853792	0.872260	0.920548
Bebygd	Adam	0.3	0.001	0.851179	0.864066	0.918999
Bebygd	Adam	0.5	0.001	0.848461	0.873166	0.917316
Bebygd	Adam	0.2	0.001	0.847708	0.858330	0.916946
Bebygd	Adam	0.5	0.01	0.846658	0.874271	0.916199
Bebygd	Adam	0.3	0.01	0.842556	0.855448	0.913804
Bebygd	RMSprop	0.0	0.0	0.825260	0.842655	0.903387
Bebygd	Adam	0.3	0.0	0.823408	0.847053	0.902167
Jordbruk	Adam	0.2	0.001	0.810793	0.820818	0.894127
Jordbruk	Adam	0.3	0.01	0.809435	0.823213	0.893256
Jordbruk	Adam	0.5	0.01	0.794629	0.766959	0.883988
Jordbruk	Adam	0.3	0.001	0.791823	0.807943	0.882300
Jordbruk	Adam	0.5	0.001	0.787707	0.813947	0.879682
Jordbruk	Adam	0.2	0.01	0.762143	0.783511	0.863126
Jordbruk	Adam	0.2	0.0	0.760137	0.738591	0.861701
Jordbruk	RMSprop	0.2	0.01	0.753678	0.777062	0.857333
Skog	Adam	0.3	0.001	0.815938	0.833045	0.897708
Skog	Adam	0.2	0.001	0.812557	0.828274	0.895628
Skog	Adam	0.5	0.001	0.795401	0.803044	0.884869
Skog	Adam	0.2	0.01	0.795348	0.816792	0.884976
Skog	Adam	0.5	0.01	0.794355	0.817772	0.884271
Skog	Adam	0.3	0.01	0.782995	0.766268	0.877053
Skog	Adam	0.2	0.0	0.776690	0.794967	0.873043
Skog	Adam	0.3	0.0	0.765287	0.795872	0.865819
Vann	Adam	0.2	0.001	0.971393	0.970728	0.985411
Vann	Adam	0.3	0.001	0.971202	0.975145	0.985320
Vann	Adam	0.3	0.01	0.970479	0.968539	0.984947
Vann	Adam	0.2	0.01	0.970404	0.973571	0.984904
Vann	RMSprop	0.3	0.001	0.967864	0.975008	0.983572
Vann	Adam	0.5	0.001	0.966839	0.971481	0.983026
Vann	RMSprop	0.2	0.01	0.966740	0.975599	0.982965
Vann	RMSprop	0.3	0.01	0.965363	0.974240	0.982246

3.9 Trening av nettverket, med de beste konfigurasjonene

Ettersom satellittbildet med fire bånd ga best resultat i alle konfigureringene, er det denne kombinasjonen av bånd i satellittbildet jeg bruker videre. Alle nettverk trener i 40 epoker med en batchstørrelse på 20 bilder. De samme bildene blir brukt til trening-, validering- og testdatasett i alle tilfeller. Testbildene blir ikke vist til modellen under trening.

Optimaliseringen viste at den samme konfigurasjonen ga best resultat for både flerklassedatasettet og de fire binære datasettene, med noen unntak. Derfor brukes de samme konfigurasjonene på både flerklassedatasettet og de binære datasettene.

3.9.1 U-net-4

Nettverket trenes med disse parametrene:

- Optimaliserer: Adam
- Dropout: 0.2
- Minimum læringsrate: 0.01

3.9.2 U-net-5

Nettverket trenes med disse parametrene:

- Optimaliserer: Adam
- Dropout: 0.2
- Minimum læringsrate: 0.001

For flerklassedatasettet brukes Softmax som aktiveringsfunksjon, mens for de binære datasettene brukes Sigmoid. Begge nettverkene kjøres i 40 epoker og modellene lagres som en JSON-fil, mens vektene som en HDF5-fil. Modellene kan dermed brukes ved senere anledninger.

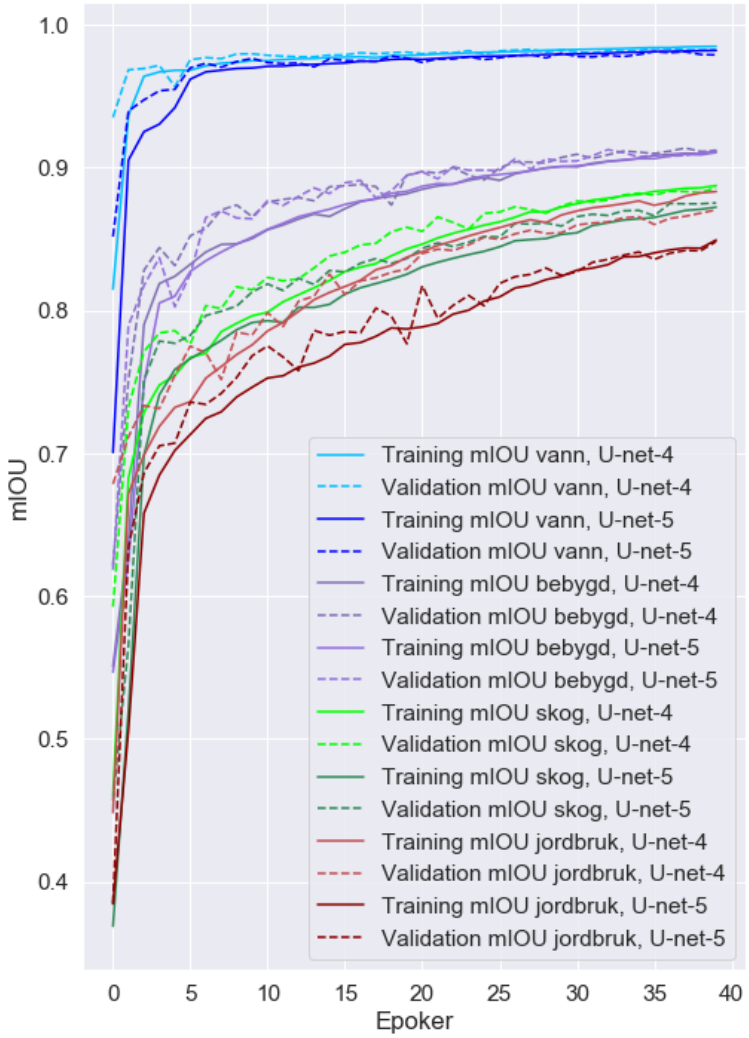
3.9.3 Resultat av treningen, flerklasse datasett

Figur 3.16 viser mIOU for trenings- og valideringsdatasettet med 40 epoker for de to U-net modellene. Figuren viser at U-net-4 gir et litt bedre trenings- og valideringsresultat enn U-net-5, men at de er nokså like.

3.9.4 Resultat av treningen, binære datasett

Figur 3.17 viser mIOU for trenings- og valideringsdatasettet med 40 epoker for de to U-net modellene. Vann-datasettet gir det beste resultatet og er relativt likt for begge modellene. Bebygd gir også relativt likt resultat for begge modellene. For både skog og jordbruk gir U-net-5 et dårligere resultat enn med U-net-4.

Figur 3.16: mIOU for trenings- og valideringsdatasettet med flerklassemodellen for U-net-4 og U-net-5.



Figur 3.17: mIOU for trenings- og valideringsdatasettet med de binær modellene for U-net-4 og U-net-5.

Kapittel 4

Resultater

I dette kapittelet presenteres resultater og prediksjoner gjort med satellittbilder fra testdatasettet med de to U-net-modellene beskrevet i kapittel 3.

For å vurdere evnen til modellene klassifisere satellittbildene i testdatasettet. Dette inneholder 99 bilder og skal klassifiseres med fem modeller på de to nettverkene, U-net-4 og U-net-5. Det er én modell basert på flerklassedatasettet, og fire modeller basert på de binære datasettene som etter klassifisering skal fusjoneres sammen og sammenlignes med modellen basert på flerklassedatasettet.

De fem modellene lastes opp fra JSON-filer og vektene legges til fra H5-filer. For å måle ytelsen til nettverket brukes mIOU og F1-score.

For hver av modellene beregnes forvirringsmatrisen for å se fordelingen av piksler og hva de blir feilklassifisert som. Diagonalen i en forvirringsmatrise forteller om antall riktige klassifiserte for hver klasse.

Prediksjonene blir lagret som georefererte tiff-bilder. Deretter sammenlignes sann label og predikerte labels for begge nettverkene, og vises sammen med satellittbildet det representerer. Eksempelprediksjonene som vises for hver av modellene er basert på de samme tre satellittbilder.

4.1 Flerklassedatasett

Tabell 4.1: Resultatet for U-net-4 og U-net-5, med flerklassedatasettet.

	mIOU	F1
U-net-4	82.31	90.12
U-net-5	81.91	89.89

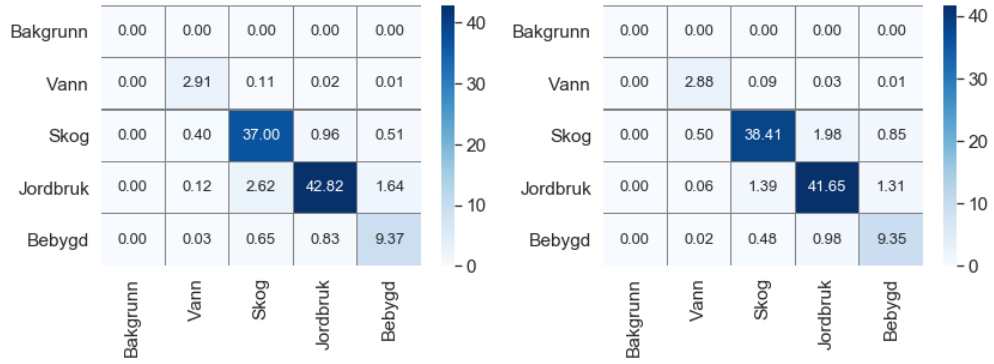
Tabell 4.1 viser at U-net-4 gir et bedre resultat enn U-net-5. Forvirringsmatrisen vist i figur 4.1 viser et mer detaljert resultatet for U-net-4 og U-net-5. Ved å sammenligne figur 4.1 a) og figur 4.1 b) kan man se at U-net-4 klassifiserer bebygd, jordbruk og vann bedre enn U-net-5, mens U-net-5 klassifiserer skog bedre enn U-net-4.

Det er også interessant å se hva pikslene innenfor de fire arealtypene oftest blir feilklassifisert som.

- Bebygd: feilklassifisert som jordbruk.
- Jordbruk: feilklassifisert som bebygd og skog.
- Skog: feilklassifisert som jordbruk.
- Vann: feilklassifisert som skog.

4.1.1 Eksempel på prediksjoner

Figur 4.2 viser tre testbilder og resultatet av klassifiseringen i U-net-4 og U-net-5. Figuren viser at resultatet tilsynelatende er bra, men at begge modellene sliter

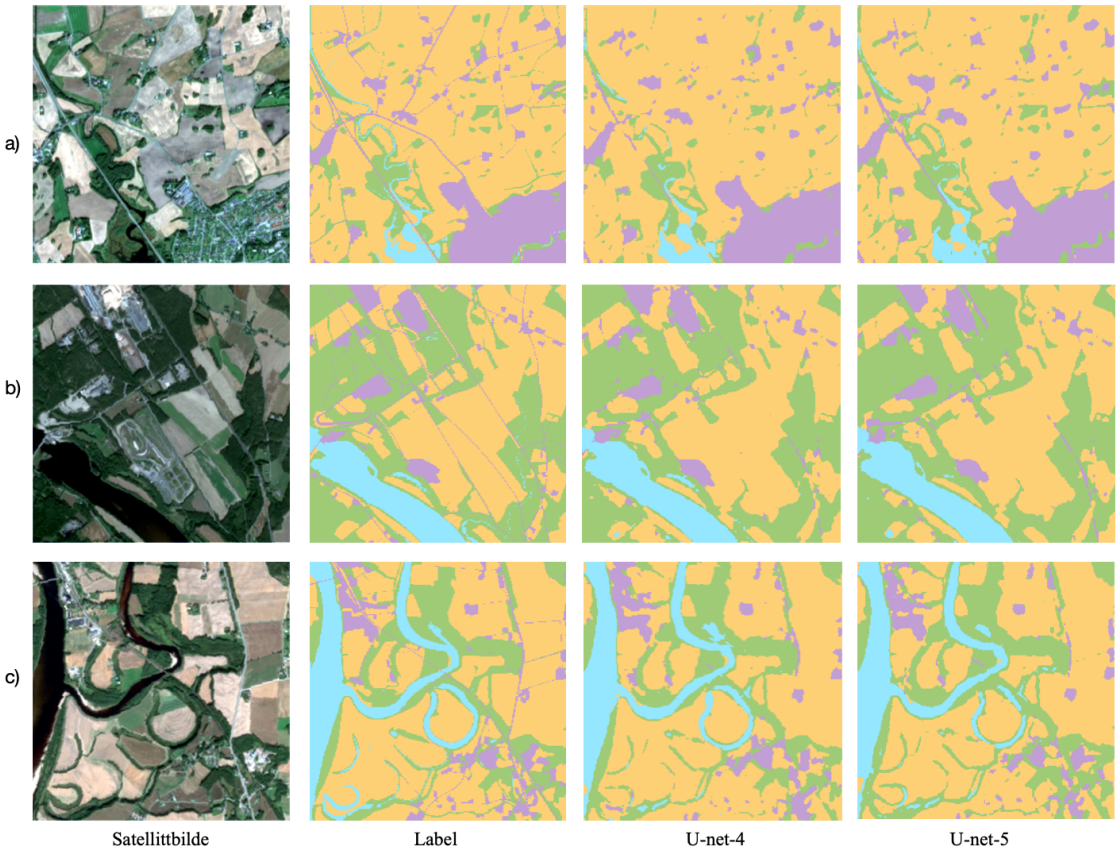


(a) Forvirringsmatrise for U-net-4.

(b) Forvirringsmatrise for U-net-5.

Figur 4.1: Forvirringsmatrise for klassifisering av flerklassedatasettet. Verdier vist i prosentandel.

med å klassifisere bilveger. Smale skogstriper langs vann eller mot jorder er også delvis feilklassifiserte. Nederst i venstre hjørne i prediksjonen i c) kan man se at vannkanalene blir klassifisert som skog. Når man ser på satellittbildet kan det se ut som det faktisk er blitt skog der, og at klassifiseringen derfor er riktig. Dette kan bety at for noen områder er det arealressurskartet som er feil, ikke prediksjonen.



Figur 4.2: Klassifisering basert på flerklassedatasettet. Grønn er skog, blå er vann, lilla er bebygd og gult er skog.

Tabell 4.2: Resultatet for U-net-4 og U-net-5, bebygd.

	mIOU	F1
U-net-4	91.75	95.67
U-net-5	91.98	95.79

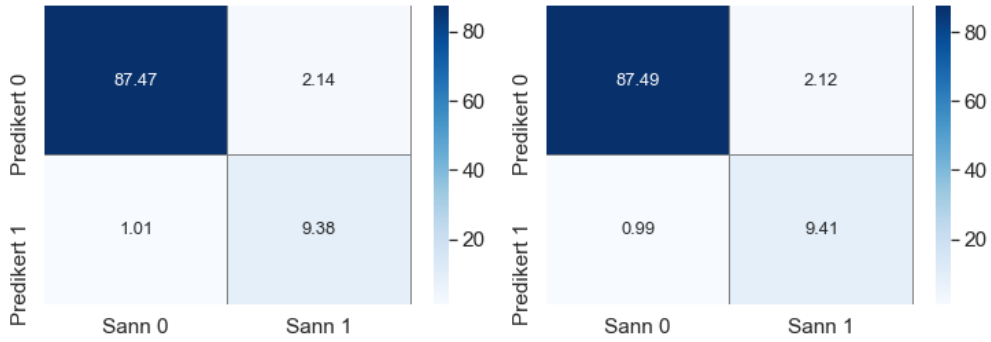
4.2 Binære klasser

For de binære datasettene vurderes modellene opp mot hverandre for hver av arealklassene. De klassifiserte områdene fusjoneres sammen for deretter å sammenlignes med resultatet fra flerklasse-klassifiseringen.

4.2.1 Resultat

Bebygde områder

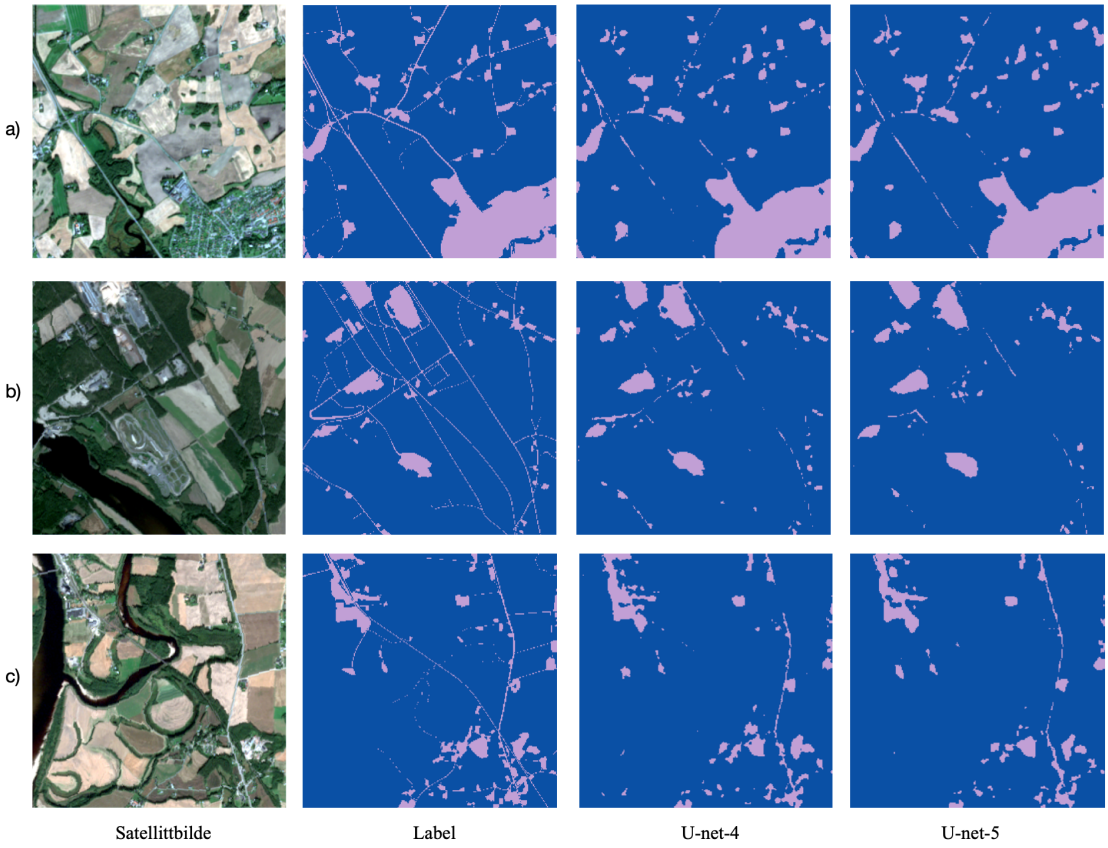
Tabell 4.2 viser at U-net-4 og U-net-5 gir et relativt likt resultat, men at U-net-5 gir et litt bedre resultat. Forvirringsmatrisene vist i figur 4.3 viser at U-net-5 predikerer 0.03% flere piksler som sann bebygd, enn U-net-4.



(a) U-net-4 **(b)** U-net-5

Figur 4.3: Forvirringsmatrise for klassifisering av bebygde områder. Verdier vist i prosentandel.

Figur 4.4 viser eksempler på prediksjoner for begge modellene sammen med sann label. Store arealer klassifiseres i hovedsak riktig, men begge modellene sliter med å klassifisere veger, som tydelig kommer fram i prediksjonen i 4.4 b). Prediksjonene i 4.4 a) og 4.4 c) viser at hovedvegen klassifiseres best i U-net-5, som også totalt sett er den modellen som gir best resultat for bebygde områder. I de tre eksempel-lablene kan man se at veger generelt forsvinner i datasettet. Dette er fordi vegene er smalere enn 10 meter og derfor ikke dekker en hel piksel, som videre bidrar til at veger klassifiseres dårlig i U-net-4 og U-net-5.



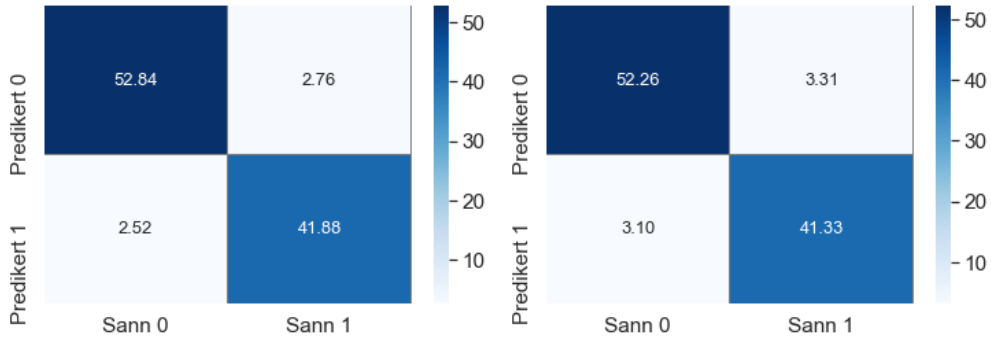
Figur 4.4: Klassifisering basert på binært bebygddatasett. Blått er bakgrunn, lilla er bebygd.

Tabell 4.3: Resultatet for U-net-4 og U-net-5, jordbruk.

	mIOU	F1
U-net-4	88.21	93.62
U-net-5	85.75	92.18

Jordbruk

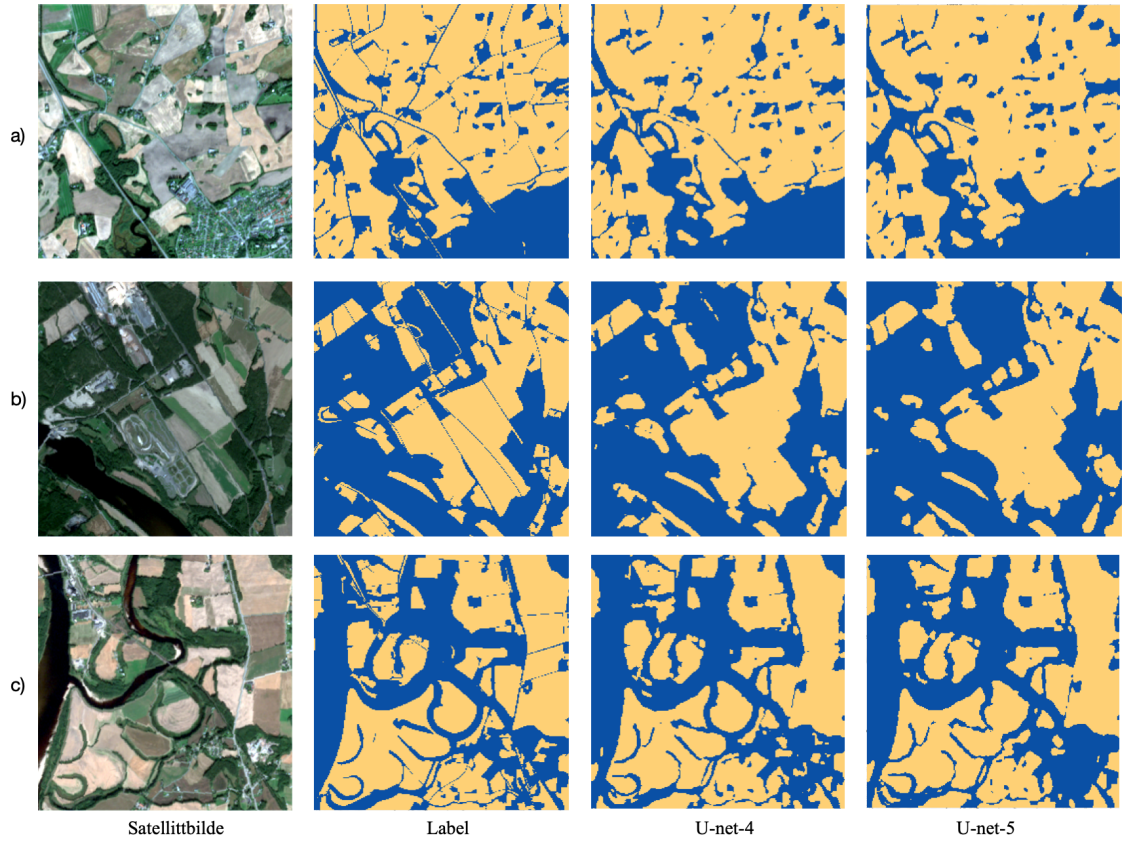
Tabell 4.3 viser at U-net-4 gir det beste resultatet for jordbruk. Forvirringsmatrisen vist i figur 4.5 viser at U-net-4 predikerer 0.55% flere piksler som sann jordbruk, enn U-net-5.



(a) U-net-4 **(b)** U-net-5

Figur 4.5: Forvirringsmatrise for klassifisering av jordbruksareal. Verdier vist i prosentandel.

Figur 4.6 viser eksempler på prediksjoner for begge modellene sammen med sann label. Begge modellene gjør gode prediksjoner, men klassifiserer veger som jordbruk i mange tilfeller som man tydelig kan se i prediksjon 4.6 a), hvor smale streker er klassifisert som jordbruk. Totalt sett inneholder prediksjonene med U-net-4 flere detaljer og gir de beste prediksjonene.



Figur 4.6: Klassifisering basert på binært jordbrukdatasett. Blå er bakgrunn, gult er jordbruk.

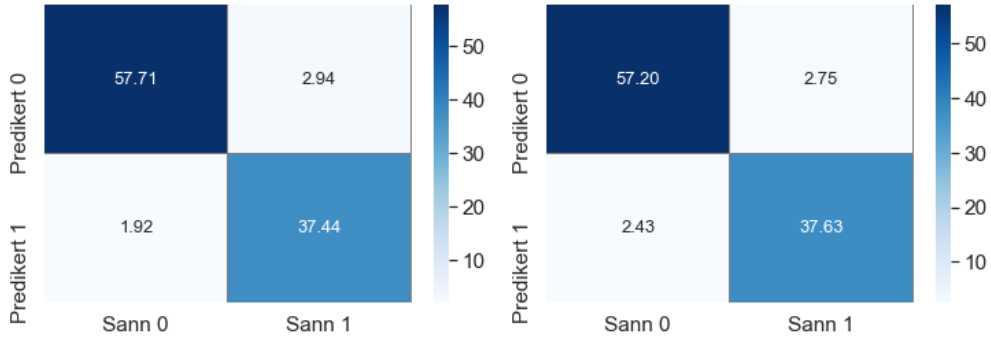
Tabell 4.4: Resultatet for U-net-4 og U-net-5, skog.

	mIOU	F1
U-net-4	88.94	94.06
U-net-5	87.86	93.44

Skog

Tabell 4.4 viser at U-net-4 og U-net-5 gir veldig likt resultat for skog, men U-net-4 gir det beste resultatet. Forvirringsmatrisen vist i figur 4.7 viser at U-net-5 predikerer 0.19% flere piksler som sann skog, enn U-net-4.

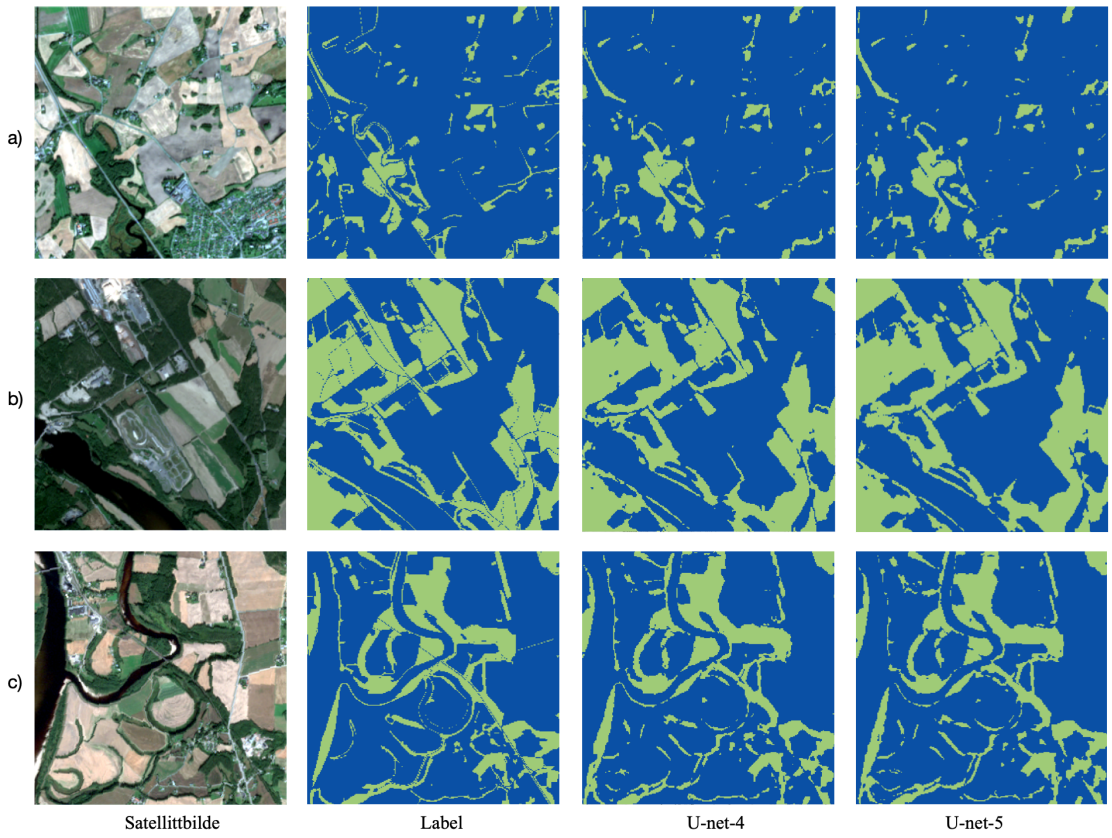
Figur 4.8 viser eksempler på prediksjoner for begge modellene sammen med sann label. Begge modellene klassifiserer skog bra, men sliter med å klassifisere smale skogstriper langs vann eller jorder. Prediksjonen i 4.8 a) viser klassifisering av skog i et område med mye jordbruk, og er et godt eksempel på dette problemet.



(a) U-net-4 **(b)** U-net-5

Figur 4.7: Forvirringsmatrise for klassifisering av skogareal. Verdier vist i prosentandel.

Totalt sett er det vanskelig å skille mellom de to modellene fordi U-net-4 gir det beste resultatet, mens U-net-5 klassifiserer flere skogpiksler riktig. I figur 4.8 klassifiserer U-net-4 flere detaljer enn U-net-5.



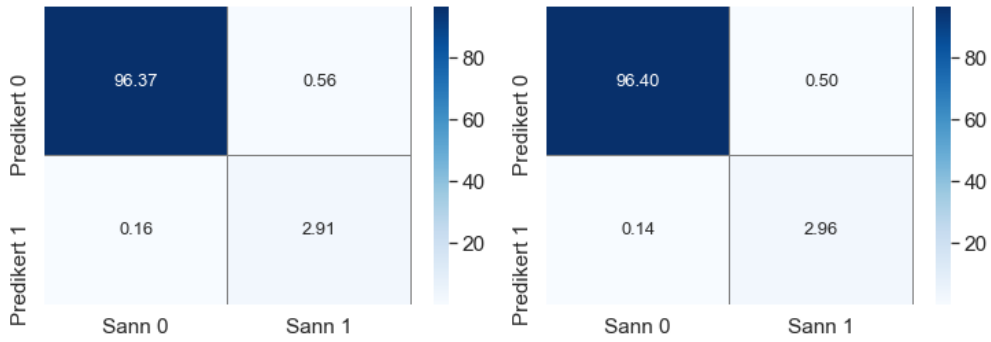
Figur 4.8: Klassifisering basert på binært jordbrukdatasett. Blå er bakgrunn, grønn er skog

Tabell 4.5: Resultatet for U-net-4 og U-net-5, vann.

	mIOU	F1
U-net-4	97.77	98.87
U-net-5	98.17	99.07

Vann

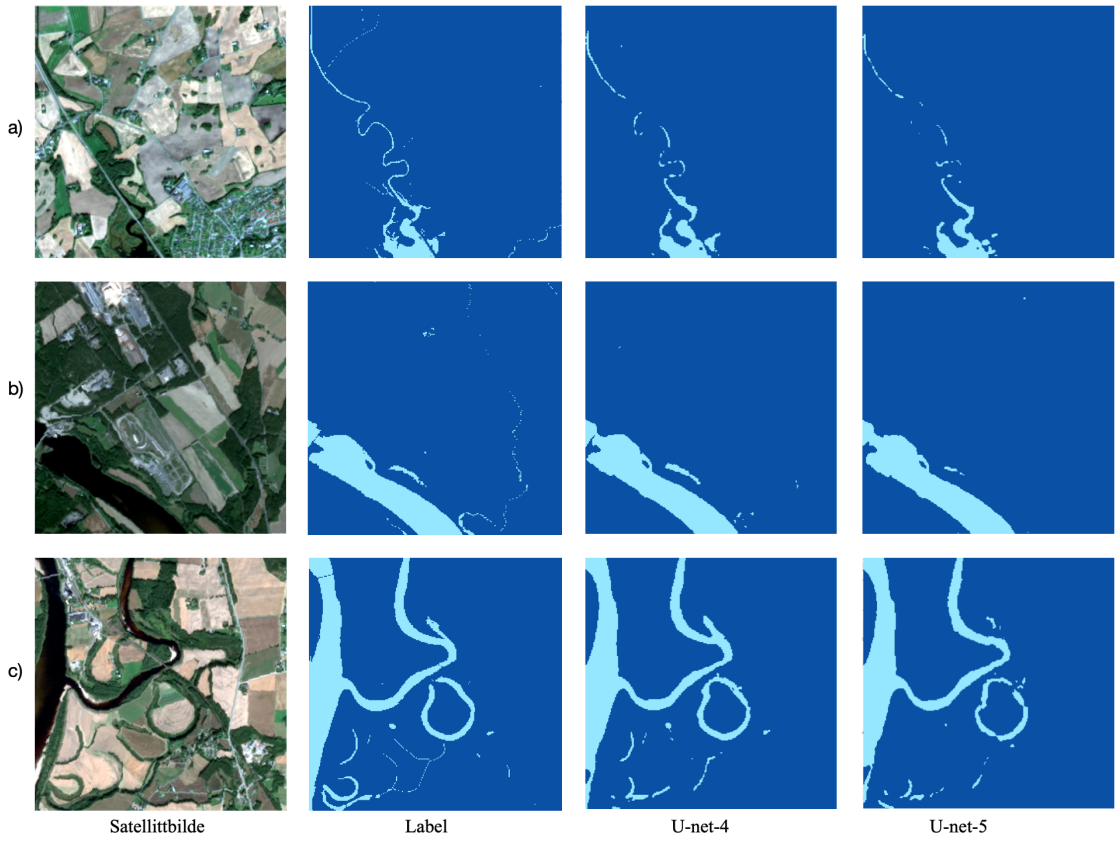
Tabell 4.5 viser at U-net-4 og U-net-5 gir et veldig likt resultat for vann, men at U-net-5 gir et litt bedre resultat. Tatt i betraktning at over 95% av datasettet består av bakgrunn, må man gå mer i dybden for å forstå resultatet.
Forvirringsmatrisen vist i figur 4.9 viser at U-net-5 predikerer 0.05% flere piksler som sann vann, enn U-net-4.



(a) U-net-4 **(b)** U-net-5

Figur 4.9: Forvirringsmatrise for klassifisering av vann. Verdier vist i prosentandel.

Figur 4.10 viser eksempler på prediksjoner for begge modellene sammen med sann label. Modellene sliter med å klassifisere smale striper med vann, som man kan se i 4.10 b). De tre prediksjonenene viser alle at U-net-4 inneholder flere detaljer og klassifiserer vann best.



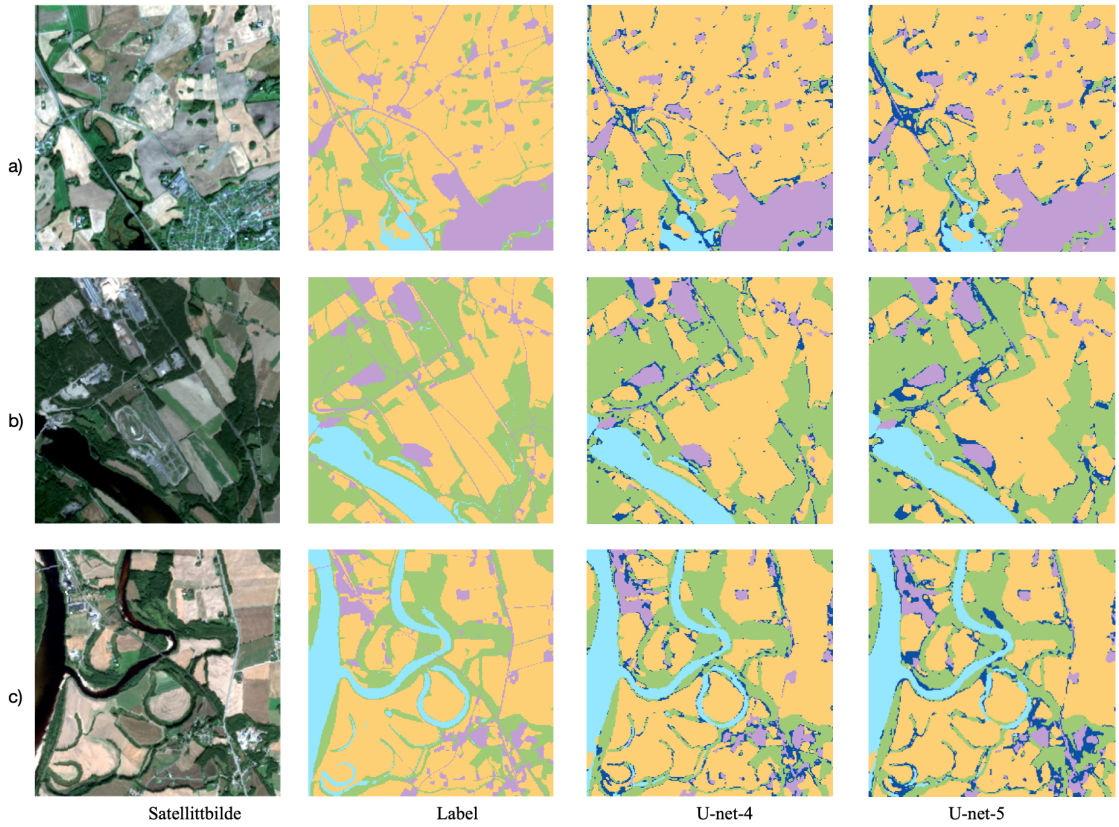
Figur 4.10: Klassifisering basert på binært vanndatasett. Mørk blå er bakgrunn, lys blå er vann.

4.2.2 Fusjon av prediksjonene

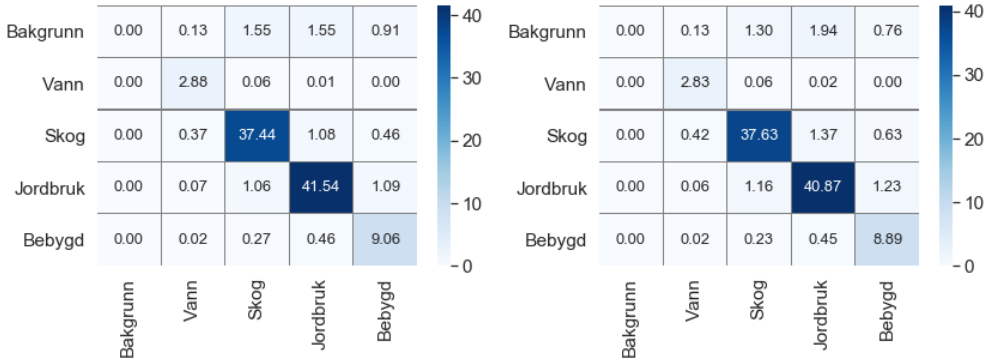
De predikerte pikslene fusjoneres sammen for å lage et datasett med alle arealtypene. Fusjonen baserer seg på pikselverdien og rangeringen arealtypene fikk i tabell 3.2.

Figur 4.11 viser eksempler på prediksjoner. Det kommer tydelig frem at noen områder ikke blir klassifisert som noen av arealtypene i prediksjonen, og derfor blir klassifisert som «Bakgrunn» i datasettet. Dette er noe man ikke får ved flerklasseprediksjon. Ser man bort fra dette, gir de binære klassifiseringene et godt resultat. Basert på prediksjonenene 4.11 a), 4.11 b) og 4.11 c) kan det se ut som U-net-4 klassifiserer mindre som bakgrunn og får med flere detaljer.

Figur 4.12 viser forvirringsmatrise for det fusjonerte binære datasettet. Det viser at U-net-4 klassifiserer flere piksler riktig, som vann, jordbruk og bebygd areal, mens U-net-5 klassifiserer flere piksler riktig som skog.



Figur 4.11: De fire binære datasettene fusjonert sammen. Mørk blå er piksler som er klassifisert som bakgrunn i alle datasettene.



(a) Forvirringsmatrise for U-net-4.

(b) Forvirringsmatrise for U-net-5.

Figur 4.12: Forvirringsmatrise for det fusjonerte binære datasettet. Verdier er vist i prosentandel.

4.3 Sammenligning av klassifisering med flerklassedatasettet og det binære datasettet

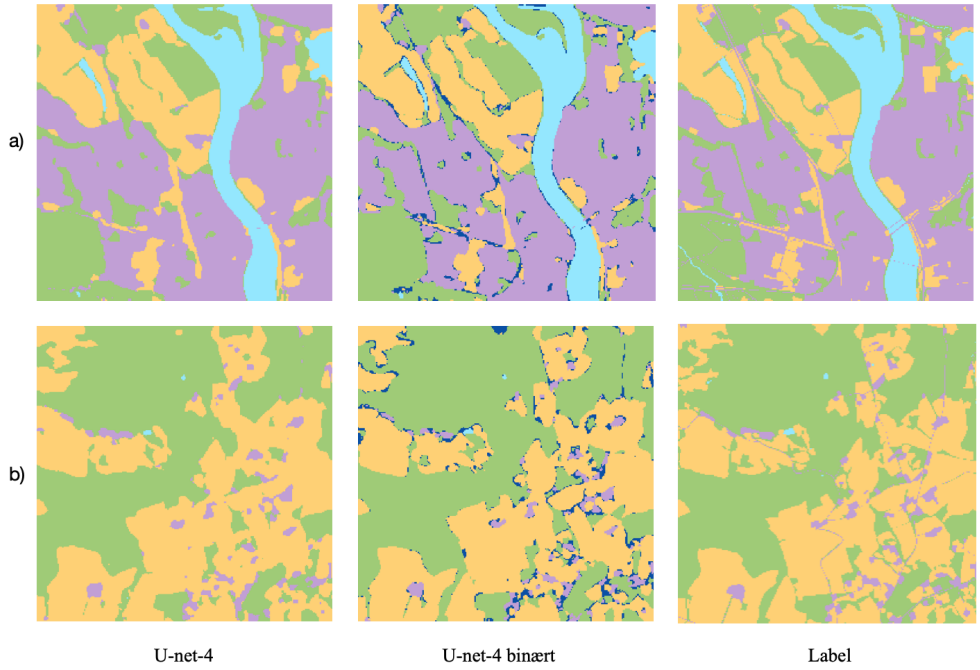
For å sammenligne prediksjonene med flerklassedatasettet og fusjon av det binære datasettet, settes prediksjonene ved siden av hverandre. Figur 4.13 viser satellittbildet som blir klassifisert med U-net-4 i figur 4.14 og U-net-5 i figur 4.14. Begge figurene viser at det fusjonerte binære datasettet fanger opp flere detaljer enn klassifisering med flerklassedatasettet, men mye av detaljene er klassifisert som bakgrunn. Dette gjør også at områder som er vanskelig å klassifisere blir uthevet og på den måten også klassifisert.

Ved å sammenligne forvirringsmatrisene for de to datasettene får man forståelse for hvordan modellene klassifiserer. Figur 4.1 viser forvirringsmatrisen for klassifisering med flerklassedatasettet, mens figur 4.12 viser forvirringsmatrise for det fusjonerte binære datasettet. Det binære datasettet klassifiserer mye som bakgrunn, noe flerklassedatasettet ikke gjør. Bakgrunn eksisterer i utgangspunktet ikke i label-datasettet. Basert på forvirringsmatrisene kan det se ut som at pikslene som blir klassifisert som bakgrunn bidrar til at færre piksler blir feilklassifisert som andre arealtyper.

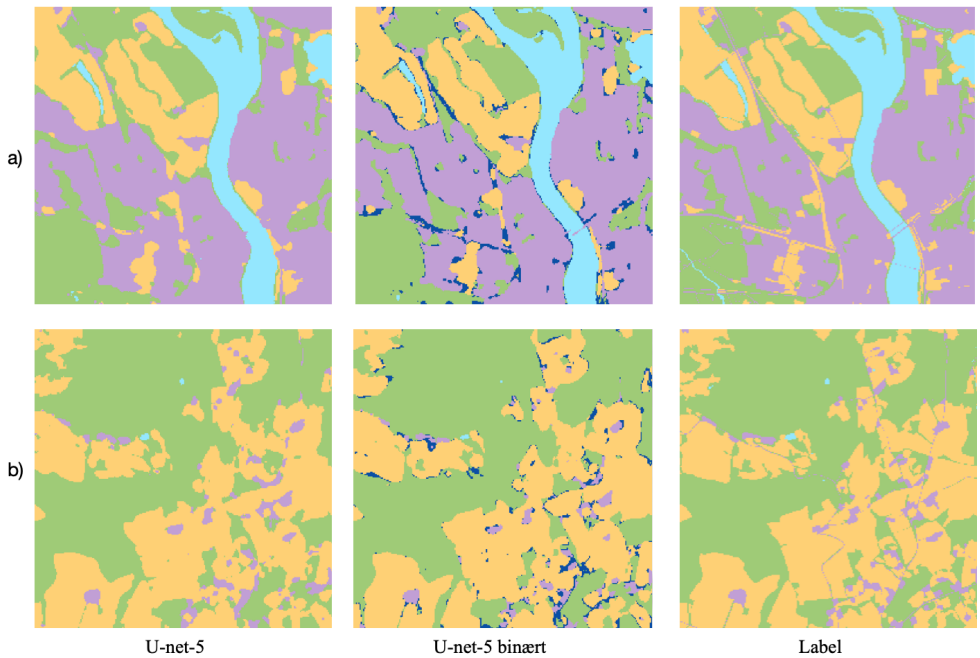
a) b)

Figur 4.13: Satellittbilde brukt til prediksjonene som er vist i figur 4.14 og figur 4.15

Begge U-net modellene, både binært og med flerklassedatasett sliter med å klassifisere veger, skogstriper og små elver.



Figur 4.14: Prediksjon med U-net-4 modellen for flerklassedatasettet og det binære datasettet.



Figur 4.15: Prediksjon med U-net-5 modellen for flerklassedatasettet og det binære datasettet.

Tabell 4.6: mIOU og F1 for U-net-4, ved klassifisering av satellittbilde over Nittedal.

	Flerklasse	Bebygd	Jordbruk	Skog	Vann
mIOU	75.68	88.18	81.96	85.71	97.74
F1	86.02	93.66	90.02	92.28	98.86

Tabell 4.7: mIOU og F1 for U-net-5, ved klassifisering av satellittbilde over Nittedal.

	Flerklasse	Bebygd	Jordbruk	Skog	Vann
mIOU	76.22	88.19	81.31	85.23	97.56
F1	86.38	93.67	89.60	92.00	98.76

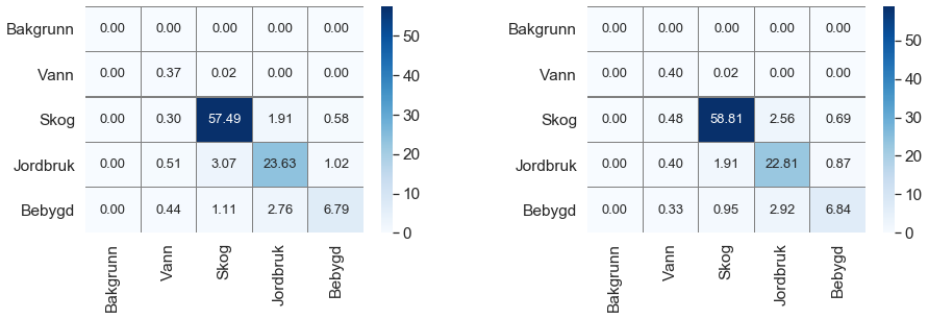
4.4 Produksjon av arealressurskart fra satellittbilde

Ettersom testdatasettet inneholder satellittbilder som er fra samme satellittbilde som trenings- og valideringsdatasettet, er det interessant å se hvordan nettverket presterer på satellittbilder over nye områder. Et satellittbilde over deler av Nittedal er brukt som eksempel her. Satellittbildet blir delt opp i kvadrater på 256*256 piksler før det predikeres med modellene og sammenlignes med fasitdata. Tabell 4.6 og tabell 4.7 viser mIOU og F1-score for nettverkene U-net-4 og U-net-5 ved de ulike modellene. Her ser en at U-net-5 gir det beste resultatet med flerklassedatasettet. Videre kan man se at U-net-4 gjør det bedre en U-net-5 for de tre binære datasettene; jordbruk, skog og vann, mens det binære datasettet, bebygd, har omtrent det samme resultatet for begge.

Figur 4.16 viser forvirringmatrise for prediksjon med flerklassedatasettet for U-net-4 og U-net-5. Her ser en at U-net-5 klassifiserer flere piksler riktig enn U-net-4 for arealklassene vann, skog og bebygd.

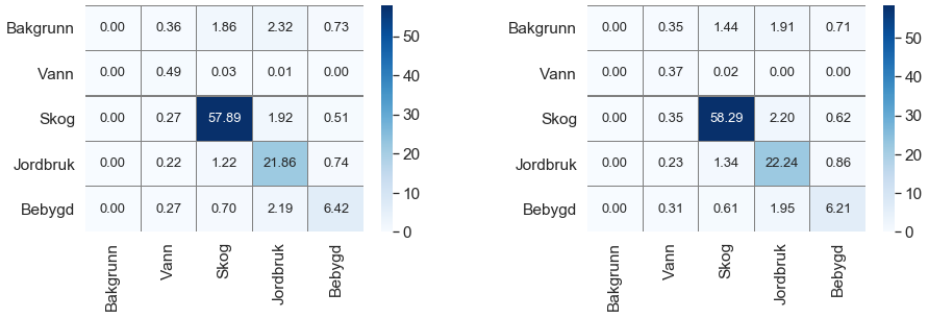
Figur 4.17 viser forvirringsmatrise for det fusjonerte binære datasettet for U-net-4 og U-net-5. Her ser en at U-net-4 predikerer flere piksler riktig for arealklassene vann og bebygd, mens U-net-5 predikerer flere piksler riktig for arealklassene skog og jordbruk. En ser også at mye blir klassifisert som bakgrunn i stedet for en av arealtypene. Generelt er mIOU for U-net-4 og U-net-5 relativt likt for alle klassene, men U-net-4 gjør det totalt sett litt bedre.

De predikerte bildene fusjoneres sammen til den originale størrelsen på satellittbildet. I figur 4.18 vises satellittbildet som er brukt for å lage et predikert arealressurskart sammen med sann label. Figur 4.19 viser de predikerte arealressurskartene ved flerklassemodellen og de fusjonerte binære modellene ved U-net-4 og U-net-5. Her kan en se at elven som renner gjennom Nittedal ikke blir



(a) U-net-4 **(b)** U-net-5

Figur 4.16: Forvirringsmatrise for klassifisering satellittbilde over Nittedal, med flerklassedatasettet.

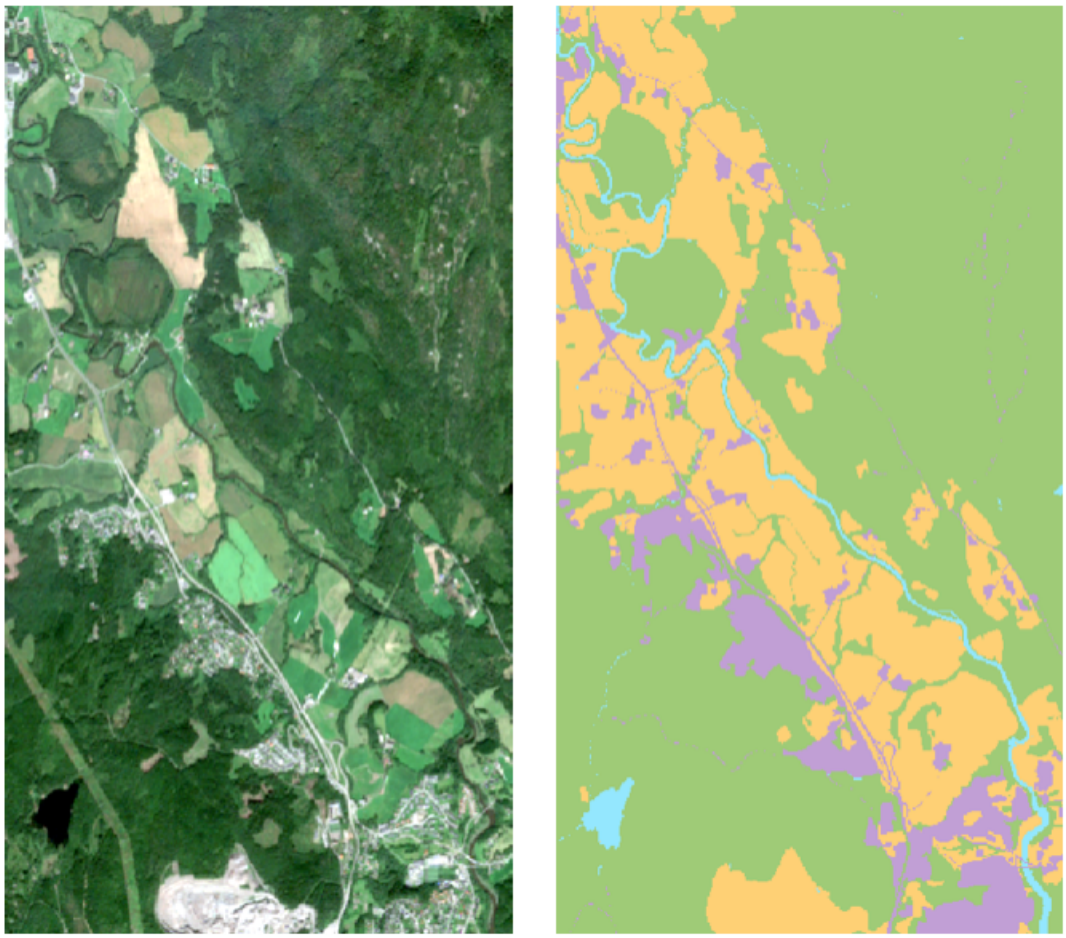


(a) U-net-4 **(b)** U-net-5

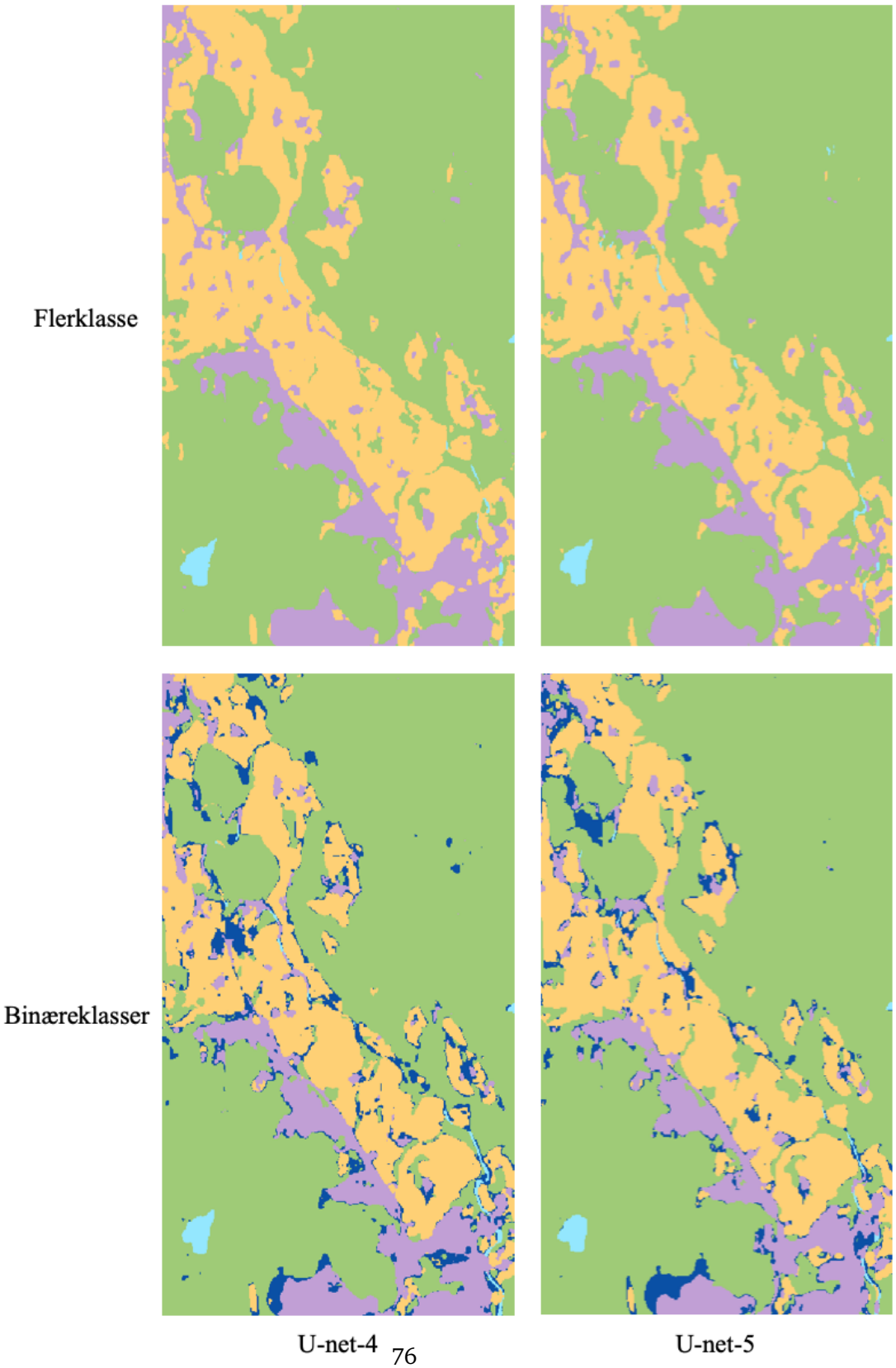
Figur 4.17: Forvirringsmatrise for klassifisering satellittbilde over Nittedal, med fusjonering av binære datasett.

klassifisert i noen av prediksjonene. Prediksjonen med binære klasser og U-net-4 er modellen som klassifiserer elven best.

Når man sammenligner de to prediksjonene gjort med flerklassemodellen vist i figur 4.19 kan en se at U-net-5 gjør en bedre prediksjon enn ved U-net-4 når det sammenlignes med sann label. Når man sammenligner de to prediksjonene gjort med de binære modellene kan en se at U-net-4 gjør en bedre prediksjon enn U-net-5. Dette er en bedre klassifisering i tillegg til at færre områder er klassifisert som bakgrunn.



Figur 4.18: Området som skal bli klassifisert og sann label.



Figur 4.19: Predikerte områder med flerklassedatasett og fusjonert binære datasett for U-net-4 og U-net-5.

Kapittel 5

Diskusjon

I dette kapittelet diskuteres foreslått datasett, framgangsmåte og resultat fra kapittel 4.

I denne studien er det vist at nettverkene U-net-4 og U-net-5 kan brukes til å predikere arealressurskart til en ganske nøyaktig grad. Spørsmålet er videre hvordan bruksområdet på det predikerte arealressurskartet er sammenlignet med det originale arealressurskartet. Ettersom satellittbildene har en pikseloppløsning på 10 meter kan ikke det predikerte arealressurskartet få en høyere oppløsning enn dette. Men jeg mener at det predikerte arealressurskartet kan være til stor nytte for å finne områder i arealressurskart som har behov for oppdatering. Det kan også være et nyttig verktøy for oppgaver som ikke krever høy nøyaktighet. I tillegg kan det predikerte datasettet være til nytte for noen områder mer enn andre - i eksempelvis skog og jordbruk er det høyere nøyaktighet enn for bebygde områder og vann.

5.1 Datasettet

Datasettet som brukes for å trene nettverkene består av to satellittbilder fra østlandet. Dette gjør at nettverkene ikke får variert data i form av ulike satellittbilder eller variasjon i arealtypene. Mangelen på variasjon kan føre til at modellene ikke klassifiserer områder med vegetasjon og landskap andre steder enn østlandet, like godt som resultatet i denne studien viser. Satellittbildene er lastet ned som L1C-bilder som gir refleksjon over atmosfæren. I utgangspunktet hadde det vært å foretrekke å bruke L2A-bilder som gir refleksjon under atmosfæren for å få klarere bilder uten dis. Denne korreksjonen er forøvrig kun gjort på bilder som er tatt etter 28.03.2017. For denne studien har det derimot vært viktigere å prioritere satellittbilder som er tatt nært i tid oppdateringsdatoen i arealressurskartet, heller enn å ta i bruk bilder uten dis. Denne korreksjonen er det mulig å gjøre selv, men for å unngå behandling av satellittbildene er dette ikke gjort. Tanken med å ikke behandle satellittbildene er at dataene skal være lite komplekse og etterprøvbare. Dessuten kan databehandling være tidkrevende og gi variert utseende fra satellittbilde til satellittbilde.

Norge består av mye jordbruks- og skogområder, noe som gjenspeiles i sammensetningen av arealtyper i det foreslåtte datasettet. Her kunne det vært aktuelt å samle inn mer data fra tettbygde strøk og områder med vann og elver, for å få et mer sammensatt datasett.

I denne studien er arealressurskart i målestokk 1:5 000 brukt. Dette kartet inneholder tolv arealtyper, men for denne studien er det valgt å bruke et datasett der de tolv arealtypene er slått sammen til fire arealtyper. Satellittbildene inneholder mye god informasjon og nettverkene kan nok klare å klassifisere flere arealtyper enn fire. Det kunne ha vært interessant å se hvordan modellenes ytelse er med alle tolv arealtyper. Det er mulig at modellene kunne gitt bedre resultateter med bruk av de originale klassene uten sammenslåingen gjort i denne studien. Det kan også vurderes om arealtypene er riktig slått sammen eller om

noen skulle vært fjernet fra datasettet og blitt brukt som bakgrunn. En feil som er gjort er et arealtypen åpen fastmark er kategorisert som arealtypen jordbruk. Åpen fastmark er områder med lav vegetasjon, fjell og viddeområder og passer derfor i utgangspunktet ikke inn i noen av de fire arealtypene i studien. Likevel har jeg feiltolket arealtypen og klassifisert den som jordbruk. Dette kan ha hatt en negativ påvirkning på resultatet. Et eksempel på dette kan en se nederst i figur 4.18 der et område skal være klassifisert som arealtypen jordbruk, men er predikert som bebygd-område i alle modellene. Ved nærmere undersøkelse av området, ser en at dette er et masseuttak og har arealtypen åpen fastmark. I forhold til arealressurskartet er dette en feilklassifisering, men utseendemessig kan en forstå hvorfor den ble predikert som bebygd-område. Derfor burde åpen fastmark være en selvstendig klasse i modellen, eller vært utelatt fra datasettet.

Ved sammenslåing av datasettet fikk arealtypene nye verdier som også fungerte som en rangering av prioritering, når de binære datasettene ble fusjonert etter klassifisering. Resultatet baserer seg derfor på valg av rangering ettersom en annen rangering ville gitt andre resultater enn vist i prediksjonen. For eksempel er jordbruk rangert høyere enn skog, og derfor vil en piksel klassifisert som både jordbruk og skog med de binære datasettene, bli klassifisert som jordbruk ved fusjonering. Dette kan føre til feilklassifiseringer hvis pikselen egentlig tilhørte arealtypen skog. Ved flerklasse-klassifisering vil pikselen få arealtype basert på hvilken arealklasse som scorer høyest. Ettersom arealtypene klassifiseres hver for seg i de binære datasettene kan man ikke sammenligne output prosentverdiene, noe som kunne gitt et mer realistisk resultat.

Satellittbildene som er brukt i studien er tatt omtrent samtidig som arealressurskartet sist ble oppdatert for et bestemt område. Dette er for å unngå at endringer i arealtypene skal ha skjedd i tiden fra arealressurskartet sist ble oppdatert til satellittbildet ble tatt. Som vist i kapittel 3, inneholder arealressurskartet feilklassifiseringer som fører til at fasitgrunnlaget som brukes til trening av modellen inneholder feil. Dette er en feilkilde som er vanskelig å gjøre noe med, ettersom man antar at arealressurskartet er fullstendig. Med dette menes at det er tatt utgangspunkt i at arealressurskart er fullstendig selv om det finnes feil i dette. En annen feilkilde oppstår når det nøyaktige vektordatasettet av arealressurskart gjøres om til rasterdata med 10 meters pikseloppløsning. Informasjon går da tapt. Dette kan man tydelig se ved smale områder som veger, skogstriper, bekker og smale områder som grenser til andre arealtyper, hvor stripen blir en ufullstendig linje med pikselhull. Men, ved å bruke arealressurskart i målestokk 1:5000 i stedet for i målestokk 1:50 000, får en med mer informasjon selv om noe går tapt.

En viktig faktor for omfanget av datamengde som kan brukes i studien, er maskinvare. Ettersom datakraft i PCen som brukes er den avgrensende faktoren for hvor stort datasettet kan være, ville jeg med mer datakraft kunne tatt med flere

deler av Norge. Det ville gitt et datasett med mer variasjon og større bruksområde. NMBU har tilgjengelige GPUer fra våren 2020, men det ble for sent for meg å ta den i bruk.

I studien ble tre ulike båndkombinasjoner anvendt og vurdert. Satellittbildet med fire bånd ga best resultat og ble derfor brukt videre i studien, men satellittbildet med tre og fem bånd ga også gode resultater. Ved å bruke satellittbilder med fire bånd kan alle båndene lastes ned direkte fra Landviewer, noe som gjør det lettere å ta i bruk modellene. For å lage det femte båndet måtte Arcgis brukes, noe som kan være en ulempe for andre som vil ta i bruk modellene.

5.2 Trening av nettverkene

I denne studien er det kun tatt i bruk to nettverk av U-net, men det kan vært interessant å se på ytelsen ved bruk av andre nevrale nettverk. Flere nettverk ble vurdert, men U-net ble valgt på grunn av evnen til rask og presis segmentering av bilder. Det ble også vurdert å ta i bruk ResNet med ulike lengder, men på grunn av begrensing i maskinvare ble det vurdert at U-net var det beste valget. Ved optimaliseringen kan det også utforskes andre endringer i nettverket eller legges til flere parametere for å se om dette påvirker ytelsen til modellene .

5.3 Resultat fra prediksjonen og predikert arealressurskart.

For flerklassemodellen viser figur 4.1 at U-net-4 gir et litt bedre resultat enn U-net-5. Ved klassifisering med de binære datasettene og fusjonering av de predikerte testdatasettene, viser figur 4.12 at U-net-4 også her gir ett litt bedre resultat enn U-net-5. Når en sammenligner disse resultatene kan en se at U-net-4 med klassifisering av flerklassedatasettet totalt sett gir det beste resultatet. U-net-4 er derfor i denne studien det beste nettverket å bruke.

Det predikerte arealressurskartet over Nittedal er vist i figur 4.19. Figuren viser at arealressurskart kan predikeres fra modellen. Resultatet for prediksjonen er noe dårligere enn resultatet fra testdatasettet. Dette kan tyde på at modellene ikke har fått nok variert data. Dette kan være som følge av at treningsdatasettet som er brukt til å trene nettverkene, bare inneholder to satellittbilder. Ettersom testbildene er fra samme satellittbilder som treningsdatasettet, vil de ha likere verdier enn om testdatasettet var fra et nytt satellittbilde. Ulike solforhold og tidspunkt for når satellittbildet er tatt på året vil blant annet påvirke hvordan satellittbildet ser ut.

Prediksjonen vist i figur 4.19 viser at U-net-4 basert på de binære modellene gir den beste prediksjonen. Prediksjonen er mer detaljert enn for prediksjonen ved flerklassemodellen, men den inneholder også mye piksler klassifisert som bakgrunn.

U-net-4 gir det beste resultatet både for testdatasettet og ved prediksjon av satellittbilde fra Nittedal. Det er interessant å se at testdatasettet med flerklassemodellen ga det beste resultatet ved trening og test. Men, ved klassifisering av et nytt satellittbilde ga fusjonering av de binære modellene det beste resultatet. I utgangspunktet har de fire modellene; flerklassemodellen, og de fusjonert binære modellene, begge med U-net-4 og U-net-5, vist seg å gi gode resultater ved trening og test. Men, ved prediksjon av satellittbilde fra Nittedal viste det seg at de fusjonert binære modeller med U-net-4 ga det beste resultatet. Dette illustrerer viktigheten av å prøve ulike modeller før en velger endelig modell.

5.4 Forslag til videre arbeid

I denne studien er det bare klassifisert fire av totalt tolv arealklasser fra arealressurskartet. Det kan være interessant å undersøke hvordan nettverket fungerer ved bruk av alle tolv arealklassene, ettersom det gir et bedre arealressurskart hvis det inneholder alle klassene.

En annen interessant studie er å se hvordan resultatene endrer seg ved tilgang på mer data. Et større datasett med satellittbilder fra alle Norges byer ville bidratt til et mer variert datasett. Det er interessant å se om flere bilder styrker nettverkets klassifiseringer.

Videre vil det og være interessant å se på flere nettverk eller endre parametere i optimaliseringen.

Kapittel 6

Konklusjon

I dette kapitlet sammenstilles resultatet med oppgavens problemstillinger for å konkludere forskningsspørsmålet.

Det ble innledningsvis i oppgaven presentert to forskningsspørsmål:

- Hvor nøyaktig kan satellittbilder klassifiseres etter arealtyper ved hjelp av konvolusjonelle nevrale nettverk.
- Hva kan det predikerte arealressurskartet brukes til.

Resultatene viser at satellittbilder kan klassifiseres ved bruk av konvolusjonelle nevrale nettverk, hvor U-net med en dybde på fire blokker ga det beste resultatet i denne studien. Resultatet er basert på fasitdata med 10 meters pikseloppløsning som innebærer at på tross av gode resultater, kan ikke nøyaktigheten sammenlignes fullt med det originale arealressurskartet. I utgangspunktet er arealressurskartet svært nøyaktig, mens det predikerte arealressurskartet i denne studien er langt mer utydelig. Dette fører til at bruksområdet ikke blir det samme som ved det originale arealressurskartet.

Et bruksområde for det predikerte arealressurskartet er å bruke det for å kartlegge områder i AR5 med oppdateringsbehov. Ved å sammenligne pikselverdier fra det predikerte arealressurskartet med det originale arealressurskartet, kan en få ut et nytt kart som inneholder ulikheter i de to arealressurskartene. Dette kan for eksempel brukes til å effektivt lokaliserer områder som har behov for oppdatering i det originale arealressurskartet hvis en ser at prediksjonen ikke stemmer overens med fasit. Prediksjonen har en kvalitet som kan bidra i det manuelle arbeidet med arealressurskart.

Referanser

[1] A. P. Ahlstrøm, K. Bjørklo og K. Fadnes, «AR5 Klssifikasjonssystem», *NIBIO Bok*, årg. 5, 2019.

[2] *Arealstatistikk*. URL: https://www.ssb.no/natur-og-miljo/statistikker/arealstat (Lest: 04.05.2020).

[3] H. A. Arief, G.-H. Strand, H. Tveite og U. G. Indahl, «Land cover segmentation of airborne LiDAR data using stochastic atrous network», *Remote Sensing*, årg. 10, nr. 6, s. 973, 2018.

[4] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[5] F. Chollet, *Deep learning with Python*. Manning Publications Co., 2018.

[6] Ø. B. Dick, *Vegetasjonsindeks*, feb. 2009. URL: https://snl.no/vegetasjonsindeks (Lest: 26.04.2020).

[7] Ø. B. Dick, «Spektralkurver», feb. 2009. URL: https://snl.no/spektralkurver (Lest: 24.04.2020).

[8] J. Duchi, E. Hazan og Y. Singer, «Adaptive subgradient methods for online learning and stochastic optimization», *Journal of machine learning research*, årg. 12, nr. Jul, s. 2121–2159, 2011.

[9] H. Dvergsdal, *Nevralt nettverk*, nov. 2019. URL: https://snl.no/nevralt_nettverk (Lest: 07.02.2020).

[10] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn og A. Zisserman, «The Pascal Visual Object Classes Challenge: A Retrospective», *International Journal of Computer Vision*, årg. 111, nr. 1, s. 98–136, jan. 2015.

[11] *Export training data for deep learning*. URL: https://pro.arcgis.com/en/pro-app/tool-reference/image-analyst/export-training-data-for-deep-learning.htm (Lest: 26.04.2020).

[12] *FKB*, mar. 2019. URL: https://snl.no/FKB (Lest: 06.03.2020).

[13] M. Gašparović og T. Jogun, «The effect of fusing Sentinel-2 bands on land-cover classification», *International journal of remote sensing*, årg. 39, nr. 3, s. 822–841, 2018.

[14] *Geomatikk*, 2018. URL: https://snl.no/geomatikk (Lest: 25.05.2020).

[15] *Geospatial Data Science [White paper]*, Open Geospatial Consortium, 2020. (Lest: 23.05.2020).

[16] I. Goodfellow, Y. Bengio og A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[17] *Gradient Descent and Stochastic Gradient Descent Algorithms for Neural Networks*. URL: https://medium.com/konvergen/gradient-descent-and-stochastic-gradient-descent-algorithms-for-neural-networks-e817f3c411ef (Lest: 23.05.2020).

[18] K. He, X. Zhang, S. Ren og J. Sun, «Deep residual learning for image recognition», i *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, s. 770–778.

[19] Y. Heryadi og E. Miranda, «Land Cover Classification Based on Sentinel-2 Satellite Imagery Using Convolutional Neural Network Model: A Case Study in Semarang Area, Indonesia», i *Asian Conference on Intelligent Information and Database Systems*, Springer, 2019, s. 191–206.

[20] V. Iglovikov, S. Mushinskiy og V. Osin, «Satellite imagery feature detection using deep convolutional neural network: A kaggle competition», *arXiv preprint arXiv:1706.06169*, 2017.

[21] Kartverket, «SOSI Del 3 Produktspesifikasjon for Felles KartdataBase (FKB)», nr. 4.6, 2020.

[22] *Kilden, NIBIO*. URL: kilden.nibio.no (Lest: 21.04.2020).

[23] D. P. Kingma og J. Ba, «Adam: A method for stochastic optimization», *arXiv preprint arXiv:1412.6980*, 2014.

[24] A. Krizhevsky, I. Sutskever og G. E. Hinton, «Imagenet classification with deep convolutional neural networks», i *Advances in neural information processing systems*, 2012, s. 1097–1105.

[25] A. Krizhevsky, I. Sutskever og G. E. Hinton, «Imagenet classification with deep convolutional neural networks», 2012, s. 1097–1105.

[26] *Landviewer, EOS*. URL: https://eos.com/lv/ (Lest: 26.04.2020).

[27] Li Fei-Fei, R. Fergus og P. Perona, «Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories», i *2004 Conference on Computer Vision and Pattern Recognition Workshop*, 2004, s. 178–178.

[28] J. Long, E. Shelhamer og T. Darrell, «Fully convolutional networks for semantic segmentation», i *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, s. 3431–3440.

[29] R. S. Mällberg og V. E. S. Rolfsen, «Map Creation from Semantic Segmentation of Aerial Images Using Deep Convolutional Neural Networks», masteroppg., Norwegian University of Science og Technology, 2018.

[30] *NDVI colorized function*. URL: https://pro.arcgis.com/en/pro-app/help/data/imagery/ndvi-colorized-function.htm (Lest: 26.04.2020).

[31] M. Papadomanolaki, M. Vakalopoulou, S. Zagoruyko og K. Karantzalos, «Benchmarking deep learning frameworks for the classification of very high resolution satellite multispectral data.», *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, årg. 3, nr. 7, 2016.

[32] D. M. Powers, «Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation», 2011.

[33] S. Raschka og V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Birmingham: Packt Publishing Ltd, 2017, bd. 2.

[34] *Reduse Learningrate on Plateau*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau (Lest: 02.05.2020).

[35] *RMSprop*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop (Lest: 18.05.2020).

[36] O. Ronneberger, P. Fischer og T. Brox, «U-Net: Convolutional Networks for Biomedical Image Segmentation», *CoRR*, årg. abs/1505.04597, 2015. URL: http://arxiv.org/abs/1505.04597.

[37] S. Ruder, «An overview of gradient descent optimization algorithms», *arXiv preprint arXiv:1609.04747*, 2016.

[38] *Semantic Segmentation, Popular Architectures*. URL: https://towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0 (Lest: 07.05.2020).

[39] *Sentinel missions*. URL: https://sentinel.esa.int/web/sentinel/missions (Lest: 25.05.2020).

[40] *Sentinel-2*. URL: https://gdal.org/drivers/raster/sentinel2.html (Lest: 18.05.2020).

[41] *Sentinel-2 overview*. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2/overview (Lest: 20.04.2020).

[42] *Sentinel-2 products*. URL: https://medium.com/sentinel-hub/sentinel-2-l2a-products-available-on-sentinel-hub-beab58903285 (Lest: 20.04.2020).

[43] K. Simonyan og A. Zisserman, «Very deep convolutional networks for large-scale image recognition», *arXiv preprint arXiv:1409.1556*, 2014.

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke og A. Rabinovich, «Going deeper with convolutions», i *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, s. 1–9.

[45] A. Tidemann, *Dyp læring*, 20. februar 2018. URL: https://snl.no/dyp_l%5C%C3%5C%A6ring (Lest: 10.02.2020).

[46] *Types of Convolution Kernels Simplified*. URL: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37 (Lest: 23.04.2020).

[47] C. Yang, F. Rottensteiner og C. Heipke, «Classification of land cover and land use based on convolutional neural networks», *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 4 (2018), Nr. 3*, årg. 4, nr. 3, s. 251–258, 2018.

[48] M. D. Zeiler, «Adadelta: an adaptive learning rate method», *arXiv preprint arXiv:1212.5701*, 2012.

[49] A. Zhang, Z. C. Lipton, M. Li og A. J. Smola, *Dive into Deep Learning*. 2020, https://d2l.ai.

Vedlegg A

Resultat

A.1 Flerklasseklassifisering

Tabell A.1: U-net-4, flerklassedatasett

Antall bånd	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
5	Adam	0.2	0.0	0.6782561	0.697937	0.806170
5	Adam	0.2	0.01	0.7519615	0.759732	0.857130
5	Adam	0.2	0.001	0.74648577	0.765077	0.853524
5	Adam	0.3	0.0	0.66386884	0.664749	0.795604
5	Adam	0.3	0.01	0.7204643	0.734210	0.835833
5	Adam	0.3	0.001	0.7314224	0.731307	0.843492
5	Adam	0.5	0.0	0.6581733	0.650903	0.791571
5	Adam	0.5	0.01	0.70894605	0.723420	0.827884
5	Adam	0.5	0.001	0.7057137	0.723261	0.825682
5	RMSprop	0.2	0.0	0.6514303	0.668647	0.786581
5	RMSprop	0.2	0.01	0.6541811	0.687067	0.787747
5	RMSprop	0.2	0.001	0.6370079	0.695305	0.774769
5	RMSprop	0.3	0.0	0.6491444	0.661581	0.784802
5	RMSprop	0.3	0.01	0.66642517	0.701882	0.797060
5	RMSprop	0.3	0.001	0.6558541	0.691901	0.789322
5	RMSprop	0.5	0.0	0.64085793	0.658296	0.778578
5	RMSprop	0.5	0.01	0.6529753	0.692799	0.787036
5	RMSprop	0.5	0.001	0.6553343	0.698274	0.788731
4	Adam	0.2	0.0	0.65735847	0.667819	0.790858
4	Adam	0.2	0.01	**0.759446**	0.766429	0.862121
4	Adam	0.2	0.001	0.739733	0.752157	0.849045
4	Adam	0.3	0.0	0.66247416	0.676256	0.794773
4	Adam	0.3	0.01	0.74638224	0.765088	0.853368
4	Adam	0.3	0.001	0.7270427	0.738456	0.840254

4	Adam	0.5	0.0	0.6383056	0.686600	0.776705
4	Adam	0.5	0.01	0.7201837	0.750911	0.835676
4	Adam	0.5	0.001	0.7079117	0.717258	0.827313
4	RMSprop	0.2	0.0	0.6646781	0.672719	0.796250
4	RMSprop	0.2	0.001	0.64809257	0.712022	0.783622
4	RMSprop	0.2	0.01	0.6736226	0.706290	0.802213
4	RMSprop	0.3	0.0	0.6617191	0.679021	0.793985
4	RMSprop	0.3	0.01	0.6689356	0.711854	0.798942
4	RMSprop	0.3	0.001	0.6478182	0.695141	0.783379
4	RMSprop	0.5	0.0	0.6529591	0.667968	0.787632
4	RMSprop	0.5	0.01	0.65834457	0.690149	0.791327
4	RMSprop	0.5	0.001	0.64493215	0.694772	0.781033
3	Adam	0.2	0.0	0.6435434	0.675691	0.780903
3	Adam	0.2	0.01	0.729402	0.750230	0.841980
3	Adam	0.2	0.001	0.7445704	0.762210	0.852343
3	Adam	0.3	0.0	0.6162724	0.635810	0.760152
3	Adam	0.3	0.01	0.715132	0.738276	0.832189
3	Adam	0.3	0.001	0.7317594	0.753832	0.843629
3	Adam	0.5	0.0	0.6249748	0.617872	0.766474
3	Adam	0.5	0.01	0.7156036	0.684868	0.832466
3	Adam	0.5	0.001	0.70141125	0.739316	0.822743
3	RMSprop	0.2	0.0	0.6300682	0.638058	0.770652
3	RMSprop	0.2	0.01	0.6282421	0.694652	0.767529
3	RMSprop	0.2	0.001	0.6331019	0.683242	0.772016
3	RMSprop	0.3	0.0	0.62800604	0.668797	0.769294
3	RMSprop	0.3	0.01	0.6251438	0.698702	0.765249
3	RMSprop	0.3	0.001	0.6294138	0.672459	0.769196
3	RMSprop	0.5	0.0	0.62170106	0.613615	0.763981
3	RMSprop	0.5	0.01	0.602482	0.667695	0.746549
3	RMSprop	0.5	0.001	0.6263664	0.677499	0.766250

Tabell A.2: U-net-5, flerklassedatasett

Antall bånd	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
5	Adam	0.2	0.0	0.676153	0.686503	0.804638
5	Adam	0.2	0.01	0.735025	0.742998	0.845845
5	Adam	0.2	0.001	0.744010	0.757408	0.851859
5	Adam	0.3	0.0	0.653973	0.688806	0.788402
5	Adam	0.3	0.01	0.731175	0.734747	0.843175
5	Adam	0.3	0.001	0.726384	0.744395	0.839787
5	Adam	0.5	0.0	0.637103	0.647459	0.775984
5	Adam	0.5	0.01	0.701702	0.714210	0.822799
5	Adam	0.5	0.001	0.699165	0.701370	0.821093
5	RMSprop	0.2	0.0	0.276455	0.294817	0.397623
5	RMSprop	0.2	0.01	0.666651	0.707495	0.797010
5	RMSprop	0.2	0.001	0.669775	0.715197	0.799497
5	RMSprop	0.3	0.0	0.668593	0.691857	0.799074
5	RMSprop	0.3	0.01	0.660916	0.713455	0.792872
5	RMSprop	0.3	0.001	0.644821	0.685891	0.780807
5	RMSprop	0.5	0.0	0.652526	0.630301	0.787176
5	RMSprop	0.5	0.01	0.646916	0.686657	0.782336
5	RMSprop	0.5	0.001	0.662954	0.705806	0.794160
4	Adam	0.2	0.0	0.660401	0.652729	0.793211
4	Adam	0.2	0.01	0.742358	0.747034	0.850771
4	Adam	0.2	0.001	**0.756624**	0.768917	0.860227
4	Adam	0.3	0.0	0.672925	0.684854	0.802251
4	Adam	0.3	0.01	0.732183	0.748859	0.843864
4	Adam	0.3	0.001	0.735193	0.758160	0.845904
4	Adam	0.5	0.0	0.648412	0.615072	0.784313
4	Adam	0.5	0.01	0.708333	0.732572	0.827399
4	Adam	0.5	0.001	0.710587	0.739315	0.829053
4	RMSprop	0.2	0.0	0.659504	0.661292	0.792510
4	RMSprop	0.2	0.01	0.661924	0.711756	0.793699
4	RMSprop	0.2	0.001	0.650146	0.705744	0.784719
4	RMSprop	0.3	0.0	0.660408	0.683202	0.793045
4	RMSprop	0.3	0.01	0.663436	0.710709	0.794711
4	RMSprop	0.3	0.001	0.650446	0.708849	0.784912
4	RMSprop	0.5	0.0	0.641775	0.650484	0.779321
4	RMSprop	0.5	0.01	0.660038	0.709797	0.792419
4	RMSprop	0.5	0.001	0.644030	0.704632	0.780313
3	Adam	0.2	0.0	0.673607	0.673801	0.802864
3	Adam	0.2	0.01	0.722935	0.752568	0.837524
3	Adam	0.2	0.001	0.745640	0.760351	0.852865

3	Adam	0.3	0.0	0.593481	0.625609	0.742402
3	Adam	0.3	0.01	0.724199	0.729081	0.838402
3	Adam	0.3	0.001	0.724440	0.741323	0.838533
3	Adam	0.5	0.0	0.624498	0.670169	0.766513
3	Adam	0.5	0.01	0.703576	0.733852	0.824159
3	Adam	0.5	0.001	0.694027	0.709536	0.817500
3	RMSprop	0.2	0.0	0.650102	0.657166	0.785773
3	RMSprop	0.2	0.01	0.645791	0.706744	0.781142
3	RMSprop	0.2	0.001	0.629582	0.696161	0.768865
3	RMSprop	0.3	0.0	0.604638	0.631461	0.750505
3	RMSprop	0.3	0.01	0.632117	0.674221	0.770857
3	RMSprop	0.3	0.001	0.613624	0.660825	0.756535
3	RMSprop	0.5	0.0	0.627695	0.597330	0.762880
3	RMSprop	0.5	0.01	0.631770	0.692001	0.770371
3	RMSprop	0.5	0.001	0.626028	0.698094	0.766233

A.2 Binærklassifisering

Tabell A.3: U-net-4, binære datasett.

Datasett	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
Bebygd	Adam	0.2	0.0	0.838017	0.850852	0.911156
Bebygd	Adam	0.2	0.01	**0.870537**	0.877914	0.930438
Bebygd	Adam	0.2	0.001	0.860956	0.872806	0.924798
Bebygd	Adam	0.3	0.0	0.821226	0.854630	0.900780
Bebygd	Adam	0.3	0.01	0.856992	0.877173	0.922435
Bebygd	Adam	0.3	0.001	0.854111	0.880114	0.920736
Bebygd	Adam	0.5	0.0	0.822394	0.845833	0.901498
Bebygd	Adam	0.5	0.01	0.852531	0.869290	0.919717
Bebygd	Adam	0.5	0.001	0.850476	0.867477	0.918585
Bebygd	RMSprop	0.2	0.0	0.752724	0.809363	0.844837
Bebygd	RMSprop	0.2	0.01	0.819061	0.855900	0.899147
Bebygd	RMSprop	0.2	0.001	0.782391	0.827647	0.875472
Bebygd	RMSprop	0.3	0.0	0.824725	0.843500	0.902919
Bebygd	RMSprop	0.3	0.01	0.815288	0.848943	0.896719
Bebygd	RMSprop	0.3	0.001	0.812988	0.858746	0.895392
Bebygd	RMSprop	0.5	0.0	0.758132	0.807634	0.849679
Bebygd	RMSprop	0.5	0.01	0.813396	0.864572	0.895681
Bebygd	RMSprop	0.5	0.001	0.809536	0.862201	0.893331
Jordbruk	Adam	0.2	0.0	0.777951	0.765828	0.873092
Jordbruk	Adam	0.2	0.01	**0.827075**	0.827268	0.904082
Jordbruk	Adam	0.2	0.001	0.817455	0.815711	0.898252
Jordbruk	Adam	0.3	0.0	0.748046	0.766042	0.854023
Jordbruk	Adam	0.3	0.01	0.803479	0.807692	0.889532
Jordbruk	Adam	0.3	0.001	0.818063	0.816944	0.898560
Jordbruk	Adam	0.5	0.0	0.755955	0.743504	0.859032
Jordbruk	Adam	0.5	0.01	0.796497	0.801938	0.885252
Jordbruk	Adam	0.5	0.001	0.798546	0.804593	0.886440
Jordbruk	RMSprop	0.2	0.0	0.756539	0.769426	0.859515
Jordbruk	RMSprop	0.2	0.01	0.731555	0.750951	0.842529
Jordbruk	RMSprop	0.2	0.001	0.745045	0.768199	0.851702
Jordbruk	RMSprop	0.3	0.0	0.743416	0.758722	0.850623
Jordbruk	RMSprop	0.3	0.01	0.742140	0.777989	0.849696
Jordbruk	RMSprop	0.3	0.001	0.758828	0.781886	0.860686
Jordbruk	RMSprop	0.5	0.0	0.737332	0.758603	0.846933
Jordbruk	RMSprop	0.5	0.01	0.744727	0.759078	0.851348
Jordbruk	RMSprop	0.5	0.001	0.757303	0.782950	0.859774

Skog	Adam	0.2	0.0	0.768483	0.793892	0.867778
Skog	Adam	0.2	0.01	0.819212	0.837601	0.899711
Skog	Adam	0.2	0.001	**0.827252**	0.843089	0.904744
Skog	Adam	0.3	0.0	0.769308	0.797751	0.868349
Skog	Adam	0.3	0.01	0.820212	0.840929	0.900373
Skog	Adam	0.3	0.001	0.817581	0.834102	0.898674
Skog	Adam	0.5	0.0	0.763512	0.764961	0.864516
Skog	Adam	0.5	0.01	0.808203	0.831959	0.892936
Skog	Adam	0.2	0.001	0.824831	0.844510	0.903138
Skog	RMSprop	0.2	0.0	0.766250	0.784433	0.866303
Skog	RMSprop	0.2	0.01	0.764272	0.801943	0.864920
Skog	RMSprop	0.2	0.001	0.749921	0.788833	0.855221
Skog	RMSprop	0.3	0.0	0.759240	0.762603	0.861703
Skog	RMSprop	0.3	0.01	0.760947	0.796580	0.862561
Skog	RMSprop	0.3	0.001	0.765588	0.798070	0.865571
Skog	RMSprop	0.5	0.0	0.751010	0.765313	0.856166
Skog	RMSprop	0.5	0.01	0.766085	0.790916	0.865974
Skog	RMSprop	0.5	0.001	0.742758	0.781794	0.850424
Vann	Adam	0.2	0.0	0.964522	0.964909	0.981785
Vann	Adam	0.2	0.01	**0.973283**	0.973137	0.986401
Vann	Adam	0.2	0.001	0.972386	0.972978	0.985933
Vann	Adam	0.3	0.0	0.962706	0.968869	0.980836
Vann	Adam	0.3	0.01	0.969668	0.974294	0.984517
Vann	Adam	0.3	0.001	0.972034	0.972458	0.985759
Vann	Adam	0.5	0.0	0.924930	0.941235	0.955309
Vann	Adam	0.5	0.01	0.967107	0.968929	0.983164
Vann	Adam	0.5	0.001	0.969420	0.962708	0.984386
Vann	RMSprop	0.2	0.0	0.966581	0.968665	0.982890
Vann	RMSprop	0.2	0.01	0.970032	0.975123	0.984706
Vann	RMSprop	0.2	0.001	0.970080	0.976972	0.984743
Vann	RMSprop	0.3	0.0	0.963618	0.971943	0.981311
Vann	RMSprop	0.3	0.01	0.967085	0.972681	0.983159
Vann	RMSprop	0.3	0.001	0.923816	0.941236	0.955422
Vann	RMSprop	0.5	0.0	0.925306	0.941236	0.956114
Vann	RMSprop	0.5	0.01	0.926798	0.941236	0.957271
Vann	RMSprop	0.5	0.001	0.962553	0.970191	0.980725

Tabell A.4: U-net-5, binære datasett.

Datasett	Optimaliserer	Dropout	Læringsrate	mIOU	Validering mIOU	F1
Bebygd	Adam	0.2	0.0	0.815989	0.833423	0.897570
Bebygd	Adam	0.2	0.01	**0.853792**	0.872260	0.920548
Bebygd	Adam	0.2	0.001	0.847708	0.858330	0.916946
Bebygd	Adam	0.3	0.0	0.823408	0.847053	0.902167
Bebygd	Adam	0.3	0.01	0.842556	0.855448	0.913804
Bebygd	Adam	0.3	0.001	0.851179	0.864066	0.918999
Bebygd	Adam	0.5	0.0	0.811925	0.863450	0.895056
Bebygd	Adam	0.5	0.01	0.846658	0.874271	0.916199
Bebygd	Adam	0.5	0.001	0.848461	0.873166	0.917316
Bebygd	RMSprop	0.0	0.0	0.825260	0.842655	0.903387
Bebygd	RMSprop	0.0	0.01	0.800117	0.837851	0.886621
Bebygd	RMSprop	0.0	0.001	0.813613	0.857869	0.895812
Bebygd	RMSprop	0.2	0.0	0.804528	0.823348	0.890481
Bebygd	RMSprop	0.2	0.01	0.802868	0.861578	0.888581
Bebygd	RMSprop	0.2	0.001	0.813575	0.858887	0.895546
Bebygd	RMSprop	0.5	0.0	0.811587	0.835911	0.894790
Bebygd	RMSprop	0.5	0.01	0.793246	0.836001	0.882779
Bebygd	RMSprop	0.5	0.001	0.799143	0.854312	0.886284
Jordbruk	Adam	0.2	0.0	0.760137	0.738591	0.861701
Jordbruk	Adam	0.2	0.01	0.762143	0.783511	0.863126
Jordbruk	Adam	0.2	0.001	**0.810793**	0.820818	0.894127
Jordbruk	Adam	0.3	0.0	0.744542	0.752336	0.851508
Jordbruk	Adam	0.3	0.01	0.809435	0.823213	0.893256
Jordbruk	Adam	0.3	0.001	0.791823	0.807943	0.882300
Jordbruk	Adam	0.5	0.0	0.745161	0.741314	0.851972
Jordbruk	Adam	0.5	0.01	0.794629	0.766959	0.883988
Jordbruk	Adam	0.5	0.001	0.787707	0.813947	0.879682
Jordbruk	RMSprop	0.2	0.0	0.748035	0.763076	0.853819
Jordbruk	RMSprop	0.2	0.01	0.753678	0.777062	0.857333
Jordbruk	RMSprop	0.2	0.001	0.751353	0.776649	0.855783
Jordbruk	RMSprop	0.3	0.0	0.736629	0.757334	0.846315
Jordbruk	RMSprop	0.3	0.01	0.737711	0.781100	0.846439
Jordbruk	RMSprop	0.3	0.001	0.722207	0.763109	0.836056
Jordbruk	RMSprop	0.5	0.0	0.732373	0.726749	0.843437
Jordbruk	RMSprop	0.5	0.01	0.739925	0.757470	0.848429
Jordbruk	RMSprop	0.5	0.001	0.736442	0.767999	0.845811
Skog	Adam	0.2	0.0	0.776690	0.794967	0.873043
Skog	Adam	0.2	0.01	0.795348	0.816792	0.884976

Skog	Adam	0.2	0.001	0.812557	0.828274	0.895628
Skog	Adam	0.3	0.0	0.765287	0.795872	0.865819
Skog	Adam	0.3	0.01	0.782995	0.766268	0.877053
Skog	Adam	0.3	0.001	**0.815938**	0.833045	0.897708
Skog	Adam	0.5	0.0	0.752644	0.731880	0.857456
Skog	Adam	0.5	0.01	0.794355	0.817772	0.884271
Skog	Adam	0.5	0.001	0.795401	0.803044	0.884869
Skog	RMSprop	0.2	0.0	0.741564	0.758995	0.850070
Skog	RMSprop	0.2	0.01	0.758486	0.800431	0.860919
Skog	RMSprop	0.2	0.001	0.755602	0.781752	0.859181
Skog	RMSprop	0.3	0.0	0.702077	0.521672	0.823017
Skog	RMSprop	0.3	0.01	0.761212	0.797085	0.862742
Skog	RMSprop	0.3	0.001	0.738485	0.774775	0.847453
Skog	RMSprop	0.5	0.0	0.748369	0.775203	0.854514
Skog	RMSprop	0.5	0.01	0.753024	0.774572	0.857306
Skog	RMSprop	0.5	0.001	0.733777	0.769799	0.844806
Vann	Adam	0.2	0.0	0.961520	0.960900	0.980204
Vann	Adam	0.2	0.01	0.970404	0.973571	0.984904
Vann	Adam	0.2	0.001	**0.971393**	0.970728	0.985411
Vann	Adam	0.3	0.0	0.958377	0.958519	0.978530
Vann	Adam	0.3	0.01	0.970479	0.968539	0.984947
Vann	Adam	0.3	0.001	0.971202	0.975145	0.985320
Vann	Adam	0.5	0.0	0.928739	0.940154	0.961620
Vann	Adam	0.5	0.01	0.964454	0.969022	0.981742
Vann	Adam	0.5	0.001	0.966839	0.971481	0.983026
Vann	RMSprop	0.2	0.0	0.958021	0.965530	0.978333
Vann	RMSprop	0.2	0.01	0.966740	0.975599	0.982965
Vann	RMSprop	0.2	0.001	0.965288	0.974902	0.982201
Vann	RMSprop	0.3	0.0	0.958614	0.949955	0.978654
Vann	RMSprop	0.3	0.01	0.965363	0.974240	0.982246
Vann	RMSprop	0.3	0.001	0.967864	0.975008	0.983572
Vann	RMSprop	0.5	0.0	0.961284	0.971826	0.980063
Vann	RMSprop	0.5	0.01	0.960413	0.967148	0.979598
Vann	RMSprop	0.5	0.001	0.963593	0.972770	0.981280

Vedlegg B

Pythonscript

B.1 Arkitektur

Python scriptet vist under er brukt til trening, testing og beregning av statistikk til denne masteroppgaven.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan  6 14:47:51 2020

@author: Rebecca Brekke
"""

import random
import numpy as np
from skimage import io
from matplotlib import pyplot as plt
from keras import backend as K
from keras.models import Model
from keras.layers import Input, BatchNormalization, Activation, Dropout
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from PIL import Image
from keras.callbacks import ReduceLROnPlateau
from keras.models import model_from_json
import seaborn as sn
import pandas as pd
import shutil, os
import time


def load_data(dirname_images, dirname_labels):
    """
    Laster opp data fra bilde-mappe og label-mappe, til numpy-array.
    Skalerer dataene.
    """
    # Load images:

    images = []
    im_name = []

    for fname in os.listdir(dirname_images):
        if fname.endswith("tfw"):
            None
        else:
            img = os.path.join(dirname_images, fname)
            im = io.imread(img, plugin='tifffile', as_gray=False)
            images.append(im)
            im_name.append(fname)

    images = np.asarray(images)
```

```
    # Load labels:
    labels = []

    #from keras.utils import to_categorical


    for fname in os.listdir(dirname_labels):
        if fname.endswith("tif"):
            l_path = os.path.join(dirname_labels, fname)
            lab = io.imread(l_path, plugin='tifffile', as_gray=False)
            labels.append(lab)
        else:
            None

    labels = np.asarray(labels)
    #labels = np.expand_dims(labels, axis=3)
    labels = to_categorical(labels)


    # Scale the data and changing the type to float32
    images_scaled = np.float32(images/7585)

    images = images_scaled


    return images, labels, im_name


def get_data(dirname_images, dirname_labels, batch_size):
    """
    Genererer trening-, validering- og testdata.
    """

    images, labels, im_name = load_data(dirname_images, dirname_labels)

    #Train/Test/Val split

    random.seed(4)
    random.shuffle(images)
    random.seed(4)
    random.shuffle(labels)
    random.seed(4)
    random.shuffle(im_name)

    X_train = images[:750]
    y_train = labels[:750]

    X_test = images[751:850]
    y_test = labels[751:850]
    name_test = im_name[751:850]

    X_val = images[851:]
    y_val = labels[851:]
```

```
    print('Training:   ', np.shape(X_train), np.shape(y_train))
    print('Validation: ', np.shape(X_val), np.shape(y_val))
    print('Test Set:   ', np.shape(X_test), np.shape(y_test))

    print('Type X_train: ',X_train.dtype)
    print('Type X_test: ',X_test.dtype)
    print('Type y_train: ',y_train.dtype)

    # Creating an image data generator for agumentation of the data

    seed = 123
    datagen_train = ImageDataGenerator(vertical_flip=True,
                                horizontal_flip=False)
    datagen_y_train = ImageDataGenerator(vertical_flip=True,
                                horizontal_flip=False)


    train_g = datagen_train.flow(X_train, batch_size=batch_size, seed=seed)

    train_y_g = datagen_y_train.flow(y_train, batch_size=batch_size, seed=seed)

    # Function that puts togheter the agumentet images with the
    #correct label/mask
    def combine_generator(gen1,gen2):
        new_generator = zip(gen1,gen2)
        for (gen1, gen2) in new_generator:
            yield(gen1, gen2)

    train_generator = combine_generator(train_g, train_y_g)

    return X_val, y_val, X_test, y_test, name_test, train_generator, len(X_train)


def dice_coef(y_true, y_pred, smooth=1):
    """
    F1 score
    """
    intersection = K.sum(y_true * y_pred, axis=[1,2,3])
    union = K.sum(y_true, axis=[1,2,3]) + K.sum(y_pred, axis=[1,2,3])
    dice = K.mean((2. * intersection + smooth)/(union + smooth), axis=0)
    return dice


def iou_coef(y_true, y_pred, smooth=1):
    """
    mIOU score
    """
    intersection = K.sum(K.abs(y_true * y_pred), axis=[1,2,3])
    union = K.sum(y_true,[1,2,3])+K.sum(y_pred,[1,2,3])-intersection
    iou = K.mean((intersection + smooth) / (union + smooth), axis=0)
    return iou

# Hentet fra:
#https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2
```

```python
def conv2d_block(input_tensor, n_filters, kernel_size = 3, 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 unet_4(input_img, n_filters = 16, dropout = 0.1, batchnorm = True,
           n_classes = 5, last_activation="sigmoid"):
    """
    U-net med 4 blokker.
    """
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters = n_filters * 16, kernel_size = 3, batchnorm = batchnc

    # Expansive Path
    u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2), padding = 'same')(c5)
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)

    u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2), padding = 'same')(c6)
```

```
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)

    u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2), padding = 'same')(c7)
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)

    u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2), padding = 'same')(c8)
    u9 = concatenate([u9, c1])
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)

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


def unet_5(input_img, n_filters = 16, dropout = 0.1, batchnorm = True,
           n_classes = 5, last_activation="sigmoid"):
    """
    U-net med 5 blokker.
    """
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters * 16, kernel_size = 3, batchnorm = batchnorm)
    p5 = MaxPooling2D((2, 2))(c5)
    p5 = Dropout(dropout)(p5)

    c6 = conv2d_block(p5, n_filters = n_filters * 32, kernel_size = 3, batchnorm = batchnorm)

    # Expansive Path

    u7 = Conv2DTranspose(n_filters * 16, (3, 3), strides = (2, 2), padding = 'same')(c6)
    u7 = concatenate([u7, c5])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 16, kernel_size = 3, batchnorm = batchnorm)
```

```
    u8 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2), padding = 'same')(c7)
    u8 = concatenate([u8, c4])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)

    u9 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2), padding = 'same')(c8)
    u9 = concatenate([u9, c3])
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)

    u10 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2), padding = 'same')(c9)
    u10 = concatenate([u10, c2])
    u10 = Dropout(dropout)(u10)
    c10 = conv2d_block(u10, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)

    u11 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2), padding = 'same')(c10)
    u11 = concatenate([u11, c1])
    u11 = Dropout(dropout)(u11)
    c11 = conv2d_block(u11, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)

    outputs = Conv2D(n_classes, (1, 1), activation=last_activation)(c11)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model

    # Functions from lectures in DAT300 about CNN


def run_model(train_generator, X_val, y_val, unet_X, unet_name, batch_size,
              epochs, reduce_lr, min_lr, dropout, optimizer, X_train_len,
              band_num, last_activ, filename='model_cnn_model',
              plot=False, n_classes=5):
    """
    Funksjon for å kjøre U-net modellene.
    """
    input_img = Input((256, 256, band_num), name='img')
    model = unet_X(input_img, n_filters=20, dropout=dropout, batchnorm=False,
                   n_classes = n_classes, last_activation=last_activ)

    if n_classes == 5:
        loss='categorical_crossentropy'
    elif n_classes == 2:
        loss='binary_crossentropy'


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

    # Fit the model on the agumentet training data
    historyFlow2 = model.fit_generator(train_generator,
        epochs=epochs, steps_per_epoch=X_train_len/batch_size,
        validation_data=(X_val, y_val),
        validation_steps=len(X_val)/batch_size, callbacks=[reduce_lr])

    # lagrer modell til JSON
```

```
    model_json = model.to_json()
    with open("%s.json" % filename, "w") as json_file:
        json_file.write(model_json)
    # lagrer vekter til HDF5
    model.save_weights("%s.h5" % filename)
    print("Saved model to disk")

    if plot is True:

        #Plotting Loss and Dice score
        iou_ = historyFlow2.history['iou_coef']
        val_iou_ = historyFlow2.history['val_iou_coef']

        loss = historyFlow2.history['loss']
        val_loss = historyFlow2.history['val_loss']

        epochs_range = range(epochs)

        plt.figure(figsize=(8, 8))
        plt.subplot(1, 2, 1)
        plt.plot(epochs_range, iou_, label='Training Accuracy, iou')
        plt.plot(epochs_range, val_iou_, label='Validation Accuracy, iou')
        plt.legend(loc='lower right')
        plt.title("optimizer=%s, min_lr=%s, dropout=%s, U_net=%s" % (optimizer,
                                                min_lr, dropout, unet_name))

        plt.subplot(1, 2, 2)
        plt.plot(epochs_range, loss, label='Training Loss')
        plt.plot(epochs_range, val_loss, label='Validation Loss')
        plt.legend(loc='upper right')
        #plt.title()
        plt.show()

    return (max(historyFlow2.history['iou_coef']),
            max(historyFlow2.history['val_iou_coef']),
            max(historyFlow2.history['dice_coef']), iou_, val_iou_,
            historyFlow2.history['dice_coef'])


def forvirringsmatrise_flerklasse(y_classes, y_true):
    cm = []
    for l in range(len(y_classes)):
        cm_l = [0]*25
        for i in range(len(y_classes[l])):

            for j in range(256):
                pred = y_classes[l][i][j]
                true_y = y_true[l][i][j]

                if true_y == 0:
                    if pred == 0:
                        cm_l[0]+=1
                    elif pred == 1:
                        cm_l[1]+=1
```

```
            elif pred == 2:
                cm_l[2]+=1
            elif pred == 3:
                cm_l[3]+=1
            elif pred == 4:
                cm_l[4]+=1

        if true_y == 1:
            if pred == 0:
                cm_l[5]+=1
            elif pred == 1:
                cm_l[6]+=1
            elif pred == 2:
                cm_l[7]+=1
            elif pred == 3:
                cm_l[8]+=1
            elif pred == 4:
                cm_l[9]+=1

        if true_y == 2:
            if pred == 0:
                cm_l[10]+=1
            elif pred == 1:
                cm_l[11]+=1
            elif pred == 2:
                cm_l[12]+=1
            elif pred == 3:
                cm_l[13]+=1
            elif pred == 4:
                cm_l[14]+=1

        if true_y == 3:
            if pred == 0:
                cm_l[15]+=1
            elif pred == 1:
                cm_l[16]+=1
            elif pred == 2:
                cm_l[17]+=1
            elif pred == 3:
                cm_l[18]+=1
            elif pred == 4:
                cm_l[19]+=1

        if true_y == 4:
            if pred == 0:
                cm_l[20]+=1
            elif pred == 1:
                cm_l[21]+=1
            elif pred == 2:
                cm_l[22]+=1
            elif pred == 3:
                cm_l[23]+=1
            elif pred == 4:
                cm_l[24]+=1
```

```
            cm.append(cm_l)

        cm1 = [sum(i) for i in zip(*cm)]

        sum_cm1 = sum(cm1)
        percent_cm1 = [((i/sum_cm1)*100) for i in cm1]


        df_cm =  pd.DataFrame({"Bakgrunn": percent_cm1[:5],
                               "Vann": percent_cm1[5:10],
                               "Skog": percent_cm1[10:15],
                               "Jordbruk": percent_cm1[15:20],
                               "Bebygd": percent_cm1[20:]},
                              index=["Bakgrunn", "Vann", "Skog", "Jordbruk", "Bebygd"])

        sn.set(font_scale=1.4) # for label size
        sn.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='d') # font size
        plt.show()


    def forvirringsmatrise_binær(y_classes, y_true):
        cm = []
        for l in range(len(y_classes)):
            cm_l = [0]*4
            for i in range(len(y_classes[l])):

                for j in range(256):
                    pred = y_classes[l][i][j]
                    true_y = y_true[l][i][j]

                    if true_y == 0:
                        if pred == 0:
                            cm_l[0]+=1
                        elif pred == 1:
                            cm_l[1]+=1

                    if true_y == 1:
                        if pred == 0:
                            cm_l[2]+=1
                        elif pred == 1:
                            cm_l[3]+=1
            cm.append(cm_l)

        cm1 = [sum(i) for i in zip(*cm)]

        sum_cm1 = sum(cm1)
        percent_cm1 = [((i/sum_cm1)*100) for i in cm1]

        df_cm =  pd.DataFrame({"Sann 0": percent_cm1[:2],
                               "Sann 1": percent_cm1[2:4]},
                              index=["Predikert 0", "Predikert 1"])

        sn.set(font_scale=1.4) # for label size
        sn.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='d') # font size
```

```
    plt.show()


def test_data(model_name, X_test, y_test, name_test, batch_size,
              dirname_lab_pred, dirname_labels):
    """
    Kjøre flerklasse-testdata på modellene.
    """

    # load json and create model
    json_file = open('%s.json' % model_name, 'r')
    loaded_model_json = json_file.read()
    json_file.close()

    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("%s.h5" % model_name)

    # evaluate loaded model on test data
    loaded_model.compile(optimizer="adam", loss='categorical_crossentropy',
                         metrics=[iou_coef, dice_coef])
    score = loaded_model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
    print("%s: %.2f%%" % (loaded_model.metrics_names[2], score[2]*100))

    y_pred = loaded_model.predict(X_test, batch_size = batch_size)
    y_classes = y_pred.argmax(axis=-1)
    y_true = y_test.argmax(axis=-1)

    forvirringsmatrise_flerklasse(y_classes, y_true)

    for i in range(len(y_classes)):
        name = name_test[i]
        Image.fromarray(y_classes[i].astype(np.uint8)).save(dirname_lab_pred
                                                +"\\%s.tif" % name[:-4])

    for iname in os.listdir(dirname_lab_pred):
        print(iname[:-3])
        path_label = os.path.join(dirname_labels, iname[:-3]+"tfw")
        # # fpath = os.path.join(dirname_labels, iname)

        shutil.copy(path_label, dirname_lab_pred)

    return y_true, y_classes


def test_data_binary(model_name, X_test, y_test, name_test, batch_size, value,
                     dirname_lab_pred, dirname_labels):
    """
    Kjøre binær-testdata på modellene.
    """

    # load json and create model
    json_file = open('%s.json' % model_name, 'r')
```

```
    loaded_model_json = json_file.read()
    json_file.close()

    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("%s.h5" % model_name)

    # evaluate loaded model on test data
    loaded_model.compile(optimizer="adam", loss='binary_crossentropy',
                         metrics=[iou_coef, dice_coef])
    score = loaded_model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
    print("%s: %.2f%%" % (loaded_model.metrics_names[2], score[2]*100))

    y_pred = loaded_model.predict(X_test, batch_size = batch_size)
    y_classes = y_pred.argmax(axis=-1)
    y_true = y_test.argmax(axis=-1)

    forvirringsmatrise_binær(y_classes, y_true)

    #Gjør om verdien fra 1 til verdien til arealklassen, før det lagres som et bilde.
    y_classes = np.where(y_classes==1, value, y_classes)

    for i in range(len(y_classes)):
            name = name_test[i]
            Image.fromarray(y_classes[i].astype(np.uint8)).save(dirname_lab_pred
                                                    +"\\%s.tif" % name[:-4])

    for iname in os.listdir(dirname_lab_pred):
        print(iname[:-3])
        path_label = os.path.join(dirname_labels, iname[:-3]+"tfw")

        shutil.copy(path_label, dirname_lab_pred)


def fusjon(dirname_lab_bebygd, dirname_lab_skog, dirname_lab_jord,
           dirname_lab_vann, dirname_lab_alle, model_num):
    """
    Fusjonerer de fire prediksjonene fra de binære datasettene sammen til ett datasett.
    """
    dirname = [dirname_lab_bebygd, dirname_lab_skog, dirname_lab_jord,
               dirname_lab_vann]

    labels_bebygd = []
    labels_skog = []
    labels_jord = []
    labels_vann = []

    labels = [labels_bebygd, labels_skog, labels_jord, labels_vann]
    im_name = [[], [], [], []]

    for i in range(len(dirname)):
        for fname in sorted(os.listdir(dirname[i])):
            if fname.endswith("tif"):
```

```
                l_path = os.path.join(dirname[i], fname)
                lab = io.imread(l_path, plugin='tifffile', as_gray=False)
                labels[i].append(lab)
                im_name[i].append(fname)
            else:
                None
        labels[i] = np.asarray(labels[i])

    labels_bebygd = np.asarray(labels_bebygd)
    labels_jord = np.asarray(labels_jord)
    labels_skog = np.asarray(labels_skog)
    labels_vann = np.asarray(labels_vann)

    for l in range(len(labels_vann)):
        for i in range(len(labels_vann[l])):
            for j in range(256):
                vann = labels_vann[l][i][j]
                skog = labels_skog[l][i][j]
                jord = labels_jord[l][i][j]
                bebygd = labels_bebygd[l][i][j]

                if skog > 0:
                    labels_vann[l][i][j]=skog
                elif jord > 0:
                    labels_vann[l][i][j]=jord
                elif bebygd > 0:
                    labels_vann[l][i][j]=bebygd
                else:
                    None

    for i in range(len(labels_vann)):
        name = im_name[0][i]
        Image.fromarray(labels_vann[i].astype(np.uint8)).save(dirname_lab_alle
                                                              +"\\%s.tif" % name[:-4])

    for iname in os.listdir(dirname_lab_alle):
        path_label = os.path.join(dirname_lab_vann, iname[:-3]+"tfw")
        shutil.copy(path_label, dirname_lab_alle)


def optimalisering(dirname_images, dirname_labels, n_classes, band_num,
                   unet_X, unet_name):
    """
    Funksjon for optimalisering for nettverkene.
    """
    batch_size = 20
    epochs = 20
    result_list = []
    optimizer = ["adam", "RMSprop"]
    dropout=[0.2, 0.3, 0.5]
    min_lr=[0.0, 0.01, 0.001]

    X_val, y_val, X_test, y_test, name_test, train_generator, X_train_len = get_data(
                dirname_images, dirname_labels, batch_size)
```

```
for opt in optimizer:
    for drop in dropout:
        for lr in min_lr:
            reduce_lr = ReduceLROnPlateau(monitor='iou_coef',
                                          factor=0.2, patience=4, min_lr=lr)
            (r_iou_coef, r_val_iou_coef, r_acc, r_dice_coef, iou_p, val_iou_p,
                                      dice_p) = run_model(train_generator,
                                          X_val, y_val, unet_X, unet_name, batch_size, epochs,
                                          reduce_lr, lr,  drop, opt, X_train_len, band_num,
                                         filename='model_', plot=True, n_classes=n_classes)

            result = [opt, drop, lr, r_iou_coef, r_val_iou_coef,
                      r_acc, r_dice_coef]
            result_list.append(result)
            print(result)
```

B.2 Prosessering

Python scriptet vist under er brukt til prosessering av treningsdatasettet og sammenslåing av arealressurskart ved generering av kart fra satellittbiler.

```
import arcpy
import os
import time


def export_trainingdata(image, folder, feature_data):
    """
    Eksporterer treningsdata. Splitter opp satellittbilder
    i kvadrater på 256*256 piksler.
    """
    arcpy.ia.ExportTrainingDataForDeepLearning(image, folder, feature_data, "TIFF",
                256, 256, 200, 200, "ONLY_TILES_WITH_FEATURES", "Classified_Tiles", 0,
                "classvalue", 0, None, 0, "MAP_SPACE", "PROCESS_AS_MOSAICKED_IMAGE",
                "NO_BLACKEN", "FIXED_SIZE")


def clip_ar5(feature_data, image_data_folder, output_folder):
    """
    Klipper sann label basert på utklippet av satellittbildene lagd i funksjonen over.
    """
    t0 = time.time()
    raster_files = [image_data_folder + "\\" + x for x in os.listdir(image_data_folder)]
    i = 0
    for image in raster_files:
        if image.endswith("tfw"):
            None
        else:
            if i % 10 == 0:
                t1 = time.time()
                print("Tid: ", (t1-t0)/(3600))

            top = arcpy.GetRasterProperties_management(image, "TOP")
            left = arcpy.GetRasterProperties_management(image, "LEFT")
            right = arcpy.GetRasterProperties_management(image, "RIGHT")
            bottom = arcpy.GetRasterProperties_management(image, "BOTTOM")

            name_line = image.split("\\")  # År og linje fra path name.

            name = "%s" % (name_line[-1][:-4])
            print(name, "antall prosessert:", i)
            i += 1

            arcpy.management.Clip(feature_data,
                                  "%s %s %s %s" % (left, bottom, right, top),
                                  output_folder + "\%s.tif" % name, image, "100"
                                  "", "NONE", "MAINTAIN_EXTENT")


def merge(raster_files, folder, name, nr_bands):
    arcpy.management.MosaicToNewRaster(raster_files,
```

```
                folder, "%s.png" % name, None,
                "8_BIT_UNSIGNED", 10, nr_bands, "LAST", "FIRST")
```

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

Postboks 5003
NO-1432 Ås
Norway