



IT DIPLOMUDDANNELSEN

BIG DATA - FORÅR 2019



REMOTE SENSING: IDENTIFICERING AF AKTIVE GRAVEOMRÅDER I RÅSTOFGRAVE VED HJÆLP AF FRIE OFFENTLIGE GEODATA

STUDERENDE: Allan Kötter

STUDIENUMMER: s140381

UNDERSØGELSE:

Jacob Nordfalk og John Aasted Sørensen

DATO: 27. maj 2019

ANSLAG: 45099 tegn (7068 ord)

Indhold

1	Indledning og formål	2
2	Datakilder, infrastruktur og feature extraction	3
2.1	Datakilder og afgrænsning af projektet	3
2.2	Modelområder, datamodel og infrastruktur	3
2.3	Feature extraction	4
3	Indledende dataanalyse	6
3.1	Visualisering	6
3.2	PCA analyse	8
3.3	Delkonklusion	8
4	Teori: Neurale Netværk og Remote Sensing	12
4.1	Pixel-baserede metoder vs. objekt-baserede metoder	12
4.2	Neurale netværk	13
5	Simpel billede segmentering med Scikit-image	16
5.1	Threshold segmentering	16
5.2	Watershed segmentering med udgangspunkt i thresholds	16
5.3	Segmentering baseret på Clustering (K-means)	19
5.4	Delkonklusion på simpel segmentering	21
6	Forsøg med Mask R-CNN	22
6.1	Opsætning af modellen	22
6.2	Træning af modellen	23
6.3	Modevaluering	23
7	Distribueret modelkørsel	27
7.1	Distribueret GPU-processering	27
7.2	I den nære fremtid?	28
8	Konklusion og perspektivering	30
	Litteratur	32
	Bilag	38
A	Anvendte værktøjer	39
B	Notebook: EDA	40
C	Notebook: Simpel billedesegmentering med Scikit-Image	54

D Notebook: Clustering med R-Tree	68
E Udskrift af koden: Mask R-CNN modelopsætning	73
F Notebook: Mask R-CNN inspect data	82
G Notebook: Mask R-CNN inspect model	102

1. Indledning og formål

Siden 2016 har regionerne haft ansvar for myndighedsopgaven omkring råstofgrave, herunder indvindingstilladelser og tilsyn med råstofindvindingernes overholdelse af vilkår i indvindingstilladelserne. Det er en stor og tidskrævende opgave og regionerne har ikke ressourcer til hyppige fysiske tilsyn.

Regionerne fører derfor tilsyn med råstofgravene i en ligelig fordelt rækkefølge, der samtidig bliver tilpasset den øvrige arbejdsbelastning. Der kan derfor forekomme ulovlige forhold i en råstofgrav gennem længere tid, før dette bliver identificeret og påtalt af regionen, og det er derfor ønskeligt hvis tilsynsopgaven kunne effektiviseres og målrettes.

Der findes i dag en række muligheder for at anvende remote sensing som supplement til myndighedernes fysiske tilsynsopgaver. Hvert år udgiver Styrelsen for Dataforsyning og Effektivisering (SDFE) et opdateret ortofoto (luftfoto) i høj opløsning for hele danmark, der bl.a. er målrettet offentlige myndigheders tilsynsvirksomhed. Desuden vedligeholder SDFE også en Digital Terræn Model (DTM) for hele danmark, baseret på LIDAR målinger. Der er i dag også adgang til store mængder af højopløselige satellitdata, der løbende stilles til rådighed med en høj opdateringsfrekvens. og flere og flere myndigheder tager også droner i brug, til indsamling af billede- og højdedata i forbindelse med tilsyn.

Jeg ønsker med dette projekt, at udforske mulighederne for at anvende Machine Learning på højopløselige ortofoto, som grundlag for et varslingsystem og beslutningsstøtteværktøj, der kan hjælpe regionerne med at målrette tilsynsopgaven i råstofgrave. Jeg vil med projektet forsøge at belyse den nyeste viden inden for remote sensing med fokus på Kunstig Intelligens (AI) og Deep Learning algoritmer.

Projektet kommer til at tage udgangspunkt i frie geodata fra SDFE, fordi disse er let tilgængelige og anvendes i forvejen af de fleste offentlige myndigheder. Ved hjælp af ortofoto fra SDFE vil jeg forsøge at segmentere og klassificere et areal som "*Aktivt graveområde*" - dvs. det område hvor der foregår aktiv råstofudvinding. Perspektivet er, at området kan sammenlignes med det areal, der er givet tilladelse til at grave råstof i og varsle, hvis der bliver gravet uden for det tilladte område. Arealet ville muligvis også kunne danne grundlag for andre beregninger f.eks. opgørelse af indvundne råstofmængder inden for området.

Dataprocessering, analyser og modellering gennemføres i Python. Selvstudiet vil fokusere på udviklingen af Deep Learning algoritmer inden for analyse af højopløselige billeder herunder nyeste viden inden for remote sensing.

2. Datakilder, infrastruktur og feature extraction

2.1 Datakilder og afgrænsning af projektet

Projektet her skal opfattes som et *"Proof of Concept"* for anvendelse af remote sensing i råstofgrave, og der vil blive taget udgangspunkt i ortofotos fra Kortforsyningen som datakilde. Ortotos udstilles som en del af statens frie grunddata (se <https://sdfe.dk/data-skaber-vaerdi/grunddata/>), og disse data har en høj kvalitet, kan tilgås på en nem og standardiseret måde, og de indgår i forvejen som baggrundskort i de fleste offentlige myndigheders administration.

Det landsdækkende ortofotos bliver opdateret en gang om året, så i forhold til et varslings-system der ideelt set kunne detekterer løbende ændringer i en råstofgrav, er opdateringsfrekvensen for lav. Til dette formål ville f.eks. satellitdata være bedre egnet, da disse i dag kan fås med en opdateringsfrekvens på ned til et par dage.

API'erne til forskellige satellitdata ville dog kræve en del arbejde at sætte sig ind i, og de satellitdata, der stilles til rådighed med høj opløsning og høj opdateringsfrekvens, koster penge. Som datakilde til Remote Sensing er satellitbilleder sammenlignelige med Ortotos fra Kortforsyningen, og de koncepter og algoritmer som afprøves i dette projekt, vurderes relativt nemt at kunne overføres til satellitdata.

Råstofteamet har desuden adgang til en drone, som også kan indsamle luftfotos og måle terrænhøjder. Arbejdet med dronedata er dog også væsentligt mere omfattende, idet disse data indebærer bearbejdning bl.a. opretning og sammenstykning af fotos, og dette ligger uden for projektets fokusområde. Koncepter og metoder fra projektet vurderes også at kunne overføres til dronedata, hvis disse skal inddrages på et senere tidspunkt.

2.2 Modelområder, datamodel og infrastruktur

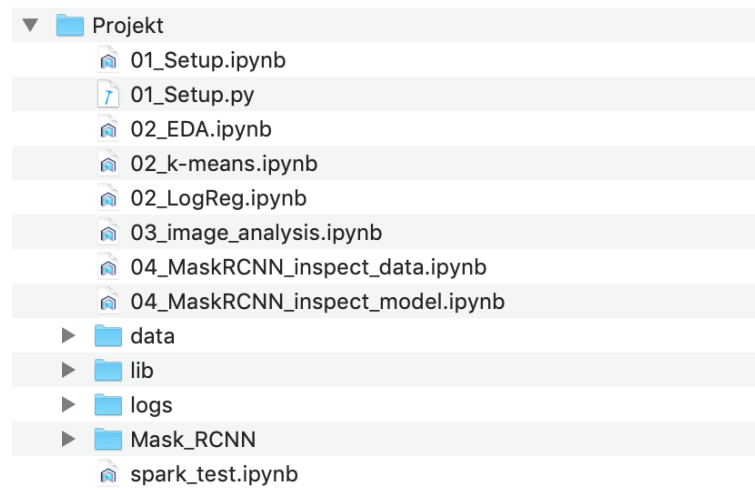
Når man arbejder med *"Change Detection"* inden for Remote Sensing, er metoden typisk at udlægge et *"Area Of Interest" (AOI)*, der er et rektangulært område, hvor data kan indsamles og ændringer over tid registreres. I dette projekt udlægges derfor AOI'er omkring de aktive råstofgrave i Region Hovedstaden, så disse kan danne grundlag for en klassifikationsmodel for det *"aktive graveområde"* i grusgrave.

Polygonerne, som definerer de områder, hvor der er blevet givet tilladelse til at grave inden for, skal også kunne håndteres og tilgås. De vil i dette projekt tilgås fra en fil i projekt-mappen, men findes i råstofteamets administrative database *RIT* og vil ideelt set skulle tilgås fra kilden.

Når AOI'erne er udlagt, kan ortofotos og evt. terrænmodel downloades for disse arealer, via Kortforsynings webservices. Da projektet fokuserer på ændringer over tid, vil AOI'erne fremadrettet også skulle bruges til at downloade nye opdaterede data, så ændringer i det aktive graveområde kan detekteres vha modellen. Datastrukturen skal dermed kunne håndtere tidsspecifikke versioner af data i et modelområde.

For at understøtte disse behov, har jeg programmeret en Controller-klasse, der giver adgang til alle AOI'erne, samt en række hjælpeklasser og metoder til at vælge et specifikt område ud og arbejde med dette - herunder downloade data fra Kortforsynings webservices. Klasserne implementerer bl.a. metoder til at downloade nye tidsspecifikke data og gemme disse på en struktureret måde. Til styring af de tilgængelige data der relaterer sig til et AOI, implementerer controlleren et register over tilgængelige billedfiler og træningsdata, samt persistent lagring i en relationel database.

Filstrukturen for projektet er vist på figur 2.1



Figur 2.1: Projektet filstruktur

Under mappen *data/* på figur 2.1, er der for hvert AOI oprettet en mappe, navngivet med en GUID for det pågældende areal. I dette projekt har jeg kun ét datasæt pr. AOI, men forhold til fremadrettet download af nye tidsspecifikke ortofoto, vil der i hver AOI-mappe kunne oprettes en undermappe til hvert datasæt, hvor et datasæt består af tilgængelige billedfiler fra et specifikt tidspunkt. Mapperne vil kunne navngives med datakilde og dato f.eks. *KF20190213*, for data hentet fra Kortforsyningen den 13. april 2019.

Med denne struktur, vil infrastrukturen kunne håndtere udvidelse med nye datakilder - f.eks. satellitdata - og samtidig understøtte løbende indsamling af tidsspecifikke datasæt fra samme datakilde for hvert område.

2.3 Feature extraction

For hvert datasæt genereres en tabel / matrice, bestående af en række for hver pixel / celle, og med features for det specifikke datasæt som kolonner. Tabellens størrelse bliver derfor $N \times M$, hvor N vil være billedets højde gange bredde i pixels, mens M vil være antallet af features i datasættet. Fordelen ved denne metode er, at selv om AOI'er udlægges med varierende rektangler, vil de kunne repræsenteres i samme tabelstruktur - dvs. med samme

antal features men med varierende antal rækker.

Tabellen gemmes som en csv-fil sammen med rådata, og vil senere kunne indlæses og give let adgang til features for hver enkelt celle / pixel. Tabelstrukturen for er vist på i tabel 2.1

Tabel 2.1: Datastruktur for matrice med features i datasættet

Felt	Datatype	Bemærkninger
aoi_id	GUID	ID for området, giver sammen med række- og kolonneindex en unik identifikation af den enkelte celle
row_index	int	Rækkeindex for celle i datasættet
column_index	int	Kolonneindex for celle / pixel i datasættet
x	float	X-værdi for cellens centerkoordinat
y	float	Y-værdi for cellens centerkoordinat
r	int	Rød kanal fra ortofoto
g	int	Grøn kanal fra ortofoto
b	int	Blå kanal fra ortofoto
grey	int	Beregnet gråtone i ortofoto
nir	int	Nær infrarød kanal fra ortofoto
ndvi	float	Beregnet vegetationsindex
terraenkote	float	Terrænkote fra terrænmodel
aktiv	bool	Angivelse af om cellen ligger inden eller uden for det aktive graveområde i billedet. Baseret på manuel tolkning af billede

2.3.1 Beregning af vegetationsindex (NDVI)

Inden for Remote Sensing, er vegetation en vigtig parameter i forhold til identificering af arealanvendelse. Til dette formål bruges målinger af det røde og det infrarøde spektrum, idet planter absorberer det røde lys og reflekterer det infrarøde. Da der er adgang både til målinger af det røde (R) og det *nær infrarøde (NIR)* spektrum i datasættet, har jeg beregnet et *vegetationsindex (NDVI)* ud fra standartformlen:

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

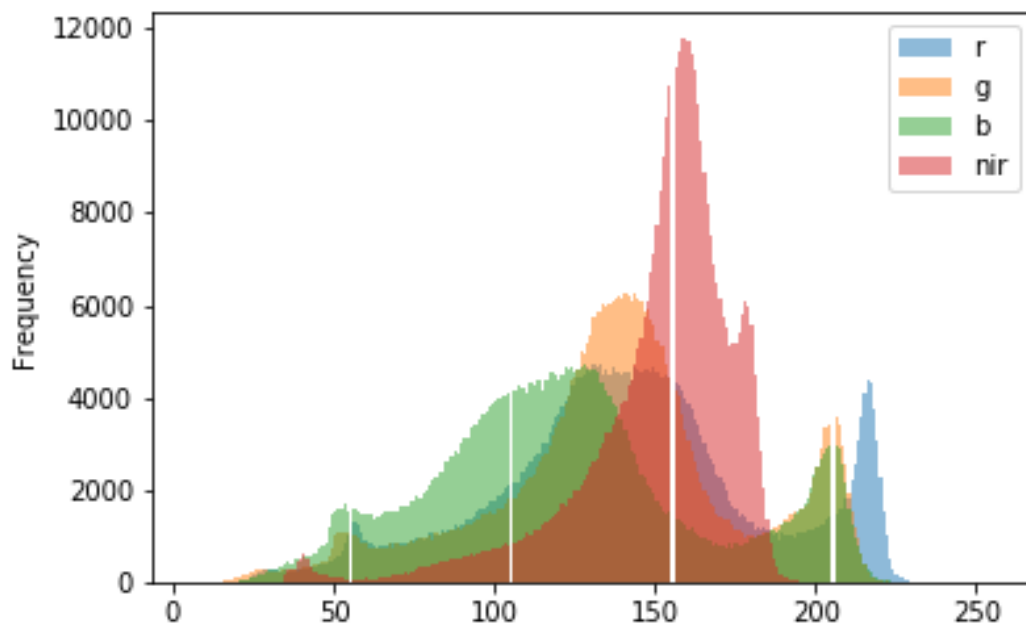
NDVI stå for *Normalized Difference Vegetation Index*, og er et udtryk for den relative forskel mellem hhv. det røde og det infrarøde spektrum (se bl.a. [19])

3. Indledende dataanalyse

Som udgangspunkt er det let visuelt at identificere det aktive graveområde i en råstofgrav, når man kikker på et ortofoto. I dette kapitel vil jeg indledningsvist bruge nogle eksplorative analysemetoder til at undersøge, om det pixelbaserede datasæt kan understøtte dette og dermed om almindelige statistiske modeller og klassiske Machine Learning algoritmer kan anvendes til identificering og klassificering af det aktive graveområde i en råstofgrav.

3.1 Visualisering

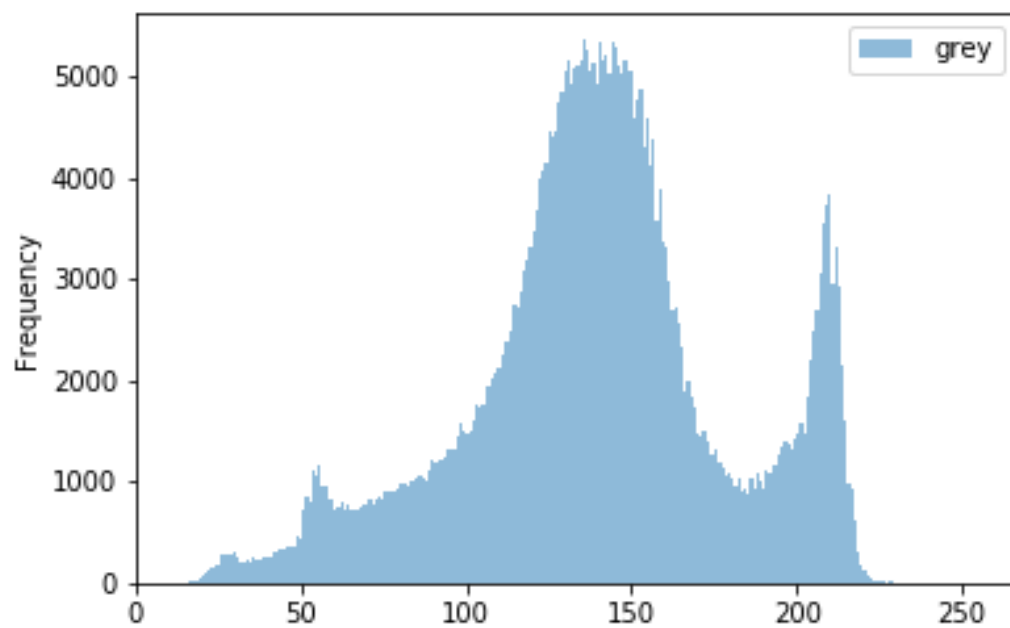
Indledningsvist gennemføres en dataanalyse på en feature-matrice fra en af områderne. Den Jupyter Notebook, som analysen er gennemført med, er vist i bilag B.



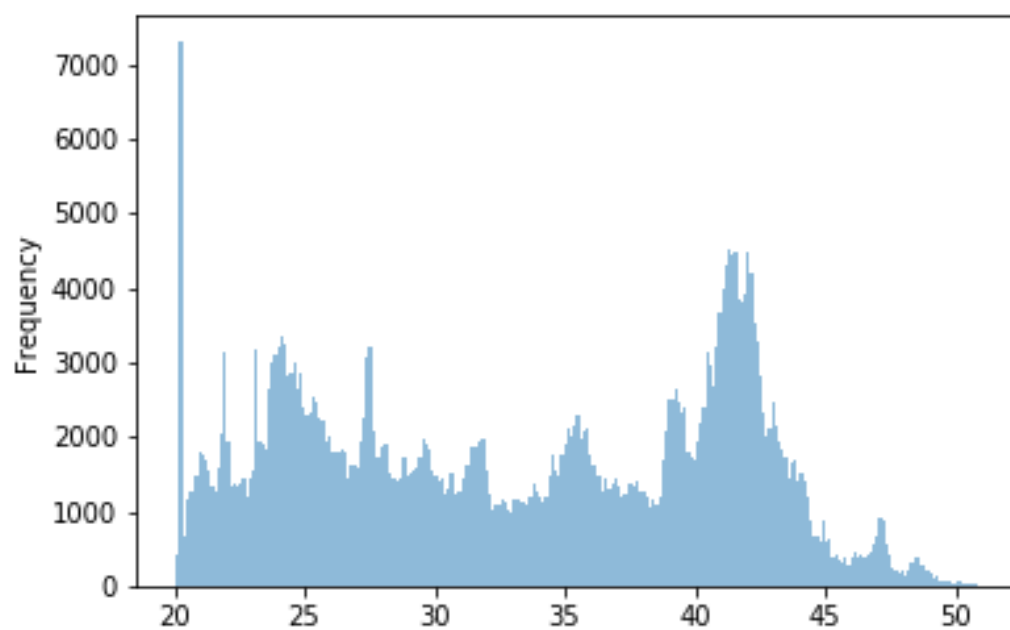
Figur 3.1: Histogram for rød, grøn, blå og nir

3.1.1 Korrelerende features i datasættet

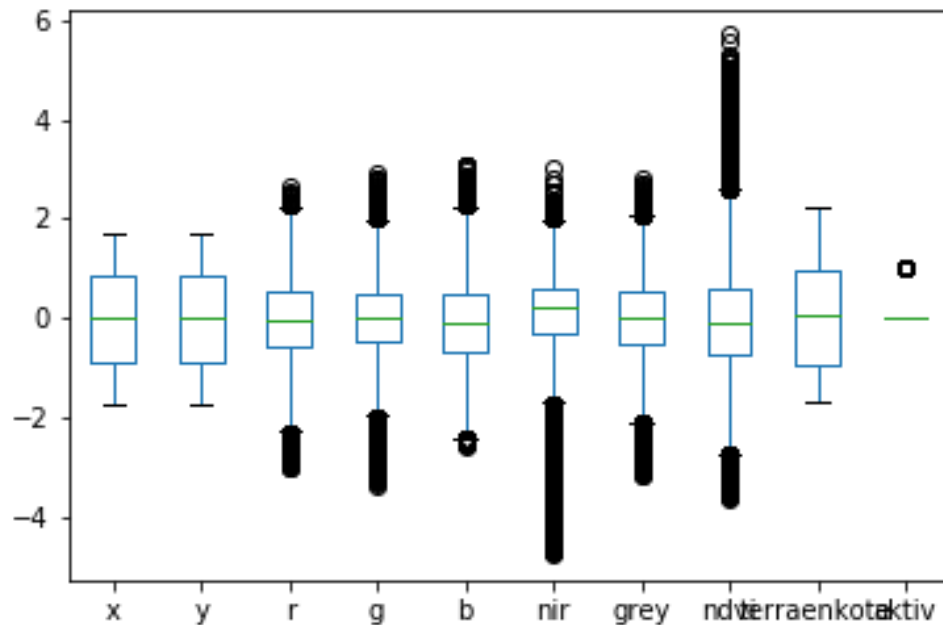
For at teste mulige korrelerende features i datasættet, er der i figur 3.5 vist en scatterplot matrice for kolonnerne x, y, r, g, b, nir, gray, terraenkote, ndvi og aktiv. Plottet viser at kolonnerne r, g, b og gray har en lineær korrelation, hvilket ikke er overraskende. Figuren viser også at der ikke umiddelbart ses korrelation mellem de øvrige kolonner - heller ikke mellem kolonnen "aktiv" og de øvrige kolonner.



Figur 3.2: Histogram for gråtone



Figur 3.3: Histogram for terrænmodel



Figur 3.4: Boxplot - grupperet box and whiskers på normaliserede data

De punkter, hvor aktiv er 1 (dvs. inden for det aktive graveområde) er generelt relateret til et isoleret værdi-område i de øvrige parametre, men for dette værdiområde findes der alle steder tilsvarende pixels, hvor aktiv er 0 (dvs. uden for det aktive graveområde). Den parameter med det mindste overlap mellem aktiv 0 og aktiv 1 er ndvi, hvilket er forventeligt. Dog er der også her et væsentligt overlap mellem ndvi for aktiv 0 og aktiv 1.

3.2 PCA analyse

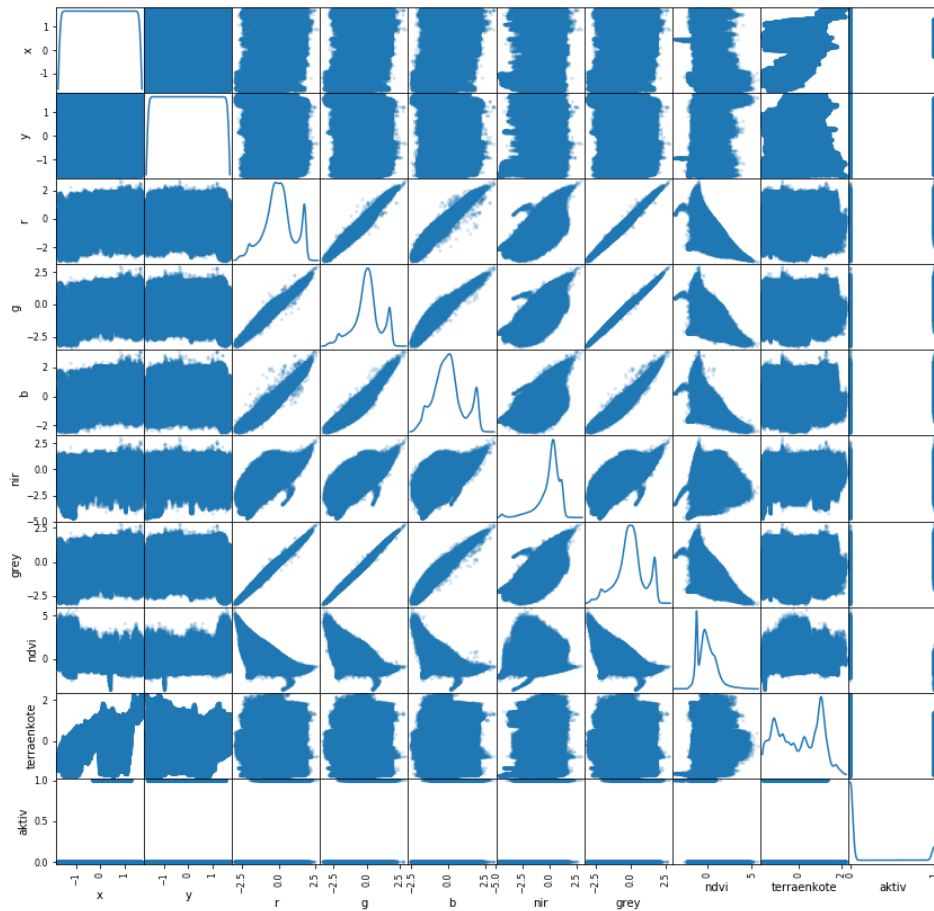
For at få et indtryk af, hvordan de enkelte features relaterer sig til den totale variation i datasættet, er der lavet en PCA-analyse. På figur 3.6 er den totale variation i datasættet plottet mod PC-akserne, og figuren viser at den første PC akse kan forklare omkring 60% af den totale variation, men ca. 90% af den totale variation i datasættet kan forklares med de første 3 PC akser.

På figur 3.7 er variationen for hhv. PC1, 2 og 3 for udvalgte features i datasættet plottet. Plottet kan vise om udvalgte features evt. bidrager mere eller mindre til en PC akse i datasættet, og hvilke features der evt. bidrager til samme PC. Plottet tegner nogenlunde samme billede som scatterplot matricen. r, g, b og gray ligger samlet i en klump og bidrager sammen med nir og ndvi til variationen i PC1, mens x, y og terræn primært bidrager til variationen i PC2. X, y, nir og ndvi bidrager desuden til variationen i PC3.

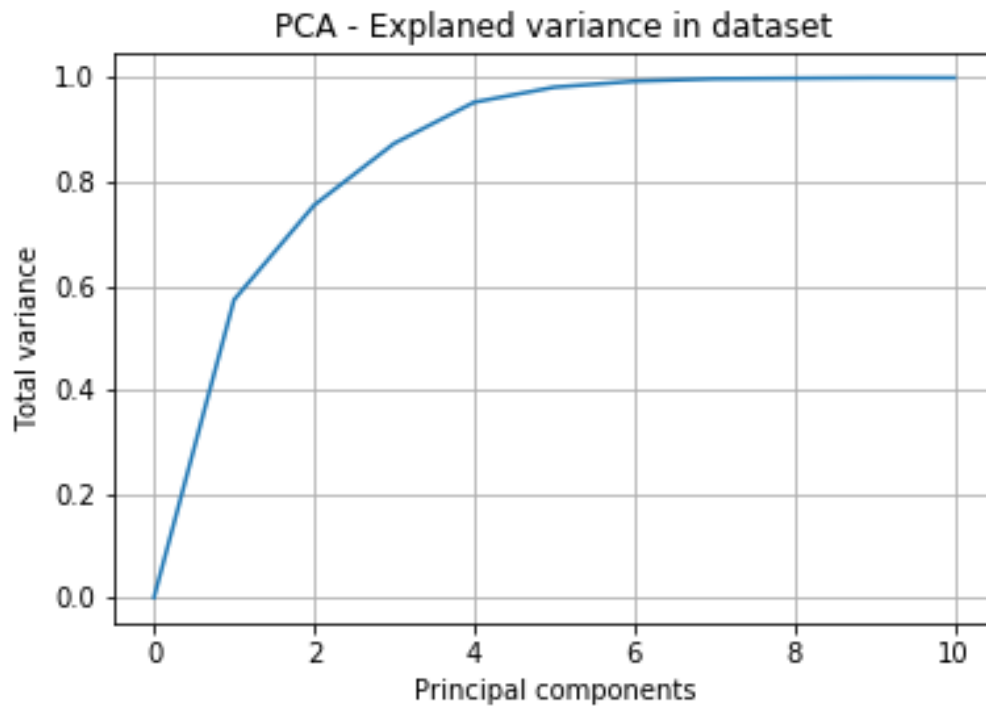
I begge plot ses "aktiv" at ligge tæt på 0,0, hvilket betyder at den samlede variation i datasættet ikke kan relateres til om en pixel ligger inden for eller uden for et aktivt graveområde.

3.3 Delkonklusion

Konklusionen på denne indledende eksplorativende dataanalyse er, at de isolerede pixel-værdier i datasættet ikke umiddelbart kan bruges til at identificere det aktive graveområde.



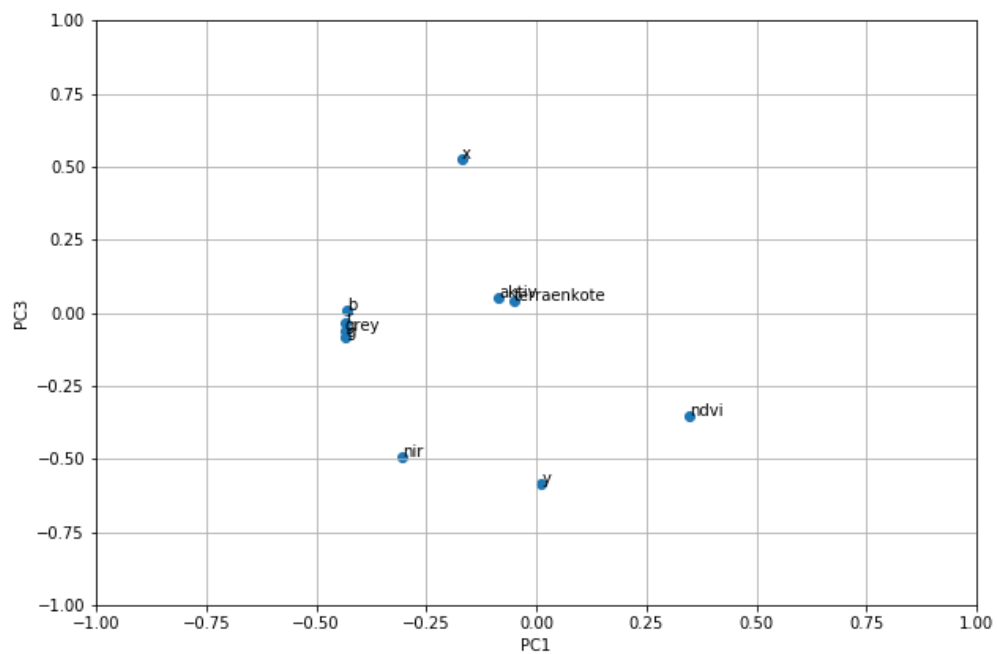
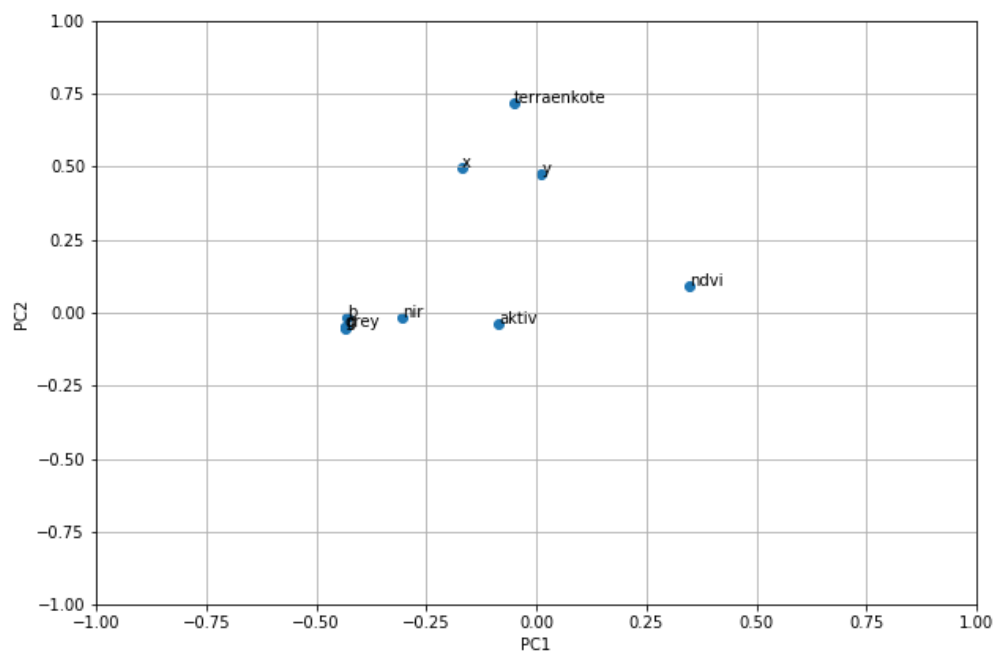
Figur 3.5: Scatterplot matrix



Figur 3.6: PCA - total variation i datasættet som funktion af PC akser

med. Dette kan virke som et dilemma, nå nu det er relativt enkelt visuelt at identificere det aktive graveområde i billedet. Forklaringen på dette er dog, at det menneskelige øje i højere grad opfatter sammenhængende klynger af pixels som billedeobjekter og formår at tolke disse ud fra klyngernes form og tekstur. Denne menneskelige præprocessering af billedets data mangler i denne pixelbaseret analyse.

På baggrund af ovenstående analyse vurderes det ikke at være formålstjeneligt at baserer de videre analyser på isolerede pixel-værdier, og I den resterende analyse vil jeg derfor fokuserer på algoritmer, der i højere grad retter sig mod objektbaserede analysemetoder.



Figur 3.7: PCA - features vist i rummet mellem PC1, PC2 og PC3

4. Teori: Neurale Netværk og Remote Sensing

4.1 Pixel-baserede metoder vs. objekt-baserede metoder

Traditionelle klassifikationsmetoder har med held været anvendt til pixelbaseret billedeklassifikation igennem mange år [8], og tidligere har disse pixelbaserede metoder også været anvendt meget inden for remote sensing. Årsagen til dette har været at tidligere tiders satellitfotos har haft mange kanaler, men en meget grov opløsning (100 m - 500 m), og udstrækningen af de geografiske objekter, som er forsøgt klassificeret, har typisk været i samme størrelsesorden som en pixel - f.eks. en mark. Dermed har der været en korrelation mellem farvekanalerne i billedets pixels og de objekter man har forsøgt at klassificere.

I dag er satellitbilleder og andre ortofoto så højpåkløselige (0,5 m - 10 m), at man kan identificere og dermed klassificere mindre objekter, men hvert objekt består ofte af et stort antal pixels. Som demonstreret i 3 giver dette en udfordring i forhold til pixel-baserede metoder, fordi kanalerne i de pixels, som objektet består af, typisk har en stor variation, og samtidig vil den "kanal-signatur", som man finder i pixels inden for et objekt, også optræde i stort antal uden for objektet.

Som følge af dette og som følge af at algoritmerne inden for billede- og objektklassifikation har udviklet sig væsentligt, anvendes basale pixel-baserede klassifikationsmetoder sjældent inden for remote sensing i dag, men istedet anvendes objektbaserede metoder. Fundamentet for denne udvikling er bl.a. udviklingen af hardware, processerings-kraft og mængden af tilgængelige data, som også har drevet den generelle udvikling inden for Machine Learning og dataanalyse.

Tre niveauer af billedeklassifikation

Man kan overordnet inddele billedeklassifikation i 3 kategorier - her eksemplificeret med billeder af katte:

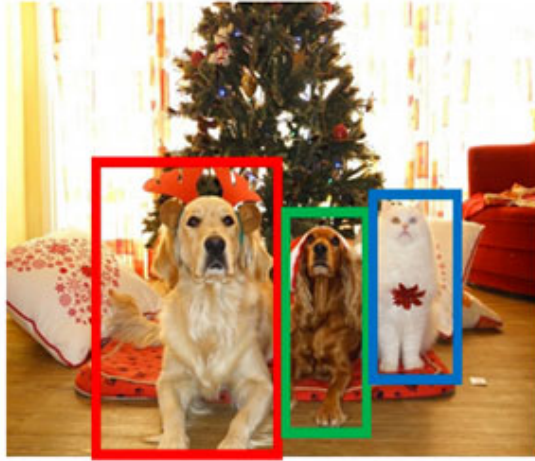
Basal billedeklassifikation: billedet klassificeres som "et billede af en kat".

Objekt detektion: Alle områder i billedet med katte findes og markeres.

Instance segmentering: Alle katte i billedet findes og omridset af hver kat afgrænses som et separat objekt (se figur 4.1)

Alle tre typer af klassificering kan være relevante, og bliver anvendt i forskellige algoritmer, men i forhold til at klassificere det aktive graveområde i en råstofgrav, vil det være *instance segmentering* der er relevant, idet det vil være det aktive graveområdes udbredelse, som er væsentligt for regionens tilsynsvirkning. Ved valg af klassificerings algoritme, er det

Object Detection



Instance Segmentation



Figur 4.1: Kilde: *cs231n.stanford.edu*

derfor vigtigt at være opmærksom på, hvilken type klassifikation den pågældende algoritme resulterer i.

Vellykket segmentering / afgrænsning af det aktive graveområde som objekt, vil være en forudsætning for, om en given model vil kunne optimeres tilstrækkeligt. Segmentering som disciplin falder inden for kategorien Computer Vision og ikke Machine Learning, og i kapitel 5 har jeg afprøvet nogle forskellige metoder og algoritmer på et af graveområderne, og vil også beskrive nogle af de udfordringer der er ved segmentering inden for Remote Sensing.

4.2 Neurale netværk

Neurale Netværk (NN) er en af de algoritme-komplekser, der inden for de sidste 10 år har fået enorm interesse inden for dataanalyse, herunder også inden for Remote Sensing. Den grundlæggende ide bag Neurale Netværk er at lade computeren imitere den måde den menneskelige hjerne lærer på. Psykologen Frank Rosenblatt beskrev i slutningen af 1950'erne den menneskelige perception, som en simpel matematisk model, hvor neuroner modtager binære input fra sanser eller andre neuroner, vægter disse input via synapsen, for derefter at evaluere summen af de vægtede input ud fra en fastlagt grænseværdi. Denne evaluering overføres til et output-lag af neuroner på binær form (0 og 1).

Dette princip med et koblede lag af neuroner, demonstrerede Rosenblatt at man kunne implementere som algoritme i elektriske kredsløb, og derved lade en maskine (computer) lære at kategoriserer simple figurer i et billede [50], [35]. I 1969 publicerede Marvin Minsky og Seymour Papert dog en skeptisk analyse, som dokumenterede nogle matematiske begrænsninger i Rosenblatt's algoritme (kendt som "*the XOR problem*") [37]. Minsky og Papert's skeptiske artikel indehold dog samtidig nogle grundlæggende ideer til, hvordan disse begrænsninger kunne overvindes, men artiklens kritiske vinkel gjorde alligevel, at interessen for Neurale Netværk forsvandt [28], [56], [35].

Der skulle gå ca. 20 år før der igen opstod fornyet interesse for Neurale Netværk, og ideerne fra Minsky og Papert's artikel blev videreudviklet. Den grundlæggende løsning var at implementerer såkaldte "hidden layers" som kobling mellem input- og output-lagene. Sammen med konceptet om "back-propagation" [51], er dette de bærende elementer, som alle moderne Neurale Netværks-algoritmer bygger på [28], [56], [35].

Neurale netværk og Remote Sensing

Neurale netværk har fra starten haft fokus på billede-klassifikation og parallelt med videreudviklingen af neurale netværk, er der også sket en voldsom udvikling af satellitteknologien og anvendelsen af satellitbilleder inden for Remote Sensing, og der har derfor hele tiden være stor interesse for mulighederne med Neurale Netværk inden for Remote Sensing [4].

Typen af Neurale Netværk der typisk anvendes inden for remote sensing kaldes *The feed-forward back-propagation multilayer perceptron*, og er typisk blevet anvendt på ortofotos til klassifikation af arealanvendelse (f.eks. marker, skove, søer m.m.). Desuden er identifikation af skydække på ortofotos også en klassisk problemstilling, som disse metoder bruges til .

En af fordelene ved Neurale Netværk er, at de ikke er baseret på en specifik statistisk eller matematisk model, og de tillader derfor, at klassifikationsmodeller kan baseres på et sæt data og efterfølgende udbygges med andre typer af data [4]. Muligheden for at anvende data fra flere forskellige kilder passer specielt godt til Remote Sensing, hvor billededata f.eks. kan kombineres med højdemodeller (DEM), for at give en mere detaljeret beskrivelse af et område, og dermed en mere kompleks klassifikationsmodel.

Convolutional Neural Network (CNN)

En gren inden for Neurale Netværk, der i de seneste år har fået stor opmærksomhed, er Convolutional Neural Network (CNN). I 2012 demonstrerede Krizhevsky et. al. [27] ved ImageNet's *Large Scale Visual Recognition Challenge* at de med et Convolutional Neural Network kunne klassificerer 1,3 mio billeder i 1000 klasser og opnå en en betydelig lavere valideringsfejl end der var opnået med andre algoritmer. De vandt selvfølgelig konkurrencen, og dette bidrog til at CNN algoritmerne blev den nye "state-of-the-art" inden for billede-klassifikation.

Kort fortalt underinddeles hele billeder i en lang række mindre udsnit eller "konvolutter" vha. såkaldte "filtre". Udsnittene fungerer som *feature extraction* fra billedet, men hvor den indbyrdes relation og variation mellem pixels bevares, og hvor udsnittene samles til et "*feature map*". Forskellige filtre kan trække forskellige typer af information ud af billedet f.eks. kanter. Alle genererede feature maps puljes i nye billeder med reducerede opløsning og processen gentages i næste neurale netværkslag. Tilsidst evalueres alle de genererede featuremaps i et *fully connected layer* og klassificeres ud fra vægte [26].

R-CNN er en videreudvikling af den basale CNN algoritme fra 2014, hvor R står for regioner - "*Regions with Convolutional Neural Networks*". R-CNN starter med at udfører en "selective search", og giver derefter forslag til hvilke regioner af billedet der skal lægges til grund for feature maps og klassifikation. Man opnår derved dels en optimering af algoritmen fordi feature maps ikke skal omfatte hele billedet og dels at resultatet bliver en objekt detektering i billedet (se 4.1) frem for bare en klassifikation af hele billedet [18], [43], [26].

R-CNN er efterfølgende blevet optimeret yderligere med algoritmerne *Fast R-CNN* i 2015 [17], og *Faster R-CNN* i 2017 [46]. Seneste skud på stammen er Mask R-CNN, der blev udviklet af FaceBook i 2017 [23], og som ud over objekt detektering også resulterer i Instans segmentering - dvs. algoritmen kan både lokalisere og afgrænse objekter i billedet og klassificere disse.

Forsøg med Mask R-CNN

Mask R-CNN er blevet implementeret i TensorFlow af firmaet Matterport [39] og er derved blevet tilgængelig for alle, der ønsker at afprøve denne algoritme. Algoritmen er med held blevet brugt inden for Remote Sensing til f.eks. at detektere bygninger og skibe samt til klassificering af sprækker i arktisk havis vha. satellit- og ortofotos [74], [75], [36], [76], og jeg vil i dette projekt forsøge at anvende algoritmen til klassificering af det aktive graveområde i råstofgrave (se 6).

5. Simpel billede segmentering med Scikit-image

Billedesegmentering er en væsentlig del af de fleste algoritmer, der bruges inden for billedanalyse, Computer Vision (CV), billede klassifikation, objekt detektering og instanssegmentering. Der findes mange forskellige algoritmer, der hyppigt anvendes inden for forskellige områder, og i dette kapitel har jeg med udgangspunkt i [54] og [72] afprøve nogle af disse. Til formålet har jeg valgt at anvende python-modulet Scikit-Image, som implementerer en lang række forskellige algoritmer (se bilag C).

5.1 Threshold segmentering

Threshold segmentering er en meget simpel, men også meget anvendt metode til regionsbaseret billedesegmentering. Det der kan identificeres som det aktive graveområde i et gråtone-ortofoto, synes netop at være defineret af lysere nuancer i et billede. Så som et muligt udgangspunkt for at segmentere billedet i objekter, testes derfor forskellige thresholds i gråtoneværdierne på ortofoto for et udvalgt testområde (se figur 5.1).

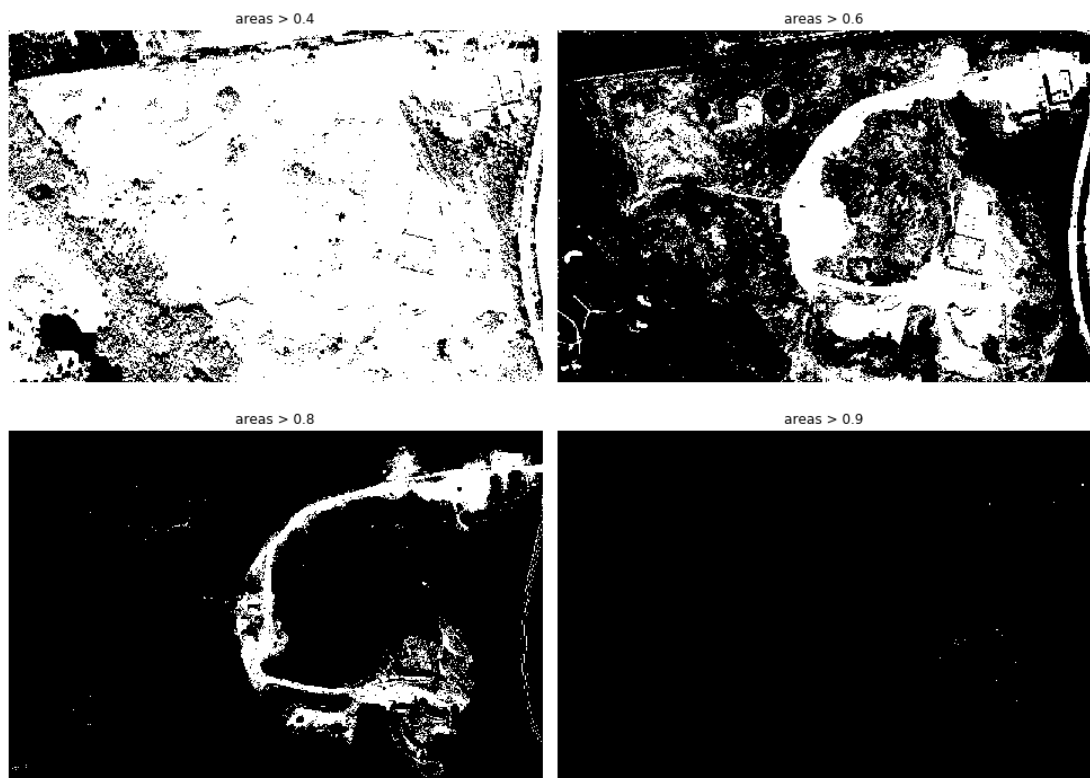
Figuren viser at den optimale threshold for dette område ligger i intervallet mellem 0,6 og 0,8. Den præcise værdi må dog forventes at varierer fra billede til billede, idet både lysforhold og jordtype vil influere på værdien. På figur 5.2 er der vist en "maske" baseret på pixelværdier under 0,7 og over 0,8 - vist med hhv. grøn og hvid. Sorte pixels i billedet er værdier mellem 0,7 og 0,8.

5.2 Watershed segmentering med udgangspunkt i thresholds

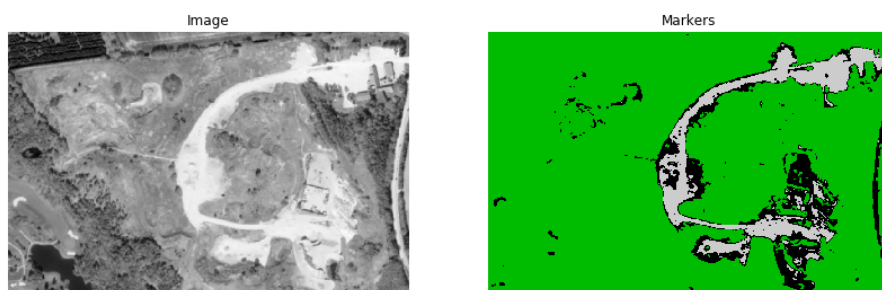
På baggrund af masken genereret i foregående afsnit, kan vi nu anvende watershed algoritmen til at isolere og lable separate segmenter i billedet. Dette er vist i figur 5.3. På figuren ses segmenterne som hvide områder, og det kan bemærkes at der ud over vores aktive graveområde også ses forskellige mindre områder inden og uden for graveområdet.

Hvis disse mindre områder betragtes som støj, kan de fjernes ved at bort-filtrerer områder (både "øer" og "søer"), der er mindre end en valgt grænse. På figur 5.4 er der anvendt en grænse på 500 pixels (svarende til 500 m²), og det rensede, segmenterede billede er vist som "Cleaned".

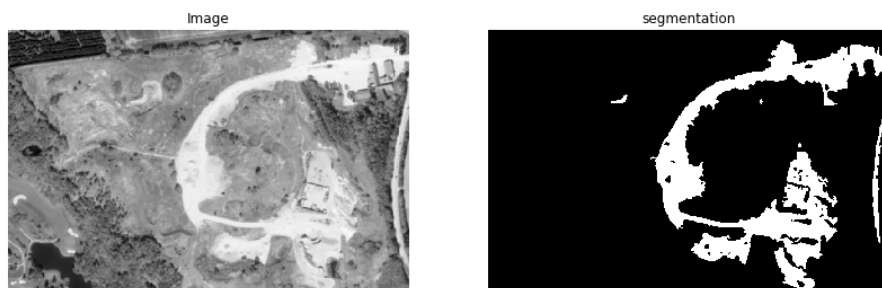
Endelig kan dette billede gemmes som en maske til senere anvendelse, eller objekterne kan eksporteres som geografiske objekter til videre bearbejdning.



Figur 5.1: Visualisering af forskellige thresholds i gråtoneværdier



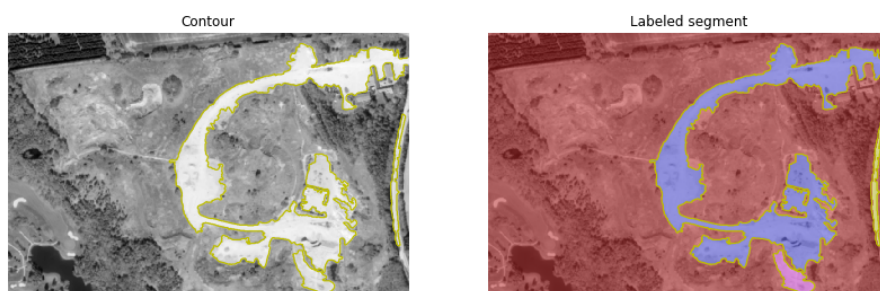
Figur 5.2: Anvendelse af threshold under 0,7 (grøn) og over 0,8 (hvid)



Figur 5.3: Segmentering vha. watershed algoritmen



Figur 5.4: Segmenterede billede renset for små objekter



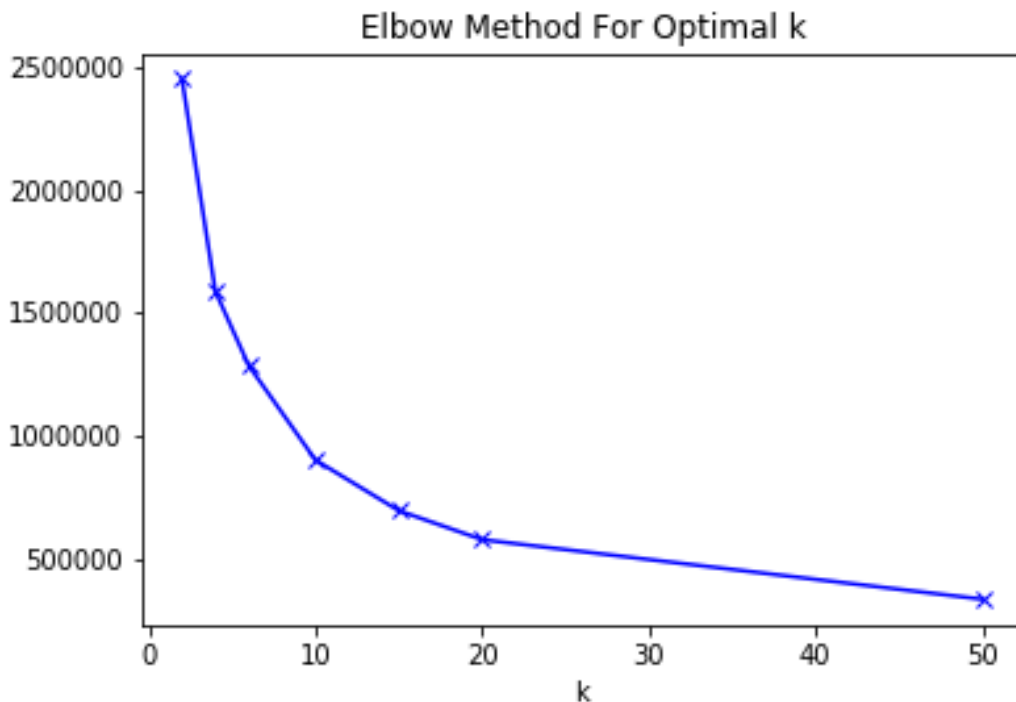
Figur 5.5: afgrænsning vist som polygon oven på ortofoto og med separate objekter farvelagt

5.3 Segmentering baseret på Clustering (K-means)

Der er gennemført en clusteranalyse med algoritmen K-Means på datasættet fra et af graveområderne. K-means kan hjælpe med at undersøge og evaluere naturlige grupperinger eller "klynger" i datasættet. Clusteranalysen er gennemført med hhv. 2, 6, 10, 20 og 50 klynger, og modellerne er evalueret med *Sum Of Squared Distances (SOSDist)* for at vurdere, antallet af naturlige grupper.

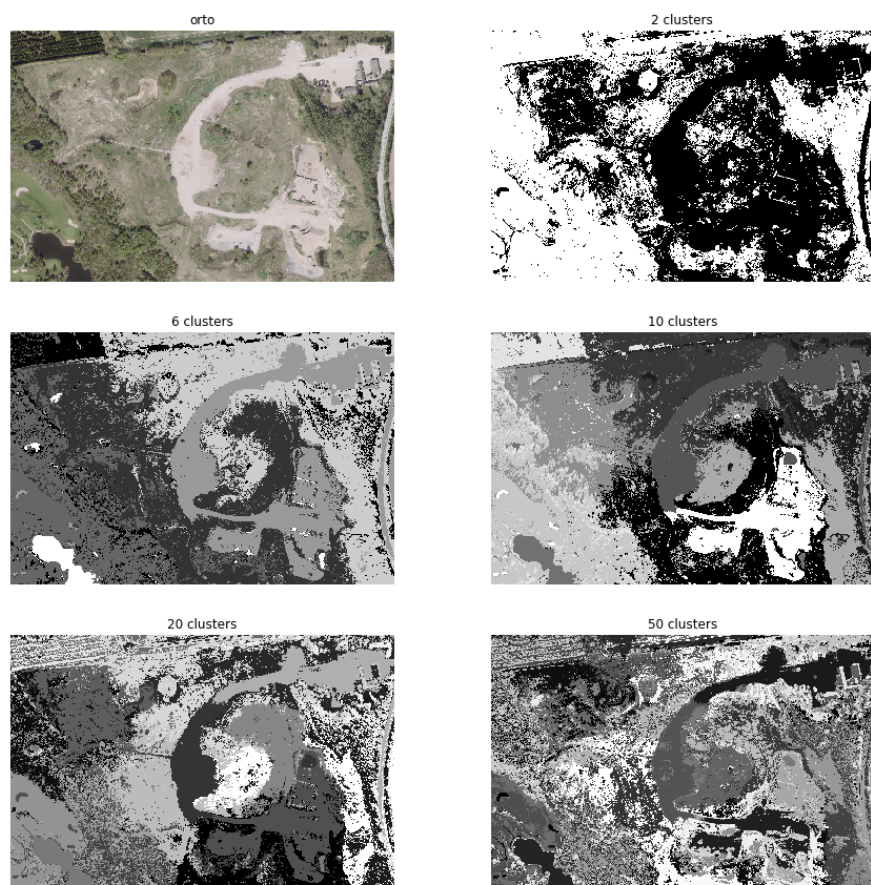
Figur 5.6 viser antallet af grupper plottet mod SOSDist, og kaldes for *the "Elbow Method For Optimal k"*. Plottet bruges normalt ved at finde "albuen" dvs. den k -værdi, hvor kurven knækker skarpt, og dette kan så bruges som udgangspunkt for antallet af naturlige grupper i datasættet.

På plottet i figur 5.6 ses der ikke noget skarpt knæk og generelt er SOSDist meget højt. Dette indikerer at der ikke findes et åbenlyst antal af naturlige grupper i datasættet, og at en analyse baseret på features i isolerede pixels ikke er en optimal tilgang til klassifikation af objekter i billeder



Figur 5.6: Evaluering af k -means som funktion af antal clusteres

På figur 5.7 har jeg vist et billede for hver af klyngerne 2, 6, 10, 20 og 50, hvor pixels er farver efter den klynge de falder inden for. På figuren er ortofotoet også vist til sammenligning. Figur 5.7 viser at graveområdet træder tydeligst frem ved 6 og 10 klynger, men at f.eks. vejen mod øst falder i samme klynge som graveområdet, og dermed ikke kan adskilles. Ved 10 klynger ses der en opdeling i graveområdet, hvor dele af graveområdet falder i separate klynger, hvilket kan skyldes at datasættet også indeholder højdedata. Clustering med K-Means er udført i en Jupyter-notebook, som er vist i bilag D.



Figur 5.7: Beregnede clusters visualiseret som billede

5.4 Delkonklusion på simpel segmentering

Det vurderes at ovenstående metoder virker til at separerer og isolere et sammenhængende aktivt graveområde i et billede. Metoderne er dog ikke tilstrækkelig robust til at kunne automatiseres, idet thresholds skal tilpasses det enkelte billedede, og der i nogen tilfælde også vil være behov for efterbehandling af objekterne. Threshold segmentering vurderes dog at kunne anvendes til at genererer masker, der efter en manuel bearbejdning evt. vil kunne bruges til træning af mere avancerede modeller.

Generelt er objekt segmentering en udfordring i Remote Sensing med højoypløselige billeder, fordi de geografiske objekter man ofte er interesserede i - f.eks. et graveområde - har meget varierende former og nødvendigvis træder tydeligt frem fra baggrunden, som objekter i andre typer af billeder. Ofte vil afgrænsningen af objekterne også være forstyrret af "støj" som f.eks. skygger fra træer og skrænter, og skygger vil have en helt anden pixel-signatur end objektet [36].

Den anvendte kode er vist i bilag C og bilag D. For læringens skyld har jeg i samme notesbog afprøvet andre segmenteringsmetoder, men disse har ikke vist sig at være lige så effektive. Dette kan handle om, at nogle af metoderne kræver en del konfiguration og tilpasning af parametre for at virke optimalt.

6. Forsøg med Mask R-CNN

6.1 Opsætning af modellen

Med udgangspunkt i demoprojektet ”Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow” fra Matterport Engineering Techblog [1], har jeg trænet en Mask R-CNN model til segmentering af aktive graveområder i råstofgrave. Udgangspunktet er ortofoto fra mine 22 AOI’er, som jeg hele tiden har arbejdet med, samt mine tolkningsmasker, som jeg har genereret ved manuelt at indtegne det formodede graveområde i et GIS-program. Maskerne er derfor gemt som Shape-filer.

Udgangspunktet for modelopsætningen var lidt anderledes en Matterport’s demoprojekt. Matterport havde flere billeder i deres datasæt, og havde opdelt billederne statistisk i et trænings- og et valideringssæt. Matterport havde ved annoteringen af masker opbygget en enkelt JSON fil, der - ud over alle de digitaliserede masker – også indeholdt referencer til hvert enkelt billede.

Mine data er alle sammen samlet i mappen *data/*, men i en undermappe for hvert graveområde og maskerne ligger opdelt sammen med hvert enkelt billede. For at kunne konfigurere og træne modellen, var jeg derfor nød til at ændre en del i metoderne *load_balloons* og *load_mask* i klassen *BalloonDatasæt*, som i min model er kommet til at hedde *Graveomr-Dataset* med metoderne *load_graveomr* og *load_mask*. Efter en del fejl-rettelser lykkedes det at få modellen til at køre.

Konfiguration af modellen

Modellen er konfigureret i filen *mrcnn.py* med følgende parametre:

- `IMAGES_PER_GPU = 1`
- `NUM_CLASSES = 1 + 1 # Graveområde + Ikke graveområde`
- `STEPS_PER_EPOCH = 100`
- `DETECTION_MIN_CONFIDENCE = 0.9`

... samt til at anvende vægte fra MS COCO datasættet, som udgangspunkt for modellen. Opsætning og konfiguration af model-klassen er vist i bilag E, mens bilag F er en Jupyter Notebook, der inspicerer at data er indlæst korrekt i modellen.

Transfere learning med MS COCO datasættet

Transfer learning, betyder, at man - i stedet for at træne sit neurale netværk fra bunden - lader modellen tage udgangspunkt i vægte fra en allerede trænet model, der er blevet

trænet på et stort billededatasæt, med en bred variation af forskellige objekter.

MS COCO datasættet indeholder billeder af mennesker, biler, hunde, katte og meget andet, men indeholder ikke billeder af råstofgrave. Grunden til at datasættet alligevel kan anvendes som udgangspunkt for en model, der skal klassificere råstofgrave er, at råstofgrave-objekterne kan "holdes op imod" alle de andre objekter i datasættet, som modellen allerede har lært at genkende. Herved bliver det lettere for modellen at differentiere råstofgrave fra andre typer af objekter og etablerer en vægt for råstofgravene.

Man "*overfører*" således den læring, der er opnået i andre modeller (deraf navnet Transfer learning) og dette bidrager til en væsentlig hurtigere og mere præcis træning af modellen. MS COCO datasættet er open source og er beskrevet i [32].

6.2 Træning af modellen

Jeg har trænet modellen på min linux-maskine, der har en intel i7 cpu med 6 kerner og 12 tråde samt 64 Gb RAM, og med Ubuntu 18 som styresystem. Indledningsvist satte jeg træningen af modellen i gang uden at have et grafikkort (GPU) i maskinen. Efter at have ladet træningen køre i ca. 1½ døgn, kunne jeg konstatere, at der kun var blevet gennemført 6 ud af 30 iterationer, hvorefter jeg besluttede at afslutte processen.

På deres wiki-side skriver Matterport, at mask R-CNN er designet til præcision og ikke hukommelseoptimering, og at det ikke er en "letvægts-model", og det kan jeg nu konstatere er rigtigt! Modellen er som udgangspunkt sat op til at køre TensorFlow's multiprocessor mode, og som det fremgår af figur 6.1, er alle kerner / tråde i maskinen på overarbejde.

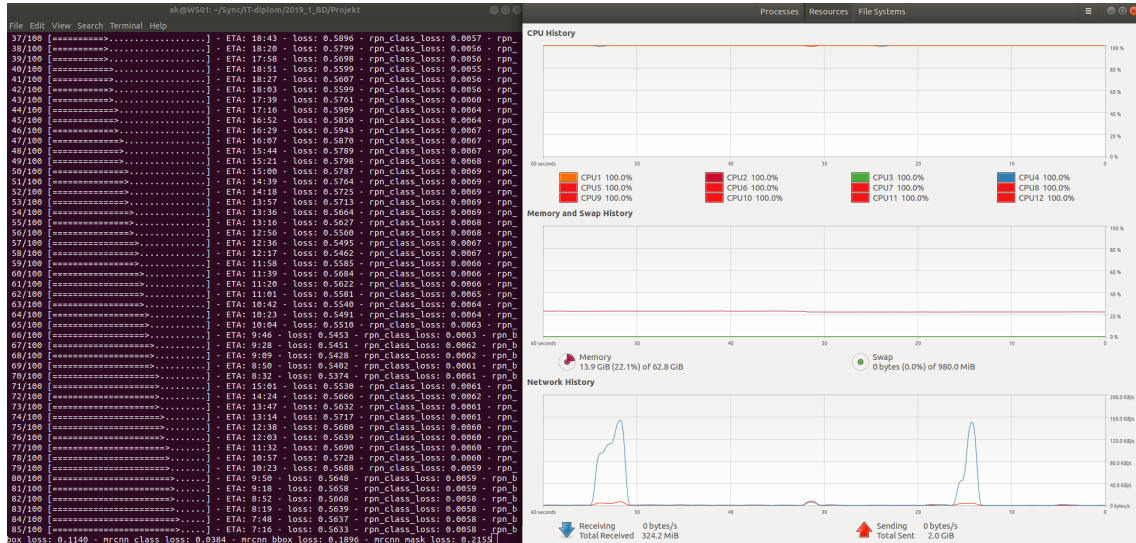
Mange studier har vist, at tunge beregninger kan køre væsentligt hurtigere på en GPU end på en CPU og Matterport anbefaler at bruge en GPU på mindst 12 Gb til træning af modellen, men at modellen kan køres på mindre h.v. konfiguration. Jeg valgte derfor at sætte et højtydende NVIDIA grafikkort med 6 Gb i min maskine, og så forsøge at træne modellen igen.

Med en GPU i maskinen kunne jeg efter et døgn konstatere, at der var blevet udført 8 ud af 30 gennemkørsler. Jeg besluttede at forsøge at lade træningen køre færdig med dette setup, og træningen af modellen var gennemført efter ca. 4 døgn.

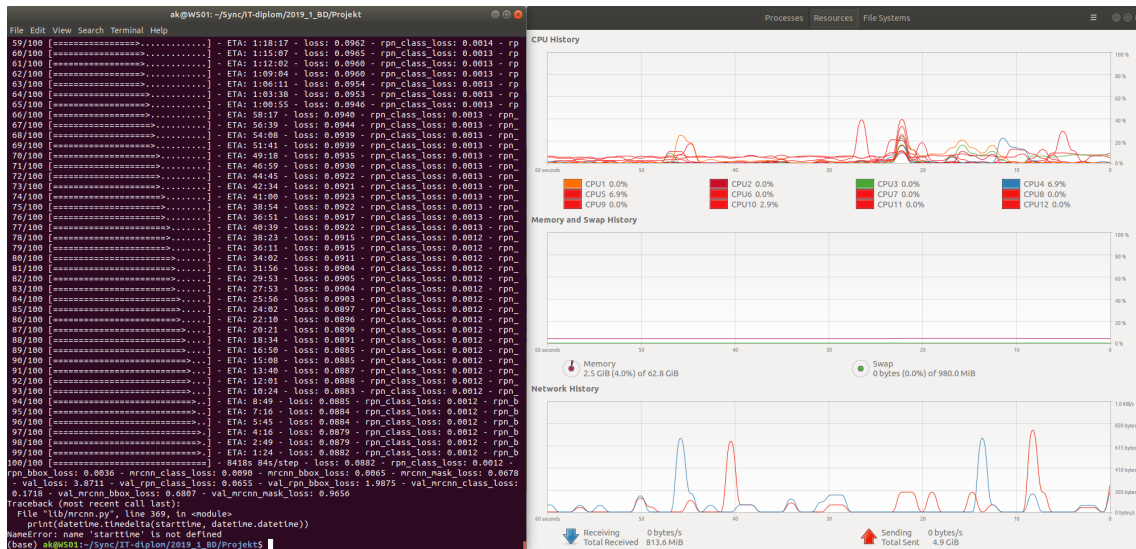
6.3 Modevaluering

For at evaluere modellen, er der i bilag G lavet et plot, der viser forholdet mellem precision og recall (se fig. 6.3), hvor recall er plottet på x-aksen og precision er plottet på y-aksen. *Precision* er et mål for, hvor stor del af det modellerede aktive graveområde ligger inden for det faktuelle aktive graveområde, mens *Recall* viser, hvor stor del af det faktuelle aktive graveområde der bliver modelleret som aktivt graveområde.

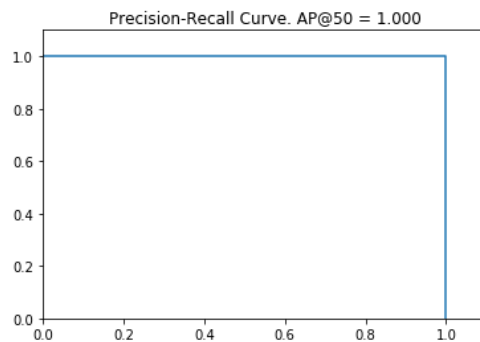
Figuren viser at precision er 1 for alle Recall-værdier og at en Recall-værdi på 1 også giver en precision på 1, hvilket ikke vurderes at være korrekt! Ud fra fig. 6.5 er der forskelle i det faktuelle område og det modellerede område. Nogen steder ligger det modellerede område uden for det faktuelle område (vist med rød på fig. 6.5), så Precision-værdien burde derfor falde til under 1. Der er desuden både falsk positive og falsk negative.



Figur 6.1: Første forsøg på træning uden GPU. Bemærk alle CPU'ens 12 tråde er fuldt udnyttet



Figur 6.2: Træning af modellen med GPU blev gennemført på ca. 4 døgn



Figur 6.3: Plot af Precision og Recall

Test af objekt segmentering med modellen

For visuelt at se, hvordan modellen performer, har jeg prøvet at anvende modellen på et af billederne, og visualiseret det område, som modellen klassificerer som "*Aktivt graveområde*" (se figur 6.4).

Billedet - sammenlignet med figur 6.5 viser, at det klassificerede område i overvejende grad ligger inden for det, der visuelt kan opfattes som det aktive graveområde. Afgrænsningen af området er ikke specielt nøjagtig - hjørner er f.eks. afrundede og området går generelt ikke helt ud til kanten af råstofgraven. Et enkelt sted synes det klassificerede område at gå lidt ud over kanten. Dog må det konstateres, at det område som modellen har afgrænset som "*aktivt*", ligger ret tæt på det, som man visuelt kan opfatte som aktivt.

Det billede som modellen her er testet på, er det billede fra samme datasæt, som modellen er trænet på. For at få et retvisende billede af modellens evne til at finde aktive graveområder, bør testen laves på et billede, der ikke har indgået i hverken trænings- eller valideringsdatasættet. Jupyter Notebook med test og evaluering af modellen er vist i bilag G.



Figur 6.4: Test af detektion af aktivt graveområde med modellen



Figur 6.5: Forskellen på modelleret og faktisk graveområde

7. Distribueret modelkørsel

Som demonstreret i kapitel 6, er deep learning algoritmer ofte meget ressourcetunge og træningen af modeller kan tage dage uden de nødvendige ressourcer. Dette er uheldigt, fordi en central del af arbejdet med Machine Learning er at teste forskellige konfigurationer af en model, for at få den til at performe optimalt. Hvis man skal vente flere dage på resultater, for at justere parametrene yderligere og evt. ende med en fejl, bliver arbejdet træt og ikke særligt operationelt.

Dette er en typisk *Big Data* problemstilling, og burde kunne løses ved udskalering dvs. distribueret træning af modellen i et kluster. Da Neurale Netværk som udgangspunkt afvikles med parallel processing, kan det undre at der primært har været fokus på at optimere deep learning modeller med opskalering dvs. - dvs. sætte flere og større grafikkort i maskinen.

Forklaringen er, at GPU-processing er ekstremt effektive til matrice-beregninger, som er det som deep learning algoritmer er baseret på. Udviklingen af GPU'er i spilleindustrien har gjort, at selv meget store og tunge deep learning algoritmer kan køre lige så effektivt på en håndfuld kraftige GPU'er, som på et stort antal distribueret klustre af CPU-baserede maskiner, og med en lavere samlet omkostning [11].

7.1 Distribueret GPU-processing

Selv om der i de seneste år har været lavet en række forsøg og demonstrationer med at køre deep learning netværk som distribuerede GPU beregninger, er det ikke noget der i dag kan gøres out-of-the-box. Først med Hadoop version 3 er det blevet muligt at konfigurere YARN med GPU-ressourcer, og en forudsætning for dette er, at GPU-ressourcerne findes i hardwaren og at GPU-typen understøttes. Apache Spark nuværende version (2.4.3) understøtter som udgangspunkt ikke GPU-processing, men dette kan opnås med 3-parts distributioner fra f.eks. Databricks.

I dag tilbyder de fleste cloud-udbydere muligheden for at sætte GPU-understøttede virtuelle maskiner op, og dermed er GPU-klustre en reel mulighed. Men næste udfordring er, at afvikle GPU-understøttede kodebiblioteker f.eks. TensorFlow over Hadoop eller Spark. Typisk kræver distribuerede beregninger specielle kodebiblioteker, og at den algoritme man ønsker at anvende, er implementeret i disse specialbiblioteker. SparkFlow er et eksempel på sådan et kodebibliotek fra 2018, der muliggør at køre TensorFlow i Spark. Jeg har i forbindelse med dette projekt ikke kunne finde eksempler på, at Mask R-CNN modeller er blevet kørt med SparkFlow.

Hadoop-versionen på kursets BigData-server er version 3, og understøtter dermed GPU-processing, men serverens grafikkort er af typen *Matrox MGA G200EH* (se figur 7.1). TensorFlow kræver et NVIDIA grafikkort med CUDA 3.5 eller højere installeret og det

har derfor ikke givet mening at forsøge at arbejde distribueret med Neurale Netværk på kursets bigdata-server.

[illegible]

Figur 7.1: Test af miljø på `bigdata.javapoq.dk`

7.2 I den nære fremtid?

Samlet set er distribueret GPU-proccesering vha. Spark og/eller Hadoop et nyt område, hvor der i øjeblikket sker en omfattende teknologisk udvikling. Teknologien er stadig ikke særligt moden og de projekter jeg har kunne finde på netop omkring distribueret processe- ring af GPU-accelererede deep learning, har karakter af forsknings- og pilot-projekter.

Der går dog rygter om, at Apache Spark version 3 kommer til at understøtte GPU-accelererede beregninger via *Accelerator-aware task scheduling for Spark (SPARK-24615 på Spark JIRA)* og få bedre integration til eksterne deep learning frameworks som f.eks. TensorFlow via *Standardize Optimized Data Exchange between Spark and DL/AI frameworks (SPARK-24579 på Spark JIRA)* [60].

Og mens Spark allerede nu kan bruge kubernetes som kluster manager, giver projekter som KubeFlow [60] mulighed for at bl.a. køre GPU accelererede deep learning modeller distribueret vha. containerteknologi. Samlet set må det forventes at distribuerede GPU-accelererede beregninger vil blive væsentligt lettere tilgængelige i den nærmeste fremtid.

8. Konklusion og perspektivering

Jeg har i dette projekt arbejdet med en klassifikationsmodel, til at detekterer og afgrænse det aktive graveområde i en råstofgrav. Datagrundlaget for projektet har været frie offentlige data fra kortforsyningen og en indledende dataanalyse har vist, at pixelbaserede klassifikationsmetoder ikke kan bruges på højopløselige ortofoto, men at der i stedet skal anvendes objektbaserede klassifikationsmetoder.

I dag er stort set al billede-klassifikation baseret på Neurale Netværk, og i dette projekt har jeg opsat og trænet en model baseret på Mask R-CNN algoritmen. Neurale Netværk kræver mange computer-ressourcer, men det er lykkedes at træne modellen ved at installere en højtydende GPU på min PC, og køre træningen via denne.

Modellen er testet og kan i et vist omfang klassificerer det aktive graveområde i udvalgte billeder. Der er stor dog forskel på, hvor godt modellen detekterer det aktive graveområde i det enkelte billede. For at kunne anvende modellen som en effektiv støtte i tilsynsopgaven med råstofgrave, kræver det derfor yderligere arbejde i forhold til at forbedre modellens performance.

Det vurderes, at følgende tiltag vil kunne forbedre modellen:

- Et større datasæt, med større variation i graveområderne. Der er i dette projekt kun arbejdet med 22 billeder.
- At der inddrages flere parametre i modellen. I dette projekt er modellen kun baseret på RGB-værdierne i ortofoto, men hvis flere bånd kunne inddrages f.eks. nær-infrarød, vurderes dette at kunne forbedre performance.
- Flere GPU-ressourcer. I dette projekt er træningen kun baseret på et enkelt gennemløb, som tog omkring 4 døgn. Hvis der blev gennemført flere gennemløb, baseret på dynamiske split-sampels af datasættet i trænings- og validerings-datasæt, vurderes dette også at kunne forbedre modellen performance. Dette vil dog kræve flere GPU-ressourcer for at være operationelt.

Endelig mangler der at blive opbygget en applikation, der kan stille modellens tolkninger til rådighed over for slutbrugeren, samt samle feed back ind fra brugeren, sådan at dette kan anvendes til *Continous Learning*, og fortsat forbedring modellens performance.

Perspektivering

Samlet set vurderes projektet at være lykkedes. I Machine Learning og Big Data ligger der et stort potentiale for moderne datadrevne virksomheder, og værktøjer til at arbejde med disse teknologier bliver mere og mere tilgængelige.

I dette projekt har jeg arbejdet med Python, som ud over at være en fortræffelig platform til dataprocessering og -analyse, også er et generisk programmeringssprog, med en kæmpe

portefølge af moduler og frameworks til alt mellem himmel og jord. Python vil således kunne bruges til at bygge hele pipeline fra den indledende dataanalyse til applikationen, der stiller data-produktet til rådighed for slutbrugeren.

Vi har i min afdeling netop fået adgang til Azure, som er Microsofts Cloud Miljø. Her er alle de Big Data teknologier, som er blevet introduceret på kurset tilgængelige, sammen med mange andre teknologier. Der er bl.a. mulighed for at oprette virtuelle maskiner, konfigureret med væsentligt flere GPU-ressourcer end jeg har brugt i dette projekt. Jeg havde et ønske om at forsøge at træne mit Neurale Netværk på en virtuel maskine i Azure i dette projekt, men der har der ikke været tid.

Samlet set har kurset giver mig et godt fundament til at kunne komme videre med Cloud Computing og Big Data teknologier, og jeg forventer at kunne arbejde videre med noget af dette i mit afgangsprøveprojekt til efteråret.

Litteratur

- [1] Waleed Abdulla. Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow, 2018.
- [2] S M Abdullah, Peter Tischer, Sudanthi Wijewickrema, and Andrew Paplinski. Parameter-free hierarchical image segmentation. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, dec 2017.
- [3] Anonymous. Automatic image segmentation. *Official Gazette of the United States Patent and Trademark Office Patents*, 2018.
- [4] P. M. Atkinson and A. R.L. Tatnall. Introduction neural networks in remote sensing. *International Journal of Remote Sensing*, 18(4):699–709, mar 1997.
- [5] P. M. Atkinson and A. R.L. Tatnall. Introduction neural networks in remote sensing. *International Journal of Remote Sensing*, 18(4):699–709, mar 1997.
- [6] Alando Ballantyne. Remote Sensing: An Overview of Common Pixel Classification Techniques. [\url{https://alan.do/remote-sensing-an-overview-of-common-pixel-classification-techniques-4827c4c3a9ff}](https://alan.do/remote-sensing-an-overview-of-common-pixel-classification-techniques-4827c4c3a9ff), 2018.
- [7] Simon Brooker. Spatial epidemiology of human schistosomiasis in Africa: risk models, transmission dynamics and control. *Transactions of the Royal Society of Tropical Medicine and Hygiene*, 101(1):1–8, jan 2007.
- [8] James B. Campbell. *Introduction to Remote Sensing*. Taylor & Francis, second edition, 2006.
- [9] Yang-lang Chang, Amare Anagaw, Lena Chang, Yi Chun Chun Wang, Chih-Yu Hsiao, Wei-hong Lee, Yang-lang Chang, Amare Anagaw, Lena Chang, Yi Chun Chun Wang, Chih-Yu Hsiao, and Wei-hong Lee. Ship Detection Based on YOLOv2 for SAR Imagery. *Remote Sensing*, 11(7):786, apr 2019.
- [10] Fen Chen, Ruilong Ren, Tim Van de Voorde, Wenbo Xu, Guiyun Zhou, Yan Zhou, Fen Chen, Ruilong Ren, Tim Van de Voorde, Wenbo Xu, Guiyun Zhou, and Yan Zhou. Fast Automatic Airport Detection in Remote Sensing Images Using Convolutional Neural Networks. *Remote Sensing*, 10(3):443, mar 2018.
- [11] Chris Davies. NVIDIA GPU neural network makes Google’s cat-spotter look dumb - SlashGear, 2013.
- [12] Zhi Ding, Xiaohan Liao, Fenzhen Su, and Dongjie Fu. Mining coastal land use sequential pattern and its land use associations based on association rule mining. *Remote Sensing*, 9(2):1–18, 2017.
- [13] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. mar 2016.

- [14] Fubiao Feng, Wei Li, Qian Du, and Bing Zhang. Dimensionality reduction of hyper-spectral image with graph-based discriminant analysis considering spectral similarity. *Remote Sensing*, 9(4), 2017.
- [15] Giles M. Foody, Feng Ling, Doreen S. Boyd, Xiaodong Li, and Jessica Wardlaw. Earth observation and machine learning to meet Sustainable Development Goal 8.7: Mapping sites associated with slavery from space. *Remote Sensing*, 11(3):266, jan 2019.
- [16] Gabriel Garza. Mask R-CNN for Ship Detection & Segmentation, 2019.
- [17] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 Inter, pages 1440–1448, apr 2015.
- [18] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, nov 2014.
- [19] GISGeography. What is NDVI (Normalized Difference Vegetation Index)? - GIS Geography, 2018.
- [20] Xiaobing Han, Yanfei Zhong, Liqin Cao, and Liangpei Zhang. Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification. *Remote Sensing*, 9(8), 2017.
- [21] Xiaobing Han, Yanfei Zhong, and Liangpei Zhang. An efficient and robust integrated geospatial object detection framework for high spatial resolution remote sensing imagery. *Remote Sensing*, 9(7), 2017.
- [22] Xiaobing Han, Yanfei Zhong, Liangpei Zhang, Xiaobing Han, Yanfei Zhong, and Liangpei Zhang. An Efficient and Robust Integrated Geospatial Object Detection Framework for High Spatial Resolution Remote Sensing Imagery. *Remote Sensing*, 9(7):666, jun 2017.
- [23] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 2980–2988, mar 2017.
- [24] Xiaohui Huang, Lizhe Wang, Jining Yan, Ze Deng, Shaoyuan Wang, and Yan Ma. Towards Building a Distributed Data Management Architecture to Integrate Multi-Sources Remote Sensing Big Data. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 83–90. IEEE, jun 2018.
- [25] Robert Kabacoff. *R in action : data analysis and graphics with R*. Manning, second edition.
- [26] Ujjwal Karn. An Intuitive Explanation of Convolutional Neural Networks, 2016.
- [27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks, 2012.
- [28] Andrey Kurenkov. A 'Brief' History of Neural Nets and Deep Learning, 2015.
- [29] Noam Levin, Saleem Ali, David Crandall, and Salit Kark. World Heritage in danger: Big data and remote sensing can help protect sites in conflict zones. *Global Environmental Change*, 55:97–104, mar 2019.

- [30] Chuan Li. Transfer learning with Mask R-CNN, 2018.
- [31] Feimo Li, Shuxiao Li, Chengfei Zhu, Xiaosong Lan, Hongxing Chang, Feimo Li, Shuxiao Li, Chengfei Zhu, Xiaosong Lan, and Hongxing Chang. Cost-Effective Class-Imbalance Aware CNN for Vehicle Localization and Categorization in High Resolution Aerial Images. *Remote Sensing*, 9(5):494, may 2017.
- [32] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. may 2014.
- [33] Zhu LingLing, Cao FuBin, and Li Jie. Remote sensing image segmentation based on Hadoop cloud platform. In Guohai Situ, Liquan Dong, Xun Cao, and Wolfgang Osten, editors, *2017 International Conference on Optical Instruments and Technology: Optoelectronic Imaging/Spectroscopy and Signal Processing Technology*, volume 10620, page 7. SPIE, jan 2018.
- [34] Peng Liu, Liping Di, Qian Du, and Lizhe Wang. Remote Sensing Big Data: Theory, Methods and Applications. *Remote Sensing*, 10(5):711, 2018.
- [35] Jean-Christophe B. Loiseau. Rosenblatt’s perceptron, the very first neural network, 2019.
- [36] Tingting Lu, Dongping Ming, Xiangguo Lin, Zhaoli Hong, Xueding Bai, and Ju Fang. Detecting Building Edges from High Spatial Resolution Remote Sensing Imagery Using Richer Convolution Features Network. *Remote Sensing*, 10(9):1496, sep 2018.
- [37] M. Minsky and Seymour Papert. Perceptrons, Expanded Edition An Introduction to Computational Geometry. *MIT Press*, page 292, 1969.
- [38] Ernestina Martel, Raquel Lazcano, José López, Daniel Madroñal, Rubén Salvador, Sebastián López, Eduardo Juárez, Raúl Guerra, César Sanz, and Roberto Sarmiento. Implementation of the Principal Component Analysis onto high-performance computer facilities for hyperspectral dimensionality reduction: Results and comparisons. *Remote Sensing*, 10(6), 2018.
- [39] Matterport Inc. Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow, 2018.
- [40] Jakub Nalepa, Michal Myller, and Michal Kawulok. Validating Hyperspectral Image Segmentation. *IEEE Geoscience and Remote Sensing Letters*, pages 1–5, 2019.
- [41] Shanlan Nie, Zhiguo Jiang, Haopeng Zhang, Bowen Cai, and Yuan Yao. Inshore ship detection based on mask r-cnn. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, volume 2018-July, pages 693–696. IEEE, jul 2018.
- [42] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza. A new deep convolutional neural network for fast hyperspectral image classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 145:120–147, nov 2018.
- [43] Dhruv Parthasarathy. A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN, 2017.
- [44] Kunlun Qi, Chao Yang, Qingfeng Guan, Huayi Wu, and Jianya Gong. A multiscale deeply described correlations-based model for land-use scene classification. *Remote Sensing*, 9(9):19–22, 2017.

- [45] Kunlun Qi, Chao Yang, Qingfeng Guan, Huayi Wu, Jianya Gong, Kunlun Qi, Chao Yang, Qingfeng Guan, Huayi Wu, and Jianya Gong. A Multiscale Deeply Described Correlatons-Based Model for Land-Use Scene Classification. *Remote Sensing*, 9(9):917, sep 2017.
- [46] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, jun 2017.
- [47] Yun Ren, Changren Zhu, and Shunping Xiao. Deformable Faster R-CNN with Aggregating Multi-Layer Features for Partially Occluded Object Detection in Optical Remote Sensing Images. *Remote Sensing*, 10(9):1470, sep 2018.
- [48] V.F. Rodríguez-Galiano, F. Abarca-Hernández, B. Ghimire, M. Chica-Olmo, P.M. Atkinson, and C. Jeganathan. Incorporating Spatial Variability Measures in Land-cover Classification using Random Forest. *Procedia Environmental Sciences*, 3:44–49, 2011.
- [49] V.F. Rodríguez-Galiano, E. Pardo-Igúzquiza, M. Chica-Olmo, and J.P. Rigol-Sánchez. Increasing the spatial resolution of thermal infrared images using cokriging. *Procedia Environmental Sciences*, 3:117–122, 2011.
- [50] Frank Rosenblatt. THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN. *Psychological Review*, 65(6):1–23, 1958.
- [51] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986.
- [52] David Schweitzer and Rajeev Agrawal. Multi-Class Object Detection from Aerial Images Using Mask R-CNN. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, pages 3470–3477. IEEE, dec 2019.
- [53] Katherine Scott. Python from Space: Analyzing Open Satellite Imagery Using the Python Ecosystem. <https://github.com/kscottz/PythonFromSpace>, 2017.
- [54] Himanshu Singh. *Practical Machine Learning and Image Processing*. Apress, Berkeley, CA, 2019.
- [55] Yuheng Song and Hao Yan. Image Segmentation Techniques Overview. In *2017 Asia Modelling Symposium (AMS)*, pages 103–107. IEEE, dec 2017.
- [56] John Aasted Sørensen. Lecture note: The Time Gap in Neural Network Research. 2019.
- [57] Suraj Srinivas, Ravi Kiran Sarvadevabhatla, Konda Reddy Mopuri, Nikita Prabhu, Srinivas S. S. Kruthiventi, and R. Venkatesh Babu. A Taxonomy of Deep Convolutional Neural Nets for Computer Vision. *Frontiers in Robotics and AI*, 2, jan 2016.
- [58] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearsons, first edit edition, 2014.
- [59] Xicheng Tan, Song Guo, Liping Di, Meixia Deng, Fang Huang, Xinyue Ye, Ziheng Sun, Weishu Gong, Zongyao Sha, and Shaoming Pan. Parallel Agent-as-a-Service (P-AaaS) based geospatial service in the cloud. *Remote Sensing*, 9(4):1–16, 2017.
- [60] The Kubeflow Authors. Kubeflow, 2019.

- [61] Adriana Tufa, Ionut Boicu, Ion-Dorinel Filip, Catalin Negru, and Florin Pop. Remote Sensing Computing Model for Forest Monitoring in Cloud. In *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 138–144. IEEE, oct 2018.
- [62] Mark Watson. Keeping Your Machine Learning Models Up-To-Date – IBM CODAIT – Medium, 2018.
- [63] Erik Westra. *Python Geospatial Development*. Packt Publishing, third edit edition, 2016.
- [64] Bingfang Wu, Miao Zhang, Nana Yan, Qiang Xing, Weiwei Zhu, and Xin Zhang. Cloud Based Cropwatchglobal Remote Sensing Monitoring Online System. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 8181–8182. IEEE, jul 2018.
- [65] Ke Wu, Qian Du, Yi Wang, and Yetao Yang. Supervised sub-pixel mapping for change detection from remotely sensed images with different resolutions. *Remote Sensing*, 9(3), 2017.
- [66] Ke Wu, Qian Du, Yi Wang, Yetao Yang, Ke Wu, Qian Du, Yi Wang, and Yetao Yang. Supervised Sub-Pixel Mapping for Change Detection from Remotely Sensed Images with Different Resolutions. *Remote Sensing*, 9(3):284, mar 2017.
- [67] Reynold Xin. Keynote: Announcing Koalas Open Source Project.
- [68] Fang Xu, Jinghong Liu, Mingchao Sun, Dongdong Zeng, and Xuan Wang. A hierarchical maritime target detection method for optical remote sensing imagery. *Remote Sensing*, 9(3):1–23, 2017.
- [69] Jiangqiao Yan, Hongqi Wang, Menglong Yan, Wenhui Diao, Xian Sun, Hao Li, Jiangqiao Yan, Hongqi Wang, Menglong Yan, Wenhui Diao, Xian Sun, and Hao Li. IoU-Adaptive Deformable R-CNN: Make Full Use of IoU for Multi-Class Object Detection in Remote Sensing Imagery. *Remote Sensing*, 11(3):286, feb 2019.
- [70] Lingbo Yang, Lamin R. Mansaray, Jingfeng Huang, and Limin Wang. Optimal segmentation scale parameter, feature subset and classification algorithm for geographic object-based crop recognition using multisource satellite imagery. *Remote Sensing*, 11(5):514, mar 2019.
- [71] Chengxiao Yu, Xin Wang, Junqiu Wang, and Hongbin Zha. Uncertainty-aware RGBD image segmentation. In *2017 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 97–102. IEEE, oct 2017.
- [72] Song Yuheng and Yan Hao. Image Segmentation Algorithms Overview. jul 2017.
- [73] Ling Zhang, Wei You, Q. M. Jonathan Wu, Shengbo Qi, and Yonggang Ji. Deep learning-based automatic clutter/Interference detection for HFSWR. *Remote Sensing*, 10(10):1517, sep 2018.
- [74] Shaoming Zhang, Ruize Wu, Kunyuan Xu, Jianmei Wang, and Weiwei Sun. R-CNN-Based Ship Detection from High Resolution Remote Sensing Imagery. *Remote Sensing*, 11(6):631, mar 2019.
- [75] Weixing Zhang, Chandi Witharana, Anna K. Liljedahl, Mikhail Kanevskiy, Weixing Zhang, Chandi Witharana, Anna K. Liljedahl, and Mikhail Kanevskiy. Deep convolutional neural networks for automated characterization of arctic ice-wedge polygons in very high spatial resolution aerial imagery. *Remote Sensing*, 10(9):1487, sep 2018.

- [76] Kang Zhao, Jungwon Kang, Jaewook Jung, and Gunho Sohn. Building Extraction from Satellite Images Using Mask R-CNN with Building Boundary Regularization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 242–2424. IEEE, jun 2018.
- [77] Xiaodong Zhao, Qichao Zhang, Dongbin Zhao, and Zhonghua Pang. Overview of image segmentation and its application on free space detection. In *Proceedings of 2018 IEEE 7th Data Driven Control and Learning Systems Conference, DDCLS 2018*, pages 1164–1169. IEEE, may 2018.

Bilag

A. Anvendte værktøjer

Projektet er gennemført ved brug af følgende værktøjer:

- Anaconda Distribution version 2019.03:
 - Python 3.6
 - Jupyter Notebook / Jupyter Lab
 - Spyder
- Python moduler:
 - Pandas / Geopandas
 - Scikit-Learn / Scikit-Image
 - TensorFlow m.fl.
- QGIS
- SQLite / Spatialite

Rapporten er skrevet med LaTeX med håndtering af referencer i BibTeX og Mendeley.

B. Notebook: EDA

```
[1] %matplotlib inline

import os
from pylab import *
import pandas as pd
import geopandas as gpd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from PIL import Image
import time
```

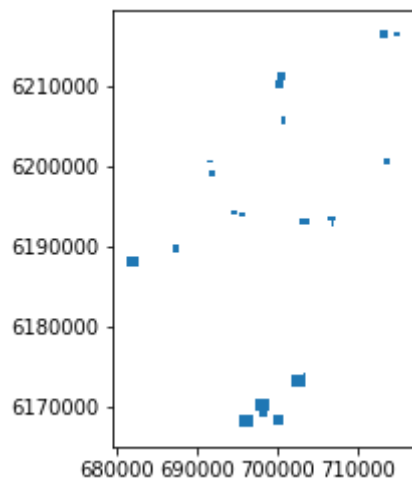
```
[2] # Workdir ændres til Data-mappen
wd = "./data/"
os.chdir(os.path.expanduser(wd))
os.getcwd()
imgdir = "../..Rapport/Figurer/"
```

```
[5] os.listdir("./02_proc/")
```

```
['1F5D59FB-C80B-4789-B252-82EA00ABB3DC',
'C39F5252-966C-4F03-B090-D51F93AAF3EF',
'8587B280-9EEF-4FD7-9A9C-1E140FAC7038',
'754A0A4D-41FF-4D96-AFED-5320D061FB01',
'.DS_Store',
'046E1060-E828-4AB0-8A4C-08EE908A7802',
'C3C3E540-E463-45C0-A7FF-2E45CD65FE8D',
'3492FBF8-E73A-4A8F-965F-A8357972062C',
'B6881AE3-B5D3-4285-982A-10F2F5210469',
'A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F',
'C95E8154-1F97-4210-A2B5-9C04589E24B2',
'80ABB9ED-B4B6-4B68-902B-5F073617A1D4',
'E01454F7-579A-424F-95B2-734C47A6FEC1',
'3291B727-CF8E-4593-BAAF-087EBE35ACF8',
'8ECC0084-CC94-4EE2-839F-4FD9E7D9EBC9',
'1D19A5B9-73D1-4337-B8FF-3940EC422400',
'6C24B0D3-7F97-4ED4-9637-33C96932BCB8',
'2AC54EE2-4935-42A8-8840-DC80F884547D',
'BDCA49B7-DD34-4B65-8822-414A7CF4C554',
'1422133C-84E2-467D-B35A-B415EF2E01FB',
'AB5B0287-1321-48C9-A073-B8D3E38A7ECB',
'A4481F38-9384-4B59-9AAF-CD7C5FF47238',
'F43B642E-655E-4985-848F-BD223A4BFC35']
```

```
[6] # Indlæs modelområderne - Areas Of Interest fra shape-fil
modelomr = gpd.GeoDataFrame.from_file("./01_proc/aoi.shp")
modelomr.plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x1a1b2d02e8>



```
[7] data = pd.read_csv("02_proc/A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F/data.csv")
data.head()
```

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	aoi_id	row_index	col
0	0	0	0	A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F	0	0
1	1	1	1	A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F	0	1
2	2	2	2	A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F	0	2
3	3	3	3	A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F	0	3
4	4	4	4	A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F	0	4

```
[8] # Vis opsummerende statistik for datasettet
data.describe()
```

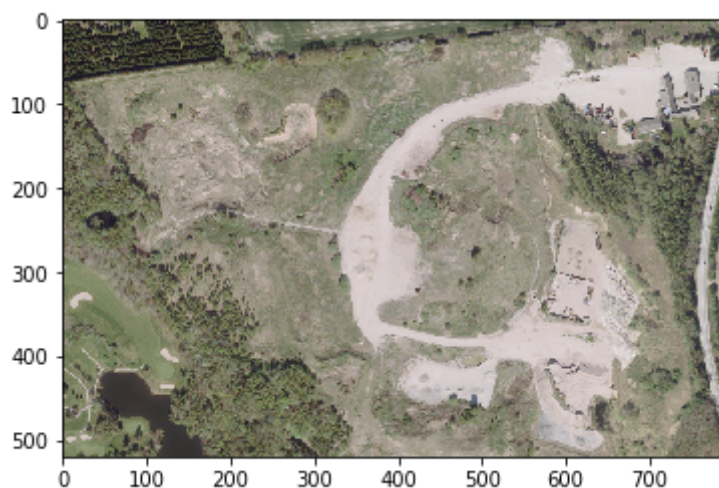
	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	row_index	col
count	410800.000000	410800.000000	410800.000000	410800.000000	410800.000000
mean	205399.500000	205399.500000	205399.500000	259.500000	39.500000

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	row_index	col
std	118587.889628	118587.889628	118587.889628	150.110975	228
min	0.000000	0.000000	0.000000	0.000000	0.0
25%	102699.750000	102699.750000	102699.750000	129.750000	19
50%	205399.500000	205399.500000	205399.500000	259.500000	39
75%	308099.250000	308099.250000	308099.250000	389.250000	59
max	410799.000000	410799.000000	410799.000000	519.000000	78

```
[9] # HLV-Test: kan data i csv-filen stadig kan fortolkes som et billede?
orto = data[['r', 'g', 'b']].values
omr = modelomr[modelomr['aoi_id']=='A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F']
width = omr['width'].values[0]
height = omr['height'].values[0]
print(width, height)
orto = orto.reshape(height,width,3)
orto = np.array(orto,dtype=int)
imshow(orto)
```

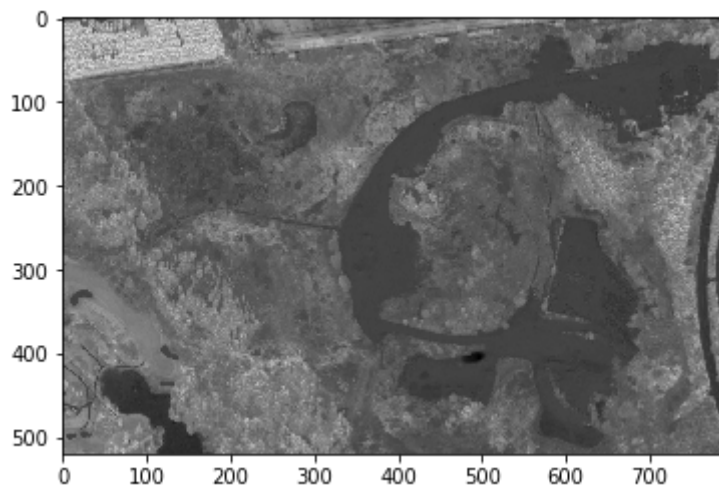
790 520

<matplotlib.image.AxesImage at 0x1a1b6d5048>



```
[10] # Vis ndvi som foto
ndvi = data[['ndvi']].values
ndvi = ndvi + abs(ndvi.min())
ndvi = ndvi * (255/ndvi.max())
omr = modelomr[modelomr['aoi_id']=='A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F']
width = omr['width'].values[0]
height = omr['height'].values[0]
ndvi = ndvi.reshape(height,width)
ndvi = np.array(ndvi,dtype=int)
imshow(ndvi, cmap=plt.cm.gray)
```

<matplotlib.image.AxesImage at 0x1029b5828>



```
[11] import skimage.color as color
      orto_gray = color.rgb2gray(orto)

      aktiv = data[['aktiv']].values
      omr = modelomr[modelomr['aoi_id']=='A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F']
      width = omr['width'].values[0]
      height = omr['height'].values[0]
      aktiv_mask = aktiv.reshape(height,width)
      aktiv_mask = np.array(aktiv_mask,dtype=int)

      fig, axes = plt.subplots(1, 2, figsize=(14, 4))
      axes[0].imshow(orto_gray, cmap=plt.cm.gray, interpolation='nearest')
      axes[0].axis('off')
      axes[1].imshow(aktiv_mask, cmap=plt.cm.gray, interpolation='nearest')
      axes[1].axis('off')

      orto_gray.shape
      aktiv_mask.shape
```

```
/Users/ak/anaconda3/lib/python3.7/site-packages/skimage/util/dtype.py:141:
UserWarning: Possible precision loss when converting from int64 to float64
  .format(dtypeobj_in, dtypeobj_out))
```

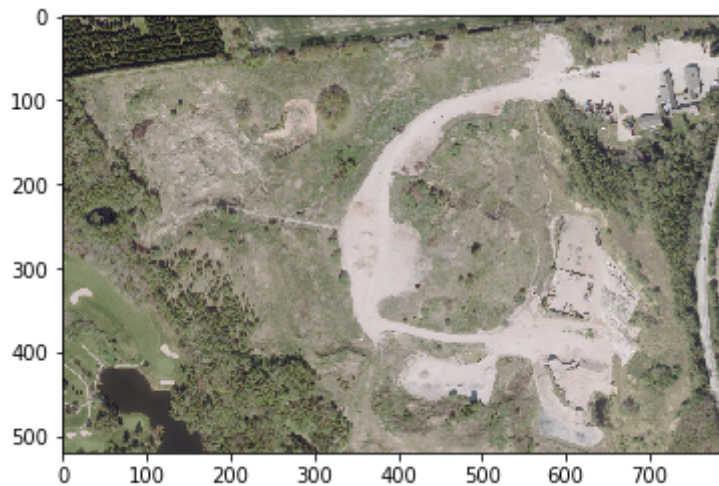
(520, 790)



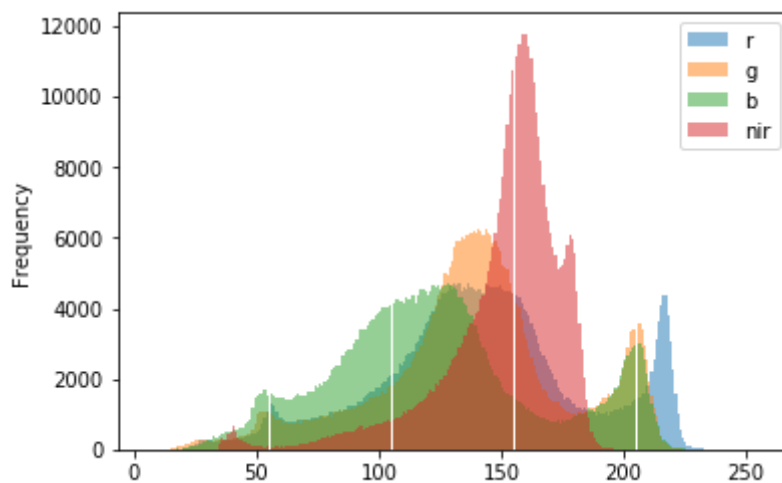
```
[12] # Sammenlignes med selve billede-filen - jep, de er ens!
ortoimg = array(Image.open('02_proc/A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F/'))
dim = ortoimg.shape
print(dim)
imshow(ortoimg)
```

(520, 790, 4)

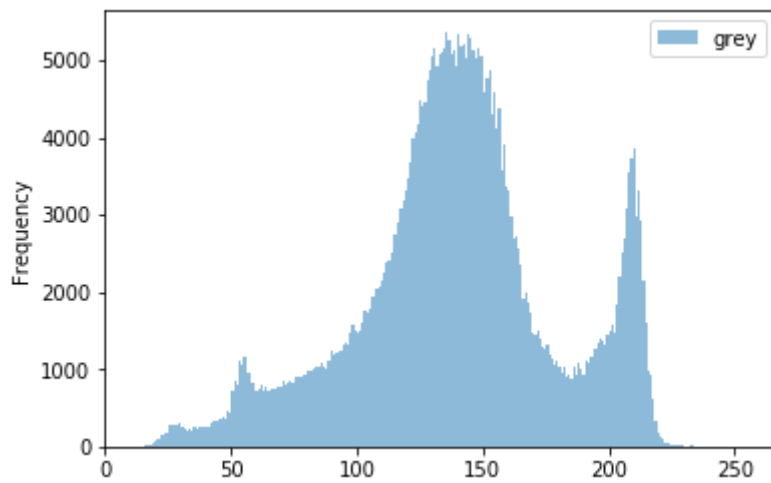
<matplotlib.image.AxesImage at 0x1a1c21f048>



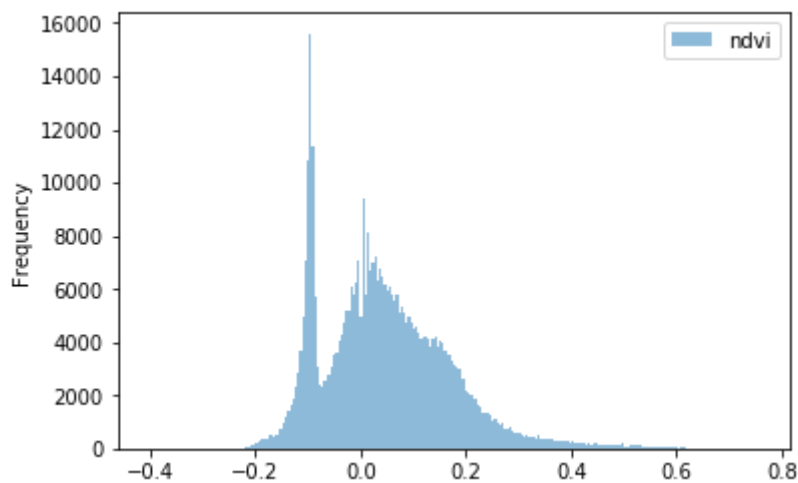
```
[13] # Histogram for ortofoto
# NB! Hvorfor er der mrkelige striber?? er der farveværdier der ikke bruges
data[['r','g','b','nir']].plot.hist(bins=254, alpha=0.5)
filepath = os.path.join(imgdir,"hist_rgbn.png")
plt.savefig(filepath)
```



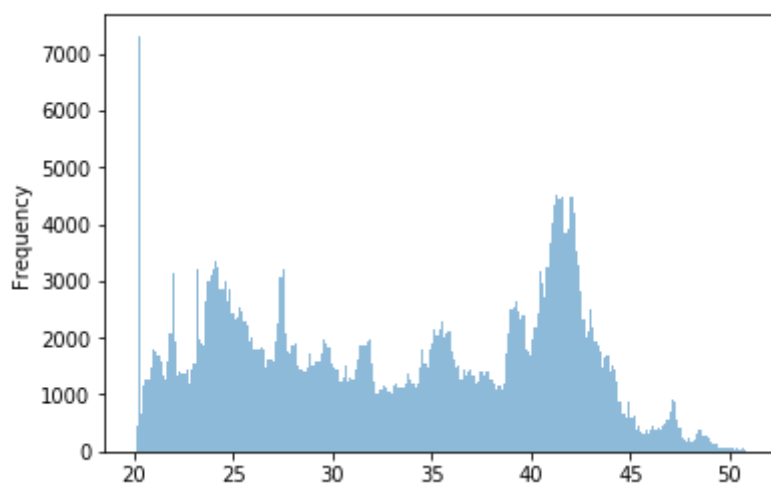
```
[18] # Histogram for gråtone
# NB! ingen mærkelige striber
data[['grey']].plot.hist(bins=254, alpha=0.5)
filepath = os.path.join(imgdir,"hist_grey.png")
plt.savefig(filepath)
```



```
[19] # Histogram for gråtone
# NB! ingen mærkelige striber
data[['ndvi']].plot.hist(bins=254, alpha=0.5)
filepath = os.path.join(imgdir,"hist_ndvi.png")
plt.savefig(filepath)
```

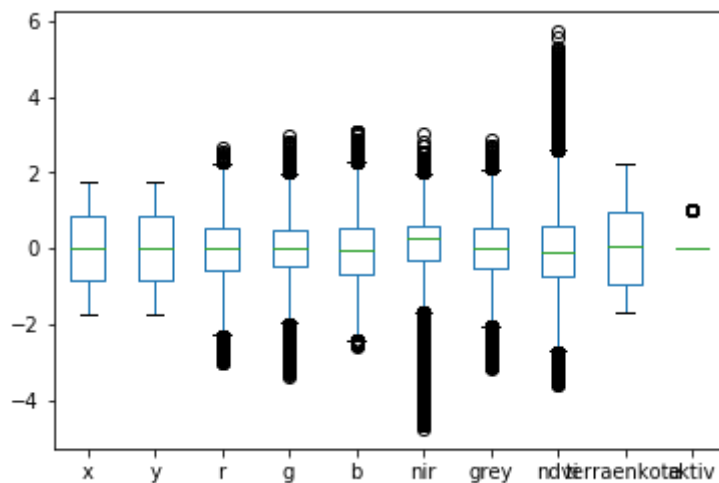


```
[21] # Histogram for terræknoten
data['terraenkote'].plot.hist(bins=254, alpha=0.5)
filepath = os.path.join(imgdir,"hist_terraen.png")
plt.savefig(filepath)
```

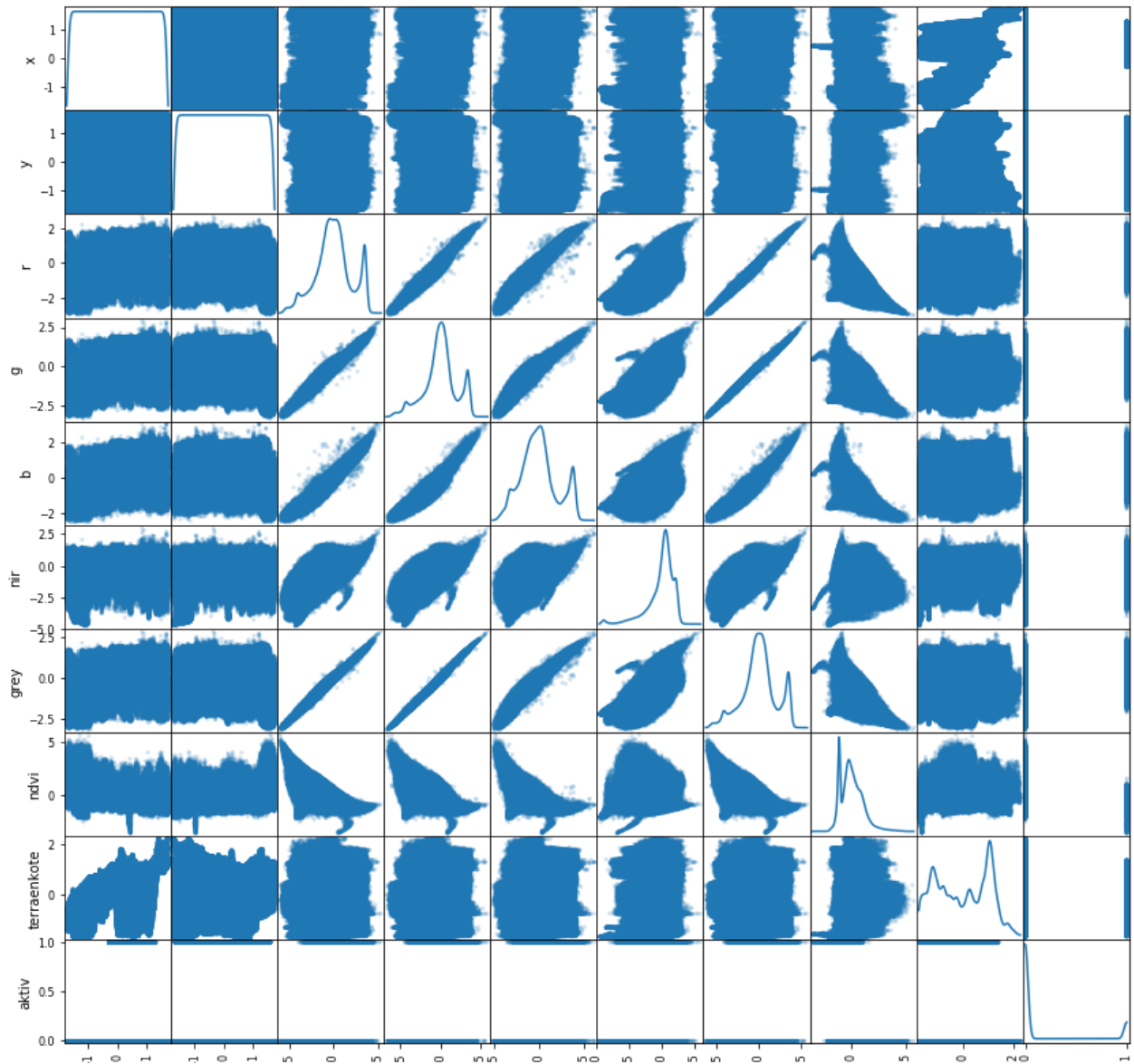


```
[22] # Normalisering af data & udvælgelse af kolonner
cols = ['x', 'y', 'r', 'g', 'b', 'nir', 'grey', 'ndvi', 'terraenkote']
data_features = data[cols]
x = data_features.values
x = StandardScaler().fit_transform(x)
target = data[['aktiv']] / 255
cols.append('aktiv')
x = np.concatenate((x,target.values),axis=1)
data_norm = pd.DataFrame(x, columns=cols)
```

```
[23] data_norm.plot.box()
filepath = os.path.join(imgdir,"boxplot.png")
plt.savefig(filepath)
```



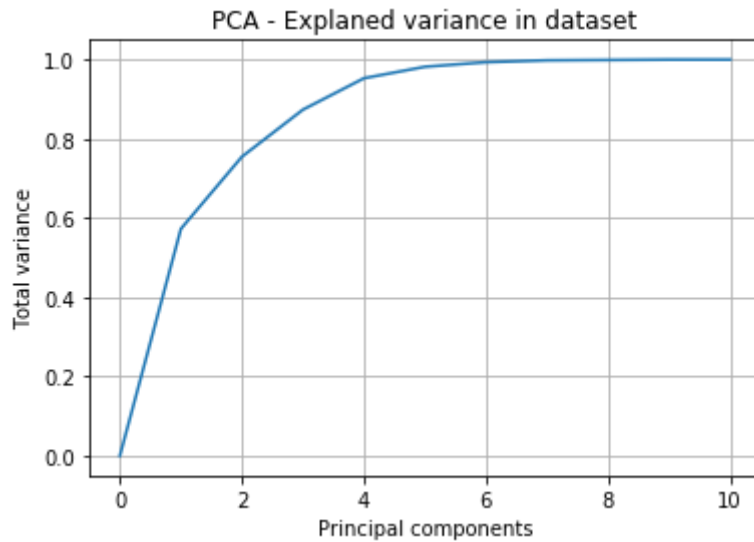
```
[24] pd.plotting.scatter_matrix(data_norm, alpha=0.2, figsize=(14, 14), diagonal
filepath = os.path.join(imgdir,"scatter_matrix.png")
plt.savefig(filepath_)
```



```
[25] pca = PCA()
principalComp = pca.fit_transform(x)
```

```
[26] col = []
for pc in range(principalComp.shape[1]):
    col.append("PC"+str(pc+1))
pc_df = pd.DataFrame(data = principalComp, columns = col)
pc_df = pd.concat([pc_df, target], axis=1)
```

```
[27] expl_var = np.cumsum(pca.explained_variance_ratio_)
expl_var = np.append([0],expl_var)
plt.plot(expl_var)
plt.xlabel('Principal components', fontsize = 10)
plt.ylabel('Total variance', fontsize = 10)
plt.title('PCA - Explained variance in dataset', fontsize = 12)
plt.grid()
filepath = os.path.join(imgdir,"pca_expl.png")
plt.savefig(filepath)
```



```
[28] v = 0
      for i, pc in enumerate(expl_var[1:]):
          print("PC" + str(i+1), str(round((pc-v)*100,1)) + "%")
          v = pc
```

```
PC1 57.3%
PC2 18.2%
PC3 11.9%
PC4 7.9%
PC5 2.9%
PC6 1.2%
PC7 0.5%
PC8 0.1%
PC9 0.1%
PC10 0.0%
```

```
[33] comp = pca.components_[0:3, :]
      pc1 = comp[0]
      pc2 = comp[1]
      pc3 = comp[2]

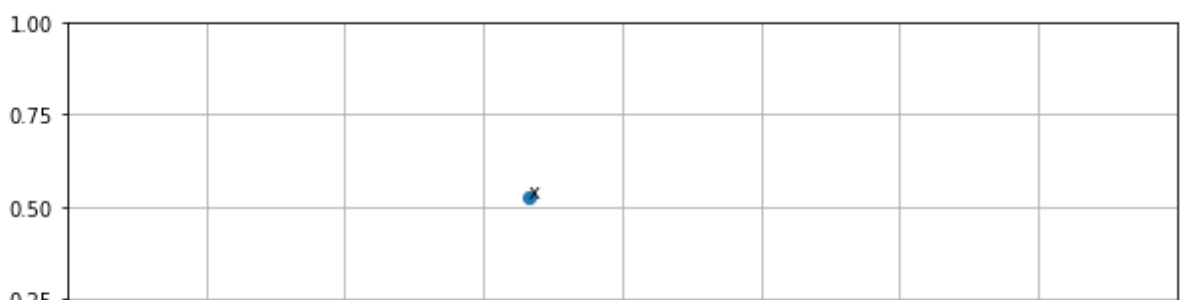
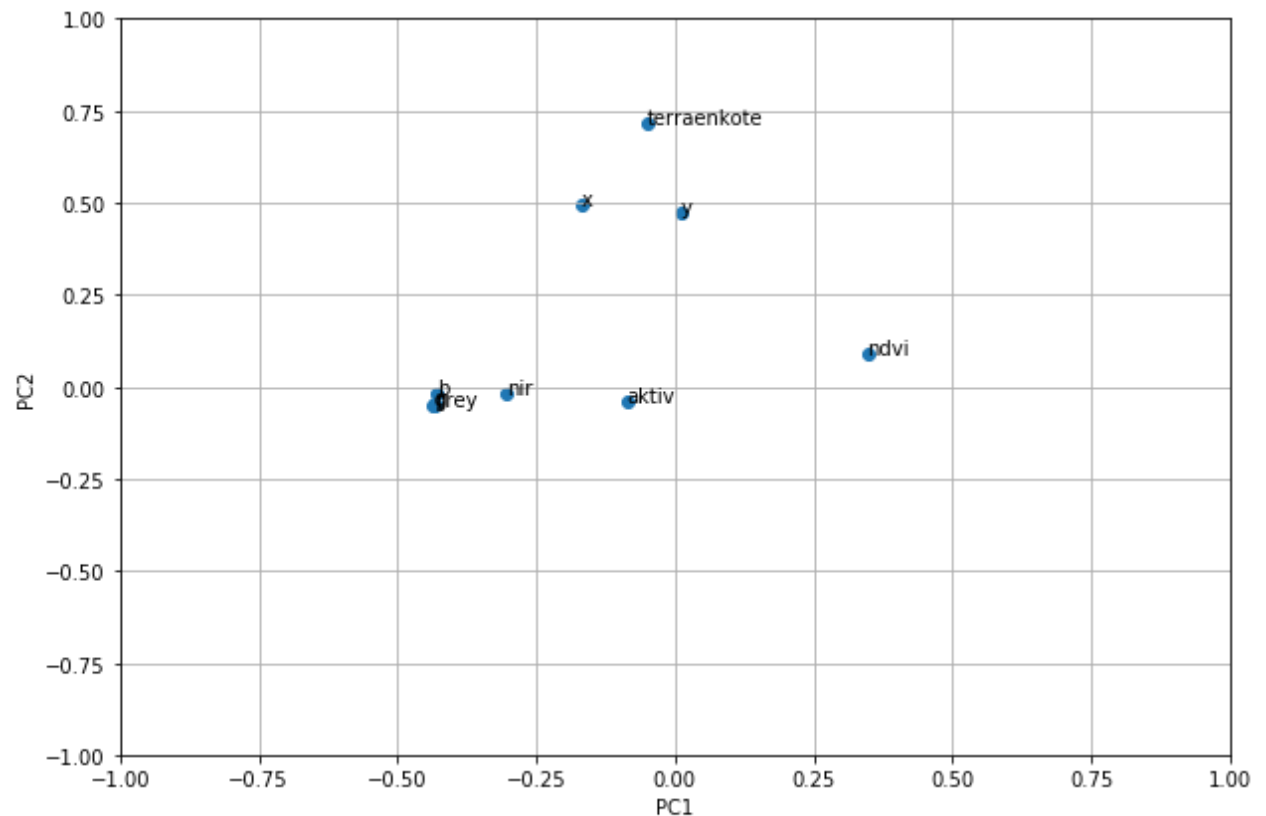
      fig, ax = plt.subplots(2, 1, figsize=(10, 15))
      ax[0].scatter(pc1, pc2)
      ax[1].scatter(pc1, pc3)

      for i, txt in enumerate(cols):
          ax[0].annotate(txt, (pc1[i], pc2[i]))
          ax[1].annotate(txt, (pc1[i], pc3[i]))

      ax[0].set(xlabel="PC{}".format(1), ylabel="PC{}".format(2))
      ax[0].set_xlim(-1,1)
      ax[0].set_ylim(-1,1)
      ax[0].grid()
      ax[1].set(xlabel="PC{}".format(1), ylabel="PC{}".format(3))
      ax[1].set_xlim(-1,1)
      ax[1].set_ylim(-1,1)
      ax[1].grid()
```

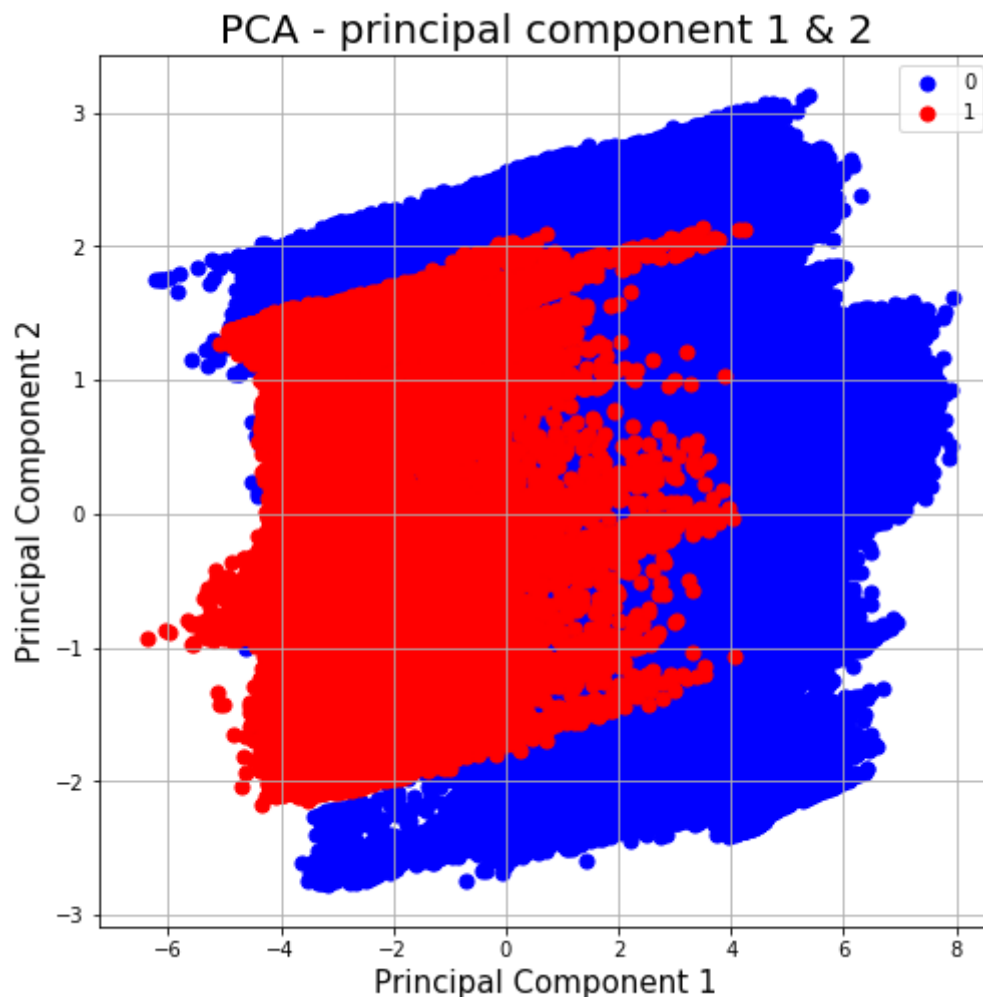


```
filepath = os.path.join(imgdir,"pca_pc.png")
plt.savefig(filepath)
```



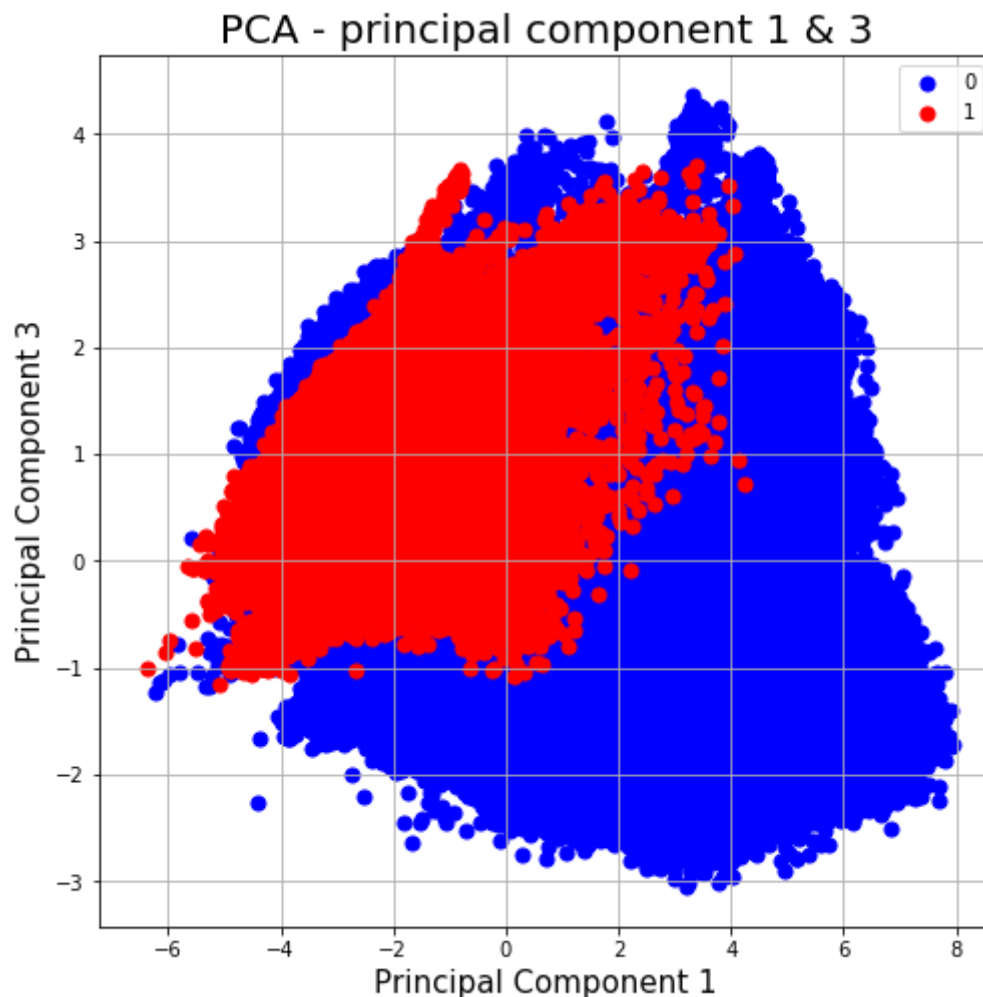
```
[34] fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('PCA - principal component 1 & 2', fontsize = 20)

targets = [0, 1]
colors = ['b', 'r']
for target, color in zip(targets,colors):
    indicesToKeep = pc_df['aktiv'] == target
    ax.scatter(pc_df.loc[indicesToKeep, 'PC1']
               , pc_df.loc[indicesToKeep, 'PC2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



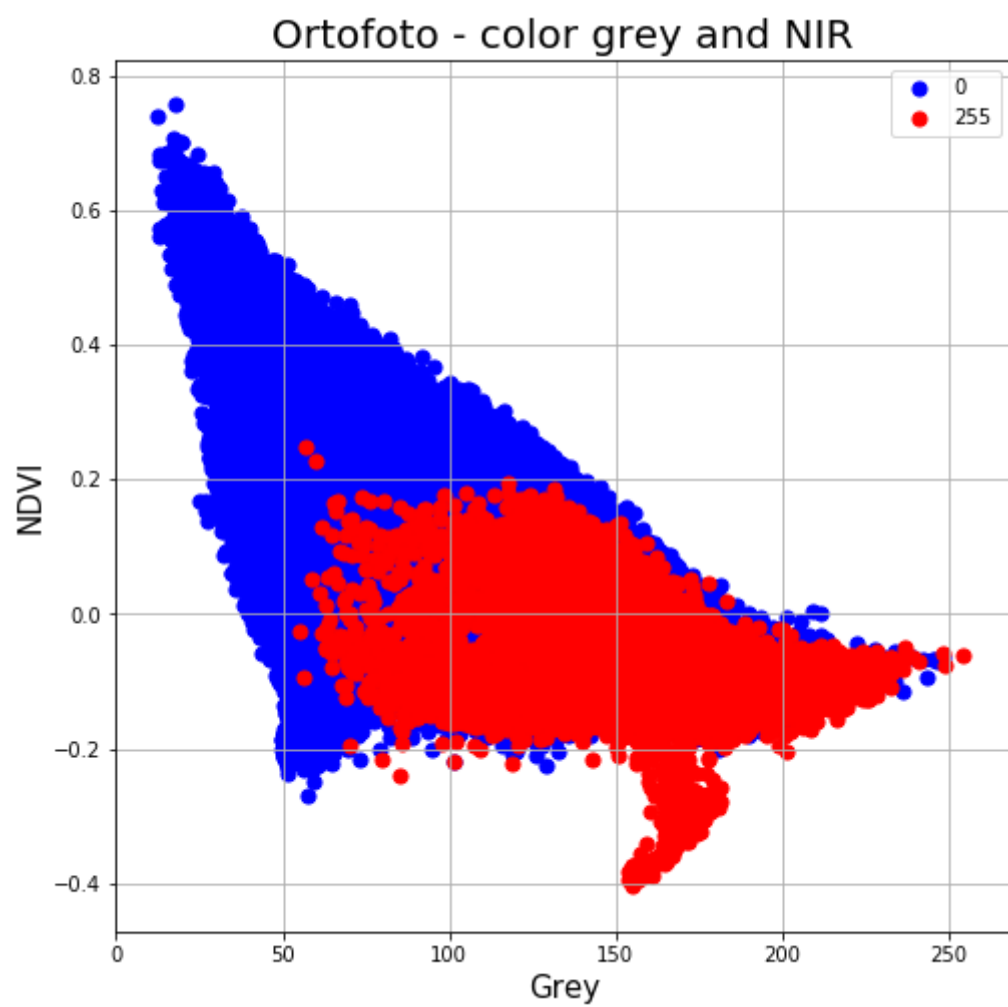
```
[35] fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 3', fontsize = 15)
ax.set_title('PCA - principal component 1 & 3', fontsize = 20)

targets = [0, 1]
colors = ['b', 'r']
for target, color in zip(targets, colors):
    indicesToKeep = pc_df['aktiv'] == target
    ax.scatter(pc_df.loc[indicesToKeep, 'PC1']
               , pc_df.loc[indicesToKeep, 'PC3']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



```
[55] fig = plt.figure(figsize = (8,8))
      ax = fig.add_subplot(1,1,1)
      ax.set_xlabel('Grey', fontsize = 15)
      ax.set_ylabel('NDVI', fontsize = 15)
      ax.set_title('Ortofoto - color grey and NIR', fontsize = 20)

      targets = [0, 255]
      colors = ['b', 'r']
      for target, color in zip(targets, colors):
          indicesToKeep = data['aktiv'] == target
          ax.scatter(data.loc[indicesToKeep, 'grey']
                     , data.loc[indicesToKeep, 'ndvi']
                     , c = color
                     , s = 50)
      ax.legend(targets)
      ax.grid()
```



C. Notebook: Simpel billedesegmentering med Scikit-Image

```
[1] %matplotlib inline

import os
from pylab import *
from scipy import ndimage as ndi
from skimage import io
from skimage.exposure import histogram
import skimage.segmentation as seg
import skimage.morphology as morphology
import skimage.filters as filters
import skimage.draw as draw
import skimage.color as color
import skimage.feature as feature
from skimage.future import graph
```

```
[2] # Workdir ændres til Data-mappen
wd = "~/Sync/IT-diplom/2019_1_BD/Projekt/data/02_proc/A70F9377-6FBE-49EF-,
os.chdir(os.path.expanduser(wd))
os.getcwd()
imgdir = os.path.expanduser("~/Sync/IT-diplom/2019_1_BD/Rapport/Figurer/")
```

```
[3] def image_show(image, nrows=1, ncols=1, cmap='gray'):
    fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(14, 14))
    ax.imshow(image, cmap='gray')
    ax.axis('off')
    return fig, ax
```

```
[4] ortoimg = io.imread('orto.png')
cir = io.imread('orto_cir.png')
ortoimg.shape
```

(520, 790, 4)

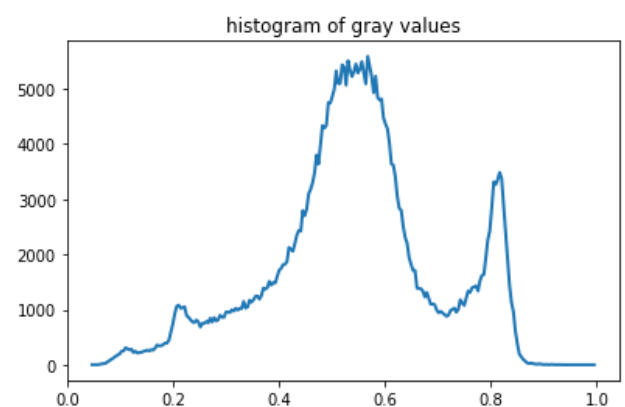
```
[5] orto_gray = color.rgb2gray(ortoimg)
img = orto_gray
image_show(img);
```



```
[6] hist, hist_centers = histogram(img)

fig, axes = plt.subplots(1, 2, figsize=(14, 4))
axes[0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
axes[0].axis('off')
axes[1].plot(hist_centers, hist, lw=2)
axes[1].set_title('histogram of gray values')
```

Text(0.5, 1.0, 'histogram of gray values')



```
[7] fig, axes = plt.subplots(2, 2, figsize=(14, 10), sharey=True)

axes[0,0].imshow(img > 0.4, cmap=plt.cm.gray, interpolation='nearest')
axes[0,0].set_title('areas > 0.4')
axes[0,0].axis('off')

axes[0,1].imshow(img > 0.65, cmap=plt.cm.gray, interpolation='nearest')
```

```

axes[0,1].set_title('areas > 0.6')
axes[0,1].axis('off')

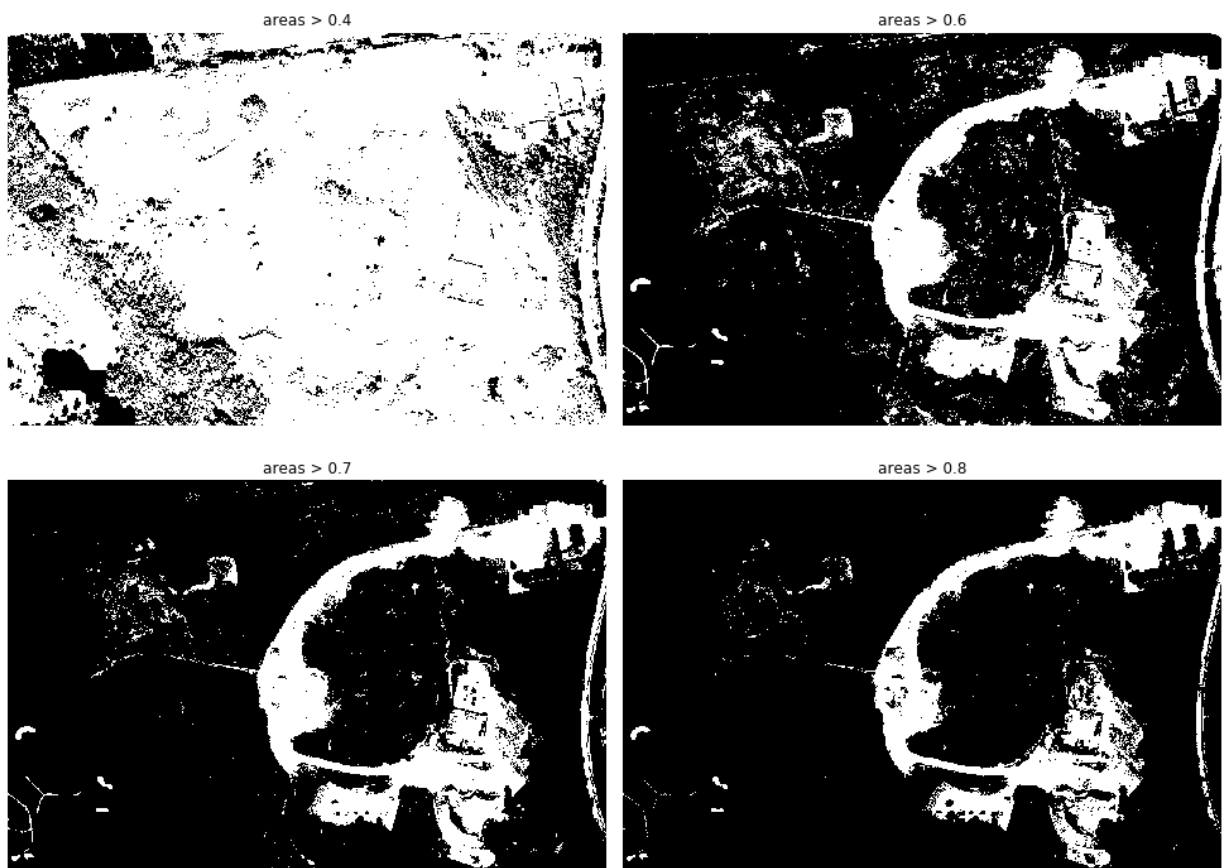
axes[1,0].imshow(img > 0.7, cmap=plt.cm.gray, interpolation='nearest')
axes[1,0].set_title('areas > 0.7')
axes[1,0].axis('off')

axes[1,1].imshow(img > 0.75, cmap=plt.cm.gray, interpolation='nearest')
axes[1,1].set_title('areas > 0.8')
axes[1,1].axis('off')

plt.tight_layout()

imgpath = os.path.join(imgdir,"thresholds.png")
plt.savefig(imgpath)

```



```

[8] img = filters.gaussian(img, 1)
    edge_map = filters.sobel(img)

fig, ax = plt.subplots(1, 2, figsize=(14, 5))

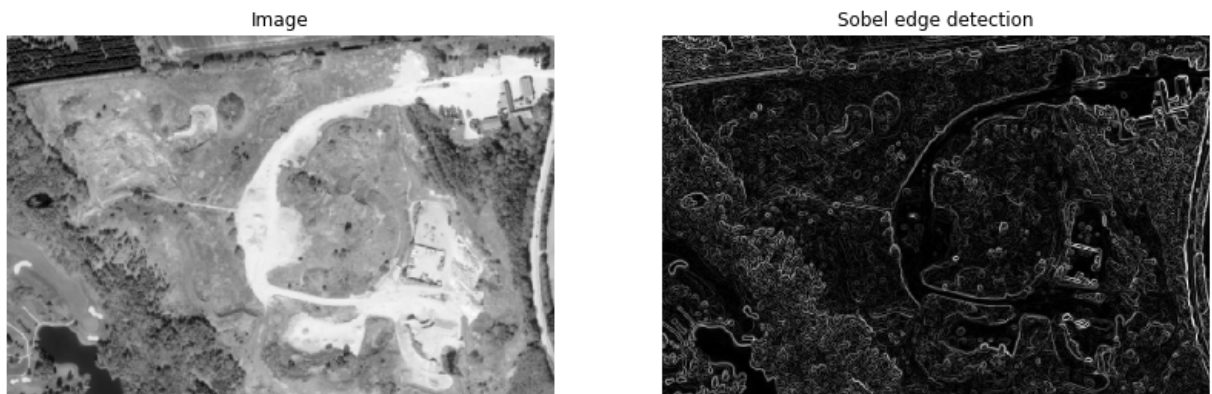
ax[0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
ax[0].set_title('Image')
ax[0].axis('off')

ax[1].imshow(edge_map, cmap=plt.cm.gray, interpolation='nearest')
ax[1].set_title('Sobel edge detection')
ax[1].axis('off')

```



```
imgpath = os.path.join(imgdir,"sobel.png")
plt.savefig(imgpath)
```



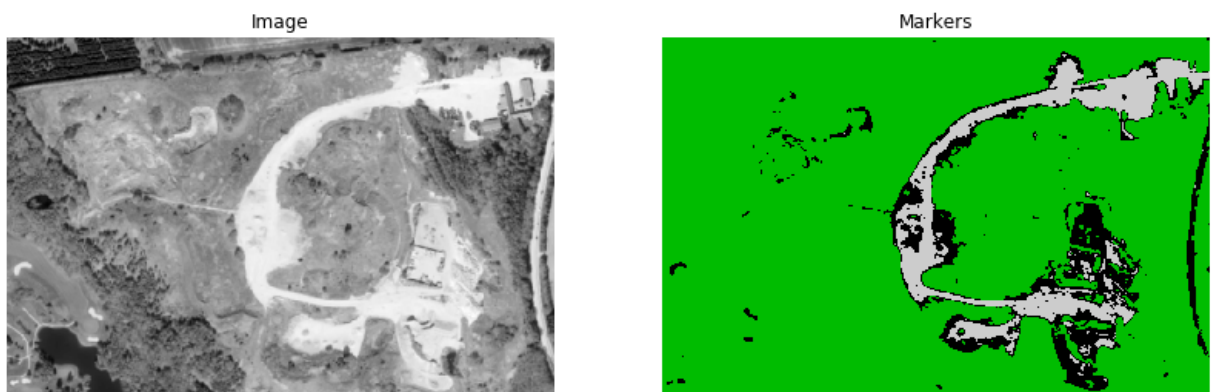
```
[9] markers = np.zeros_like(img)
markers[img < 0.7] = 1
markers[img > 0.8] = 2

fig, ax = plt.subplots(1, 2, figsize=(14, 5))

ax[0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
ax[0].set_title('Image')
ax[0].axis('off')

ax[1].imshow(markers, cmap=plt.cm.nipy_spectral, interpolation='nearest')
ax[1].set_title('Markers')
ax[1].axis('off')

imgpath = os.path.join(imgdir,"basic_seg1.png")
plt.savefig(imgpath)
```



```
[10] segmentation = morphology.watershed(edge_map, markers)

fig, ax = plt.subplots(1, 2, figsize=(14, 5))

ax[0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
ax[0].set_title('Image')
ax[0].axis('off')
```

```
ax[1].imshow(segmentation, cmap=plt.cm.gray, interpolation='nearest')
ax[1].set_title('segmentation')
ax[1].axis('off')

imgpath = os.path.join(imgdir,"basic_seg2.png")
plt.savefig(imgpath)
```



```
[11] filled = ndi.binary_fill_holes(segmentation-1)
      cleaned = morphology.remove_small_objects(filled, 500)
      cleaned = cleaned + 1

      fig, ax = plt.subplots(1, 2, figsize=(14, 5))

      ax[0].imshow(segmentation, cmap=plt.cm.gray, interpolation='nearest')
      ax[0].set_title('Segmentation')
      ax[0].axis('off')

      ax[1].imshow(cleaned, cmap=plt.cm.gray, interpolation='nearest')
      ax[1].set_title('Cleaned')
      ax[1].axis('off')

      imgpath = os.path.join(imgdir,"basic_seg3.png")
      plt.savefig(imgpath)
```



```
[12] labeled_areas, _ = ndi.label(cleaned-1)
      image_label_overlay = color.label2rgb(labeled_areas, image=img)
```

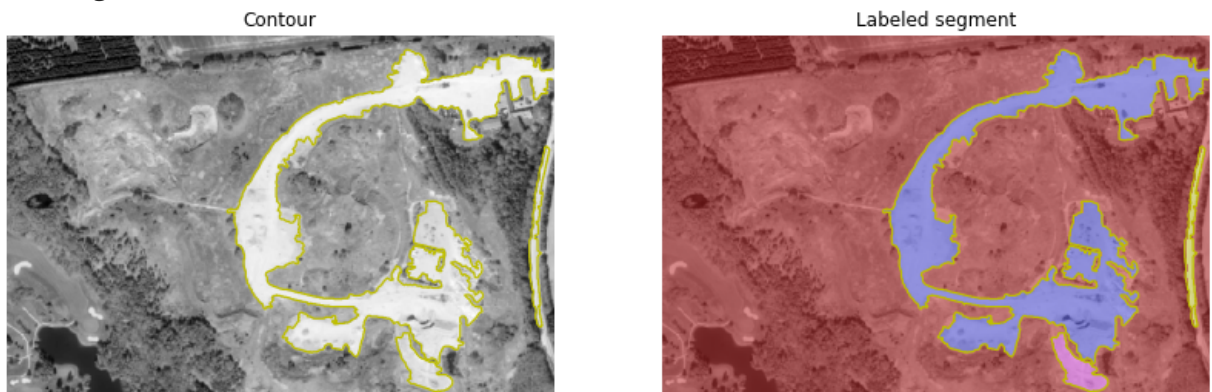
```
fig, ax = plt.subplots(1, 2, figsize=(14, 5), sharey=True)
ax[0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
ax[0].contour(cleaned, levels=[1], linewidths=1.2, colors='y')
ax[0].set_title("Contour")

ax[1].imshow(image_label_overlay, interpolation='nearest')
ax[1].contour(cleaned, levels=[1], linewidths=1.2, colors='y')
ax[1].set_title("Labeled segment")

for a in ax:
    a.axis('off')

imgpath = os.path.join(imgdir, "basic_seg4.png")
plt.savefig(imgpath)
```

/Users/ak/anaconda3/lib/python3.7/site-packages/matplotlib/contour.py:1243:
 UserWarning: No contour levels were found within the data range.
 warnings.warn("No contour levels were found")



SLIC(Simple Linear Iterative Clustering)

SLIC algorithm actually uses a machine learning algorithm called K-Means under the hood. It takes in all the pixel values of the image and tries to separate them out into the given number of sub-regions. Read the Reference Paper here: <https://ieeexplore.ieee.org/document/6205760>

SLIC works in color so we will use the original image.

```
[13] image_slic = seg.slic(ortoimg,n_segments=5000)
# label2rgb replaces each discrete label with the average interior color
slic_avg=color.label2rgb(image_slic, ortoimg, kind='avg')
fig, ax = plt.subplots(1, 1, figsize=(7, 5), sharey=True)
ax.imshow(slic_avg, cmap=plt.cm.gray, interpolation='nearest')
ax.axis("off")
```

(-0.5, 789.5, 519.5, -0.5)



Felzenszwalb

This algorithm also uses a machine learning algorithm called minimum-spanning tree clustering under the hood. Felzenszwaib doesn't tell us the exact number of clusters that the image will be partitioned into. It's going to run and generate as many clusters as it thinks is appropriate for that given scale or zoom factor on the image. The Reference Paper can be accessed here: <http://cs.brown.edu/people/pfelzens/segment/>

```
[14] image_felzenszwalb = seg.felzenszwalb(ortoimg)
      image_show(image_felzenszwalb);
```

```
/Users/ak/anaconda3/lib/python3.7/site-
packages/skimage/segmentation/_felzenszwalb.py:60: RuntimeWarning: Got
image with third dimension of 4. This image will be interpreted as a
multichannel 2d image, which may not be intended.
  min_size=min_size)
```



```
[15] np.unique(image_felzenszwalb).size
```

```
5843
```

```
[16] image_felzenszwalb_colored = color.label2rgb(image_felzenszwalb, ortoimg, k  
image_show(image_felzenszwalb_colored);
```



RAG Merging

This example constructs a Region Adjacency Graph (RAG) and progressively merges regions that are similar in color. Merging two adjacent regions produces a new region with all the pixels from the merged regions. Regions are merged until no highly similar region pairs remain.

```
[17] def _weight_mean_color(graph, src, dst, n):  
    """Callback to handle merging nodes by recomputing mean color.  
  
    The method expects that the mean color of `dst` is already computed.
```

Parameters

graph : RAG
The graph under consideration.
src, dst : int
The vertices in `graph` to be merged.
n : int
A neighbor of `src` or `dst` or both.

Returns

data : dict
A dictionary with the `"weight"` attribute set as the absolute difference of the mean color between node `dst` and `n`.
"""

```
diff = graph.node[dst]['mean color'] - graph.node[n]['mean color']  
diff = np.linalg.norm(diff)  
return {'weight': diff}
```

```
def merge_mean_color(graph, src, dst):  
    """Callback called before merging two nodes of a mean color distance
```

This method computes the mean color of `dst`.

Parameters

graph : RAG
The graph under consideration.
src, dst : int
The vertices in `graph` to be merged.
"""

graph.node[dst]['total color'] += graph.node[src]['total color']
graph.node[dst]['pixel count'] += graph.node[src]['pixel count']
graph.node[dst]['mean color'] = (graph.node[dst]['total color'] /
 graph.node[dst]['pixel count'])

```
img = ortoimg[:, :, 3]  
labels = seg.slic(img, compactness=30, n_segments=400)  
g = graph.rag_mean_color(img, labels)  
  
labels2 = graph.merge_hierarchical(labels, g, thresh=35, rag_copy=False,  
                                  in_place_merge=True,  
                                  merge_func=merge_mean_color,  
                                  weight_func=_weight_mean_color)  
  
out = color.label2rgb(labels2, img, kind='avg')  
rag = seg.mark_boundaries(out, labels2, (0, 0, 0))  
  
fig, ax = plt.subplots(1, 2, figsize=(14, 8), sharey=True)
```



```

ax[0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
ax[0].contour(cleaned, [0.5], linewidths=1.2, colors='y')
ax[0].set_title('Orto med watershed kontur')

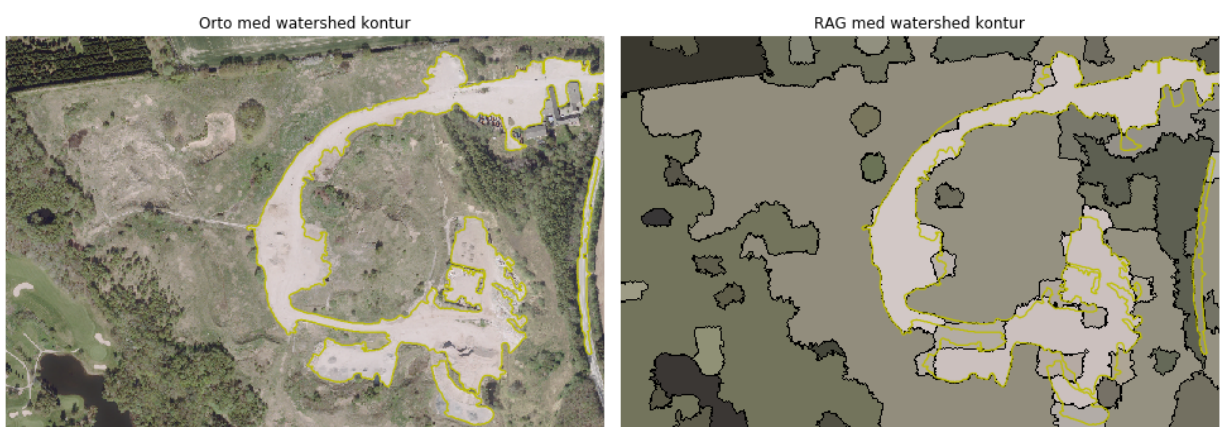
ax[1].imshow(rag, interpolation='nearest')
ax[1].contour(cleaned, [0.5], linewidths=1.2, colors='y')
ax[1].set_title('RAG med watershed kontur')

for a in ax:
    a.axis('off')

plt.tight_layout()

plt.show()

```



Active contour segmentation

Active Contour segmentation also called as snakes and is initialized using a user-defined contour or line, around the area of interest and this contour then slowly contracts and is attracted or repelled from light and edges.

For our example image, let's draw a circle around the person's head to initialize the snake.

```

[18] : funktion til at definere en cirkel ud fra center og radius. Resolution giv
: som cirklen skal tegnes med
def circle_points(resolution, center, radius):

    """
    Generate points which define a circle on an image. Centre refers to the c
    """

    radians = np.linspace(0, 2*np.pi, resolution)

    c = center[1] + radius*np.cos(radians)#polar co-ordinates
    r = center[0] + radius*np.sin(radians)

    return np.array([c, r]).T

```

```
[19] # Exclude last point because a closed path should not have duplicate point
# Opret en cirkel i billedet, som segmenteringen skal startes fra
points = circle_points(200, [290, 350], 100)[: -1]
fig, ax = image_show(ortoimg)
ax.plot(points[:, 0], points[:, 1], '--r', lw=3)
```

[<matplotlib.lines.Line2D at 0x1c26663128>]



```
[20] img_gaus = filters.gaussian(orto_gray,1)
snake = seg.active_contour(img, points, alpha=0.3, beta=0.4, gamma=0.003)

fig, ax = image_show(orto_gray)
ax.plot(points[:, 0], points[:, 1], '--r', lw=3)
ax.plot(snake[:, 0], snake[:, 1], '-b', lw=3);
```




Random walker segmentation

In this method, a user interactively labels a small number of pixels which are known as labels. Each unlabeled pixel is then imagined to release a random walker and one can then determine the probability of a random walker starting at each unlabeled pixel and reaching one of the pre-labeled pixels. By assigning each pixel to the label for which the greatest probability is calculated, high-quality image segmentation may be obtained. Read the Reference paper here: <https://ieeexplore.ieee.org/document/1704833>

```
[21] from skimage.segmentation import random_walker

img = orto_gray
img = filters.gaussian(img, 4)
markers = np.zeros_like(img)
markers[orto_gray < 0.7] = 1
markers[orto_gray > 0.75] = 2

# Run random walker algorithm and clean up
labels = random_walker(img, markers, beta=10, mode='bf')
filled = ndi.binary_fill_holes(labels-1)
cleaned = morphology.remove_small_objects(filled, 500)
segmentation = cleaned + 1

fig, ax = plt.subplots(2, 2, figsize=(14, 10))

ax[0][0].imshow(img, cmap=plt.cm.gray, interpolation='nearest')
ax[0][0].set_title('Image')
```

```

ax[0][0].axis('off')

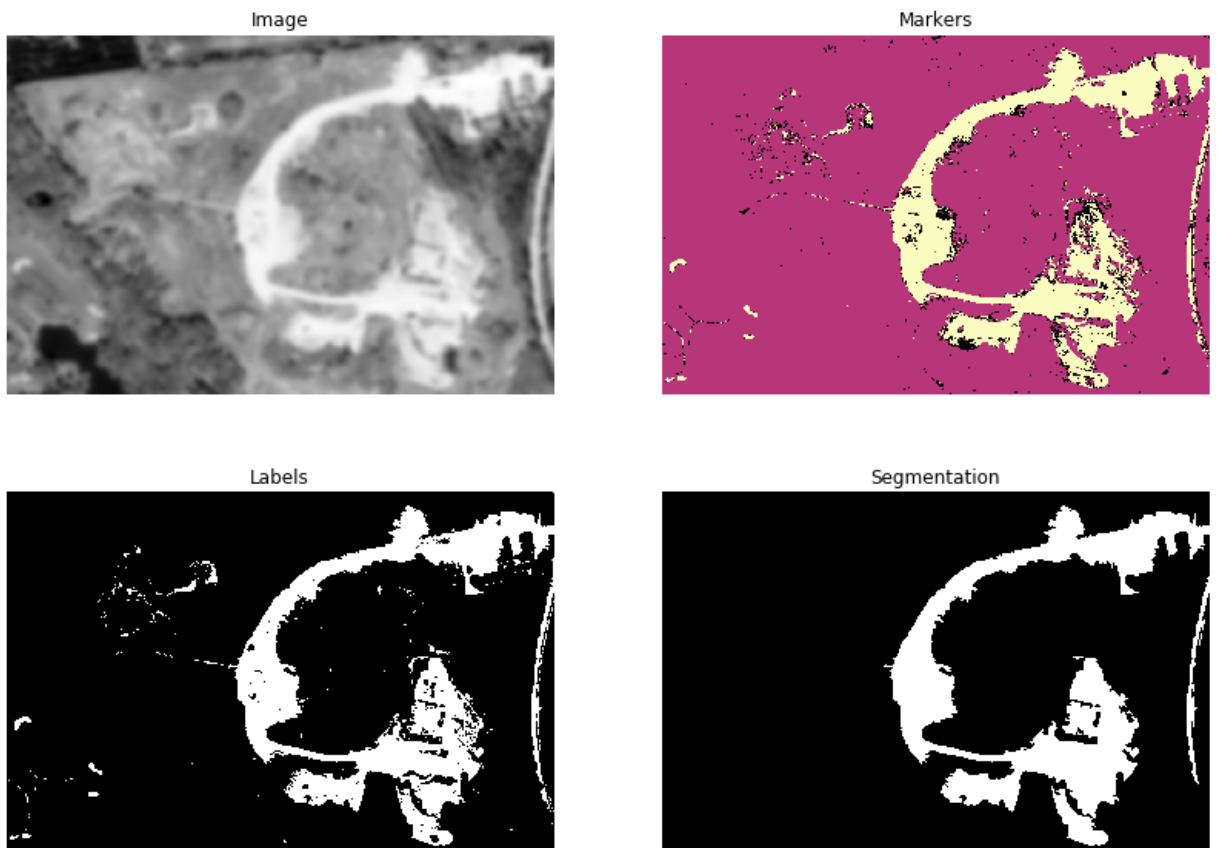
ax[0][1].imshow(markers, cmap='magma', interpolation='nearest')
ax[0][1].set_title('Markers')
ax[0][1].axis('off')

ax[1][0].imshow(labels, cmap='gray', interpolation='nearest')
ax[1][0].axis('off')
ax[1][0].set_title('Labels')

ax[1][1].imshow(segmentation, cmap='gray', interpolation='nearest')
ax[1][1].axis('off')
ax[1][1].set_title('Segmentation')

```

```
Text(0.5, 1.0, 'Segmentation')
```



D. Notebook: Clustering med R-Tree

```
[1] %matplotlib inline

import os
from pylab import *
import pandas as pd
import geopandas as gpd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from PIL import Image
import time
```

```
[2] # Workdir sættes til data-mappen
# og imagedir sættes til Figurer under Rapport-mappen
wd = os.path.abspath("data")
imgdir = os.path.abspath("../Rapport/Figurer/")
print(wd, ":", os.path.isdir(wd))
print(imgdir, ":", os.path.isdir(imgdir))

/Users/ak/Sync/IT-diplom/2019_1_BD/Projekt/data : True
/Users/ak/Sync/IT-diplom/2019_1_BD/Rapport/Figurer : True
```

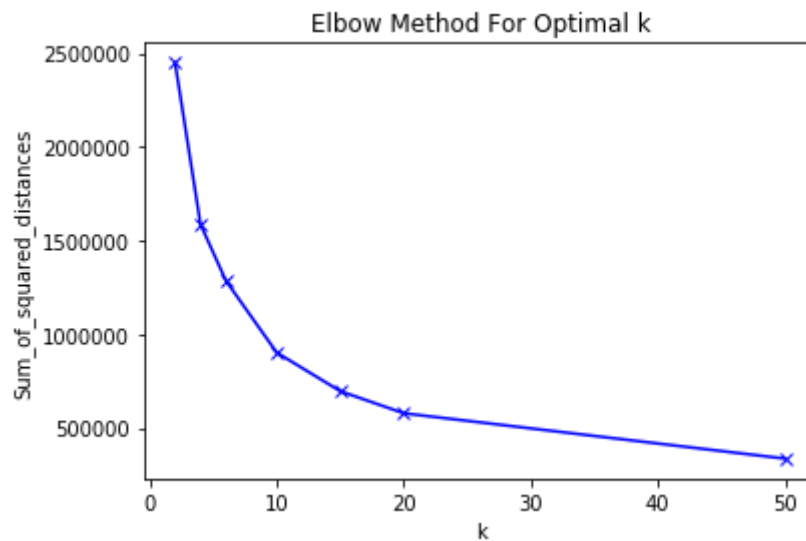
```
[3] # Indlæs modelområderne - Areas Of Interest fra shape-fil
modelomr = gpd.GeoDataFrame.from_file("data/01_proc/aoi.shp")
data = pd.read_csv("data/02_proc/A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F/dat
```

```
[4] # Normalisering af data & udvælgelse af kolonner
cols = ['x', 'y', 'r', 'g', 'b', 'nir', 'grey', 'ndvi', 'terraenkote']
data_features = data[cols]
x = data_features.values
x = StandardScaler().fit_transform(x)
y = data[['aktiv']] / 255
```

```
[5] sum_of_squared_distances = []
models = []
labels = []
K = [2,4,6,10,15,20,50]
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(x)
    models.append(km)
    labels.append(km.predict(x))
    sum_of_squared_distances.append(km.inertia_)
```

```
[6] plt.plot(K, sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
```

```
plt.title('Elbow Method For Optimal k')
filepath = os.path.join(imgdir,"kmeans_sosdist.png")
plt.savefig(filepath)
plt.show()
```



```
[7] # View clusters as an image
orto = data[['r', 'g', 'b']].values
omr = modelmr[modelmr['aoi_id']=='A70F9377-6FBE-49EF-A0A5-5B3EE9CFC39F']
width = omr['width'].values[0]
height = omr['height'].values[0]
print(width, height)
orto = orto.reshape(height,width,3)
orto = np.array(orto,dtype=int)

clusters2 = labels[0].reshape(height,width)
clusters2 = np.array(clusters2,dtype=int)

clusters4 = labels[1].reshape(height,width)
clusters4 = np.array(clusters4,dtype=int)

clusters6 = labels[2].reshape(height,width)
clusters6 = np.array(clusters6,dtype=int)

clusters10 = labels[3].reshape(height,width)
clusters10 = np.array(clusters10,dtype=int)

clusters20 = labels[5].reshape(height,width)
clusters20 = np.array(clusters20,dtype=int)

clusters50 = labels[6].reshape(height,width)
clusters50 = np.array(clusters50,dtype=int)

fig, ax = plt.subplots(3, 2, figsize=(15, 15))

ax[0][0].imshow(orto, cmap=plt.cm.gray, interpolation='nearest')
ax[0][0].axis('off')
ax[0][0].set_title("orto")
```

```

ax[0][1].imshow(clusters2, cmap=plt.cm.gray, interpolation='nearest')
ax[0][1].axis('off')
ax[0][1].set_title("2 clusters")

ax[1][0].imshow(clusters6, cmap=plt.cm.gray, interpolation='nearest')
ax[1][0].axis('off')
ax[1][0].set_title("6 clusters")

ax[1][1].imshow(clusters10, cmap=plt.cm.gray, interpolation='nearest')
ax[1][1].axis('off')
ax[1][1].set_title("10 clusters")

ax[2][0].imshow(clusters20, cmap=plt.cm.gray, interpolation='nearest')
ax[2][0].axis('off')
ax[2][0].set_title("20 clusters")

ax[2][1].imshow(clusters50, cmap=plt.cm.gray, interpolation='nearest')
ax[2][1].axis('off')
ax[2][1].set_title("50 clusters")

filepath = os.path.join(imgdir,"kmeans_images.png")
plt.savefig(filepath)

```

790 520





E. Udskrift af koden: Mask R-CNN modelopsætning

"""

Mask R-CNN

Train on the toy Balloon dataset and implement color splash effect.

Copyright (c) 2018 Matterport, Inc.

Licensed under the MIT License (see LICENSE for details)

Written by Waleed Abdulla

Usage: import the module (see Jupyter notebooks for examples), or run from the command line as such:

```
# Train a new model starting from pre-trained COCO weights
python3 balloon.py train --dataset=/path/to/balloon/dataset --weights

# Resume training a model that you had trained earlier
python3 balloon.py train --dataset=/path/to/balloon/dataset --weights

# Train a new model starting from ImageNet weights
python3 balloon.py train --dataset=/path/to/balloon/dataset --weights

# Apply color splash to an image
python3 balloon.py splash --weights=/path/to/weights/file.h5 --image=

# Apply color splash to video using the last weights you trained
python3 balloon.py splash --weights=last --video=<URL or path to file
"""
```

```
import os
import sys
import datetime
import numpy as np
import geopandas as gpd
import skimage.draw
from shapely.geometry import Point

# Root directory of the project
ROOT_DIR = os.path.expanduser("~/Sync/IT-diplom/2019_1_BD/Projekt")
os.chdir(ROOT_DIR)

# Import Mask RCNN
RCNN_DIR = os.path.join(ROOT_DIR, "Mask_RCNN")
if not RCNN_DIR in sys.path:
    sys.path.append(RCNN_DIR)

from mrcnn.config import Config
from mrcnn import model as modellib, utils

# Path to trained weights file
COCO_WEIGHTS_PATH = os.path.join(RCNN_DIR, "mask_rcnn_coco.h5")
```

```

if (os.path.isfile(COCO_WEIGHTS_PATH)):
    print("COCO_WEIGHTS_PATH ok")
else:
    print("Missing COCO_WEIGHTS_PATH")

# Directory to save logs and model checkpoints, if not provided
# through the command line argument --logs
DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")
if (os.path.isdir(DEFAULT_LOGS_DIR)):
    print("DEFAULT_LOGS_DIR ok")
else:
    print("Missing DEFAULT_LOGS_DIR")

#####
# Configurations
#####

class GraveomrConfig(Config):
    """Configuration for training on the toy dataset.
    Derives from the base Config class and overrides some values.
    """
    # Give the configuration a recognizable name
    NAME = "Graveomr"

    # We use a GPU with 12GB memory, which can fit two images.
    # Adjust down if you use a smaller GPU.
    IMAGES_PER_GPU = 1

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1 # Background + balloon

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 100

    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9

#####
# Dataset
#####

class GraveomrDataset(utils.Dataset):

    def load_graveomr(self, df, subset):
        """Load a subset of the Balloon dataset.
        df: dataframe with subset of modeliomr.
        subset: Subset to load: train or val
        """

```

```

# Add classes. We have only one class to add.
self.add_class("graveomr", 1, "graveomr")

# Train or validation dataset?
assert subset in ["train", "val"]

for omr in df.iterrows():
    omr = omr[1]
    aoi_id = omr['aoi_id']
    DATADIR = os.path.join(ROOT_DIR, "data", aoi_id)
    polygons = gpd.read_file(os.path.join(DATADIR, "mask.shp"))

    self.add_image(
        "graveomr",
        image_id = omr['aoi_id'],
        path = os.path.join(DATADIR, "orto.png"),
        width=omr['width'],
        height=omr['height'],
        minx = omr['minx'],
        miny = omr['miny'],
        maxx = omr['maxx'],
        maxy = omr['maxy'],
        polygons=polygons[polygons.geometry.notnull()].geometry.v
    )

def load_mask(self, image_id):
    """Generate instance masks for an image.
    Returns:
    masks: A bool array of shape [height, width, instance count] with
           one mask per instance.
    class_ids: a 1D array of class IDs of the instance masks.
    """
    # If not a balloon dataset image, delegate to parent class.
    image_info = self.image_info[image_id]
    if image_info["source"] != "graveomr":
        return super(self.__class__, self).load_mask(image_id)

    # Convert polygons to a bitmap mask of shape
    # [height, width, instance_count]
    info = self.image_info[image_id]
    polygons = info['polygons']
    mask = np.zeros([info["height"], info["width"], len(polygons)], dtype=bool)

    for i, geom in enumerate(polygons):
        for ri, row in enumerate(mask[:, :, i]):
            y = info["maxy"] - (ri + 0.5)
            for ci, col in enumerate(row):
                x = info["minx"] + (ci + 0.5)
                if Point(x, y).within(geom):
                    mask[ri, ci, i] = 1

```

```

        # Return mask, and array of class IDs of each instance. Since we
        # one class ID only, we return an array of 1s
        return mask.astype(np.bool), np.ones([mask.shape[-1]], dtype=np.i

def image_reference(self, image_id):
    """Return the path of the image."""
    info = self.image_info[image_id]
    if info["source"] == "graveomr":
        return info["path"]
    else:
        super(self.__class__, self).image_reference(image_id)

def train(model):
    """Train the model."""
    aoi_shp = os.path.join(ROOT_DIR, "data/aoi.shp")
    modelomr = gpd.read_file(aoi_shp)
    msk = np.random.rand(len(modelomr)) < 0.8
    train_omr = modelomr[msk]
    val_omr = modelomr[~msk]

    # Training dataset.
    dataset_train = GraveomrDataset()
    dataset_train.load_graveomr(train_omr, "train")
    dataset_train.prepare()

    # Validation dataset
    dataset_val = GraveomrDataset()
    dataset_val.load_graveomr(val_omr, "val")
    dataset_val.prepare()

    # *** This training schedule is an example. Update to your needs ***
    # Since we're using a very small dataset, and starting from
    # COCO trained weights, we don't need to train too long. Also,
    # no need to train all layers, just the heads should do it.
    print("Training network heads")
    model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=30,
                layers='heads')

def color_splash(image, mask):
    """Apply color splash effect.
    image: RGB image [height, width, 3]
    mask: instance segmentation mask [height, width, instance count]

    Returns result image.
    """
    # Make a grayscale copy of the image. The grayscale copy still

```

```

# has 3 RGB channels, though.
gray = skimage.color.gray2rgb(skimage.color.rgb2gray(image)) * 255
# Copy color pixels from the original color image where mask is set
if mask.shape[-1] > 0:
    # We're treating all instances as one, so collapse the mask into
    mask = (np.sum(mask, -1, keepdims=True) >= 1)
    splash = np.where(mask, image, gray).astype(np.uint8)
else:
    splash = gray.astype(np.uint8)
return splash

def detect_and_color_splash(model, image_path=None, video_path=None):
    assert image_path or video_path

    # Image or video?
    if image_path:
        # Run model detection and generate the color splash effect
        print("Running on {}".format(args.image))
        # Read image
        image = skimage.io.imread(args.image)
        # Detect objects
        r = model.detect([image], verbose=1)[0]
        # Color splash
        splash = color_splash(image, r['masks'])
        # Save output
        file_name = "splash_{:%Y%m%dT%H%M%S}.png".format(datetime.datetime.now())
        skimage.io.imsave(file_name, splash)
    elif video_path:
        import cv2
        # Video capture
        vcapture = cv2.VideoCapture(video_path)
        width = int(vcapture.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vcapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = vcapture.get(cv2.CAP_PROP_FPS)

        # Define codec and create video writer
        file_name = "splash_{:%Y%m%dT%H%M%S}.avi".format(datetime.datetime.now())
        vwriter = cv2.VideoWriter(file_name,
                                  cv2.VideoWriter_fourcc(*'MJPG'),
                                  fps, (width, height))

        count = 0
        success = True
        while success:
            print("frame: ", count)
            # Read next image
            success, image = vcapture.read()
            if success:
                # OpenCV returns images as BGR, convert to RGB
                image = image[..., ::-1]

```

```

        # Detect objects
        r = model.detect([image], verbose=0)[0]
        # Color splash
        splash = color_splash(image, r['masks'])
        # RGB -> BGR to save image to video
        splash = splash[..., ::-1]
        # Add image to video writer
        vwriter.write(splash)
        count += 1
    vwriter.release()
    print("Saved to ", file_name)

#####
# Training
#####

if __name__ == '__main__':
    import argparse

    # Parse command line arguments
    parser = argparse.ArgumentParser(
        description='Train Mask R-CNN to detect graveomr.')
    parser.add_argument("command",
                        metavar("<command>",
                        help="'train' or 'splash'")
    parser.add_argument('--dataset', required=False,
                        metavar="/path/to/balloon/dataset/",
                        help='Directory of the Graveomr dataset')
    parser.add_argument('--weights', required=True,
                        metavar="/path/to/weights.h5",
                        help="Path to weights .h5 file or 'coco'")
    parser.add_argument('--logs', required=False,
                        default=DEFAULT_LOGS_DIR,
                        metavar="/path/to/logs/",
                        help='Logs and checkpoints directory (default=log
    parser.add_argument('--image', required=False,
                        metavar="path or URL to image",
                        help='Image to apply the color splash effect on')
    parser.add_argument('--video', required=False,
                        metavar="path or URL to video",
                        help='Video to apply the color splash effect on')
    args = parser.parse_args()

    print("\n\nCurrent directory", os.getcwd())

    # Validate arguments
    if args.command == "train":
        assert args.dataset, "Argument --dataset is required for training
    elif args.command == "splash":
        assert args.image or args.video, \

```

```

        "Provide --image or --video to apply color splash"

print("Weights: ", args.weights)
print("Dataset: ", args.dataset)
print("Logs: ", args.logs)

# Configurations
if args.command == "train":
    config = GraveomrConfig()
else:
    class InferenceConfig(GraveomrConfig):
        # Set batch size to 1 since we'll be running inference on
        # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GP
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1
    config = InferenceConfig()
config.display()

# Create model
if args.command == "train":
    model = modellib.MaskRCNN(mode="training", config=config,
                              model_dir=args.logs)
else:
    model = modellib.MaskRCNN(mode="inference", config=config,
                              model_dir=args.logs)

# Select weights file to load
if args.weights.lower() == "coco":
    weights_path = COCO_WEIGHTS_PATH
    # Download weights file
    if not os.path.exists(weights_path):
        utils.download_trained_weights(weights_path)
elif args.weights.lower() == "last":
    # Find last trained weights
    weights_path = model.find_last()
elif args.weights.lower() == "imagenet":
    # Start from ImageNet trained weights
    weights_path = model.get_imagenet_weights()
else:
    weights_path = args.weights

# Load weights
print("Loading weights ", weights_path)
if args.weights.lower() == "coco":
    # Exclude the last layers because they require a matching
    # number of classes
    model.load_weights(weights_path, by_name=True, exclude=[
        "mrcnn_class_logits", "mrcnn_bbox_fc",
        "mrcnn_bbox", "mrcnn_mask"])
else:
    model.load_weights(weights_path, by_name=True)

```

```
# Train or evaluate
if args.command == "train":
    train(model)
elif args.command == "splash":
    detect_and_color_splash(model, image_path=args.image,
                             video_path=args.video)
else:
    print("'{}' is not recognized. "
          "Use 'train' or 'splash'".format(args.command))

print(datetime.timedelta(starttime, datetime.datetime))
```


F. Notebook: Mask R-CNN inspect data

Mask R-CNN - Inspect Balloon Training Data

Inspect and visualize data loading and pre-processing code.

```
In [11]: import os
import sys
import itertools
import math
import logging
import json
import re
import random
from collections import OrderedDict
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon
import pandas as pd
import geopandas as gpd

# Root directory of the project
ROOT_DIR = os.path.abspath(".")

# Import Mask RCNN
sys.path.append(os.path.join(ROOT_DIR, "lib")) # To find local version of the lib
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

import mask_rcnn

%matplotlib inline
```

Configurations

Configurations are defined in balloon.py

```
In [12]: config = mask_rcnn.GraveomrConfig()
DATA_DIR = os.path.join(ROOT_DIR, "data")
```

Dataset

```
In [13]: # Load dataset
# Get the dataset from the releases page
# https://github.com/matterport/Mask_RCNN/releases
aoi_shp = os.path.join(ROOT_DIR, "data/aoi.shp")
modelomr = gpd.read_file(aoi_shp)
msk = np.random.rand(len(modelomr)) < 0.8
train_omr = modelomr[msk]
val_omr = modelomr[~msk]

dataset = mask_rcnn.GraveomrDataset()
dataset.load_graveomr(train_omr, "train")

# Must call before using the dataset
dataset.prepare()

print("Image Count: {}".format(len(dataset.image_ids)))
print("Class Count: {}".format(dataset.num_classes))
for i, info in enumerate(dataset.class_info):
    print("{:3} {:50} {}".format(i, info['name'],
```

```
Image Count: 15
Class Count: 2
0. BG
1. graveomr
```

Display Samples

Load and display images and masks.

```
In [14]: # Load and display random samples
image_ids = np.random.choice(dataset.image_ids, 4)
for image_id in image_ids:
    image = dataset.load_image(image_id)
    mask, class_ids = dataset.load_mask(image_id)
    visualize_display_top_masks(image, mask, class_ids, dataset.class_names)
```



Bounding Boxes

Rather than using bounding box coordinates provided by the source datasets, we compute the bounding boxes from masks instead. This allows us to handle bounding boxes consistently regardless of the source dataset, and it also makes it easier to resize, rotate, or crop images because we simply generate the bounding boxes from the updated masks rather than computing bounding box transformation for each type of image transformation.

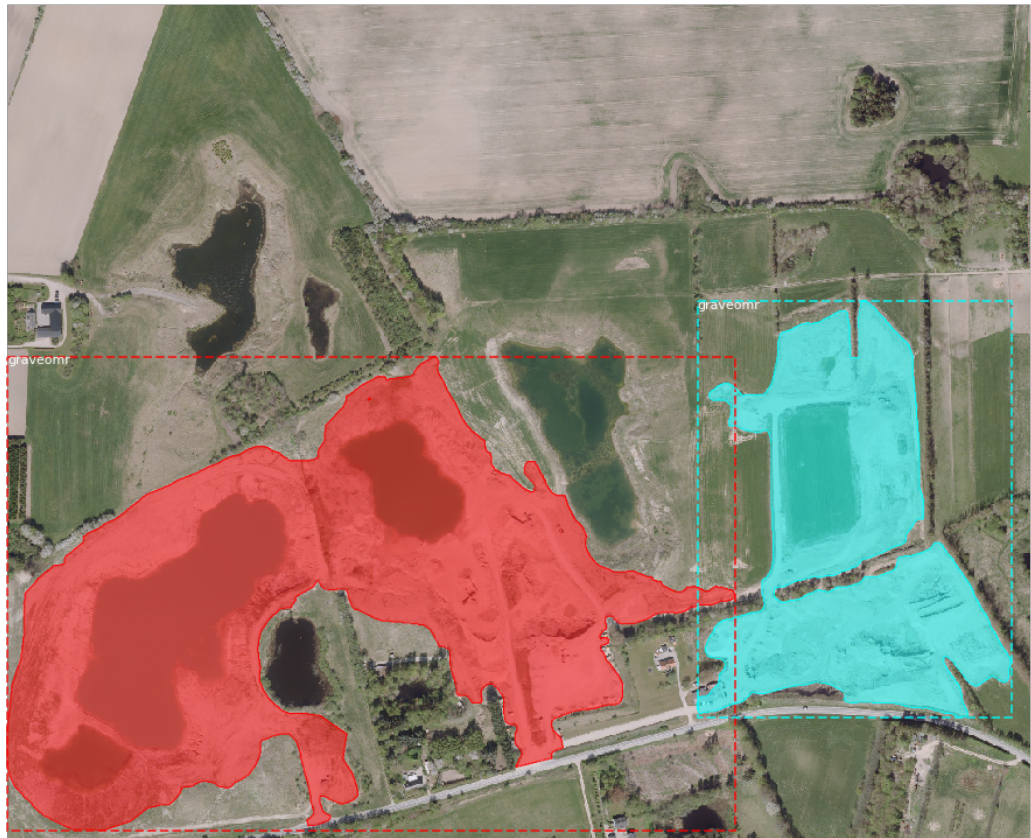
```

In [15]: # Load random image and mask.
#image_id = random.choice(dataset.image_ids)
#image = dataset.load_image(image_id)
#mask, class_ids = dataset.load_mask(image_id)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id ", image_id, dataset.image_reference(image_id))
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize_display_instances(image, bbox, mask, class_ids, dataset.class_names)

image_id 5 /home/ak/Sync/IT-diplom/2019_1_BD/Projekt/data/80ABB9ED-B4B6-4B
68-902B-5F073617A1D4/orto.png
image          shape: (887, 1085, 3)          min:    9.00000  max:
255.00000  uint8
mask          shape: (887, 1085, 2)          min:    0.00000  max:
1.00000  bool
class_ids     shape: (2,)                    min:    1.00000  max:
1.00000  int32
bbox          shape: (2, 4)                  min:    0.00000  max:
1065.00000  int32

```



Resize Images

To support multiple images per batch, images are resized to one size (1024x1024). Aspect ratio is preserved, though. If an image is not square, then zero padding is added at the top/bottom or right/left.

```

In [18]: # Load random image and mask.
#image_id = np.random.choice(dataset.image_ids, 1)[0]
#image = dataset.load_image(image_id)
#mask, class_ids = dataset.load_mask(image_id)
original_shape = image.shape
# Resize
image, window, scale, padding, _ = utils.resize_image(
    image,
    min_dim=config.IMAGE_MIN_DIM,
    max_dim=config.IMAGE_MAX_DIM,
    mode=config.IMAGE_RESIZE_MODE)
mask = utils.resize_mask(mask, scale, padding)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id: ", image_id, dataset.image_reference(image_id))
print("Original shape: ", original_shape)
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize_instances(image, bbox, mask, class_ids, dataset.class_names)

image_id: 5 /home/ak/Sync/IT-diplom/2019_1_BD/Projekt/data/80ABB9ED-B4B6-4B
68-902B-5F073617A1D4/orto.png
Original shape: (1024, 1024, 3)
image                shape: (1024, 1024, 3)        min:    0.00000    max:
250.00000  uint8
mask              shape: (1024, 1024, 2)        min:    0.00000    max:
1.00000  bool
class_ids         shape: (2,)                    min:    1.00000    max:
1.00000  int32
bbox              shape: (2, 4)                    min:    0.00000    max:
1005.00000  int32

```



Mini Masks

Instance binary masks can get large when training with high resolution images. For example, if training with 1024x1024 image then the mask of a single instance requires 1MB of memory (Numpy uses bytes for boolean values). If an image has 100 instances then that's 100MB for the masks alone.

To improve training speed, we optimize masks by:

- We store mask pixels that are inside the object bounding box, rather than a mask of the full image. Most objects are small compared to the image size, so we save space by not storing a lot of zeros around the object.
- We resize the mask to a smaller size (e.g. 56x56). For objects that are larger than the selected size we lose a bit of accuracy. But most object annotations are not very accuracy to begin with, so this loss is negligible for most practical purposes. Thie size of the mini_mask can be set in the config class.

To visualize the effect of mask resizing, and to verify the code correctness, we visualize some examples.

```
In [21]: #image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, use_mini_mask=False)

log("image", image)
log("image_meta", image_meta)
log("class_ids", class_ids)
log("bbox", bbox)
log("mask", mask)

display_images([image]+[mask[i] for i in range(min(mask.shape[-1], 7))])
```

image	shape: (1024, 1024, 3)	min: 0.00000	max:
250.00000 uint8			
image_meta	shape: (14,)	min: 0.00000	max:
1085.00000 float64			
class_ids	shape: (2,)	min: 1.00000	max:
1.00000 int32			
bbox	shape: (2, 4)	min: 0.00000	max:
1005.00000 int32			
mask	shape: (1024, 1024, 2)	min: 0.00000	max:
1.00000 bool			



In [22]: `visualize_display_instances(image, bbox, mask, class_ids, dataset_class_names)`

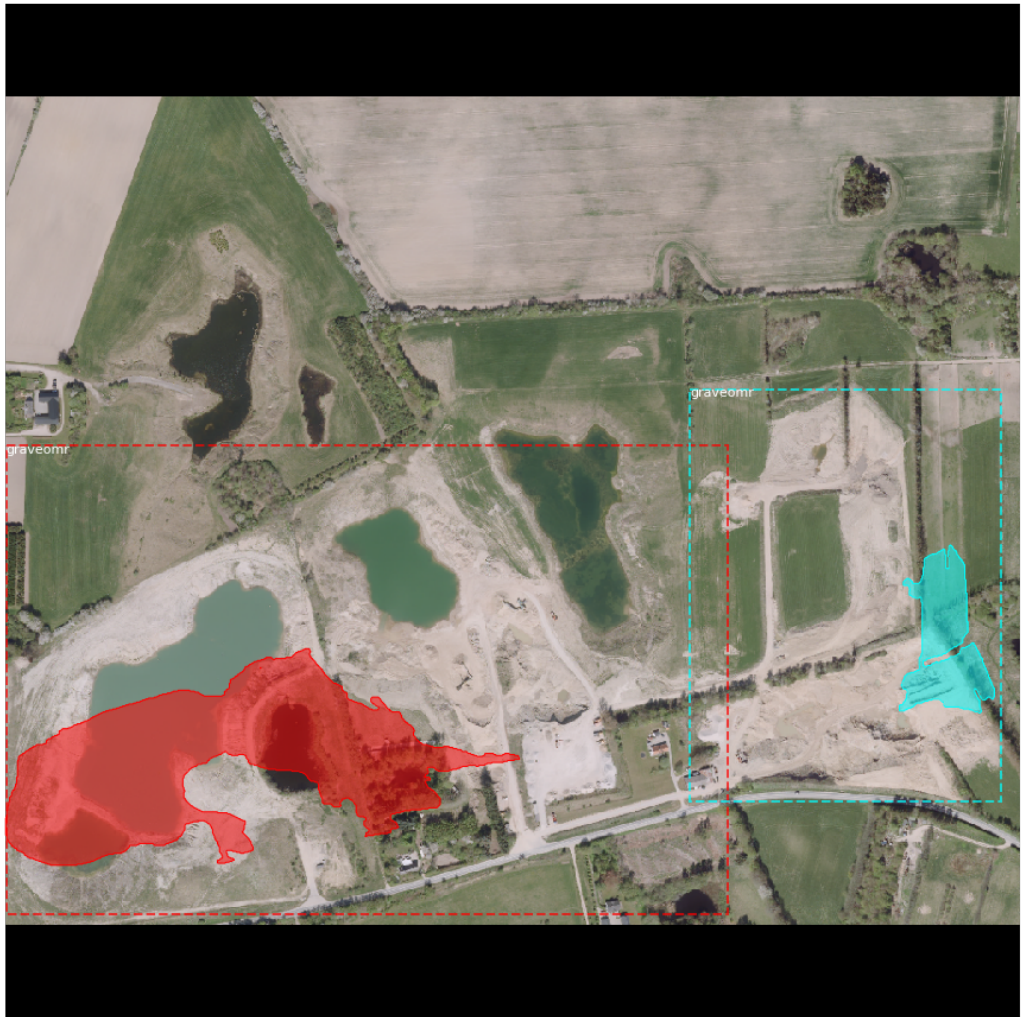


```
In [23]: # Add augmentation and mask resizing.
#image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
#    dataset, config, image_id, augment=True, use_mini_mask=True)
log("mask", mask)
display_images([image] + [mask[i] for i in range(min(mask.shape[-1], 7))])

mask
1.00000  bool          shape: (1024, 1024, 2)      min:  0.00000  max:
```



```
In [24]: mask = utils.expand_mask(bbox, mask, image.shape)
visualize_display_instances(image, bbox, mask, class_ids, dataset_class_names)
```



Anchors

The order of anchors is important. Use the same order in training and prediction phases. And it must match the order of the convolution execution.

For an FPN network, the anchors must be ordered in a way that makes it easy to match anchors to the output of the convolution layers that predict anchor scores and shifts.

- Sort by pyramid level first. All anchors of the first level, then all of the second and so on. This makes it easier to separate anchors by level.
- Within each level, sort anchors by feature map processing sequence. Typically, a convolution layer processes a feature map starting from top-left and moving right row by row.
- For each feature map cell, pick any sorting order for the anchors of different ratios. Here we match the order of ratios passed to the function.

Anchor Stride: In the FPN architecture, feature maps at the first few layers are high resolution. For example, if the input image is 1024x1024 then the feature map of the first layer is 256x256, which generates about 200K anchors (256x256). These anchors are 32x32 pixels and their stride relative to image pixels is 4 pixels, so there is a lot of overlap. We can reduce the load significantly if we generate anchors for every other cell in the feature map. A stride of 2 will cut the number of anchors by 4, for example.

In this implementation we use an anchor stride of 2, which is different from the paper.

```
In [25]: # Generate Anchors
backbone_shapes = modellib.compute_backbone_shapes(config, config.IMAGE_SHAPE)
anchors = utils.generate_pyramid_anchors(config.RPN_ANCHOR_SCALES,
                                         config.RPN_ANCHOR_RATIOS,
                                         backbone_shapes,
                                         config.BACKBONE_STRIDES,
                                         config.RPN_ANCHOR_STRIDE)

# Print summary of anchors
num_levels = len(backbone_shapes)
anchors_per_cell = len(config.RPN_ANCHOR_RATIOS)
print("Count: ", anchors.shape[0])
print("Scales: ", config.RPN_ANCHOR_SCALES)
print("ratios: ", config.RPN_ANCHOR_RATIOS)
print("Anchors per Cell: ", anchors_per_cell)
print("Levels: ", num_levels)
anchors_per_level = []
for l in range(num_levels):
    num_cells = backbone_shapes[l][0] * backbone_shapes[l][1]
    anchors_per_level.append(anchors_per_cell * num_cells // config.RPN_ANCHOR_STI
    print("Anchors in Level {}: {}".format(l, anchors_per_level[l]))
```

```
Count: 261888
Scales: (32, 64, 128, 256, 512)
ratios: [0.5, 1, 2]
Anchors per Cell: 3
Levels: 5
Anchors in Level 0: 196608
Anchors in Level 1: 49152
Anchors in Level 2: 12288
Anchors in Level 3: 3072
Anchors in Level 4: 768
```

Visualize anchors of one cell at the center of the feature map of a specific level.

In [26]: *## Visualize anchors of one cell at the center of the feature map of a specific level*

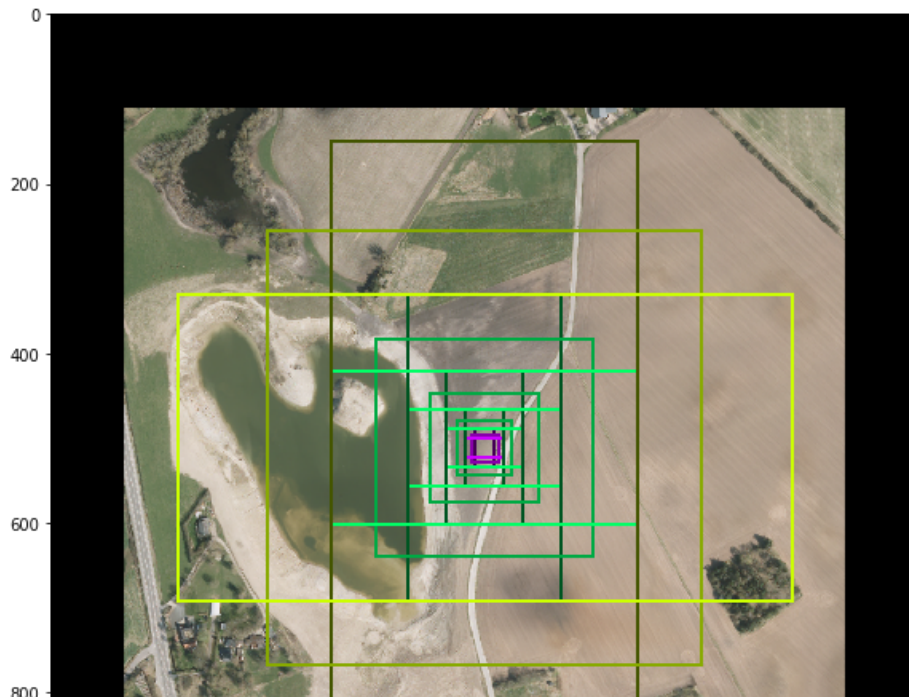
```
# Load and draw random image
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, _, _, _ = modellib.load_image_gt(dataset, config, image_id)
fig, ax = plt.subplots(1, figsize=(10, 10))
ax.imshow(image)
levels = len(backbone_shapes)

for level in range(levels):
    colors = visualize.random_colors(levels)
    # Compute the index of the anchors at the center of the image
    level_start = sum(anchors_per_level[:level]) # sum of anchors of previous levels
    level_anchors = anchors[level_start:level_start+anchors_per_level[level]]
    print("Level {}. Anchors: {:6} Feature map Shape: {}".format(level, level_anchors, backbone_shapes[level]))

    center_cell = backbone_shapes[level] // 2
    center_cell_index = (center_cell[0] * backbone_shapes[level][1] + center_cell[1])
    level_center = center_cell_index * anchors_per_cell
    center_anchor = anchors_per_cell * (
        (center_cell[0] * backbone_shapes[level][1] / config.RPN_ANCHOR_STRIDE**2
         + center_cell[1] / config.RPN_ANCHOR_STRIDE)
    )
    level_center = int(center_anchor)

    # Draw anchors. Brightness show the order in the array, dark to bright.
    for i, rect in enumerate(level_anchors[level_center:level_center+anchors_per_level[level]]):
        y1, x1, y2, x2 = rect
        p = patches.Rectangle((x1, y1), x2-x1, y2-y1, linewidth=2, facecolor='none',
                               edgecolor=(i+1)*np.array(colors[level]) / anchors_per_level[level])
        ax.add_patch(p)
```

```
Level 0. Anchors: 196608 Feature map Shape: [256 256]
Level 1. Anchors: 49152 Feature map Shape: [128 128]
Level 2. Anchors: 12288 Feature map Shape: [64 64]
Level 3. Anchors: 3072 Feature map Shape: [32 32]
Level 4. Anchors: 768 Feature map Shape: [16 16]
```



Data Generator

```
In [27]: # Create data generator
random_rois = 2000
g = modellib.data_generator(
    dataset, config, shuffle=True, random_rois=random_rois,
    batch_size=4,
    detection_targets=True)
```

```
In [28]: # Uncomment to run the generator through a lot of images
# to catch rare errors
# for i in range(1000):
#     print(i)
#     = next(g)
```

```
In [29]: # Get Next Image
if random_rois:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_class_ids, gt_boxes,
    [mrcnn_class_ids, mrcnn_bbox, mrcnn_mask] = next(g)

    log("rois", rois)
    log("mrcnn_class_ids", mrcnn_class_ids)
    log("mrcnn_bbox", mrcnn_bbox)
    log("mrcnn_mask", mrcnn_mask)
else:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_boxes, gt_masks], _ =

log("gt_class_ids", gt_class_ids)
log("gt_boxes", gt_boxes)
log("gt_masks", gt_masks)
log("rpn_match", rpn_match, )
log("rpn_bbox", rpn_bbox)
image_id = modellib.parse_image_meta(image_meta)["image_id"][0]
print("image_id: ", image_id, dataset.image_reference(image_id))

# Remove the last dim in mrcnn_class_ids. It's only added
# to satisfy Keras restriction on target shape.
mrcnn_class_ids = mrcnn_class_ids[:, : -1]
```

```
rois                                shape: (4, 200, 4)           min:    0.00000  max:
1023.00000  int32
mrcnn_class_ids                     shape: (4, 200, 1)           min:    0.00000  max:
1.00000  int32
mrcnn_bbox                           shape: (4, 200, 2, 4)       min:   -3.68421  max:
3.27273  float32
mrcnn_mask                           shape: (4, 200, 28, 28, 2)  min:    0.00000  max:
1.00000  float32
gt_class_ids                         shape: (4, 100)             min:    0.00000  max:
1.00000  int32
gt_boxes                             shape: (4, 100, 4)          min:    0.00000  max:
966.00000  int32
gt_masks                             shape: (4, 56, 56, 100)    min:    0.00000  max:
1.00000  bool
rpn_match                           shape: (4, 261888, 1)       min:   -1.00000  max:
1.00000  int32
rpn_bbox                             shape: (4, 256, 4)          min:   -3.45703  max:
3.45703  float64
image_id:  6 /home/ak/Sync/IT-diplom/2019_1_BD/Projekt/data/1F5D59FB-C80B-4789-B
252-82EA00ABB3DC/orto.png
```

In [30]: b = 0

```

# Restore original image (reverse normalization)
sample_image = modellib.unmold_image(normalized_images[b], config)

# Compute anchor shifts.
indices = np.where(rpn_match[b] == 1)[0]
refined_anchors = utils.apply_box_deltas(anchors[indices], rpn_bbox[b, :len(indices)])
log("anchors", anchors)
log("refined_anchors", refined_anchors)

# Get list of positive anchors
positive_anchor_ids = np.where(rpn_match[b] == 1)[0]
print("Positive anchors: {}".format(len(positive_anchor_ids)))
negative_anchor_ids = np.where(rpn_match[b] == -1)[0]
print("Negative anchors: {}".format(len(negative_anchor_ids)))
neutral_anchor_ids = np.where(rpn_match[b] == 0)[0]
print("Neutral anchors: {}".format(len(neutral_anchor_ids)))

# ROI breakdown by class
for c, n in zip(dataset.class_names, np.bincount(mrcnn_class_ids[b].flatten())):
    if n:
        print("{:23}: {}".format(c[:20], n))

# Show positive anchors
fig, ax = plt.subplots(1, figsize=(16, 16))
visualize.draw_boxes(sample_image, boxes=anchors[positive_anchor_ids],
                    refined_boxes=refined_anchors, ax=ax)

```

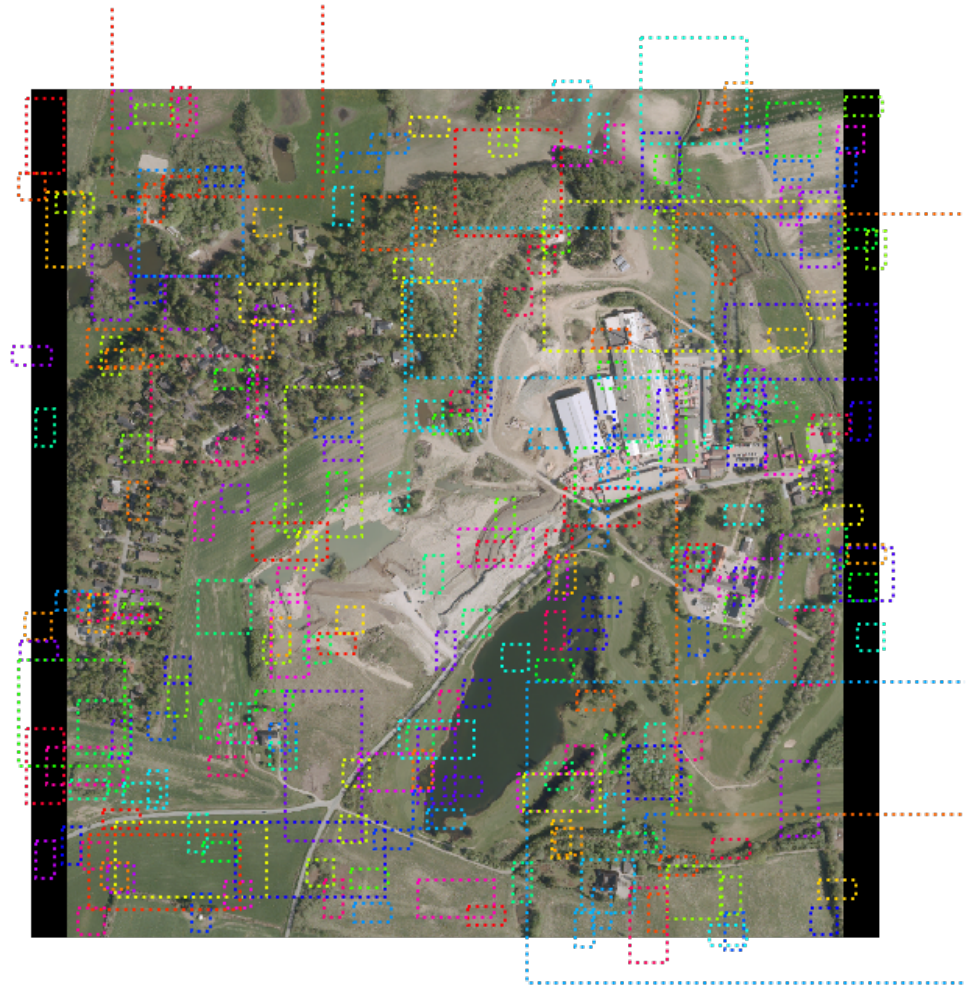
```

anchors                shape: (261888, 4)                min: -362.03867  max:
1322.03867  float64
refined_anchors        shape: (6, 4)                min: 243.000000  max:
723.000000  float32
Positive anchors: 6
Negative anchors: 250
Neutral anchors: 261632
BG                : 155
graveomr          : 45

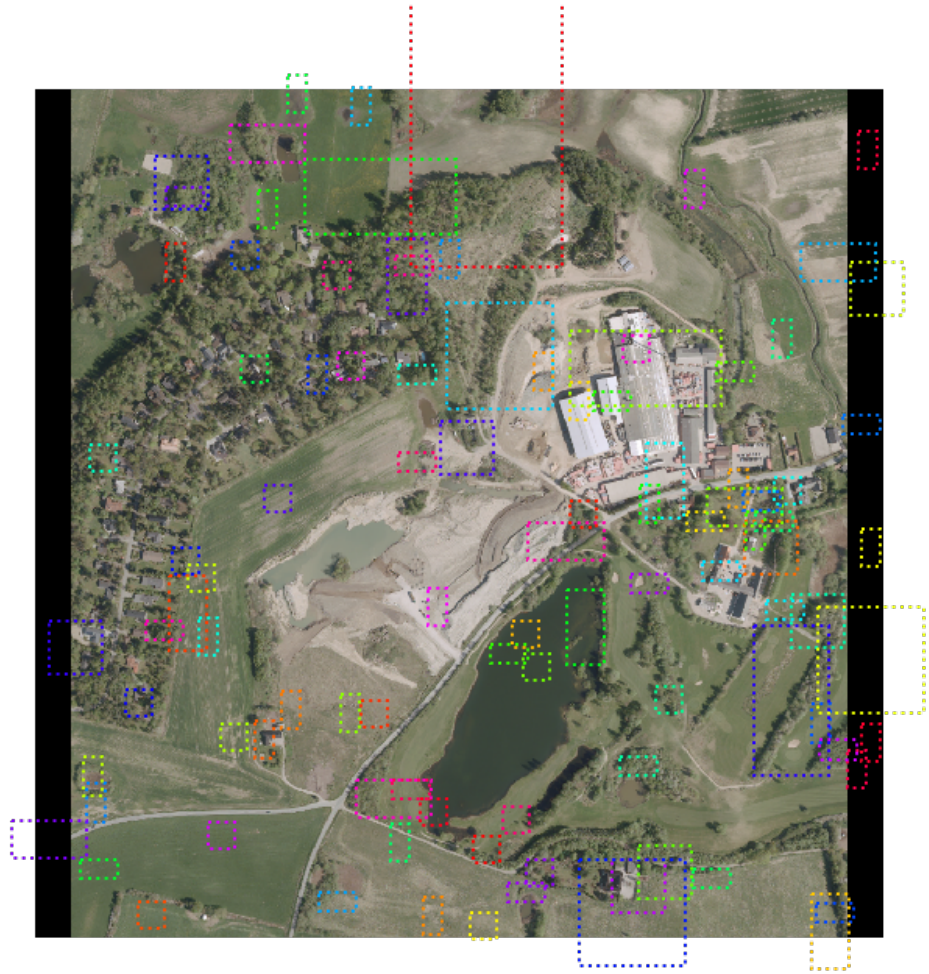
```



```
In [31]: # Show negative anchors  
visualize_draw_boxes(sample_image, boxes=anchors[negative_anchor_ids])
```




```
In [32]: # Show neutral anchors. They don't contribute to training.  
visualize_draw_boxes(sample_image, boxes=anchors[no_random_choice(neutral_anchor
```



ROIs

```
In [33]: if random_rois:
# Class aware bboxes
bbox_specific = mrcnn_bbox[b, np.arange(mrcnn_bbox.shape[1]), mrcnn_class_ids]

# Refined ROIs
refined_rois = utils.apply_box_deltas(rois[b].astype(np.float32), bbox_specific)

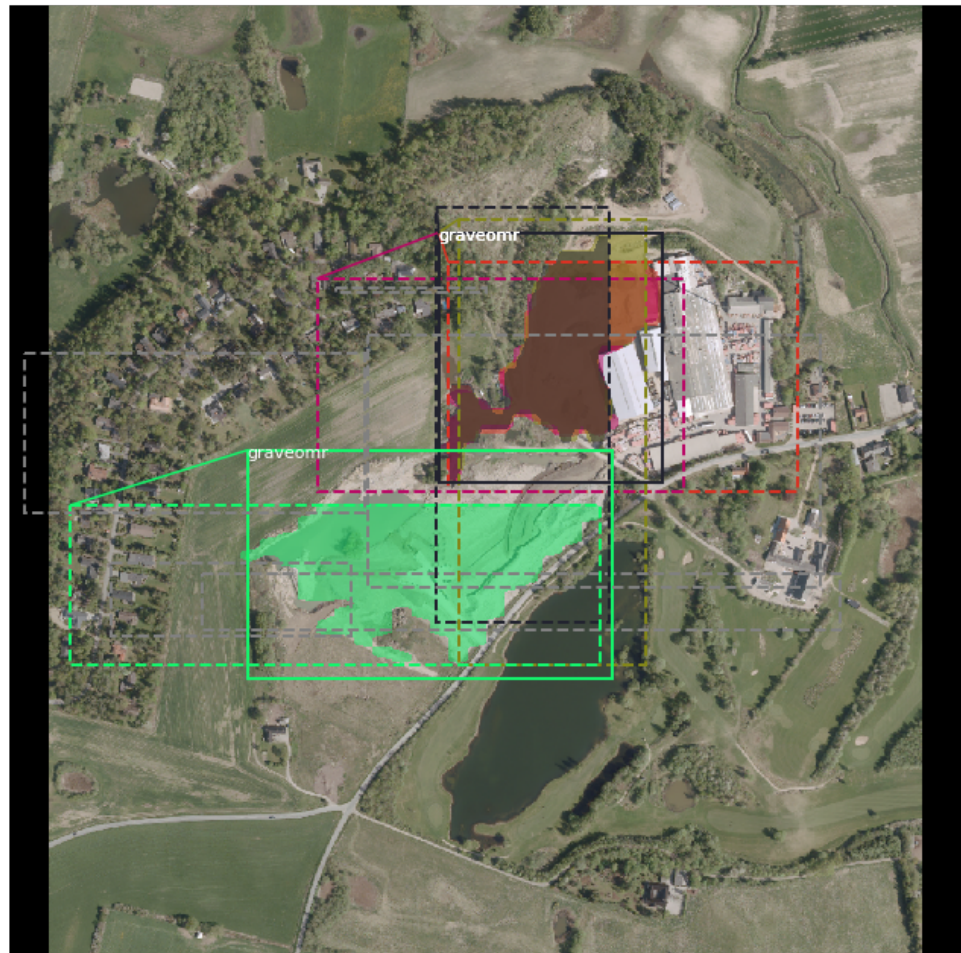
# Class aware masks
mask_specific = mrcnn_mask[b, np.arange(mrcnn_mask.shape[1]), :, :, mrcnn_class_ids]

visualize.draw_rois(sample_image, rois[b], refined_rois, mask_specific, mrcnn_class_ids)

# Any repeated ROIs?
rows = np.ascontiguousarray(rois[b]).view(np.dtype((np.void, rois.dtype.items[0].size)),
_, idx = np.unique(rows, return_index=True)
print("Unique ROIs: {} out of {}".format(len(idx), rois.shape[1]))
```

Positive ROIs: 45
 Negative ROIs: 155
 Positive Ratio: 0.23
 Unique ROIs: 200 out of 200

Showing 10 random ROIs out of 200



```
In [34]: if random_rois:
# Display ROIs and corresponding masks and bounding boxes
ids = random.sample(range(rois.shape[1]), 8)

images = []
titles = []
for i in ids:
    image = visualize.draw_box(sample_image.copy(), rois[b,i,:4].astype(np.int64))
    image = visualize.draw_box(image, refined_rois[i].astype(np.int64), [0, 255])
    images.append(image)
    titles.append("ROI {}".format(i))
    images.append(mask_specific[i] * 255)
    titles.append(dataset.class_names[mrcnn_class_ids[b,i]][:20])

display_images(images, titles, cols=4, cmap="Blues", interpolation="none")
```



```
In [35]: # Check ratio of positive ROIs in a set of images.
if random_rois:
    limit = 10
    temp_g = modellib.data_generator(
        dataset, config, shuffle=True, random_rois=10000,
        batch_size=1, detection_targets=True)
    total = 0
    for i in range(limit):
        _, [ids, _, _] = next(temp_g)
        positive_rois = np.sum(ids[0] > 0)
        total += positive_rois
        print("{:5} {:.2f}".format(positive_rois, positive_rois/ids.shape[1]))
    print("Average percent: {:.2f}".format(total/(limit*ids.shape[1])))

66 0.33
66 0.33
66 0.33
66 0.33
66 0.33
66 0.33
66 0.33
66 0.33
66 0.33
66 0.33
Average percent: 0.33
```

In []:

G. Notebook: Mask R-CNN inspect model

Mask R-CNN - Inspect Ballon Trained Model

Code and visualizations to test, debug, and evaluate the Mask R-CNN model.

```
In [1]: import os
import sys
import random
import math
import re
import time
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import pandas as pd
import geopandas as gpd

# Root directory of the project
ROOT_DIR = os.path.abspath(".")
MRCNN_DIR = os.path.abspath("./lib/")
print(MRCNN_DIR)

# Import Mask RCNN
sys.path.append(MRCNN_DIR) # To find local version of the library
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

import mask_rcnn

%matplotlib inline

# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# Path to Ballon trained weights
# You can download this file from the Releases page
# https://github.com/matterport/Mask_RCNN/releases
BALLOON_WEIGHTS_PATH = "/Mask_RCNN/mask_rcnn_balloon_h5" # TODO: update this path
/home/ak/Sync/IT-diplom/2019_1_BD/Projekt/lib
COCO_WEIGHTS_PATH ok
DEFAULT_LOGS_DIR ok

Using TensorFlow backend.
```

Configurations

```
In [2]: config = mask_rcnn.GraveomrConfig()
DATA_DIR = os.path.join(ROOT_DIR, "data")
print(config.NAME)
Graveomr
```


In [3]: *# Override the training configurations with a few
changes for inferencing.*

```
class InferenceConfig(config.__class__):
    # Run detection on one image at a time
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
```

```
config = InferenceConfig()
config.display()
```

Configurations:

```
BACKBONE                resnet101
BACKBONE_STRIDES        [4, 8, 16, 32, 64]
BATCH_SIZE              1
BBOX_STD_DEV            [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE  None
DETECTION_MAX_INSTANCES 100
DETECTION_MIN_CONFIDENCE 0.9
DETECTION_NMS_THRESHOLD 0.3
FPN_CLASSIF_FC_LAYERS_SIZE 1024
GPU_COUNT              1
GRADIENT_CLIP_NORM      5.0
IMAGES_PER_GPU          1
IMAGE_CHANNEL_COUNT      3
IMAGE_MAX_DIM           1024
IMAGE_META_SIZE         14
IMAGE_MIN_DIM           800
IMAGE_MIN_SCALE         0
IMAGE_RESIZE_MODE        square
IMAGE_SHAPE             [1024 1024   3]
LEARNING_MOMENTUM        0.9
LEARNING_RATE           0.001
LOSS_WEIGHTS             {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mr
cnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE          14
MASK_SHAPE              [28, 28]
MAX_GT_INSTANCES        100
MEAN_PIXEL              [123.7 116.8 103.9]
MINI_MASK_SHAPE         (56, 56)
NAME                    Graveomr
NUM_CLASSES              2
POOL_SIZE               7
POST_NMS_ROIS_INFERENCE 1000
POST_NMS_ROIS_TRAINING  2000
PRE_NMS_LIMIT           6000
ROI_POSITIVE_RATIO       0.33
RPN_ANCHOR_RATIOS       [0.5, 1, 2]
RPN_ANCHOR_SCALES       (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE       1
RPN_BBOX_STD_DEV        [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD       0.7
RPN_TRAIN_ANCHORS_PER_IMAGE 256
STEPS_PER_EPOCH         100
TOP_DOWN_PYRAMID_SIZE   256
TRAIN_BN                False
TRAIN_ROIS_PER_IMAGE     200
USE_MINI_MASK           True
USE_RPN_ROIS            True
VALIDATION_STEPS         50
WEIGHT_DECAY            0.0001
```

Notebook Preferences

```
In [4]: # Device to load the neural network on.
# Useful if you're training a model on the same
# machine, in which case use CPU and leave the
# GPU for training.
DEVICE = "/gpu:0" # /cpu:0 or /gpu:0

# Inspect the model in training or inference modes
# values: 'inference' or 'training'
# TODO: code for 'training' test mode not ready yet
TEST_MODE = "inference"
```

```
In [5]: def get_ax(rows=1, cols=1, size=16):
        """Return a Matplotlib Axes array to be used in
        all visualizations in the notebook. Provide a
        central point to control graph sizes.

        Adjust the size attribute to control how big to render images
        """
        _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
        return ax
```

Load Validation Dataset

```
In [6]: # Load validation dataset
aoi_shp = os.path.join(ROOT_DIR, "data/aoi.shp")
modelomr = gpd.read_file(aoi_shp)
msk = np.random.rand(len(modelomr)) < 0.8
train_omr = modelomr[msk]
val_omr = modelomr[~msk]

dataset = mask_rcnn.GraveomrDataset()
dataset.load_graveomr(val_omr, "val")

# Must call before using the dataset
dataset.prepare()

print("Images: {}\nClasses: {}".format(len(dataset.image_ids), dataset.class_names))
Images: 3
Classes: ['BG', 'graveomr']
```

Load Model

In [7]: *# Create model in inference mode*

```

with tf.device(DEVICE):
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
                              config=config)
print(model.model_dir)

```

WARNING:tensorflow:From /home/ak/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

WARNING:tensorflow:From /home/ak/Sync/IT-diplom/2019_1_BD/Projekt/lib/mrcnn/model.py:772: to_float (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

/home/ak/Sync/IT-diplom/2019_1_BD/Projekt/logs

In [8]:

```
dir_names = next(os.walk(model.model_dir))[1]
```

```
print(dir_names)
```

```
key = model.config.NAME.lower()
```

```
print(key)
```

```
print(dir_names[0], dir_names[0].startswith(key))
```

```
dir_names = filter(lambda f: f.startswith(key), dir_names)
```

```
dir_names = sorted(dir_names)
```

```
for name in dir_names:
```

```
    print(name)
```

```
['graveomr20190518T0940']
```

```
graveomr
```

```
graveomr20190518T0940 True
```

```
graveomr20190518T0940
```

In [9]: *# Set path to balloon weights file*

```
# Download file from the Releases page and set its path
```

```
# https://github.com/matterport/Mask_RCNN/releases
```

```
# weights_path = "/path/to/mask_rcnn_balloon.h5"
```

```
# Or, load the last model you trained
```

```
weights_path = model.find_last()
```

```
# Load weights
```

```
print("Loading weights ", weights_path)
```

```
model.load_weights(weights_path, by_name=True)
```

Loading weights /home/ak/Sync/IT-diplom/2019_1_BD/Projekt/logs/graveomr20190518T0940/mask_rcnn_graveomrâder_0030.h5

Re-starting from epoch 30

Run Detection

```

In [10]: image_id = random.choice(dataset.image_ids)
image, image_meta, gt_class_id, gt_bbox, gt_mask = \
    modellib.load_image_gt(dataset, config, image_id, use_mini_mask=False)
info = dataset.image_info[image_id]
print("image ID: {}.{} ({}). {}".format(info["source"], info["id"], image_id,
                                         dataset.image_reference(image_id)))

# Run object detection
results = model.detect([image], verbose=1)

# Display results
ax = get_ax(1)
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                           dataset.class_names, r['scores'], ax=ax,
                           title="Predictions")

log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)

image ID: graveomr.A4481F38-9384-4B59-9AAF-CD7C5FF47238 (2) /home/ak/Sync/IT-dip
lom/2019_1_BD/Projekt/data/A4481F38-9384-4B59-9AAF-CD7C5FF47238/orto.png
Processing 1 images
image                shape: (1024, 1024, 3)          min: 0.00000 max:
254.00000  uint8
molded_images        shape: (1, 1024, 1024, 3)      min: -123.70000 max:
150.10000  float64
image metas          shape: (1, 14)                  min: 0.00000 max:
1024.00000  int64
anchors              shape: (1, 261888, 4)              min: -0.35390 max:
1.29134  float32
gt_class_id           shape: (2,)                    min: 1.00000 max:
1.00000  int32
gt_bbox              shape: (2, 4)                   min: 119.00000 max:
791.00000  int32
gt_mask              shape: (1024, 1024, 2)          min: 0.00000 max:
1.00000  bool

```

Predictions

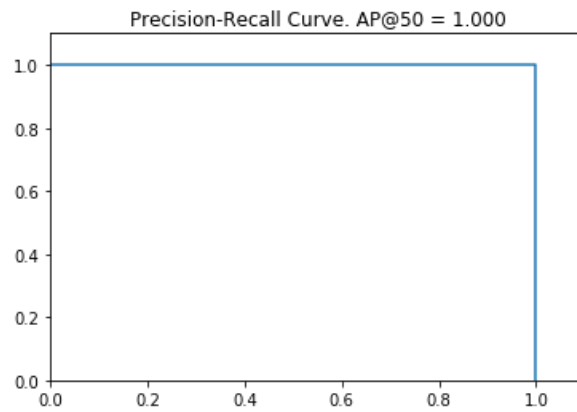


```
In [11]: visualize.display_differences(image, gt_bbox, gt_class_id, gt_mask,
                                         r['rois'], r['class_ids'], r['scores'], r['masks'],
                                         dataset.class_names, title="Display differences", a
                                         iou_threshold=0.5, score_threshold=0.5)
```

Display differences



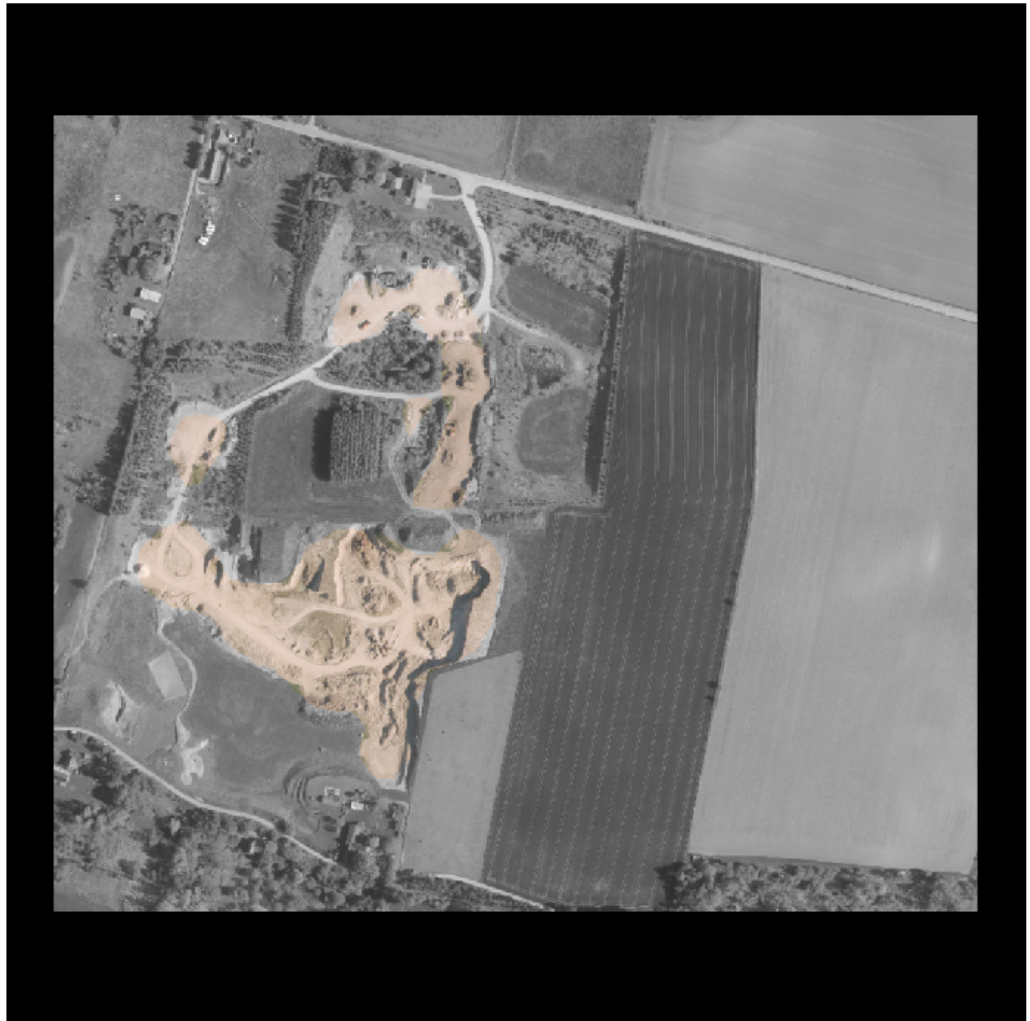
```
In [12]: # Draw precision-recall curve
AP, precisions, recalls, overlaps = utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
r['rois'], r['class_ids'], r
visualize.plot_precision_recall(AP, precisions, recalls)
```



Color Splash

This is for illustration. You can call `balloon.py` with the `splash` option to get better images without the black padding.

```
In [13]: splash = mask_rcnn.color_splash(image, r['masks'])  
display_images([splash], cols=1)
```



Step by Step Prediction

Stage 1: Region Proposal Network

The Region Proposal Network (RPN) runs a lightweight binary classifier on a lot of boxes (anchors) over the image and returns object/no-object scores. Anchors with high *objectness* score (positive anchors) are passed to the stage two to be classified.

Often, even positive anchors don't cover objects fully. So the RPN also regresses a refinement (a delta in location and size) to be applied to the anchors to shift it and resize it a bit to the correct boundaries of the object.

1.a RPN Targets

The RPN targets are the training values for the RPN. To generate the targets, we start with a grid of anchors that cover the full image at different scales, and then we compute the IoU of the anchors with ground truth object. Positive anchors are those that have an IoU ≥ 0.7 with any ground truth object, and negative anchors are those that don't cover any object by more than 0.3 IoU. Anchors in between (i.e. cover an object by IoU ≥ 0.3 but < 0.7) are considered neutral and excluded from training.

To train the RPN regressor, we also compute the shift and resizing needed to make the anchor cover the ground truth object completely.

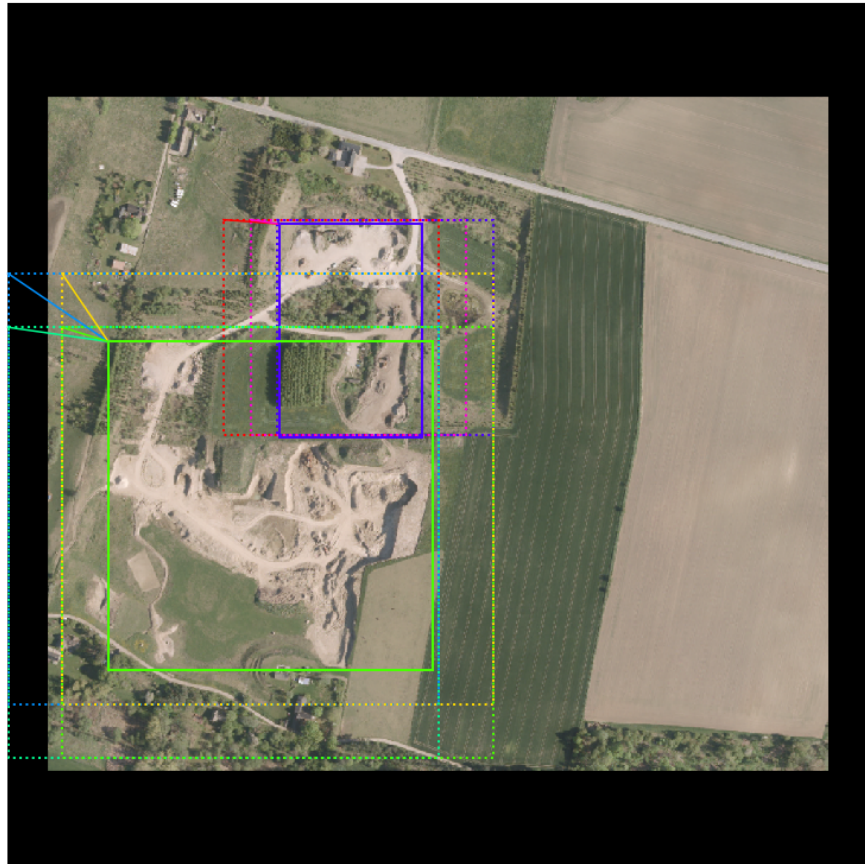
```
In [14]: # Generate RPN training targets
# target_rpn_match is 1 for positive anchors, -1 for negative anchors
# and 0 for neutral anchors.
target_rpn_match, target_rpn_bbox = modellib.build_rpn_targets(
    image.shape, model.anchors, gt_class_id, gt_bbox, model.config)
log("target_rpn_match", target_rpn_match)
log("target_rpn_bbox", target_rpn_bbox)

positive_anchor_ix = np.where(target_rpn_match[:] == 1)[0]
negative_anchor_ix = np.where(target_rpn_match[:] == -1)[0]
neutral_anchor_ix = np.where(target_rpn_match[:] == 0)[0]
positive_anchors = model.anchors[positive_anchor_ix]
negative_anchors = model.anchors[negative_anchor_ix]
neutral_anchors = model.anchors[neutral_anchor_ix]
log("positive_anchors", positive_anchors)
log("negative_anchors", negative_anchors)
log("neutral anchors", neutral_anchors)

# Apply refinement deltas to positive anchors
refined_anchors = utils.apply_box_deltas(
    positive_anchors,
    target_rpn_bbox[:positive_anchors.shape[0]] * model.config.RPN_BBOX_STD_DEV)
log("refined_anchors", refined_anchors, )
```

target_rpn_match	shape: (261888,)	min: -1.00000	max:
1.00000 int32			
target_rpn_bbox	shape: (256, 4)	min: -2.04690	max:
1.09375 float64			
positive_anchors	shape: (7, 4)	min: 0.00000	max:
896.00000 float64			
negative_anchors	shape: (249, 4)	min: -128.00000	max:
1088.00000 float64			
neutral anchors	shape: (261632, 4)	min: -362.03867	max:
1322.03867 float64			
refined_anchors	shape: (7, 4)	min: 119.00000	max:
791.00000 float32			


```
In [15]: # Display positive anchors before refinement (dotted) and  
# after refinement (solid).  
visualize_draw_boxes(image, boxes=positive_anchors, refined_boxes=refined_anchors)
```



1.b RPN Predictions

Here we run the RPN graph and display its predictions.

```

In [16]: # Run RPN sub-graph
pillar = model.keras_model.get_layer("ROI").output # node to start searching from

# TF 1.4 and 1.9 introduce new versions of NMS. Search for all names to support TF
nms_node = model.ancestor(pillar, "ROI/rpn_non_max_suppression:0")
if nms_node is None:
    nms_node = model.ancestor(pillar, "ROI/rpn_non_max_suppression/NonMaxSuppressi
if nms_node is None: #TF 1.9-1.10
    nms_node = model.ancestor(pillar, "ROI/rpn_non_max_suppression/NonMaxSuppressi

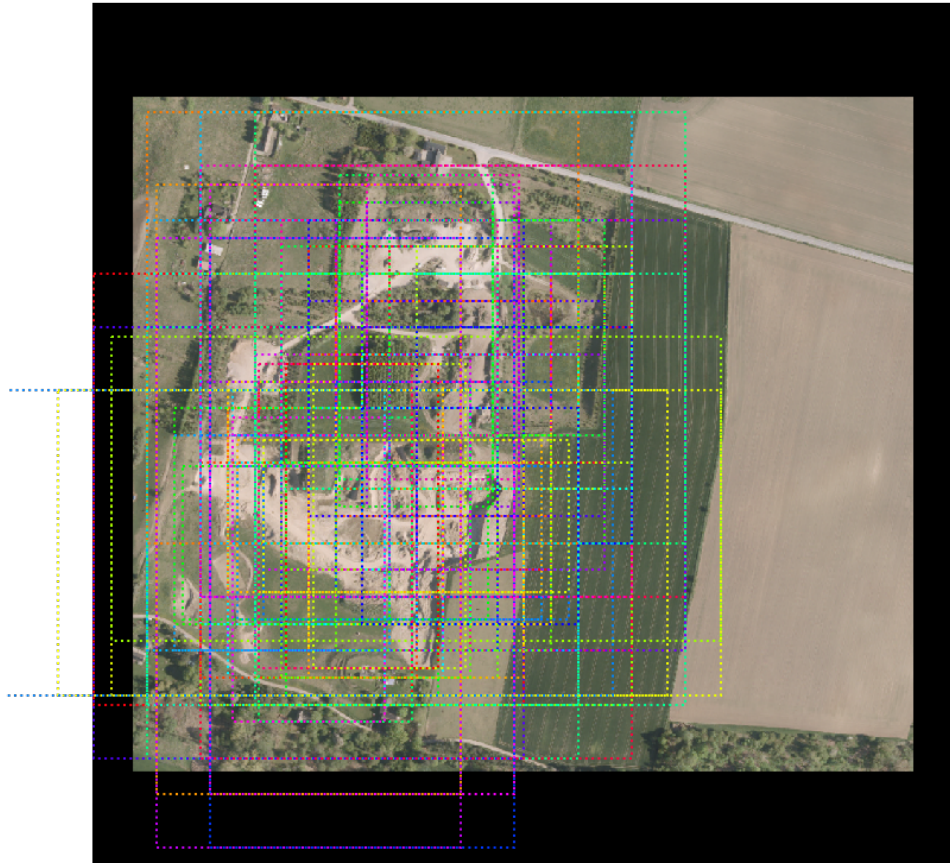
rpn = model.run_graph([image], [
    ("rpn_class", model.keras_model.get_layer("rpn_class").output),
    ("pre_nms_anchors", model.ancestor(pillar, "ROI/pre_nms_anchors:0")),
    ("refined_anchors", model.ancestor(pillar, "ROI/refined_anchors:0")),
    ("refined_anchors_clipped", model.ancestor(pillar, "ROI/refined_anchors_clipped:0")),
    ("post_nms_anchor_ix", nms_node),
    ("proposals", model.keras_model.get_layer("ROI").output),
])

rpn_class          shape: (1, 261888, 2)          min:    0.00000  max:
1.00000  float32
pre_nms_anchors    shape: (1, 6000, 4)              min:   -0.35390  max:
1.29134  float32
refined_anchors    shape: (1, 6000, 4)              min: -111.25734  max:
112.47511 float32
refined_anchors_clipped shape: (1, 6000, 4)              min:    0.00000  max:
1.00000  float32
post_nms_anchor_ix shape: (682,)                  min:    0.00000  max:
5993.00000 int32
proposals          shape: (1, 1000, 4)             min:    0.00000  max:
1.00000  float32

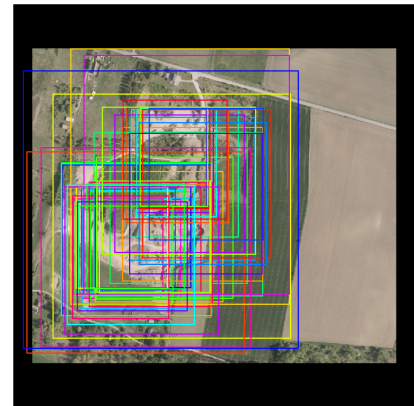
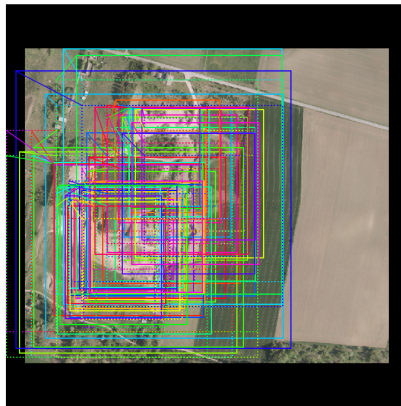
```



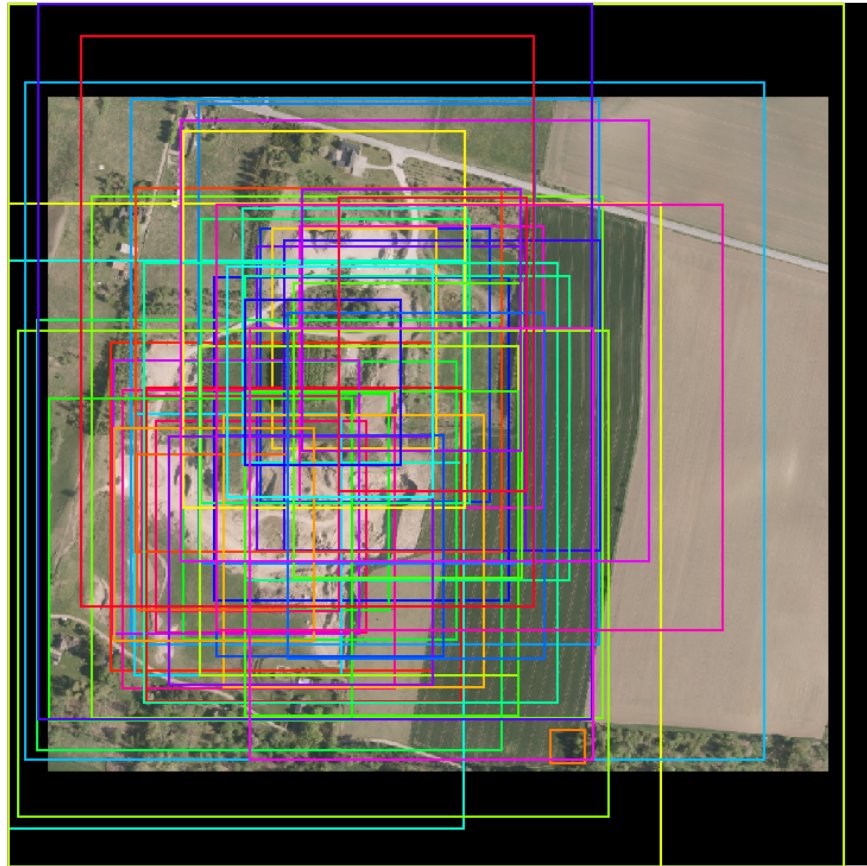
```
In [17]: # Show top anchors by score (before refinement)
limit = 100
sorted_anchor_ids = np.argsort(rpn['rpn_class'][:, :, 1].flatten())[::-1]
visualize.draw_boxes(image, boxes=model.anchors[sorted_anchor_ids[:limit]], ax=ax, ay=ay)
```



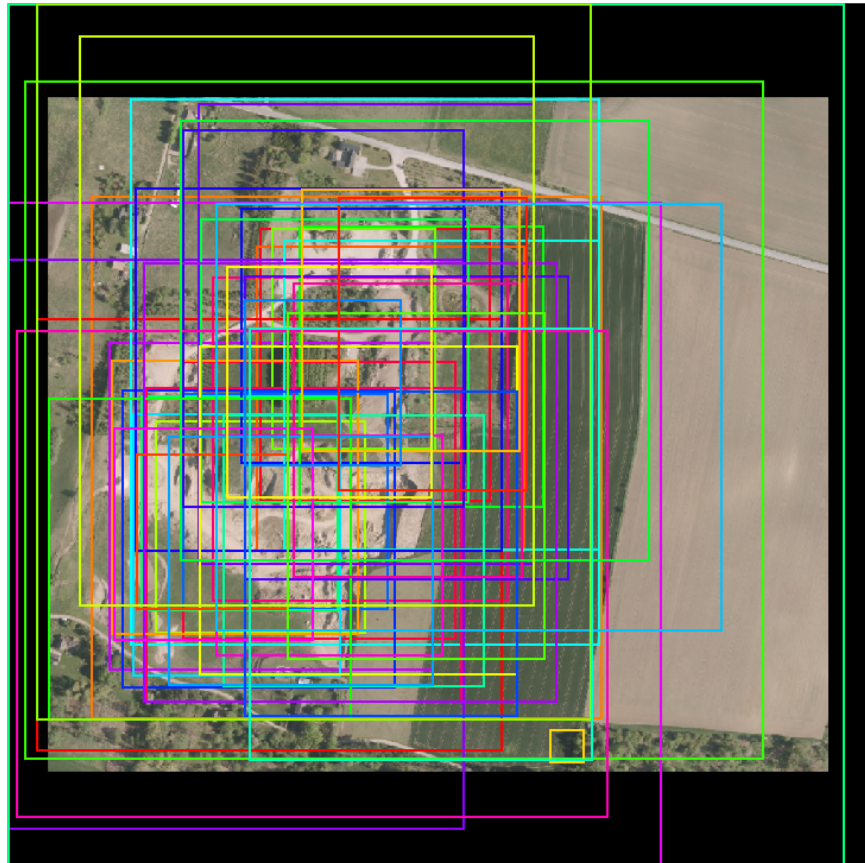
```
In [18]: # Show top anchors with refinement. Then with clipping to image boundaries
limit = 50
ax = get_ax(1, 2)
pre_nms_anchors = utils.denorm_boxes(rpn["pre_nms_anchors"][0], image.shape[:2])
refined_anchors = utils.denorm_boxes(rpn["refined_anchors"][0], image.shape[:2])
refined_anchors_clipped = utils.denorm_boxes(rpn["refined_anchors_clipped"][0], image.shape[:2])
visualize.draw_boxes(image, boxes=pre_nms_anchors[:limit],
                    refined_boxes=refined_anchors[:limit], ax=ax[0])
visualize.draw_boxes(image, refined_boxes=refined_anchors_clipped[:limit], ax=ax[1])
```



```
In [19]: # Show refined anchors after non-max suppression
limit = 50
ixs = rpn["post_nms_anchor_ix"][:limit]
visualize.draw_boxes(image, refined_boxes=refined_anchors_clipped[ixs], ax=get_ax
```



```
In [20]: # Show final proposals
# These are the same as the previous step (refined anchors
# after NMS) but with coordinates normalized to [0, 1] range.
limit = 50
# Convert back to image coordinates for display
h, w = config.IMAGE_SHAPE[:2]
proposals = rpn['proposals'][0, :limit] * np.array([h, w, h, w])
visualize.draw_boxes(image, refined_boxes=proposals, ax=get_ax())
```



Stage 2: Proposal Classification

This stage takes the region proposals from the RPN and classifies them.

2.a Proposal Classification

Run the classifier heads on proposals to generate class probabilities and bounding box regressions.

In [21]: *# Get input and output to classifier and mask heads.*

```
mrcnn = model.run_graph([image], [
    ("proposals", model.keras_model.get_layer("ROI").output),
    ("probs", model.keras_model.get_layer("mrcnn_class").output),
    ("deltas", model.keras_model.get_layer("mrcnn_bbox").output),
    ("masks", model.keras_model.get_layer("mrcnn_mask").output),
    ("detections", model.keras_model.get_layer("mrcnn_detection").output),
])
```

proposals	shape: (1, 1000, 4)	min: 0.00000	max:
1.00000 float32			
probs	shape: (1, 1000, 2)	min: 0.00000	max:
1.00000 float32			
deltas	shape: (1, 1000, 2, 4)	min: -6.33653	max:
3.89783 float32			
masks	shape: (1, 100, 28, 28, 2)	min: 0.00000	max:
1.00000 float32			
detections	shape: (1, 100, 6)	min: 0.00000	max:
1.00000 float32			


```
In [22]: # Get detection class IDs. Trim zero padding.
det_class_ids = mrcnn['detections'][0, :, 4].astype(np.int32)
det_count = np.where(det_class_ids == 0)[0][0]
det_class_ids = det_class_ids[:det_count]
detections = mrcnn['detections'][0, :det_count]

print("{} detections: {}".format(
    det_count, np.array(dataset.class_names)[det_class_ids]))

captions = ["{} {:.3f}".format(dataset.class_names[int(c)], s) if c > 0 else ""
            for c, s in zip(detections[:, 4], detections[:, 5])]
visualize.draw_boxes(
    image,
    refined_boxes=utils.denorm_boxes(detections[:, :4], image.shape[:2]),
    visibilities=[2] * len(detections),
    captions=captions, title="Detections",
    ax=get_ax())
```

2 detections: ['graveomr' 'graveomr']

Detections



2.c Step by Step Detection

Here we dive deeper into the process of processing the detections.

```
In [23]: # Proposals are in normalized coordinates. Scale them
# to image coordinates.
h, w = config.IMAGE_SHAPE[:2]
proposals = np.around(mrcnn["proposals"][0] * np.array([h, w, h, w])).astype(np.int32)

# Class ID, score, and mask per proposal
roi_class_ids = np.argmax(mrcnn["probs"][0], axis=1)
roi_scores = mrcnn["probs"][0, np.arange(roi_class_ids.shape[0]), roi_class_ids]
roi_class_names = np.array(dataset.class_names)[roi_class_ids]
roi_positive_ixs = np.where(roi_class_ids > 0)[0]

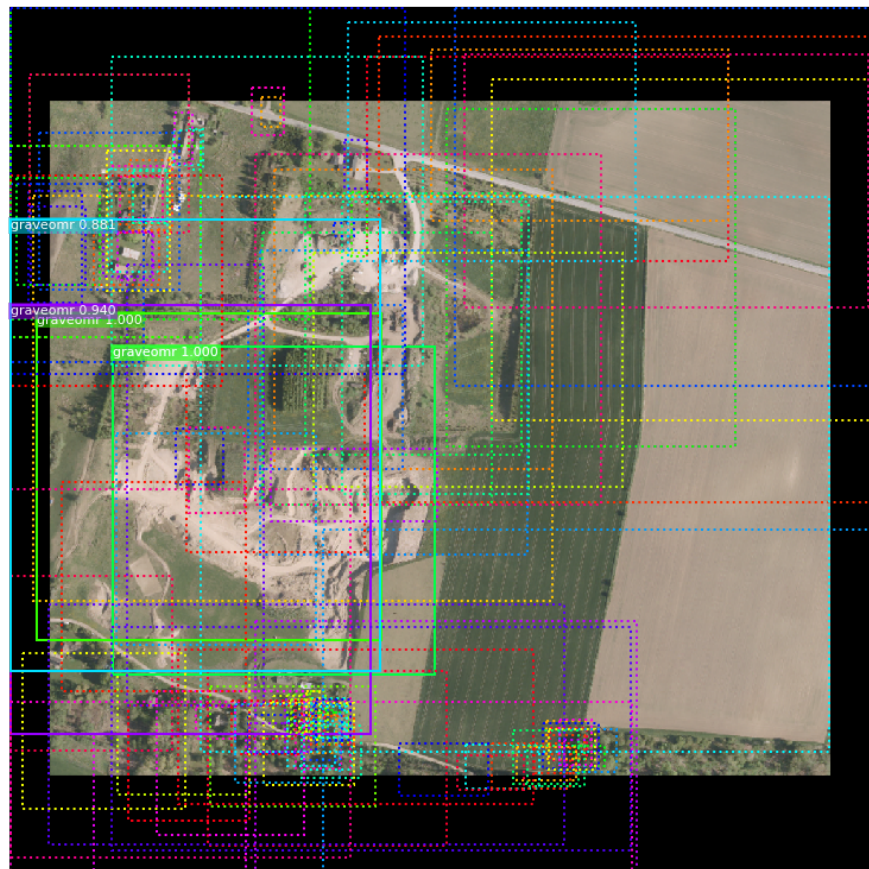
# How many ROIs vs empty rows?
print("{} Valid proposals out of {}".format(np.sum(np.any(proposals, axis=1)), proposals.shape[0]))
print("{} Positive ROIs".format(len(roi_positive_ixs)))

# Class counts
print(list(zip(*np.unique(roi_class_names, return_counts=True))))

682 Valid proposals out of 1000
21 Positive ROIs
[('BG', 979), ('graveomr', 21)]
```

```
In [24]: # Display a random sample of proposals.
# Proposals classified as background are dotted, and
# the rest show their class and confidence score.
limit = 200
ixs = np.random.randint(0, proposals.shape[0], limit)
captions = ["{} {:.3f}".format(dataset.class_names[c], s) if c > 0 else ""
            for c, s in zip(roi_class_ids[ixs], roi_scores[ixs])]
visualize.draw_boxes(image, boxes=proposals[ixs],
                    visibilities=np.where(roi_class_ids[ixs] > 0, 2, 1),
                    captions=captions, title="ROIs Before Refinement",
                    ax=ax())
```

ROIs Before Refinement



Apply Bounding Box Refinement


```
In [25]: # Class-specific bounding box shifts.
roi_bbox_specific = mrcnn["deltas"][0, np.arange(proposals.shape[0]), roi_class_ids]
log("roi_bbox_specific", roi_bbox_specific)

# Apply bounding box transformations
# Shape: [N, (y1, x1, y2, x2)]
refined_proposals = utils.apply_box_deltas(
    proposals, roi_bbox_specific * config.BBOX_STD_DEV).astype(np.int32)
log("refined_proposals", refined_proposals)

# Show positive proposals
# ids = np.arange(roi_boxes.shape[0]) # Display all
limit = 5
ids = np.random.randint(0, len(roi_positive_ixs), limit) # Display random sample
captions = ["{} {:.3f}".format(dataset.class_names[c], s) if c > 0 else ""
            for c, s in zip(roi_class_ids[roi_positive_ixs][ids], roi_scores[roi_
visualize.draw_boxes(image, boxes=proposals[roi_positive_ixs][ids],
                      refined_boxes=refined_proposals[roi_positive_ixs][ids],
                      visibilities=np.where(roi_class_ids[roi_positive_ixs][ids] >
captions=captions, title="ROIs After Refinement",
                      ax=ax())

roi_bbox_specific      shape: (1000, 4)      min:   -6.33653  max:
3.37389 float32
refined_proposals      shape: (1000, 4)      min:  -148.00000  max:
1079.00000 int32
```

ROIs After Refinement



Filter Low Confidence DetectionsIn [26]: *# Remove boxes classified as background*

```
keep = np.where(roi_class_ids > 0)[0]
print("Keep {} detections.\n{}".format(keep.shape[0], keep))
```

Keep 21 detections:

```
[ 2  6  7  9 12 13 15 16 17 26 29 30 32 34 56 66 80 147
 251 256 259]
```

In [27]: *# Remove low confidence detections*

```
keep = np.intersect1d(keep, np.where(roi_scores >= config.DETECTION_MIN_CONFIDENCE)[0])
print("Remove boxes below {} confidence. Keep {}:\n{}".format(
    config.DETECTION_MIN_CONFIDENCE, keep.shape[0], keep))
```

Remove boxes below 0.9 confidence. Keep 18:

```
[ 2  6  7  9 12 13 15 16 17 26 30 34 56 66 80 147 251 256]
```

Per-Class Non-Max SuppressionIn [28]: *# Apply per-class non-max suppression*

```
pre_nms_boxes = refined_proposals[keep]
pre_nms_scores = roi_scores[keep]
pre_nms_class_ids = roi_class_ids[keep]

nms_keep = []
for class_id in np.unique(pre_nms_class_ids):
    # Pick detections of this class
    ix = np.where(pre_nms_class_ids == class_id)[0]
    # Apply NMS
    class_keep = utils.non_max_suppression(pre_nms_boxes[ix],
                                           pre_nms_scores[ix],
                                           config.DETECTION_NMS_THRESHOLD)

    # Map indicies
    class_keep = keep[ix[class_keep]]
    nms_keep = np.union1d(nms_keep, class_keep)
    print("{:22}: {} -> {}".format(dataset.class_names[class_id][:20],
                                   keep[ix], class_keep))
```

keep = np.intersect1d(keep, nms_keep).astype(np.int32)

```
print("\nKept after per-class NMS: {}\n{}".format(keep.shape[0], keep))
```

```
graveomr : [ 2  6  7  9 12 13 15 16 17 26 30 34 56 66
80 147 251 256] -> [2 7]
```

Kept after per-class NMS: 2

[2 7]

```
In [29]: # Show final detections
ixs = np.arange(len(keep)) # Display all
# ixs = np.random.randint(0, len(keep), 10) # Display random sample
captions = ["{} {:.3f}".format(dataset.class_names[c], s) if c > 0 else ""
            for c, s in zip(roi_class_ids[keep][ixs], roi_scores[keep][ixs])]
visualize.draw_boxes(
    image, boxes=proposals[keep][ixs],
    refined_boxes=refined_proposals[keep][ixs],
    visibilities=np.where(roi_class_ids[keep][ixs] > 0, 1, 0),
    captions=captions, title="Detections after NMS",
    ax=ax)

```

Detections after NMS



Stage 3: Generating Masks

This stage takes the detections (refined bounding boxes and class IDs) from the previous layer and runs the mask head to generate segmentation masks for every instance.

3.a Mask Targets

These are the training targets for the mask branch

```
In [30]: display_images(np.transpose(gt_mask, [2, 0, 1]), cman="Blues")
```



3.b Predicted Masks

```
In [31]: # Get predictions of mask head
mrcnn = model.run_graph([image], [
    ("detections", model.keras_model.get_layer("mrcnn_detection").output),
    ("masks", model.keras_model.get_layer("mrcnn_mask").output),
])

# Get detection class IDs. Trim zero padding.
det_class_ids = mrcnn['detections'][0, :, 4].astype(np.int32)
det_count = np.where(det_class_ids == 0)[0][0]
det_class_ids = det_class_ids[:det_count]

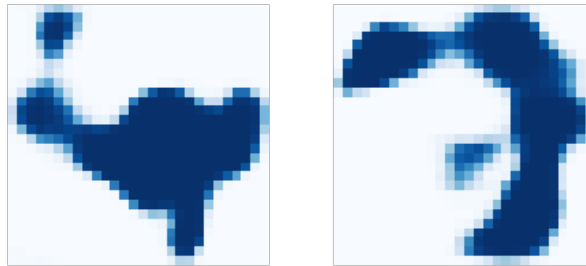
print("{} detections: {}".format(
    det_count, np.array(dataset_class_names)[det_class_ids]))

detections          shape: (1, 100, 6)          min:    0.00000  max:
1.00000 float32
masks               shape: (1, 100, 28, 28, 2)   min:    0.00000  max:
1.00000 float32
2 detections: ['graveomr' 'graveomr']
```

```
In [32]: # Masks
det_boxes = utils.denorm_boxes(mrcnn["detections"][0, :, :4], image.shape[:2])
det_mask_specific = np.array([mrcnn["masks"][0, i, :, :, c]
                              for i, c in enumerate(det_class_ids)])
det_masks = np.array([utils.unmold_mask(m, det_boxes[i], image.shape)
                      for i, m in enumerate(det_mask_specific)])
log("det_mask_specific", det_mask_specific)
log("det_masks", det_masks)

det_mask_specific    shape: (2, 28, 28)          min:    0.00000  max:
1.00000 float32
det_masks            shape: (2, 1024, 1024)       min:    0.00000  max:
1.00000 bool
```

```
In [33]: display_images(det_mask_specific[:4] * 255, cmap="Blues", interpolation="none")
```



```
In [34]: display_images(det_masks[:4] * 255, cmap="Blues", interpolation="none")
```



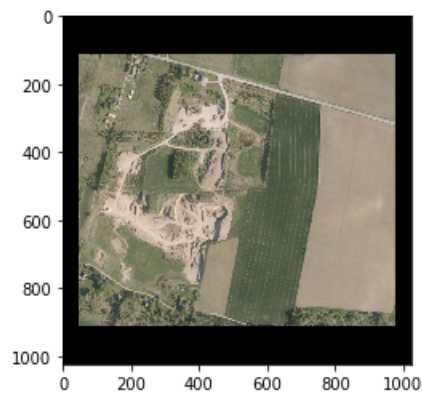
Visualize Activations

In some cases it helps to look at the output from different layers and visualize them to catch issues and odd patterns.

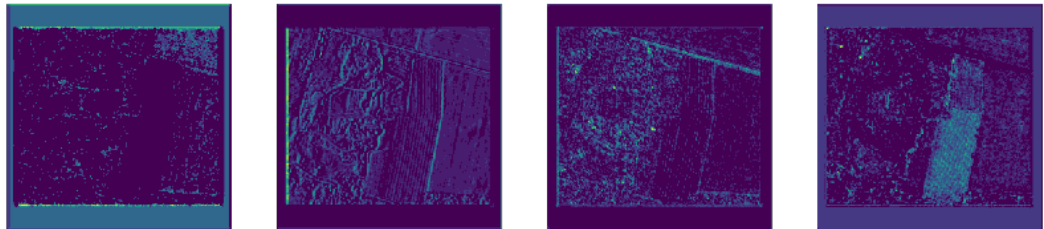
```
In [35]: # Get activations of a few sample layers
activations = model.run_graph([image], [
    ("input_image", tf.identity(model.keras_model.get_layer("input_image").output)),
    ("res2c_out", model.keras_model.get_layer("res2c_out").output),
    ("res3c_out", model.keras_model.get_layer("res3c_out").output),
    ("res4w_out", model.keras_model.get_layer("res4w_out").output), # f
    ("rpn_bbox", model.keras_model.get_layer("rpn_bbox").output),
    ("roi", model.keras_model.get_layer("ROI").output),
])
```

input_image	shape: (1, 1024, 1024, 3)	min: -123.70000	max:
150.10001 float32			
res2c_out	shape: (1, 256, 256, 256)	min: 0.00000	max:
23.13472 float32			
res3c_out	shape: (1, 128, 128, 512)	min: 0.00000	max:
28.80074 float32			
res4w_out	shape: (1, 64, 64, 1024)	min: 0.00000	max:
58.26350 float32			
rpn_bbox	shape: (1, 261888, 4)	min: -10.10481	max:
118.62450 float32			
roi	shape: (1, 1000, 4)	min: 0.00000	max:
1.00000 float32			

```
In [36]: # Input image (normalized)
         plt.imshow(modellib.unmold_image(activations["input_image"][0].confid))
```



```
In [37]: # Backbone feature map
         display_images(np.transpose(activations["res2c_out"][0 : :41][2, 0, 1]), cols=4)
```



```
In [ ]:
```