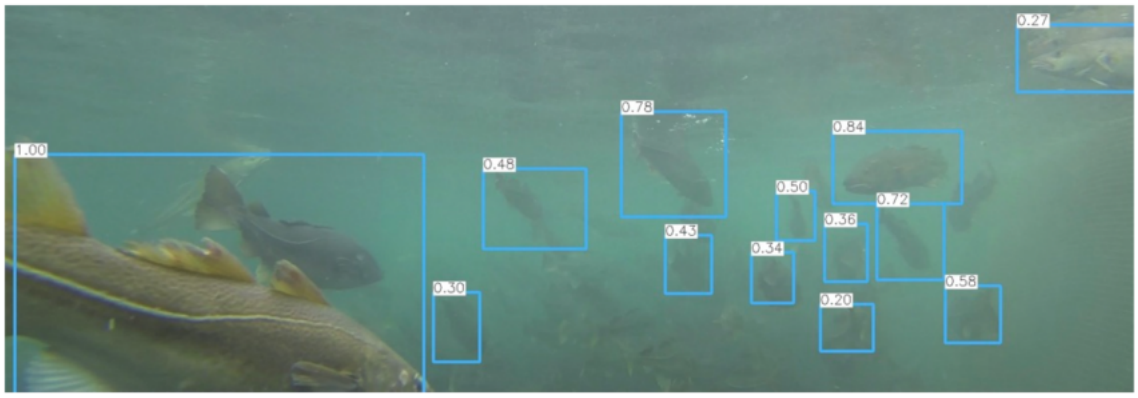
NTNU - Norges teknisk-naturvitenskapelige universitet
Institutt for bioteknologi og matvitenskap

BACHELOROPPGAVE 2020

20 studiepoeng

Bruk av maskinsyn for automatisk telling og artsbestemmelse av villfisk som beiter under oppdrettsmerder

Counting and Classification of Wild Fish Feeding in the vicinity of Fish Farms by using Computer Vision



utført av

Hans Alan Whitburn Haugen

Dette arbeidet er gjennomført som ledd i bachelorutdanningen i matteknologi ved Institutt for bioteknologi og matvitenskap, NTNU. Bruk av rapportens innhold skjer på eget ansvar.

Sammendrag

Denne bacheloroppgaven handler om å bruke maskinsyn til å telle villfisk som beiter under oppdrettsmerder. Forsøket ble gjort som en del av Sameksistens-prosjektet til Nofima. Objektdetekteringen baserer seg på dype nevrale nettverk og er i stand til å klassifisere og telle fiskene torsk og sei.

Kunstige nevrale nettverk kalles for modeller. Maskinlæring har blitt et populært paradigme innenfor informatikk. Modeller lærer fra data. YOLOv4 og RetinaNet er objektdetekteringsmodeller.

YOLOv4 og RetinaNet ble anvendt til fiskedeteksjon. Begge modellene brukte CNN-konsepter hentet fra forhåndstrente nettverk. Det ble utviklet et datasett basert på video fra og ved lagringsmerder av fisk. Gjennomsnittlig presisjon (AP50) for RetinaNet ble 66,3 %. En gjennomsnittlig objektdeteksjonspresisjon (mAP) lik 73 % ble oppnådd for YOLOv4-modellen.

Dette forsøket viser at maskinsyn kan anvendes til å løse oppgaver innenfor akvakultur. Maskinsyn kan anvendes til å finne omfanget av torsk og sei som beiter rundt oppdrettsanlegg.

Skriptene til å reprodusere forsøket er offentlig tilgjengelig fra:
https://github.com/alanhaugen/TMAT3004-Bacheloroppgave.

Abstract

This bachelor thesis describes a method for detecting and counting the wild fish feeding in the vicinity of fish farms by using computer vision. The work was done as a part of Nofima's Coexistence project. The fish detection is based on deep neural networks and can detect and count Atlantic cod and saithe.

Artificial neural networks are called models. Machine learning has become a popular new programming paradigm. Models learn from data. YOLOv4 and RetinaNet are examples of object detection models.

The networks YOLOv4 and RetinaNet were used for fish detection. Both of the models used CNN concepts from pre-trained networks. A dataset based on videos of fish from and around fish farms was generated. The average precision (AP50) for the RetinaNet model was 66,3 %. A mean average precision (mAP) of 73 % was achieved for the YOLOv4 model.

The study demonstrates that computer vision is a viable technology for solving problems within the aquaculture industry. Computer vision can be used to find the amount of Atlantic cod and saithe grazing in the vicinity of fish farms.

Scripts to reproduce this study are made publicly available at:
https://github.com/alanhaugen/TMAT3004-Bacheloroppgave.

Forkortelser og faguttrykk

Algoritme er en sekvens av instruksjoner som datamaskiner følger for å prosessere data

AP Average Precision

API Application Programming Interface

Azure nettskytjeneste fra Microsoft der en kan leie en Nvidia Titan-GPU

Backpropagation er en algoritme for å trene maskinlæringsmodeller

Bias er urettferdig vekting av konsepter i modellen

COCO er et datasett fra Microsoft med 80 klasser

CPU Central Processing Unit, kan programmeres med Python, C, C++, osv.

Darknet: Open-Source Neural Networks in C er et maskinlæringsrammeverk fra Joseph Redmon

Deep Learning er en form for maskinlæring der mange kerneler er koblet sammen

Detectron2 er et objektdetekteringssystem fra Facebook som bruker PyTorch

dnn Deep Neural Networks

FPS Frames Per Second, et mål på hvor rask et sanntids dataprogram er, YOLOv4 er laget til å gjør inferens i sanntid

GPU Graphical Processing Unit, den kan trene maskinlæringsmodeller

IDE Integrated Development Environment

ImageNet er et generelt datasett fra Stanford University

Inference eller inferens kalles også for en 'forward pass', det er når en modell forutser hva som finnes i et bilde eller en prøve

IoU Intersection over Union, mål på objektdetekteringsnøyaktighet

Klasse er også kjent som target eller label, det er en merkelapp som beskriver en bounding-box eller objekt

Kernel også kjent som filter, kunstig nevron eller perceptron er en funksjon som består av vekter og bias, den tar inn matriser, abstrakte konsepter eller «features», og gir fra seg nye abstrakte konsepter, nye «features», basert på innmatningen

Kildekode er algoritmene, instruksjonene, eller «DNA-et», til et dataprogram

Maskinlæringsbiblioteker brukes ofte som synonymt med maskinlæringsrammeverk

KI Kunstig Intelligens

Kunstige nevrale nettverk se modell

Kunstig nevron se kernel

mAP mean Average Precision, en måling på objektedekteringsnøyaktighet

Maskinsyn har i nyere tid med utviklingen innenfor kunstige nevrale nettverk (aka "deep learning") ført til maskiner som kan med høy nøyaktighet detektere og klassifisere objekter

ML Machine Learning, når en datamaskin lærer fra data

Modell er blant annet et annet ord for kunstige nevrale nettverk, en modell er trent på data, og gitt lignende data så kan en modell inferere hva den nye dataen er

Nettsky er en tjeneste der en kan leie en kraftig datamaskin, slik som en GPU-server

Nøyaktighet defineres som True positives / (True positives + False positives)

Overfitting overtilpasning er når nettverket kan detektere objekter på bilder fra treningsdatasettet, men kan ikke detektere objekter på noen andre bilder

Optical flow er bevegelsene mellom bildene i en film

R-CNN Region-Based Convolutional Neural Network

Perceptron se kernel

Presisjon defineres som True positives / (True positives + False negatives)

Programvarebibliotek er kildekode som kan anvendes til å lage dataprogrammer, det skiller seg fra programvarerammeverk ved at programmerere har mer kontroll over arkitekturen til dataprogrammet som anvender programvarebibliotek

Programvarerammeverk er programvare som kan konfigureres til å lage modeller eller nye programmer

Programvareutgivelse er typisk et Unix-miljø for programmerere som gjør det lettere å programmere datamaskiner

PyTorch er et maskinlæringsrammeverk fra Facebook

RetinaNet er en objektdetekteringsmodell fra Facebook som kan brukes sammen med Detectron2

Real-time sanntid, YOLOv4 er laget til å kunne anvendes til å gjøre maskinsyn i sanntid

Skript er et annet ord for kildekode, det brukes vanligvis om koden til enklere programmeringsspråk slik som Python

ssh Secure SHell, en protokoll og program til å koble seg til eksterne maskiner slik som en GPU-server

STL Standard Template Library, et populært programvarebibliotek for C++ med mange nyttige datastrukturer

SVM Support Vector Machine, en metode som baserer seg på kvadratisk optimering

TL Transfer learning, å anvende forhåndstrente nettverk til å forbedre nøyaktigheten av nye nettverk

Unix er et avansert operativsystem for programmerere

YOLO You Only Look Once, real-time objektdeteksjon

Takk

Jeg ønsker å takke min veileder Eirin Bar. Oppgaven ble påbegynt 9. mars og foregikk frem til i dag, jeg hadde ikke gjennomført oppgaven uten veiledningen jeg fikk. Koronaviruspandemien av 2020 førte til at arbeidet i sin helhet ble gjort i Trondheim.

Jeg ønsker også å takke Stein-Kato Lindberg fra Nofima. Det var KVASS-prosjektet[1] til Nofima som gjorde at jeg kontaktet dem og spurte om jeg kunne gjøre bacheloroppgaven i samarbeid med dem. Det har vært en drøm å jobbe på en bacheloroppgave som handler om maskinsyn. Stein-Kato foreslo at jeg tok del i sameksistensprosjektet, og har bidratt med data, hyggelige tilbakemeldinger og råd.

I tillegg ønsker jeg også å takke Satya Mallick, Vikas Gupta, Anastasia Murzova, Anna Petrovicheva, Grigory Serebryakov, Pavel Semkin, Prakash Chandra, Sergei Belousov, Tatiana Khanova og Valeriia Koriukina for utmerkede kurs ved courses.opencv.org.

Trondheim 20. mai 2020

[1] https://nofima.no/prosjekt/kvass/

Innhold

1 Innledning

Fiskeri og havbruksnæringen har en utfordring med villfisk som spiser pellets under oppdrettsmerder. Fôrsprengt fisk, også kalt «pellets-fisk» er villfisk som spiser fôrpellets fra under oppdrettsmerdene. Se figur 1. Fiskere ønsker svar på om villfisk som spiser oppdrettsfôr som inneholder medisin kan være farlig. Oppdrettsfisk er i karantene når de medisineres, men villfisk spiser fortsatt fôret og blir omsatt. Dette er beskrevet i detalj i del 2.1.

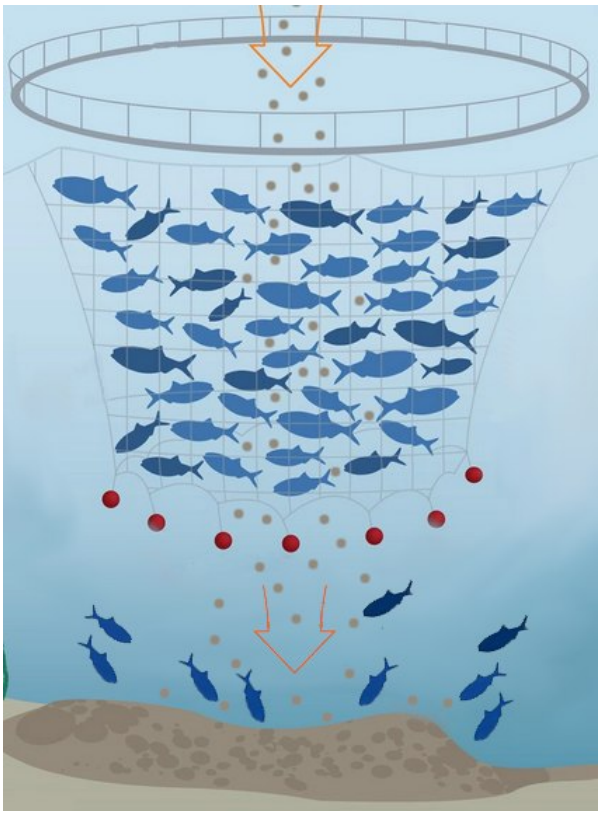
For å lage et dataprogram som kan automatisk telle og artsbestemme fisk så ble maskinlæring, en form for kunstig intelligens, anvendt. Teorien beskrives i del 2.2. Tidligere vitenskapelige forsøk der forskere har brukt maskinlæring til fiskeklassifisering og fiskedeteksjon nevnes i del 2.3.

I del 3 presenteres metoden som ble anvendt til å gjøre automatisk analyse av video. Det er først og fremst norsk torsk, *Gadus morhua*, og sei, *Pollachius virens*, som man ønsker en oversikt over. Et datasett av torsk og sei skulle utvikles for å trene opp maskinlæringsmodeller, måten dette ble gjort er beskrevet i del 3.2. Det ble planlagt å filme torsk og sei på Havbruksstasjonen i Tromsø, men koronapandemien av 2020 gjorde det vanskelig å få tilgang til data. I del 3.5.1 blir fremgangsmåten som ble brukt til å trene opp deep learning-modeller som kan kjenne igjen torsk og sei beskrevet. I del 3.8.2 forklares en av måtene en kan lage et dataprogram med en deep learning-modell som kan gi en tidsoversikt over mengden torsk og sei. Nøyaktigheten til modellene ble målt, resultatene presenteres i del 4.

Det er mulig å forbedre dataprogrammet som har blitt utviklet til å gjenkjenne torsk og sei. Nøyaktigheten til et objektdeteksjonssystem blir gitt som mAP (mean Average Precision). Modellen til dataprogrammet har en mAP på 73 %. En forbedret modell vil kunne gi oppdrettsnæringen en enda bedre tidsoversikt over torsk og sei under oppdrettsanleggene. En drøfting av resultatene er gitt i del 5.

Siste del er en konklusjon, del 6. Maskinsyn kan anvendes til å hjelpe oppdrettsnæringen med å få kontroll på fôring av oppdrettfisk ved å gi næringen en tidsoversikt over mengden torsk og sei som trekker til anleggene.

Bacheloroppgaven er gjort i samarbeid med Nofima som en del av Sameksistensprosjektet (**Robertsen 2020**).



Figur 1: *Oppdrettsbransjen er i konflikt med fiskere som mener oppdrettsnæringen ødelegger fiskeplasser i fjordene med anlegg og reduserer kvaliteten til villfisk som spiser fôrpellets under merdene (***Olsen m.fl. 2018***). Figur: Spruill (***Spruill 2011 s. 12***).*

2 Teori

2.1 Bakgrunn

Kystfiskere hevder villfisken får i seg så mye laksefôr at fiskeplassene blir ødelagt. Fisker Tor Inge Larsen i Brønnøysund sa til NRK i 2018 at han fortviler over det han mener er dårligere kvalitet hos fisken han fanger. Han «kan knapt huske sist» han fikk en sei om bord i båten uten at den hadde pellets fra lakseoppdrett i magen. Se figur 2. Han hevder fiskerne har blitt frastjålet fiskeområder på høylys dag, og mener fiskefeltene nært Brønnøysund på Sør-Helgeland i Nordland har fått betydelig lavere kvalitet på grunn av oppdrettsanleggene. (**Olsen m.fl. 2018**)

Kommunikasjonsdirektør Are Kvistad i Sjømat Norge mener laksepellets ikke utgjør noen fare for villfisk. «Det er et fiskefôr som blir sjekket av myndighetene, og er et trygt og næringsrikt fôr som også er trygt for villfisk», sa Kvistad til NRK i 2018. Kvistad mente det også er i oppdretternes interesse, både økonomisk og utslippsmessig, å redusere fôringen av oppdrettslaksen slik at den kun får den maten den trenger. (**Olsen m.fl. 2018**)

Ikke alle er negative til «pellets-fisk». Nofima gjorde en undersøkelse i 2020 som viser hvilke grupper som er negative, positive eller likegyldig til fisken. Nofima-forskerne Anette Hustad og Ragnhild Svalheim oppdaget at de fleste er nøytrale til påstander om at pelletssei smaker eller lukter annerledes enn annen sei. (**Hustad og Svalheim 2020**)

Det er forsket relativt lite på hvordan oppdrett påvirker villfisk, men seniorrådgiver Bjørn-Steinar Sæther fra Nofima ga ut en artikkel i 2017 der han viste at seien som hadde spist oppdrettsfôr får en litt bløtere og noe mer spaltet muskel enn annen sei. Ellers var kvaliteten fortsatt stort sett innenfor kategorien god kvalitet. (**Saether 2017**)

Oppdrettsnæringen er også interessert i å kartlegge hvor mye fôr som spises av villfisk. For oppdrettsnæringen er å ha kontroll på fôrmengde som går til spille viktig, fôrpellets er en stor kostnad som har økt mye de siste årene. De ønsker ikke å fôre oppdrettsfisken mer enn nødvendig. Om problemet løses vil det bli mindre oppdrettsfôr til villfisken. (**Baevre-Jensen 2019**)

I 2019 kom «pellets-fisken» i medienes søkelys igjen, denne gangen var det fiskere i Lyngen kommune i Troms som slo alarm. Villfisk som hadde spist oppdrettsfôr som inneholdt medisin ble omsatt av fiskere i fjorden. Oppdrettsfisken var i karantene i denne perioden. Fiskerne ønsket svar på om villfisken kan være farlig for forbrukere. (**Trana m.fl. 2019**)

Sjømatdivisjonen og Akvadivisjonen ved Nofima har en strategisk internsatsing som går ut på å samle kunnskap om sameksistens mellom ulike marine næringer og interesser, deriblant om hvordan man kan unngå konflikter mellom ulike interesser. Se figur 1. (**Robertsen 2020**)

Figur 2: *Dette er et eksempel på «pellets-fisken». Det lukter ekkelt og det er forferdelig å sløye en fôrsprengt fisk, ifølge flere fiskere som har vært i kontakt med NRK (***Trana m.fl. 2019***). Seien har vært under merdene, da blir den full av pellets. Fôrsprengt fisk er unaturlig, hode blir for liten og kroppen for stor. Dette har pågått i mange år. Likevel blir fisken omsatt (***Angell og Ekanger 2017***).*

En av problemstillingene er sameksistens mellom fiskeri- og oppdrettsnæringene, og hvordan oppdrettsanlegg påvirker de nærliggende fiskeplassene. I den forbindelse er det interessant å kartlegge omfanget av hvitfisk som beiter på fôr fra oppdrettsanlegg. Dette er en problemstilling som har vært i fokus hos media etter at flere fiskere har fanget fôrsprengt torsk og sei i fjorder hvor det finnes oppdrettsanlegg. Det hevdes at denne fisken er av betydelig dårligere kvalitet og den kan ha fått i seg medisiner gjennom fôret som gjør at den kan være farlig å spise. (**Olsen 2019**)

Det behøves kunnskap om hvor mange villfisk som trekker til oppdrettsanlegg, og under hvilke forhold, slik at man kan komme nærmere en løsning som kan dempe konflikten mellom fiskere og oppdrettsnæringen.

Målet med dette prosjektet var å utvikle et system som kan telle antall villfisk av ulike arter basert på en videostrøm fra et undervannskamera. Det er ønskelig å kunne vise resultatet som en fordeling av observasjoner over tid for hver art.

2.2 Kunstig intelligens

Kunstig intelligens (KI) er blitt et stort forskningsfelt med mange praktiske anvendelser og aktive forskningsprosjekter. Intelligent programvare automatiserer trivielt og repetitivt arbeid, forstår tale og kjenner igjen objekter i bilder, kan diagnostisere pasienter innenfor medisin og hjelper forskere med grunnforsk-

ning. KI kan også finne mønstre i store datamengder som det er vanskelig for mennesker å få øye på. (**Goodfellow m.fl. 2016 s. 1**)

Dette prosjektet beskriver maskiner som kan gi en tidsoversikt over torsk og sei fra en videostrøm. Den lærer konseptene en torsk eller sei består av, basert på data. Maskinlæring, å la en maskin lære fra data, er kun en av metodene innenfor kunstig intelligens. Se figur 3. (**Canziani m.fl. 2017 s. 1**)

I begynnelsen løste kunstig intelligens problemer som var vanskelige for mennesker å løse, men relativt trivielle for datamaskiner. Utfordringen med kunstig intelligens viste seg å være å løse problemer som var enkle for mennesker å utføre, men vanskelige for mennesker å beskrive formelt. Dette er problemer som vi løser intuitivt, som føles automatiske ut, slik som å forstå ordene til andre mennesker, ansiktene deres, og å kjenne igjen det en har sett tidligere i bilder. (**Goodfellow m.fl. 2016 s. 1**)

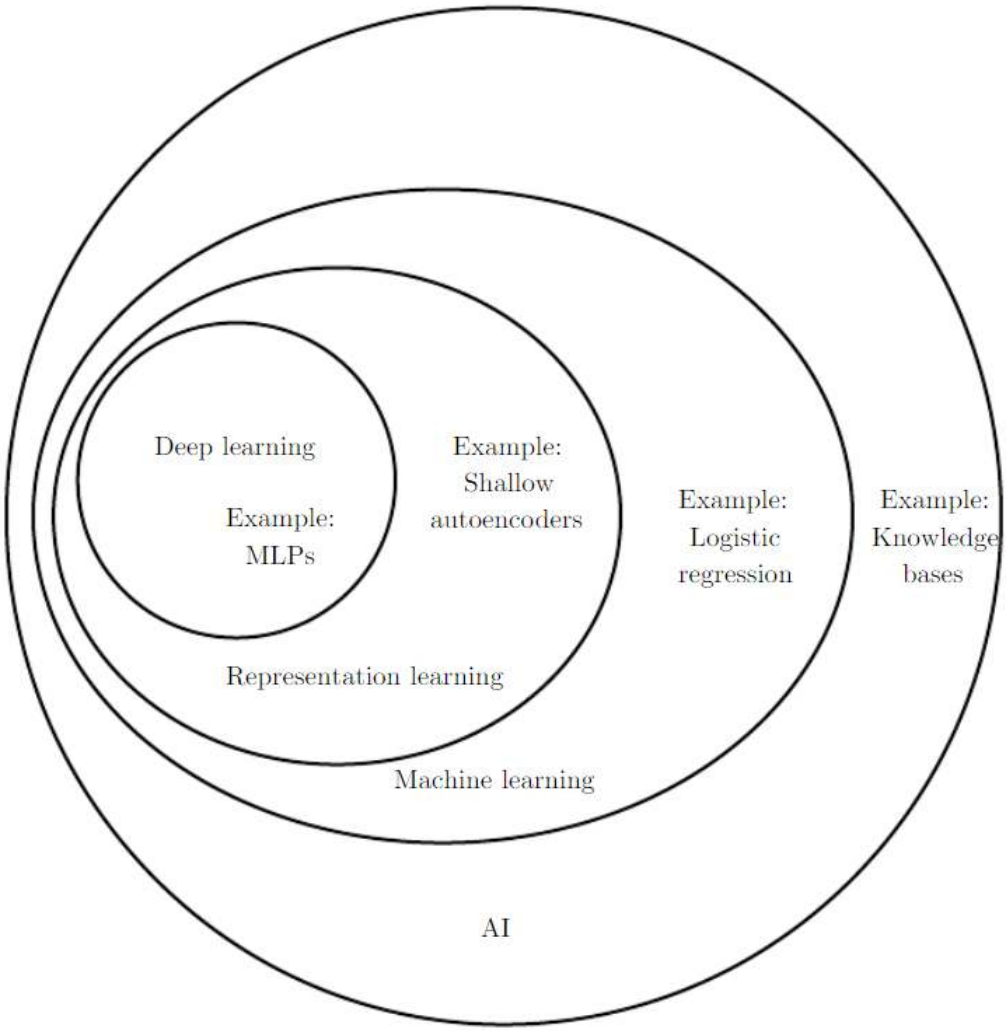
Løsningen er å la datamaskiner lære fra erfaringer og forstå verden ved å forstå konsepter, som kan igjen bli forstått med enda enklere konsepter. Denne erfaringskunnskapen gjør datamaskinene selv, uten at et menneske formelt beskriver kunnskapen til maskinen. Erfaringene kan beskrives med en graf. Se figur 4. Ettersom denne grafen kan bestå av mange lag, den er dyp, så kalles denne formen for maskinlæring «deep learning» (**Goodfellow m.fl. 2016 s. 1**).

2.2.1 Objektdeteksjon

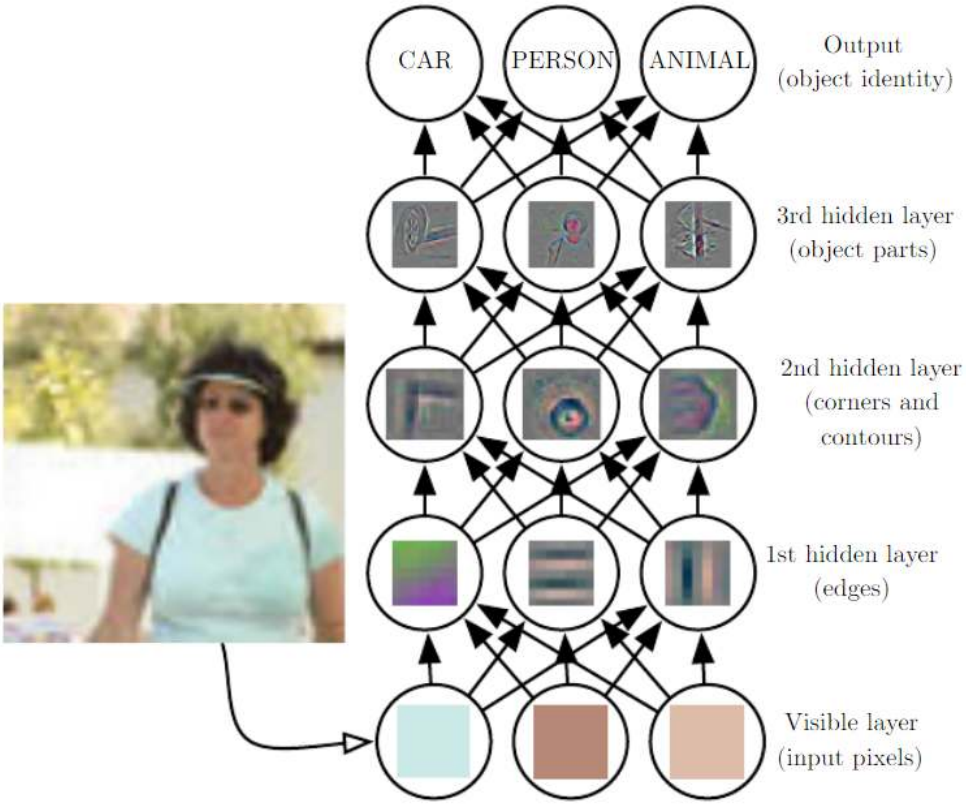
Objektdeteksjon handler om to ting. Områder av bildet som kan inneholde en klasse må genereres, og så må disse delene av et bildet klassifiseres basert på det visuelle innholdet i bildeområdet. Et bildeområde kalles for en *patch* innenfor maskinsyn (**LeCun m.fl. 1998 s. 23**). Områdene der det kan være objekter kalles for ankre. Med andre ord, målet for modellen er å gå over områder i et bilde og eventuelt gi området en merkelapp om det finnes en klasse i bildeområdet (**Jordan 2018**). Nettverket kan kun finne de objektene som den har blitt trent til å gjenkjenne.

Hubel og Wiesel forsket på synssenteret til en katt tilbake i 1959. De oppdaget at det fantes forskjellige nevroner i hjernen som reagerer til forskjellig stimulasjon. Noen nevroner aktiveres av vertikale bevegelser i synsfeltet, mens andre aktiveres av horisontale bevegelser (**Hubel og Wiesel 1959 s. 582**). Dette arbeidet kalles for den biologiske inspirasjonen til deep learning.

AlexNet fra 2012 gjorde bildeklassifisering mye mer nøyaktig, arbeidet førte til en revolusjon innenfor kunstig intelligens (**Canziani m.fl. 2017 s. 1**). I 2014 så ble Region-Based Convolutional Neural Network (R-CNN) presentert. R-CNN var det første deep learning-baserte objektdeteksjonssystemet. R-CNN var en two-stage objektdetektor (**Girshick m.fl. 2014 s. 1**). En forbedret utgave, Fast R-CNN, kom i 2015 (**Girshick 2015**). I 2016 kom den siste utgaven av R-CNN kjent som Faster R-CNN (**Ren m.fl. 2016**).



Figur 3: *Dette er et Venn-diagram som viser at deep learning er en form for representativ læring, og det er igjen en form for maskinlæring. Maskinlæring er anvendt i mange, men ikke alle, de forskjellige metodene innenfor kunstig intelligens. Hver del av Venn-diagrammet gir et eksempel av anvendelsen til teknologien.* (**Goodfellow m.fl. 2016 s. 9**)



Figur 4: *Dette er en deep learning-modell. Det er vanskelig for en datamaskin å forstå hva sensorisk rådata er. Å lage en funksjon som tar inn piksler og gir ut en objektidentitet er svært komplisert. Å utlede en slik funksjon virker nærmest umulig om en takler problemet direkte. Deep learning løser vanskelighetene ved dele opp den kompliserte funksjonenen til en samling sammenkoblede funksjoner. Disse funksjonene kalles ofte for kunstige nevroner, kerneler, perceptrons eller filtere. Utmatningene fra kernelene i ett lag blir innmatningene til neste lag i modellen. Innmatningen til modellen, slik som et bilde, kalles for det synlige laget, ettersom den består av informasjon som vi mennesker kan forstå. Dette laget er etterfulgt av en serie med skjulte lag, «hidden layers», hvert lag henter ut mer og mer abstrakte konsepter, «features», fra bildet. Disse lagene kalles for «hidden» ettersom verdiene i disse lagene ikke finnes i innmatningen til nettverket; modellen må selv bestemme hvilke konsepter som er nyttige for å beskrive sammenhengen i det synlige laget. Hvert lag danner forskjellig innsikt. Det første laget kan for eksempel lett gjenkjenne kantene i et bilde ved å sammenligne kontraster i bildet. Denne informasjonen sendes til det neste laget, et lag som finner kanter. Gitt det andre skjulte lagets beskrivelse av bildet, et bilde beskrevet med kanter, så kan objektdeteksjon bli mulig ved å sammenligne en samling kanter, konturer, med deler av kjente objekter. (***Goodfellow m.fl. 2016 s. 6***)*

I 2016 så viste Joseph Redmon at det var mulig å lage en single-stage objektdetektor (**Redmon 2016**). Isteden for å gjøre inferens per anker så ble nettverket brukt kun en gang til å finne alle objektene i et bilde. Dette systemet, You only look once (YOLO), er et av de raskeste objektdeteksjonssystemene (**Redmon 2018**).

2.2.2 Dataaugmentering

Dataen som skal brukes til å trene en maskinlæringsmodell må først normaliseres på en standard måte, for eksempel ved å subtrahere gjennomsnittet over dataen, for så å skalere alle bildene slik at de får samme størrelse. Dette gjør at nettverket kan trenes på data med ulike oppløsninger. Det er også mulig å gjøre flere andre endringer. Støy kan legges til bildene, samt å snu dem horisontalt eller vertikalt for å skape nye bilder å trene nettverket på. (**Cadieu m.fl. 2014 s. 15**)

2.2.3 Heterogene maskiner

Alle datamaskiner er i dag heterogene. De består av en CPU og en GPU, en Central Processing Unit og en Graphics Processing Unit. En programmerer CPU-en med språk slik som Python, C, C++, osv. Til å begynne med så var grafikkprosessorer fixed-function, et bestemt API tillot maskinvareakselerasjon av enkelte grafikkinstrukser. (**Buck 2006 s. 5**)

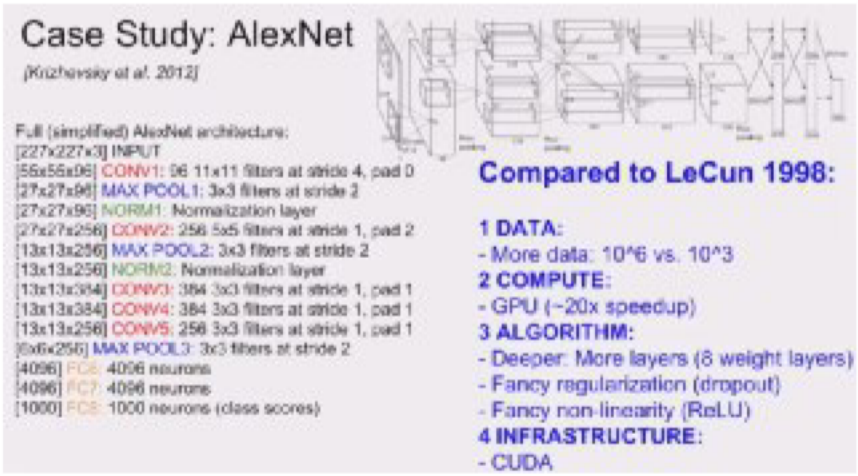
I 2006 introduserte Nvidia den første generelt programmerbare GPU-en, deres nye grafikkprosessorer kunne programmeres i CUDA, et språk som ligner på C. Dette gjorde det mulig å gjøre andre ting enn grafikk på grafikkprosessorer. Grafikkprosessorer er spesielt flinke til oppgaver som er ≪massively parallell≫. (**Buck 2006 s. 1**)

Det er ikke tilfeldig at det var i 2012 at deep learning tok av. Alex sitt team anvendte CUDA til å gjøre maskinlæring på en GPU istedet for en CPU, det hadde ikke blitt gjort tidligere (**Krizhevsky m.fl. 2012**). De anvendte en ikke-lineær aktiveringsfunksjon, noe som hadde blitt oppfunnet relativt nylig (**LeCun m.fl. 1998 s. 3**). Resultatene har ført til en revolusjon innenfor kunstig intelligens. Batch normalization har også forbedret resultatene. Se figur 5. (**Ioffe og Szegedy 2015 s. 1**)

I dette forsøket ble en GPU anvendt til å gjøre maskinlæring, å trene deep learning-modeller.

2.2.4 Maskinlæringsrammeverk

Det har blitt utviklet flere maskinlæringsrammeverk. Disse reduserer mye av kompleksiteten som går med på å lage nye modeller, og å trene modeller med ny data. Rammeverkene gjør at en får maskinvareakselerasjon av backpropagation



Figur 5: *AlexNet er på venstre side. På høyre side er nye fremskritt som har gjort deep learning populært. (***Karpathy 2014***)*

Figur 6: *Logoene til Darknet og PyTorch. Darknet er et populært rammeverk for maskinlæring skrevet i C (***Redmon 2016***). PyTorch er maskinlæringsrammeverk fra Facebook, den anvendes av objektdeteksjonssystemet Detectron2 (***Wu m.fl. 2020***).*

og andre maskinlæringsutregninger uten at en trenger å lære seg CUDA. I dette forsøket ble Detectron2 med rammeverket PyTorch og YOLO med rammeverket Darknet anvendt til å trene objektdeteksjonsmodeller. Se figur 6.

2.2.5 Azure nettsky

En virtuell maskin med en GPU ble leid fra nettskytjenesten Microsoft Azure til å trene modellene i dette forsøket. Satya Mallick presenterer hvordan en lager en virtuell maskin på nettskytjenesten Azure i kurset Deep Learning with PyTorch. Du kan lage virtuelle maskiner fra Azure Portal. Å begynne med Deep Learning er superenkelt med Azure. Azure gir deg virtuelle instanser med nesten alle de mest populære deep learning-rammeverkene ferdiginstallert på deres Data Science Virtual Machines. Da slipper man å bruke tid på installering av Nvidia-drivere og andre deep learning-biblioteker. Maskinene kommer også

med JupyterHub ferdig installert, om en ønsker å anvende Jupyter Notebooks. (**Mallick m.fl. 2020**)

2.2.6 SSH login

SSH ble anvendt til å logge inn i den GPU-akselererte nettskyinstansen. SSH er et dataprogram og protokoll som anvendes til å koble til eksterne maskiner. Den er først og fremst rettet mot Unix-maskiner, og har et tekstgrensesnitt (**Mallick m.fl. 2020**). På Windows så kan dataprogrammet PuTTY anvendes.[2]

Etter at systemet på Azure er oppe så kan man logge inn.

Å leie en GPU kan bli dyrt. Ikke glem å stoppe den virtuelle maskinen når den ikke er i bruk. Dette kan gjøres fra portalen til Azure.[3]

2.2.7 Anaconda programvareutgivelse

Python er et programmeringsspråk som er populært blant forskere.[4] (**Morris 2020**)

Anaconda er en programvareutgivelse som gjør det enkelt å bruke Python. Anaconda er også veldig nyttig for å unngå pakkekonflikter. På en Azure Data Science Virtual Machine så kommer flere Anaconda-miljøer ferdiginstallert. Anaconda PyTorch kan aktiveres med kommandoen `conda activate py37_pytorch`. (**Mallick m.fl. 2020**)

2.2.8 Label-data

Label-data er informasjon som beskriver rådataen i et datasett. En modell må ha label-data for å forstå informasjonen i treningssettet. Å lage label-data er en vesentlig del av et maskinlæringsprosjekt. Å lage label-data kan gjøres manuelt. Det kan i noen tilfeller være mulig å automatisere prosessen. Å lage label-data kan gjøres selv, eller gis som oppdrag til eksterne bedrifter. Om det er veldig mye data som må få label-data så kan prosessen bli svært tidskrevende og dyr. (**Kaller 2019**)

2.2.9 RetinaNet

RetinaNet, introdusert av Facebook-ansatte i 2017 i Focal Loss for Dense Object Detection på arXiv, ble ansett som den beste objektdeteksjonsmodellen da dette forsøket ble utført. Det var en forbedring av R-CNN. (**Lin m.fl. 2017**)

[2] https://www.putty.org/
[3] https://portal.azure.com
[4] https://www.kaggle.com/learn/python

RetinaNet er implementert i Detectron2, som bruker deep learning-rammeverket PyTorch. Grensesnittet er programmeringsspråket Python 3. Detectron2 er utgitt under en Apache 2.0-lisens, det vil si at den kan brukes til hva enn en vil på en hvilken som helst måte, gratis, så lenge man aksepterer at Facebook ikke blir ansvarlig for arbeidet som gjøres, og så lenge man ikke krediterer arbeidet en gjør til Facebook (**The Apache Software Foundation 2004**). Detectron2 installeringsinstrukser er tilgjengelig på nettet.[5]

2.2.10 YOLO

You only look once er en state-of-the-art, real-time objektdeteksjonssystem. På en Pascal Titan X, en mektig Nvidia-GPU, så prosesserer den bilder ved 30 FPS og har en mAP på 57.9 % på COCO test-dev (**Redmon 2018**). På min Macbook Pro fra 2017 så drar den 1 FPS på CPU-en på datasettet av torsk og sei. Ifølge Redmon så er YOLO like bra som Focal Loss, det vil si RetinaNet, men omtrent fire ganger raskere. Redmon mener en kan lett endre størrelsen på YOLO-modellen, den vil bli tregere men enda mer nøyaktig (**Redmon 2016**)

2.2.11 OpenCV

OpenCV (Open Source Computer Vision Library) er et et programvarebibliotek for maskinsyn og maskinlæring med åpen kildekode. OpenCV ble laget for å levere en kjent infrastruktur for maskinsynprogrammer og OpenCV-teamet ønsker å gjøre maskinsensorikk mer utbredt i kommersielle produkter. OpenCV er et produkt gitt ut under BSD-lisensen, det gjør biblioteket tilgjengelig for bedrifter som ønsker å anvende kildekoden (**OpenCV Team 2020**).

OpenCV har grensesnitt for C++, Python, Java og MATLAB og støtter Windows, Linux, Android og macOS. OpenCV-teamet forsøker først og fremst å levere sanntids maskinsyn for dataprogrammer og drar nytte av prosessorteknologi slik som MMX- og SSE-instruksjoner når de er tilgjengelige. Støtte for CUDA og OpenCL var under utvikling, men var enda ikke tilgjengelig da denne oppgaven ble skrevet. Det hadde gjort dataprogrammet mye raskere siden den ville dratt nytte av GPU-en og ikke bare CPU-en på maskinen. OpenCV er skrevet i C++ og virker uten navnekonflikter med STL namespaces. (**OpenCV Team 2020**)

2.2.12 Definisjon av nøyaktighet, presisjon og Intersection over Union

Presisjonen er et mål på hvor mange bounding boxes objektdeteksjonsystemet klarte å klassifisere riktig (**Siddiqui m.fl. 207 s. 383**). Matematisk defineres

[5] https://detectron2.readthedocs.io/

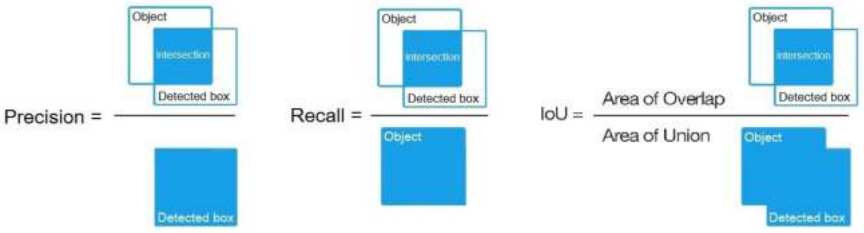
dette slik:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{1}$$

Recall er et mål på hvor mange objekter objektdetekteringssystemet klarte å klassifisere riktig (**Siddiqui m.fl. 207 s. 383**). Matematisk defineres dette slik:

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \tag{2}$$

IoU (Intersection over Union) er en evalueringsmåling som blir brukt til å måle nøyaktigheten til en objektdetektor på et gitt datasett. Det defineres som overlappingen av området modellen tror bounding box-ene finnes og der ground-truth bounding box-ene er over området modellen tror bounding box-ene finnes sammen med ground-truth bounding box-ene. Se figur 7. (**Bochkovskiy 2020**)



Figur 7: *Precision, recall og IoU. (***Bochkovskiy 2020***)*

2.2.13 Måle nøyaktigheten til Darknet-modell

Deep learning-modeller bør iterativt forbedres, og resultatene sammenlignes for hvert forsøk. Det er best å endre kun en parameter om gangen og se om modellen blir bedre. Dette er tidkrevende. TensorBoard gjør det lettere å sammenligne nye resultater med eldre resultater, grafene under viser to forsøk, hver figur har to grafer. Det nyeste forsøket er den mørkeste grafen.

Mens treningen pågår vises feilmeldinger og statusen til nettverket. Treningen kan avsluttes når training loss ikke lenger avtar. Se et eksempel på Darknet-utmatninger under:

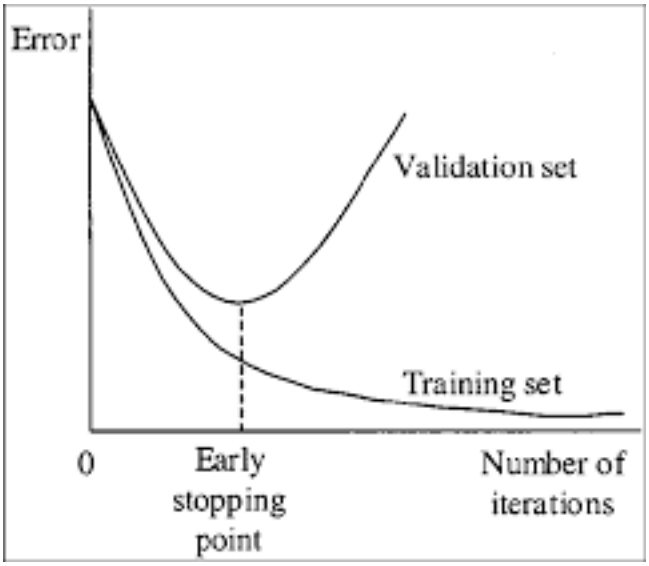
```
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg
(IOU: 0.366654, GIOU: 0.298158), Class: 0.464846,
Obj: 0.560030, No Obj: 0.549277, .5R: 0.191489,
.75R: 0.021277, count: 94, class_loss = 5464.241699,
iou_loss = 69.363770, total_loss = 5533.605469
```

Total loss er den viktigste målingen som gjøres på deep learning-modeller. Målet med treningen av deep learning-modeller er å få total loss til å bli så lav som mulig. Når modellen er ferdig trent, så vil vanligvis total loss konvergere, den blir flat. Om total loss ser ut til å fortsette å avta, så bør modellen trenes over flere iterasjoner. Learning rate har en stor innvirking på når total loss konvergerer. Total loss kan gå fra 0,05 (for en liten modell og et enkelt datasett) til 3,0 (for en stor modell og et vanskelig datasett). (**Bochkovskiy 2020**)

Treningen med optionen -map vil føre til mAP (mean Average Precision)-utregninger, da vil `Last accuracy mAP@0.5 = 18.50` og lignende dukke opp i konsollen. Mean Average Precision er gjennomsnittet av maksimumpresisjonene ved alle recall-verdiene. Det er en mye bedre indikator enn total loss på hvor bra modellen har blitt. Trening bør pågå så lenge mAP øker. (**Bochkovskiy 2020**)

Når treningen er ferdig så må den beste .weights-filen velges fra backup-mappen. Overtilpasning, eller *overfitting*, kan bli et problem. Overfitting er når modellen kan detektere objektene i treningsdatasettet, men ikke i noen andre bilder. Derfor bør trening ikke foregå over flere iterasjoner enn der Early Stopping Point forekommer, se figur 8. (**Bochkovskiy 2020**)

Velg alltid weights-filen med den høyeste mAP-skåren.



Figur 8: *Trening av en modell bør avsluttes når «Early stopping point» er nådd.* (**Bochkovskiy 2020**)

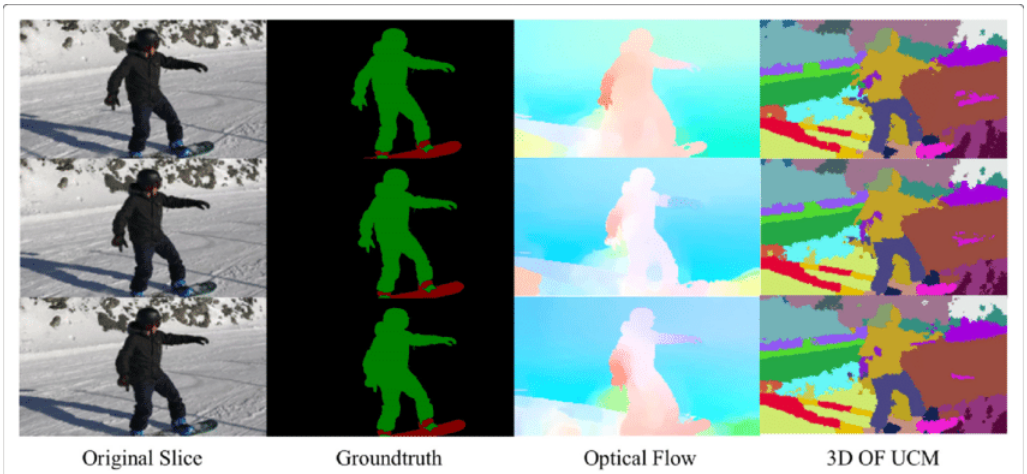
2.3 Tidligere arbeid

Noen forskere har allerede rukket å anvende deep learning til å klassifisere fisk fra undervannsvideo. I 2016 presenterte CVG Jena Fulda teamet resultater av klassifisering av fisk på SeaCLEF-datasettet. De anvendte «convolutional» nevrale nettverk, nettverk for bildeklassifisering, og klarte å gjøre objektdeteksjon samt å klassifisere forskjellige fiskearter. De testet bakgrunnssubtraksjon, en teknikk som var typisk for deteksjon av objekter i undervannsbilder på den tiden. De fant ut at å bruke bakgrunnssubtraksjon fungerte dårligere enn «object proposal classification» (OPC) for deteksjon av fisk. (**Rodner m.fl. 2016**)

Også i Norge så har det blitt gjort forsøk på maskinsyn og maskinlæring innenfor havforskning. Havforskningsinstituttet testet ut Deep Vision, et kamerasystem fra Scantrol Deep Vision AS, i 2017, 2018 og senest igjen i 2019. Kamerasystemet ble festet ved inngangen til en tråL for å måle mengde og artsbestemme fisk uten å hente den om bord og ta livet av den. Poenget med maskinlæringen var å automatisere artsbestemmelsen. Artsbestemmelsen ble gjort tidligere ved at forskere tolket hvert bilde kameraene tar manuelt. Modellen til Scantrol Deep Vision kunne gjenkjenne artene sild, kolmule, makrell og lysprikkfisk. Forsøkene viste at omtrent 90 prosent av fisken ble riktig identifisert av systemet. (**Fenstad 2019**)

Det norske kompaniet Skala Maskon har også sett på maskinsyn, de har utviklet en vaksineringsmaskin som bruker maskinsyn for å finne punktet hvor vaksinen skal settes på hver fisk. De bruker programvare fra Scorpion Vision. Skala Maskons fullautomatiske vaksineringsmaskin drives av en enkelt operatør og kan vaksinere og sortere opp til 40 000 smolt i timen. Maskinene kan vaksinere singel, dobbel, trippel og intramuskulære doser samtidig. (**Falstad 2016**)

Siddiqui sitt team presenterte i 2017 klassifisering av forskjellige fiskearter i undervannsvideoer der de brukte en modell med vekter fra forhåndstrente dype nettverk. Det er et eksempel på såkalt «transfer learning». Dette automatiske systemet kunne med høy nøyaktighet detektere, tracke og klassifisere fisk og andre marine arter i undervannsvideoer uten menneskelig overvåkning. De oppdaget at typiske maskinsynteknikker fungerer dårlig på undervannsbilder. Bakgrunnen er kompleks og både fasongen og teksturen på de forskjellige fiskeartene kan være svært like. Datadrevne programmer, slik som nevrale nettverk, krever enorme mengder kategorisert data, ellers har de en tendens til å overtilpasse treningsdataen og feiler når de blir gitt testdata som den ikke har sett før, det vil si data som ble utelatt ved treningen. I artikkelen deres «Automatic fish species classification in underwater videos» presenteres state-of-the-art maskinsynsmetoder for fine-grained klassifisering av fiskearter basert på deep learning-teknikker. Konvolveringsnettverk, nettverkene som anvendes til bildeklassifisering, ble forhåndstrent på et mye større og mer generelt datasett. Ved å forhåndstrene nettverket på et generelt datasett så kan man bruke mindre treningsdata. Support Vector Machine (SVM) ble brukt til å trene nettverket og de oppnådde en nøyaktighet på 94,3 %. De tropiske fiskeartene ble tatt fra under-

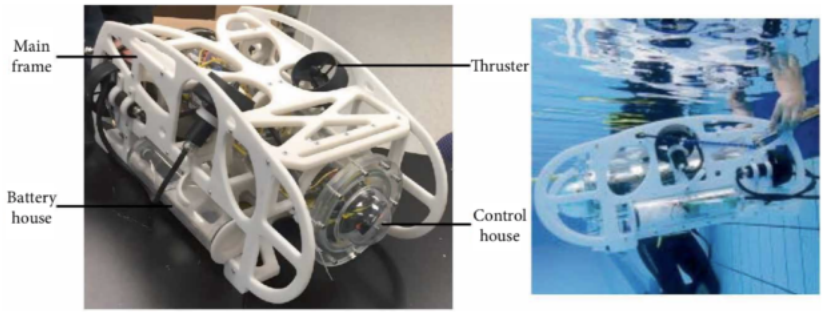


Figur 9: *Semantisk segmentering generert fra optical flow. Optical flow er bevegelsene av objektene mellom bildene i en film. (***Lin 2019***)*

vannsvideoer fra kysten av vest-Australia. Forskerne mener automatiske klassifiseringssystemer kan brukes til å identifisere fisk fra undervannsvideoer, og at det er et billig alternativ til manuell identifikasjon av fisk. (**Siddiqui m.fl. 2017**)

I 2018 presenterte Xu og Matzner forskning på deteksjon, tracking og klassifisering av fisk i undervannsvideoer. Undervannsvideoer anvendes til å studere økosystemene i havene og i elver, først og fremst for å drive med miljøforskning. Å observere mengden av liv, samt vanene til fiskene i forskjellige miljø, er eksempler på miljøforskning. Optisk video gir detaljert informasjon som mennesker kan forstå. Mennesker kan naturlig forstå verden visuelt, men manuell analyse av video tar tid. Å automatisere arbeidet er nødvendig om en har store datamengder, og en ønsker å få arbeidet fullført på en gitt tid på en effektiv måte. Dette kan være nødvendig om en skal gjøre gode avgjørelser som påvirker de marine økosystemene. Maskinsyn og maskinlæring har vist seg å være effektive metoder for å automatisere overvåking. Undervannsbilder er kjent for å være spesielt utfordrende å håndtere. Mengden lys under vann varierer over tid, og mengden lys påvirkes også av avstand fra lyskilder. Det er lite kontrast under vann og bakgrunnen er ofte svært avansert, den består av blant annet flytende vegetasjon. Videre påvirkes bildene av turbiditet. (**Xu og Matzner 2018**)

Salman sitt team viste i 2019 at de kunne estimere biomasse automatisk ved å bruke undervannsvideoer og dype nevrale nettverk. Et slikt system må takle varierende lysstyrke, vinkelen fiskene sees fra, forskjellige havbunnsstrukturer, bevegelser i vegetasjon i bakgrunnen og forskjellige former og teksturer mellom fiskearter. De anvendte en R-CNN, som er en state-of-the-art maskinlæringsteknikk, et alternativ til YOLO. Teknologien anvendes til objektdeteksjon, den gir



Figur 10: *Roboten til Cui sitt team. Den kan detektere fisk og skal brukes til havforskning. (***Cui m.fl. 2020***)*

informasjon om hvor i et bilde forskjellige objekter er samt hvor mange det er av de forskjellige objektene. De trente et nettverk og anvendte bakgrunnssubtraksjon og optical flow, sammen med råbilde ga dette områder der fisk skulle detekteres. Optical flow er bevegelsen av objektene mellom fortløpende bilder i en sekvens, det kommer av de relative bevegelsene mellom objektet og kameraet. Det brukes til bildesegmentering (**Lin 2019**). Se figur 9. De brukte to datasett, et fra Fish4Knowledge sitt Complex Scenes-datasett, den består av undervannsvideoer, og LifeCLEF 2015-datasettet. Resultatene de fikk var så gode at de anbefaler teknikkene som de har anvendt for fiskedeteksjon til andre med lignende problemer. (**Salman m.fl. 2019**)

I 2020 så skrev Cui sitt team «Fish Detection Using Deep Learning». De jobber på en robot som kan utforske havet, se figur 10. I artikkelen deres dokumenterer de hvordan en lager en modell, et nevralt nettverk, som kan utføre fiskedeteksjon. De anvendte data augmentation-teknikker. De brukte også Dropout-algoritmen for å hindre at modellen ble overtilpasset. De endret også på loss-funksjonen ettersom nettverket ble trent. Ved å bruke disse teknikkene så klarte de å redusere treningstiden og training loss. Systemet de lagde var optimalisert for et innebygd system, altså en svakere datamaskin, en AUV-robot. (**Cui m.fl. 2020**)

3 Material og metode

Trening av kunstige nevrale nettverk gjøres i seks steg. Det ble trent opp to modeller, en RetinaNet-modell og en YOLOv4-modell. RetinaNet hadde forhåndstrente vekter fra en modell trent på COCO-datasettet til Microsoft (**Lin m.fl. 2015 s. 1**). YOLOv4 hadde forhåndstrente vekter fra en «Darknet-modell», modellene ble ikke trent fra scratch.

Microsoft Azure-plattformen ble anvendt til å trene modellene. Microsoft tilbyr Nvidia Titan GPU-er i skyen. Å anvende GPU-er når en trener kunstige nevrale nettverk gjør at treningen tar mindre tid (**Dean m.fl. 2012 s. 1**).

Kildekoden kan lastes ned fra GitHub.[6]

3.1 Å forstå problemet

Mengden av to fiskeslag, torsk og sei, skulle telles med maskinsyn. Det er to klasser. Modellene RetinaNet og YOLOv4 ble valgt, de er dype nettverk som kan forstå og trenes på bilder.

3.2 Få tak i data

Datasettet bestod av bilder fra en lagringsmerd av torsk, og bilder av sei fra Havbruksstasjonen i Tromsø. Se figur 11 og 12. Opptakene av fiskene ble laget av Nofima og ble omgjort til bilder. Label-data ble lagt inn manuelt. Se figur 13. Dataen av torsk var fargebilder, så de bestod av tre kanaler (rødt, grønt og blått) og hadde størrelsen 1920 × 1080. Bildene av sei var gråtoner, kun en kanal, og hadde størrelsen 640 × 480.

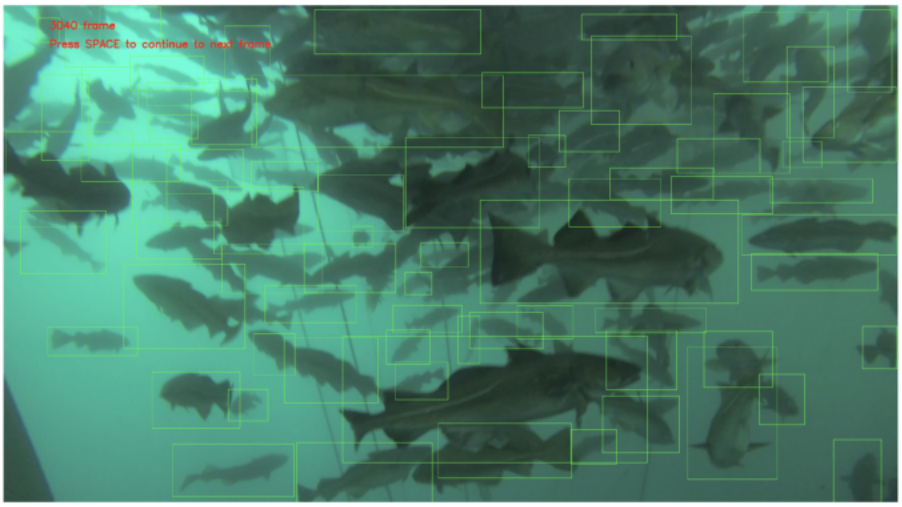
Figur 11: *Eksempelbilder fra en video av en lagringsmerd med torsk. Disse bildene er en del av datasettet anvendt for deteksjon av torsk og sei (***Nofima 2020***).*

[6] https://github.com/alanhaugen/TMAT3004-Bacheloroppgave

Figur 12: *Kamerautstyret som ble anvendt til å lage sei-datasettet (film av sei under vann). Nofima anskaffet et undervannskamera av typen Steinsvik Orbit-3300 som styres via et MB-3000 kontrollpanel. Foto: Nofima (***Lindberg og Evensen 2020***).*

Datasettet bestod av 208 bilder av torsk, med til sammen 4582 instanser av torsk i bildene. Det var 604 bilder av sei med 5525 instanser av fisken i bildene. Bildene av torsk og sei ble satt i hver sin mappe. De ble delt etter forholdet 80/20, 80 % av dataen var for trening, og 20 % for validering.

3.3 Lage label-data



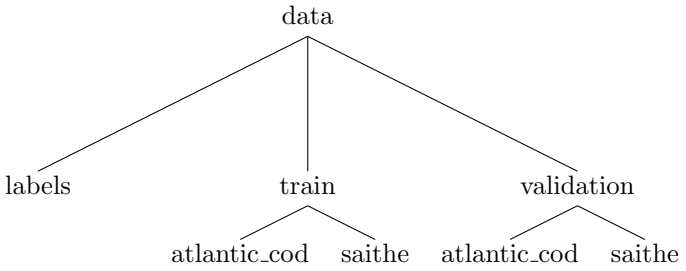
Figur 13: *Et dataprogram som ble utviklet for å lage label-data. Råbildene er basert på opptak fra Nofima (***Nofima 2020***). Figuren viser bilde nummer 3040.*

Dataprogrammet i figur 13 laster inn hvert femtiende bilde fra en video av en lagringsmerd av torsk. Denne videoen er 7 minutter lang. En modell som kan detektere objekter krever å bli trent på bilder av det den skal kjenne igjen, bilder med label-data. Label-data består av klassenummer, og koordinatene 'xmin', 'ymin', 'xmax', og 'ymax', som skaper et rektangel rundt objektene som finnes i bildet, for hvert objekt som finnes i treningsdataen. Med dette dataprogrammet så er å lage slik informasjon fra et bilde enkelt, brukeren trenger kun å lage et rektangel rundt hver torsk. Klassenummeret for torsk, her 0, og posisjonene av rektangelet, 'xmin', 'ymin', 'xmax', og 'ymax', kjent som objektets ground truth bounding box, lagres til en tekstfil. Rådataen, bilder uten rektanglene eller den røde teksten som er øverst i venstre hjørne, lagres også automatisk til en mappe.

3.3.1 Utforsk og forstå dataen

Det er viktig å se på dataen før en begynner å trene et nettverk. Det er viktig å se etter bias i bildene. Det er lurt å gå igjennom hvert eneste bilde, og se om det er noe en kan lære. Torsk- og seibildene kunne bli augmentert, de kunne blant annet bli snudd horisontalt. Det gir dobbelt så mye treningsdata. Vær obs på at ikke alle treningssett tillater dette, for eksempel et datasett med lego. Lego har brikker som er speilbilder av hverandre. Da kan ikke bildene bli snudd horisontalt ettersom det vil gjøre at nettverket ikke kan se forskjell på alle brikkene. En fisk er fortsatt samme type fisk speilvendt.

Det var to klasser i datasettet, torsk og sei. Se figur 14 og tabell 1.



Figur 14: *Et tre av mappestrukturen til datasettet.*

Tabell 1: Klassenavn og navneenkoding for datasettet

Klassekode	Klassenavn	Norsk klassenavn
0	atlantic_cod	torsk
1	saithe	sei

'`Labels`'-mappen inneholdt bounding box-data for bildene. Det var en eller flere linjer per '`.txt`'-fil. Hver linje representerer en bounding box. Representasjonen er i formatet ('xmin', 'ymin', 'xmax', og 'ymax'). Se tabell 2.

Filene som ble brukt til trening og validering ble definert i `data/fish_train.txt` og `data/fish_test.txt`. Se tabell 3.

3.4 Gjør klar dataen

Da dataen hadde blitt organisert, og label-data nøye konstruert, så ble nettverkene konfigurert og treningen ble satt i gang. Dataen ble matet inn i treningsprogrammet, inn i deep learning-rammeverkene.

Det ble trent opp to nettverk. Det ene, RetinaNet, er en del av Detectron2 fra Facebook. Den ble trent og konfigurert med PyTorch. Den andre modellen var

Tabell 2: Label-data for bildet `data/train/atlantic_cod/fish_9440.png`, lagt i filen `data/labels/fish_9440.txt`

Klassekode	xmin	ymin	xmax	ymax
0	238	643	582	882
0	80	858	368	1071

YOLOv4, den trenes med deep learning-rammeverket Darknet. Konfigurasjon gjøres ved å endre på tekstfiler og kommandolinjeparameterene til rammeverket.

3.4.1 Dataaugmentering

Det ble ikke gjort endringer på standardmåten Detectron2 og Darknet gjør dataaugmentering.

3.4.2 YOLOv4 label-data og bildeformat

YOLOv4 har et annet label-format sammenlignet med RetinaNet, dessuten krever den bilder i jpeg-formatet, med filutvidelsen '`.jpg`'. Label-dataen legges sammen med bildene. Konvertering fra RetinaNet sitt format til YOLOv4 ble gjort med et awk-skript. Formatet til YOLOv4 er:

```
[klassenavn] [objekt midtpunkt i X] [objekt midtpunkt i Y]
[objekt vidde i X] [objekt høyde i Y]
```

Et nyttig verktøy til å konvertere bilder er John Cristy sin Image Magick.[7] Her er hvordan bildene ble konvertert fra Adobe sitt png-format til jpeg:

```
mogrify -format jpg *.png
```

3.5 Tren nettverket på et lite utvalg av dataen som en test før en trener et fullt nettverk

Nettverket ble først trent opp på et utvalg av treningsdataen, for å teste nettverket. 208 bilder med 4582 instanser av torsk ble grunnlaget for de første RetinaNet- og YOLOv4-modellene. Dette steget gjøres som en sanity check før en begynner på trening som kan ta mange timer.

[7] https://imagemagick.org

Tabell 3: Eksempel på filnavn i `data/fish_train.txt` og `data/fish_test.txt`

Filnavn i `fish_test.txt`
validation/atlantic_cod/fish_590.png
validation/atlantic_cod/fish_640.png
⋮

3.5.1 RetinaNet konfigurasjon og trening

RetinaNet-modellen ble trent med en learning rate på 0,001 over 500 iterasjoner. Modellen brukte vekter fra COCO, den ble lastet ned med Detectron2 fra Detectron2 Model Zoo, den heter COCO-Detection/retinanet_R_50_FPN_3x.yam. RetinaNet sin score threshold ble satt til 0,5. Batch size var 4.

Et utvalg av programkoden som har blitt skrevet for denne oppgaven har blitt lagt ved i vedlegg 6.

Først må bibliotekene lastes inn. Se Python-koden på side 1 i vedlegg 6.

Konfigurasjonen defineres i kodesnutten på side 1 i vedlegg 6.

Konfigurasjonen av batch size, number of iterations og learning rate er de viktigste variablene, det er det neste som settes i listingen under.

```
# batch size
cfg.SOLVER.IMS_PER_BATCH = 4

# choose a good learning rate
cfg.SOLVER.BASE_LR = 0.001

# We need to specify the number of iterations for training
max_iter = 500

cfg.SOLVER.MAX_ITER = max_iter

# number of output class
cfg.MODEL.RETINANET.NUM_CLASSES = len(thing_classes)

# update create ouptput directory
cfg.OUTPUT_DIR = output_dir
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
```

Neste kodesnutt er trening, koden finnes også i vedlegg 6 side 3.

```
# Create a trainer instance with the configuration.
trainer = DefaultTrainer(cfg)

# if rseume=False, because we don't have trained model yet. It
# will download model from model url and load it
trainer.resume_or_load(resume=False)

# start training
trainer.train()
```

3.5.2 YOLOv4 trening

Alexey Bochkovskiy sin variant av Darknet ble anvendt, det er en versjon av Joseph Redmon sin Darknet: Open Source Neural Networks in C med Windows- og Linux-støtte. Den støtter nyere versjoner av OpenCV og det som var den nyeste versjonen av YOLO da denne rapporten ble skrevet, YOLOv4, som ble utgitt 23. April 2020 (**Bochkovskiy m.fl. 2020**). En god guide til Darknet og YOLO finnes på githuben til Alexey. (**Bochkovskiy 2020**)

YOLOv4 ble trent med vekter fra en forhåndstrent modell. Først må darknet installeres. Se under.

```
git clone https://github.com/AlexeyAB/darknet
cd darknet
```

Omgjør linjene GPU=0 og OPENCV=0 til GPU=1 og OPENCV=1 i Makefile.

Listing 1: De første linjene i Makefile

```
GPU=1
CUDNN=0
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=0
ZED_CAMERA=0 # ZED SDK 3.0 and above
ZED_CAMERA_v2_8=0 # ZED SDK 2.X
```

Kompiler rammeverket.

```
make
```

Dette vil kompilere Darknet for GPU-en med OpenCV-støtte.

For å få gode resultater så ble forhåndstrente vekter for konvolveringslagene lastet ned. Den forhåndstrente darknet modellen tar 162 MiB.

```
wget https://github.com/AlexeyAB/darknet/releases/download/\
darknet_yolo_v3_optimal/yolov4.conv.137
cp yolov4.conv.137 build/darknet/x64
```

Filen obj.data ble laget med følgende innhold.

Listing 2: obj.data

```
classes = 1
train  = /yolo/data/fish_train.txt
valid  = /yolo/data/fish_test.txt
names = obj.names
backup = /yolo/backup/
```

obj.names ble laget med følgende innhold.

Listing 3: obj.names

```
atlantic_cod
saithe
```

YOLOv4 må konfigureres.

```
cp cfg/yolov4-custom.cfg yolo-obj.cfg
```

Det er kun en klasse i testutvalget. Følgende endringer ble gjort i yolo-obj.cfg-filen.

- linjen batch ble satt til batch=64
- linjen subdivisions ble satt til subdivisions=16
- linjen max_batches ble satt til max_batches = 6000
- linjen steps ble satt til steps=4800,5400
- størrelsen av nettverket ble satt til width=416 og height=416
- linjen classes ble satt til classes=1 for hvert av de tre YOLO-lagene
- linjen filters ble satt til filters=18 i hvert av de tre konvolveringslagene, '`[convolutional]`', som kommer før '`[yolo]`'

For hver linje i `fish_train.txt` og `fish_test.txt` så ble filendingene omgjort til '`.jpg`'.

Nettverket ble trent med følgende kommando på Ubuntu Linux-maskinen på Azure.

```
./darknet detector train obj.data yolo-obj.cfg yolov4.conv.137 -map
```

Om en bruker Windows brukes følgende kommando:

```
darknet.exe detector train obj.data yolo-obj.cfg yolov4.conv.137 -map
```

Treningen kan ta flere dager. Etter treningen så ble den beste modellen lastet ned.

```
scp awh@public-ip:/yolo/backup/yolo-obj_final.weights yolo.weights
```

Modellen ble brukt i OpenCV-programmet beskrevet i del 3.8.2.

3.6 Trene modellen på hele datasettet

Det ble trent opp en ny RetinaNet- og en ny YOLOv4-modell med hele datasettet. Nettverkene hadde blitt testet og fungerte slik som forventet.

Hele datasettet, inkludert seidata, ble lagt til `fish_train.txt` og `fish_test.txt`. Som tidligere, 80 % av dataen var treningsdata, 20 % valideringsdata.

3.6.1 RetinaNet

RetinaNet trengte ikke å bli videre konfigurert. Koden som ble skrevet tidligere ble anvendt til å trene nettverket.

```
python train.py
```

3.6.2 YOLOv4

YOLOv4 måtte bli konfigurert for to klasser, torsk og sei. Det ble gjort følgende endringer i yolo-obj.cfg-filen.

- linjen classes ble satt til classes=2 for hvert av de tre YOLO-lagene
- linjen filters ble satt til filters=21 i hver av de tre konvolveringslagene som kommer før YOLO-lagene

Linjen classes = 1 ble omgjort til classes = 2 i obj.data.

Filendingene i `fish_train.txt` og `fish_test.txt` var '`.jpg`', som før.

3.7 Forbedre modellen

YOLO-modellen ble oppgradert til YOLOv4. RetinaNet-modellen ble mye bedre når den ble trent over flere iterasjoner. Endringene ble gjort underveis i prosjektet.

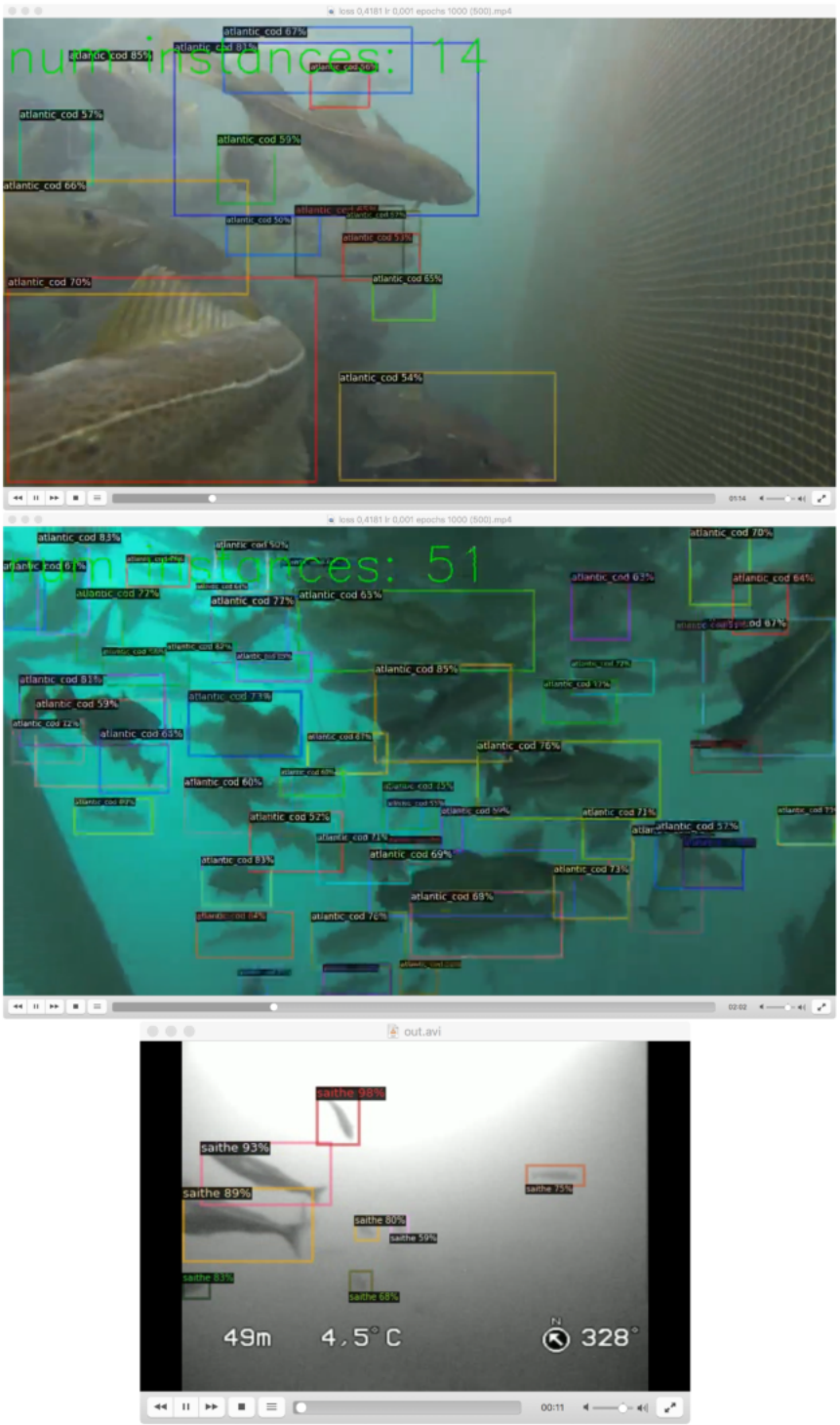
3.8 Inferens

Inferens er når en trent modell er brukt til å forutsi hva en prøve inneholder. Inferens kalles også for en forward pass. En modell gjør ikke backpropagation når den gjør inferens, backpropagation gjøres for å regne ut nøyaktigheten til modellen og til å oppdatere vektene i modellen, noe som kun skjer under trening.

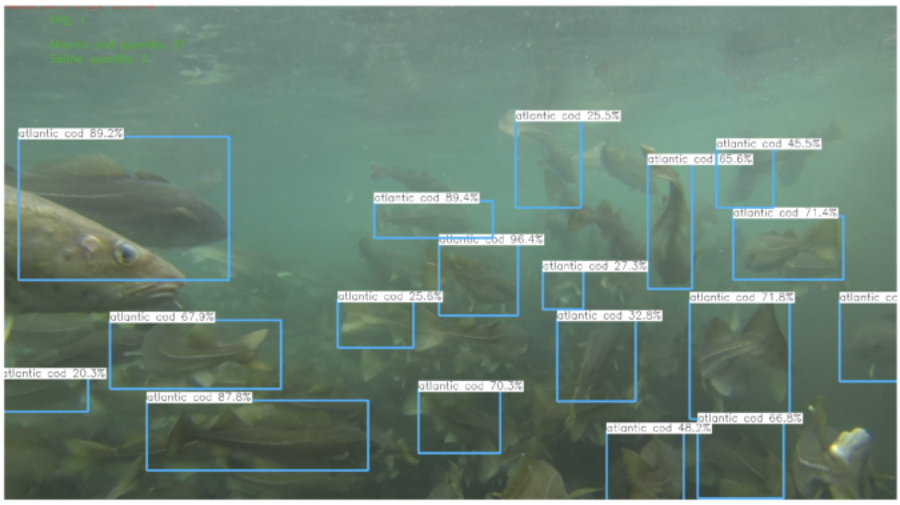
3.8.1 RetinaNet

Ettersom OpenCV ikke kunne laste inn en PyTorch-modell da denne oppgaven ble skrevet, så ble inferens gjort med Python, og det ble ikke gjort et forsøk på å gjøre inferens i sanntid på en videostrøm med RetinaNet. Se resultater i figur 15.

Koden for inferens er i vedlegg A, side 4.



Figur 15: *Inferens med en RetinaNet-model. Råbildene er fra Nofima (***Nofima 2020***).*



Figur 16: *Inferens med YOLOv4. Rundt fiskene så lager YOLOv4 bounding boxes der den tror det finnes fisk, og gir den en label basert på det den tror at det er. Her får alle fiskene labelen «atlantic cod», dette er fra en lagringsmerd med torsk. Ved siden av navnet av klassen så står det hvor sikker nettverket er på at den har funnet en torsk, presisjonen av inferensen. I grønn tekst, øverst til venstre, står det at algoritmen kan telle 31 torsk i bildet, og 0 sei. Råbildet er fra Nofima (***Lindberg og Evensen 2020***).*

3.8.2 OpenCV-program

Det ble skrevet et program i C++ som bruker OpenCV til fiskedeteksjon i sanntid med YOLOv4-modellen. OpenCV hadde fortsatt dårlig støtte av ONIX-formatet, Caffe2-formatet, og PyTorch da denne oppgaven ble skrevet. OpenCV 3.4 ble anvendt, da den nylig fikk støtte for YOLOv4. (**Batanina 2020**)

Her installeres OpenCV 3.4 på et Unix-system:

```
mkdir opencv-3.4
cd opencv-3.4
git clone https://github.com/opencv/opencv
git clone https://github.com/opencv/opencv_contrib
cd opencv_contrib
git checkout 3.4
cd modules
dir=$(pwd)
cd ../../opencv
git checkout 3.4
mkdir build
cd build
cmake OPENCV_EXTRA_MODULES_PATH=$dir ..
```

```
make
make install
```

Anaconda individual edition med Python 2 ble brukt. Det ble laget en `CMakeLists.txt`-fil for prosjektet. Prosjektet består av to programmer. OBJECT_DETECTOR gjør fiskedeteksjon. Se figur 16. DATASET_TOOL ble brukt til å lage label-data ut av videoer. Se figur 13.

OBJECT_DETECTOR gjør inferens med OpenCV dnn-modulen, den laster inn en YOLOv4-konfigurasjon, en modell, og en videostrøm. Videostrømmen kan komme fra et kamera koblet til maskinen eller en fil på filsystemet. Programmet analyserer videostrømmen og logger mengden torsk og sei som blir observert. Se figur 16.

`main.cpp` er kildekoden til OBJECT_DETECTOR. Først lastes bibliotekene inn. Se side 6 i vedlegg 6.

`getOutputsNames` henter ut label-navn fra det ytterste laget, «the fully connected layer», i modellen. Det er her utmatningene fra modellen er. Se figur 4 og side 6 i vedlegg 6.

`drawPred` lager en bounding box rundt objektene som detekteres av modellen. Se side 7 i vedlegg 6.

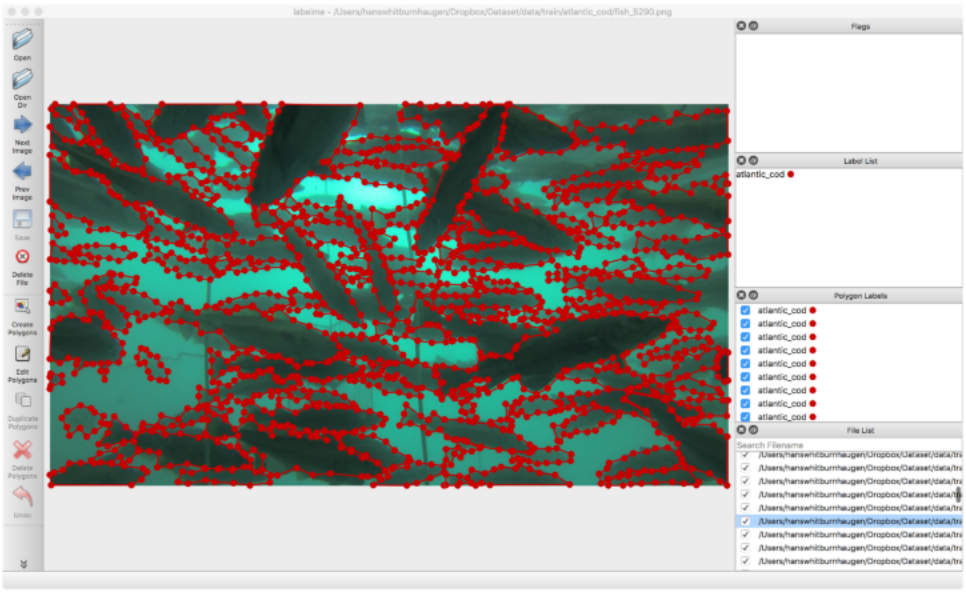
`postprocess` henter alle instanser av fiskene i et bilde som er over en non-maxima suppression confidence boundary. Deretter sjekker den om objektet er en torsk eller sei, og teller mengden torsk og sei i bildet. Se side 7 i vedlegg 6.

`main` er programmets entry point. Det er her dataprogrammet starter. Den åpner en logg, log.csv, og så åpner den en videostrøm. Deretter laster den inn YOLOv4-modellen. Se side 9 i vedlegg 6.

Den siste kodesnutten er programmets main loop. Denne koden repeteres til videoen har gått tom for bilder, eller at brukeren trykker på ESC på tastaturet. For hvert bilde i videostrømmen så lagres antall torsk og sei observert og tidskode til log.csv, med mindre det er ingen torsk eller sei som blir oppdaget. Se side 10 i vedlegg 6.

3.9 Segmentering

Det ble laget label-data for segmentering i datapgrogrammet labelme (**Wada 2016**). Se figur 17. Det ble ikke gjort et forsøk på segmentering da objektdeteksjon ga bedre resultater enn forventet.



Figur 17: *Dataprogrammet labelme (***Wada 2016***). Råbildet er fra en levendelagringsmerd med torsk (***Nofima 2020***). Denne label-dataen kan anvendes til å trene opp en modell til semantisk segmentering (***Shotton m.fl. s. 1***).*

4 Resultater

Det ble trent opp to modeller, en RetinaNet-modell og en YOLOv4-modell. Se figur 15 og 16. Dataen av torsk bestod av tre kanaler, og hadde størrelsen 1920 × 1080. Bildene av sei bestod av kun en kanal, og hadde størrelsen 640 × 480. Se figur 11.

4.1 RetinaNet

```
Average Precision (AP):
  AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small            % AP for small objects: area < 32^2
  AP^medium           % AP for medium objects: 32^2 < area < 96^2
  AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1            % AR given 1 detection per image
  AR^max=10           % AR given 10 detections per image
  AR^max=100          % AR given 100 detections per image
AR Across Scales:
  AR^small            % AR for small objects: area < 32^2
  AR^medium           % AR for medium objects: 32^2 < area < 96^2
  AR^large            % AR for large objects: area > 96^2
```

Figur 18: *Evalueringskortet til COCO. En modell som har blitt trent med vekter fra COCO kan evalueres etter dette kortet.*

AP står for ≪Average Precision≫, og AR står for ≪Average Recall≫. De er mål på hvor nøyaktig en deep learning-modell er. Se tabellene på side 31.

4.1.1 COCO deteksjonsevaluering

Resultatene i tabell 4.1 kan sammenlignes med evalueringskortet i figur 18. Resultatene bør være bedre enn COCO-evalueringskortet. AP (primary challenge metric) bør være mer enn 0,5. RetinaNet-modellen har en AP lik 0,285. Se tabell 4.1. AP for torsk var 23,1 %, og 33,9 % for sei. Se tabell 4.1.

4.1.2 TensorBoard resultater

Alle grafene under er fra TensorBoard, et verktøy fra Google som er utviklet til å hjelpe dataforskere med å trene opp deep learning-modeller og dele resultatene til forsøkene deres. TensorBoard visualiserer maskinlæringseksperimenter

Tabell 4: AP (primary challenge metric) og Average Recall

Average Precision (AP) @ [IoU=0.50:0.95	area= all	maxDets=100]	= 0.285
Average Precision (AP) @ [IoU=0.50	area= all	maxDets=100]	= 0.663
Average Precision (AP) @ [IoU=0.75	area= all	maxDets=100]	= 0.200
Average Precision (AP) @ [IoU=0.50:0.95	area= small	maxDets=100]	= 0.067
Average Precision (AP) @ [IoU=0.50:0.95	area=medium	maxDets=100]	= 0.217
Average Precision (AP) @ [IoU=0.50:0.95	area= large	maxDets=100]	= 0.402
Average Recall (AR) @ [IoU=0.50:0.95	area= all	maxDets= 1]	= 0.038
Average Recall (AR) @ [IoU=0.50:0.95	area= all	maxDets= 10]	= 0.248
Average Recall (AR) @ [IoU=0.50:0.95	area= all	maxDets=100]	= 0.433
Average Recall (AR) @ [IoU=0.50:0.95	area= small	maxDets=100]	= 0.139
Average Recall (AR) @ [IoU=0.50:0.95	area=medium	maxDets=100]	= 0.352
Average Recall (AR) @ [IoU=0.50:0.95	area= large	maxDets=100]	= 0.556

Tabell 5: Evaluation results for bbox

AP	AP50	AP75	APs	APm	APl
28.494	66.262	20.016	6.654	21.663	40.239

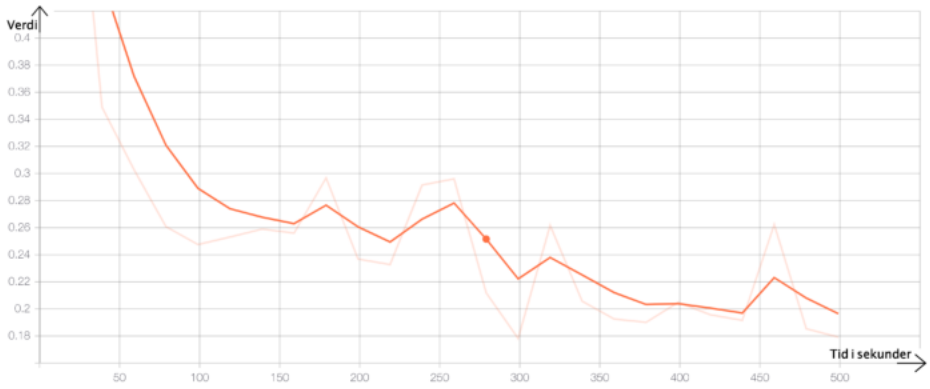
Tabell 6: Per-category bbox AP

category	AP	category	AP
atlantic_cod	23.077	saithe	33.910

og er egentlig laget for Google sin deep learning-rammeverk som heter Tensorflow, men PyTorch støtter også TensorBoard (**Oshri 2019**). Resultatene lastes opp med kommandoen `tensorboard dev upload --logdir logs --name 'TMAT3004' --description 'Bachelorprosjekt NTNU 2020'`.

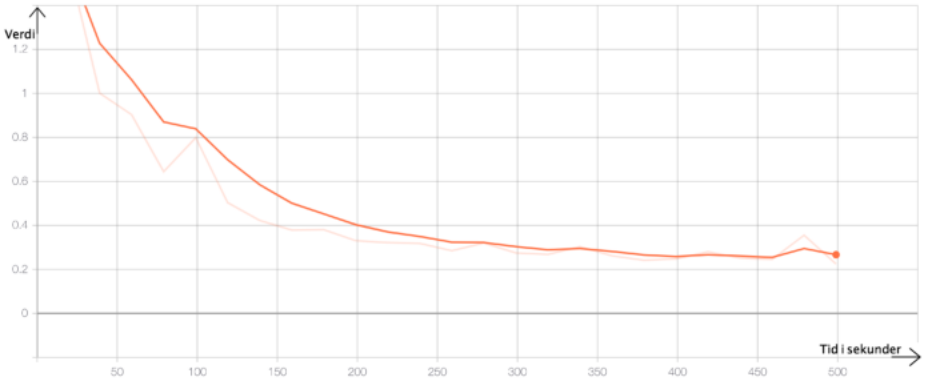
I alle grafene fra TensorBoard, figur 19, 20, 21 og 22, så er x-aksen tid i sekunder fra begynnelsen av forsøket, og y-aksen verdien til den spesifikke målingen.

Den første grafen nedenfor er «loss bounding box». Dette er et mål på hvor tett modellen satt bounding box-ene rundt objektene som fantes i bildene den gjorde inferens på.



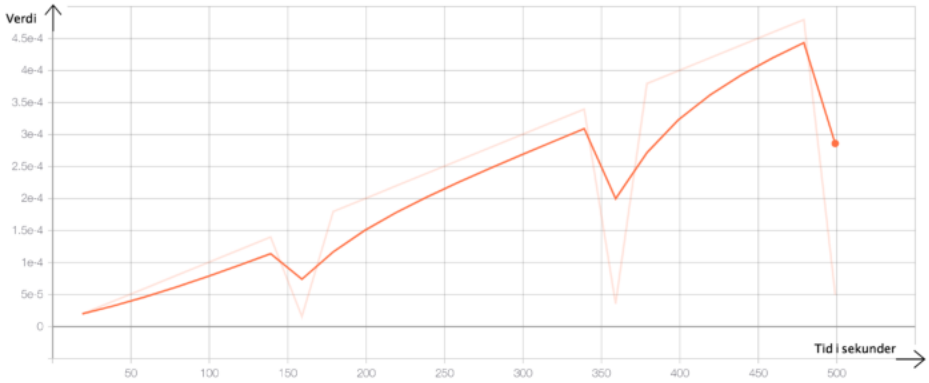
Figur 19: «*Loss bounding box*» *for RetinaNet.*

Neste graf viser «loss cls», eller «entropy loss». Entropy loss er et mål på hvor korrekt klassifiseringen av hver «bounding box» var. Hver «bounding box» kan inneholde en klasse, altså et objekt, eller være bakgrunn, og dermed være en boks som ikke inneholder en klasse.



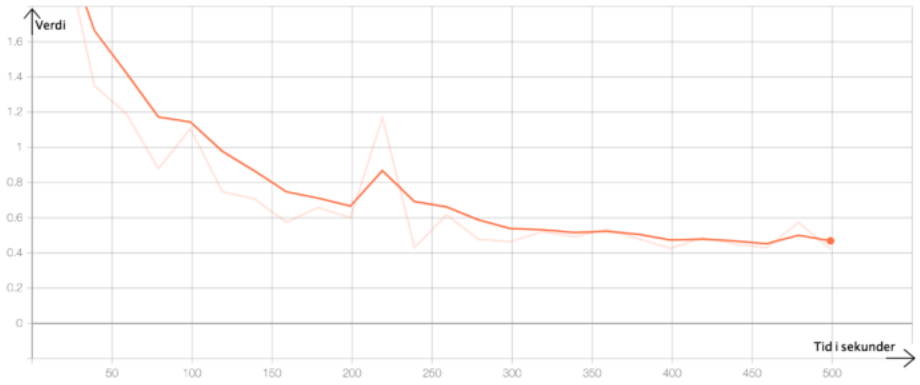
Figur 20: *≪Loss cls≫, også kjent som ≪entropy loss≫, for RetinaNet-modellen.*

Neste graf viser utviklingen av learning rate. Learning rate går stadig nedover ettersom modellen trenes, dette gjøres med en learning rate scheduler. Dette gjøres i et forsøk på å få ned total loss.



Figur 21: *Utviklingen av learning rate for RetinaNet-modellen.*

Den neste grafen viser utviklingen av total loss.



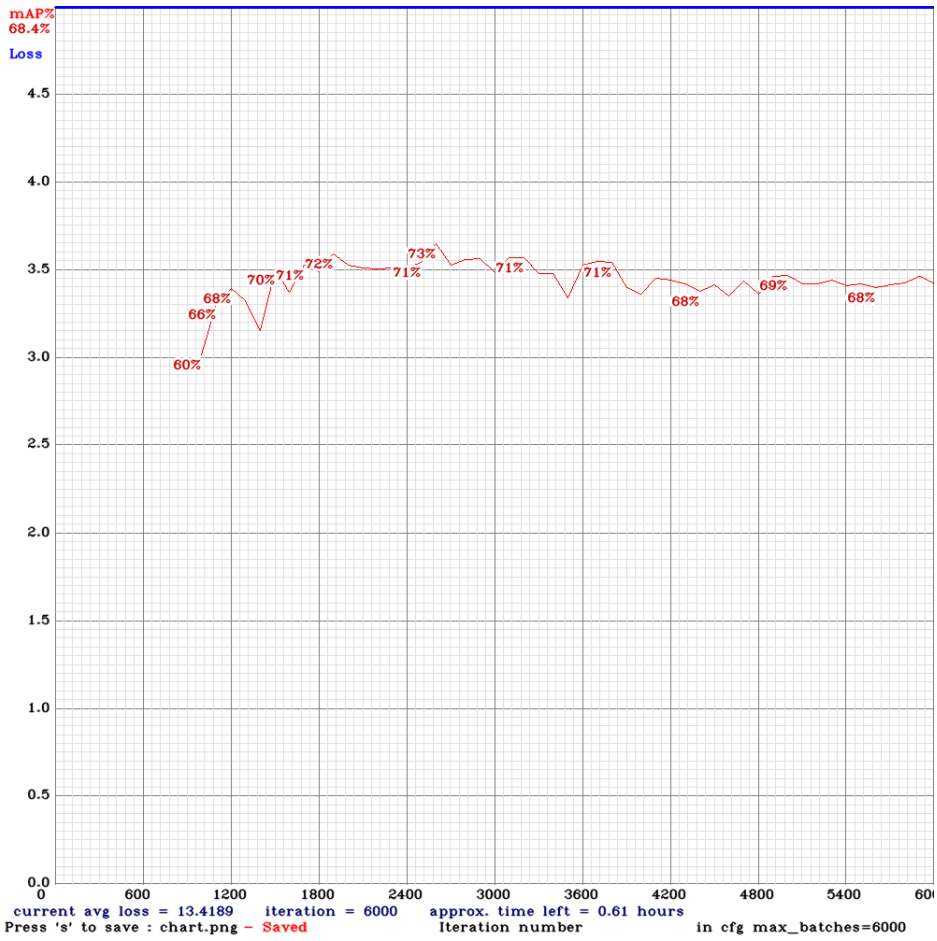
Figur 22: *Total loss for RetinaNet-eksperimentet.*

RetinaNet-resultatene for dette eksperimentet er også tilgjengelige på nettet.[8]

4.2 YOLO

Når YOLOv4-modellen ble testet på datasettet så hadde modellen en mAP på 73 %. Se figur 23. Total loss gikk fra 5533,6 ned til litt over 5,5.

[8]https://tensorboard.dev/experiment/2PxmdvA5QdavQC0YLABdgw



Figur 23: *graf YOLO.*

5 Diskusjon

Datasettet burde bestå av data uten bias. Modellene som er trent kan effektivt se forskjell på torsk og sei som finnes i bildene, men ettersom bildene ble tatt på så forskjellige måter, der sei dataen var i gråtoner og torsk var i farger, så vil modellen muligens se etter fisk av gråtoner, ikke sei, når den gjør inferens. Datasettet bestod av 208 bilder av torsk, med til sammen 4582 instanser av torsk i bildene. Det var 604 bilder av sei med 5525 instanser av fisken i bildene. Det er litt flere instanser av sei enn av torsk. Det kan også øke bias i modellen i noe grad.

Det som er lettere for mennesker å forstå er også lettere for maskiner å forstå. Data is king innenfor maskinsyn, å ha gode bilder er et must for å få gode resultater. Det betyr ikke at et bilde av gråtoner, eller av lav oppløsning er dårlige. Det er faktisk mange fordeler med gråtoner og lav oppløsning. De kan prosesseres raskt, og av og til så blir de viktigste egenskapene mer synlige.

Når en modell har blitt trent ferdig bør den forbedres. Den mest effektive måten å forbedre en deep learning-modell er å endre learning rate i konfigurasjonen, og så trene modellen på nytt. Det kan også hjelpe å bruke en ny modell, vanligvis så er en modell med flere konvolveringslag bedre. Dype nevrale nettverk blir stadig dypere ettersom forskere utvikler nye modeller (**Szegedy m.fl. s. 1**). Det er også mulig å trene modellen over flere epoker eller iterasjoner.

RetinaNet-modellen hadde overraskende lav Average Precision og Average Recall og kan sannsynligvis forbedres ved å trenes på nytt, for eksempel med et større antall iterasjoner eller ved å justere learning-rate. Det er alltid lurt å trene modeller flere ganger med forskjellige parametere.

YOLOv4-modellen hadde ganske gode resultater. Total loss burde egentlig være mellom 3,0 og 0,05, men endte opp på litt over 5,5. Modellen kan muligens forbedres ved å ved å justere learning rate.

Å eksperimentere med dataaugmentering kan også føre til bedre resultater for begge modellene. Bilder kan roteres tilfeldig med og mot klokken, lysstyrken i bildene kan bli tilfeldig endret, nye bilder kan skapes med zoom og bildene kan bli snudd horisontalt og vertikalt. Støy kan legges til bildene. Disse eksperimentene kan utføres ved å konfigurere maskinlæringsrammeverkene.

Fish4Knowledge-, LifeCLEF- og SeaCLEF-datasettene kan legges til modellen, da vil den kjenne igjen flere fiskearter.

OpenCV-programmet og YOLOv4-modellen kan testes videre, og trenes opp på data uten bias. Modellene bør testes på en videostrøm som inneholder både torsk og sei, og testes på data fra under oppdrettsanlegg for å se om den virker til den påtenkte arbeidsoppgaven den er laget for. Etter videre testing og eventuell trening på ny data så kan modellene og OpenCV-programmet bli brukt til å hjelpe oppdrettsnæringen til å ha bedre kontroll på fôringen ved å gi næringen en oversikt over mengden torsk og sei som trekker til oppdrettsanleggene.

6 Konklusjon

Maskinsyn kan anvendes til å hjelpe oppdrettsnæringen med å få kontroll på fôring av oppdrettfisk ved å gi industrien en tidsoversikt over mengden torsk og sei som trekker til anleggene.

Ved å tilpasse fôringen av oppdrettsfisken så vil fôrsprengt fisk bli et mindre problem. Kunnskapen som kan samles med automatisk videoanalyse kan hjelpe de marine næringene med å leve godt sammen langs kysten.

Maskinlæringsmodellene kan utvikles videre. En forbedret modell vil kunne gi oppdrettsnæringen en enda bedre tidsoversikt over torsk og sei under oppdrettsanleggene.

Et forbedret datasett, med flere fiskearter og mindre bias vil også forbedre resultatene. Automatisk analyse av undervannsvideoer kan gi forskere regelmessig informasjon om de marine økosystemene. Det er viktig å ha denne informasjonen når en gjør avgjørelser som kan påvirke miljøet og økonomien til de marine næringene.

Referanser

[1] Angell E., og Ekanger A. *Fekk fisk med magen full av oppdrettsfôr*. NRK, https://www.nrk.no/vestland/fekk-fisk-med-magen-full-av-oppdrettsfor-1.13677856, 2017. (hentet 24. April 2020)

[2] Batanina L. *Feature-request: State-of-art Yolo v4 Detector*. Github https://github.com/opencv/opencv/issues/17148, 2020. (hentet 6. Mai 2020)

[3] Bochkovskiy A. *YOLOv4 (v3/v2)*. Github https://github.com/AlexeyAB/darknet, 2020. (hentet 6. Mai 2020)

[4] Bochkovskiy A, Wang C.Y., og Liao H.Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv.org https://arxiv.org/abs/2004.10934, 2020. (hentet 8. Mai 2020)

[5] Buck I. *Stream Computing on Graphics Hardware*. Stanford University http://graphics.stanford.edu/~ianbuck/thesis.pdf, 2006. (hentet 6. Mai 2020)

[6] Bævre-Jensen M. *Kostnadene for norsk lakseoppdrett fortsetter å øke også i 2019*. Fiskeri- og havbruksnæringens forskningsfinansiering (FHF) https://www.fhf.no/nyheter/nyhetsarkiv/kostnadene-for-norsk-lakseoppdrett-fortsetter-aa-oeke-ogsaa-i-2019, 2019. (hentet 6. Mai 2020)

[7] Cadieu C. F., Hong H., Yamins D. L. K., Pinto N., Ardila D., Solomon E. A., Majaj N. J., og DiCarlo J. J. *Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition*. PLOS Computational Biology, 2014.

[8] Christensen T. B. *Fôrer villfisk med lusegift og medisiner*. Naturvernforbundet, https://naturvernforbundet.no/naturogmiljo/forer-villfisk-med-lusegift-og-medisiner-article39320-1024.html, 2019. (hentet 24. April 2020)

[9] Canziani A., Culurciello E., og Paszke A. *An Analysis of Deep Neural Network Models for Practical Applications*. Weldon School of Biomedical Engineering Purdue University, Faculty of Mathematics, Informatics and Mechanics University of Warsaw, 2017.

[10] Cui S., Zhou Y., Wang Y., og Zhai L. *Fish Detection Using Deep Learning*. Applied Computational Intelligence and Soft Computing, 2020.

[11] Falstad S. V. *Suksess med vaksinemaskin*. Trønder-avisa https://www.t-a.no/nyheter/2016/02/01/Suksess-med-vaksinemaskin-12104745.ece, 2016. (hentet 6. Mai 2020)

[12] Fenstad A. *Nå kan maskinen skille mellom makrell og sild*. tu.no https://www.tu.no/artikler/na-kan-maskinen-skille-mellom-makrell-og-sild/464208?utm_source=newsletter-tudaily&utm_medium=email&utm_campaign=newsletter-2019-05-06, 2019. (hentet 6. Mai 2020)

[13] Girshick R. *Fast R-CNN*. Microsoft Research, 2015.

[14] Girshick R., Donahue J., Darrell T., og Malik J. *Rich feature hierarchies for accurate object detection and semantic segmentation*. UC Berkeley, 2014.

[15] Goodfellow I., Bengio Y., og Courville A. *Deep Learning*. MIT Press www.deeplearningbook.org, 2016. (hentet 26. April 2020)

[16] He K., Zhang X., Ren S., og Sun J. *Deep Residual Learning for Image Recognition*. Microsoft Research, 2015.

[17] Hubel D. H., og Wiesel T. N. *Receptive Fields of Single Neurones in the Cat's Striate Cortex*. Wilmer Institute, The Johns Hopkins Hospital and University, Baltimore, Maryland, U.S.A., 1959.

[18] Hustad A., og Svalheim R. *Hvem bryr seg om pelletssei?*. Nofima https://www.fhf.no/nyheter/nyhetsarkiv/kostnadene-for-norsk-lakseoppdrett-fortsetter-aa-oeke-ogsaa-i-2019, 2019. (hentet 6. Mai 2020)

[19] Ioffe S., og Szegedy C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Google Inc, 2015.

[20] Jakobsen A. N. *Fikk torsk full av oppdrettsfôr: – Motbydelig fisk*. NRK, https://www.nrk.no/tromsogfinnmark/fikk-torsk-full-av-oppdrettsfor_-_-motbydelig-fisk-1.13662637, 2017. (hentet 24. April 2020)

[21] Jäger J., Rodner E., Denzler J., Wolff V., og Fricke-Neuderth K. *SeaCLEF 2016: Object proposal classification for fish detection in underwater videos*. Department of Electrical Engineering and Information Technology, Fulda University of Applied Sciences, Germany, Computer Vision Group, Friedrich Schiller University Jena, Germany, 2016.

[22] Jordan J. *An overview of object detection: one-stage methods*. Blogg https://www.jeremyjordan.me/object-detection-one-stage/, 2018.

[23] Kaller J. *Methods of Data Labeling in Machine Learning*. Blogg https://medium.com/ai%C2%B3-theory-practice-business/methods-of-data-labeling-in-machine-learning-80a34ece6c8b, 2019. (hentet 18. mai 2020)

[24] Karpathy A. *Convolutional Neural Networks*. Stanford University https://www.youtube.com/watch?v=bNb2fEVKeEo, 2014. (hentet 13. mai 2020)

[25] Krizhevsky A., Sutskever I., og Hinton G. E.. *ImageNet Classification with Deep Convolutional Neural Networks*. University of Toronto, 2012.

[26] LeCun Y., Bottou L., Bengio Y., og Haner P. *Gradient-Based Learning Applied to Document Recognition*. IEEE, 1998.

[27] Lin C.E. *Introduction to Motion Estimation with Optical Flow*. nanonets https://nanonets.com/blog/optical-flow/, 2019. (hentet 16. mai 2020)

[28] Lin T.Y., Maire M., Belongie S., Bourdev L., Girshick R., Hays J., Perona P., Ramanan D., Zitnick C. L., og Dollár P. *Microsoft COCO: Common Objects in Context*. Microsoft, 2015.

[29] Lin T.Y., Goyal P., Girshick R., He K., Dollár P. *Focal Loss for Dense Object Detection* Facebook AI Research, 2018.

[30] Lindberg S.K., og Evensen T.H. *Video av torsk og sei fra Havbruksstasjonen i Tromsø*. Nofima, 2020.

[31] Mallick S., Gupta V., Murzova A., Petrovicheva A., Serebryakov G., Semkin P., Chandra P., Belousov S., Khanova T., og Koriukina V. *Deep Learning with PyTorch*. OpenCV.org https://courses.opencv.org, 2020. (hentet 14. Mai 2020)

[32] MorrisC . *Learn Python*. kaggle.com https://www.kaggle.com/learn/python, 2020. (hentet 20. Mai 2020)

[33] NRK. *Laks med medisinrester har rømt fra Marine Harvest-anlegg*. NRK, https://www.nrk.no/trondelag/laks-har-romt-fra-marine-harvest-anlegg-i-naeroy-1.13902322, 2019. (hentet 25. April 2020)

[34] Oshri G. *Introducing TensorBoard.dev: a new way to share your ML experiment results*. Google Inc., https://blog.tensorflow.org/2019/12/introducing-tensorboarddev-new-way-to.html, 2019. (hentet 17. Mai 2020)

[35] Olsen J. E. *Ny oppmerksomhet rundt "pellets-fisken"*. fiskeribladet, https://fiskeribladet.no/nyheter/?artikkel=69867, 2019. (hentet 24. April 2020)

[36] Olsen O.C., Johansen J. I., og Grov B. *Villfisk mettes av oppdrettsmat: – Jeg blir rett og slett forbanna*. NRK, https://www.nrk.no/nordland/villfisk-mettes-av-oppdrettsmat_-_-jeg-blir-rett-og-slett-forbanna-1.14003547, 2018. (hentet 24. April 2020)

[37] OpenCV Team. *About OpenCV*. OpenCV https://opencv.org/about/, 2020. (hentet 5. Mai 2020)

[38] Redmon J. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/, 2013-2016.

[39] Redmon J. *YOLOv3: An Incremental Improvement*. arXiv, 2018.

[40] Ren S., He K., Girshick R., og Sun J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. University of Science and Technology of China, Microsoft Research, 2016.

[41] Robertsen R. *Sameksistens av marine næringer i kystsonen*. Nofima, https://nofima.no/prosjekt/sameksistens, 2020.

[42] Salman A, Siddiqui S. A. , Shafait F., Mian A., Shortis M. R., Khurshid K., Ulges A., og Schwanecke U. *Automatic fish detection in underwater videos*

by a deep neural network-based hybrid motion learning system. ICES Journal of Marine Science, 2019.

[43] Sculley D., Holt G., Golovin D., Davydov E., Phillips T., Ebner D., Chaudhary V., Young M., Crespo J.F., og Dennison D. *Hidden Technical Debt in Machine Learning Systems*. Google Inc., 2015.

[44] Shotton J., Johnson M., og Cipolla R. *Semantic Texton Forests for Image Categorization and Segmentation*. Toshiba Corporate R&D Center Kawasaki, Japan, Department of Engineering University of Cambridge, UK, 2008.

[45] Siddiqui S. A., Shafait F., Mian A. S., og Shortis M. R. *Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data*. ICES Journal of Marine Science, 2017.

[46] Simonyan K., og Zisserman A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015.

[47] Spruill V. *Environmental Impacts of Open-Ocean Aquaculture*. Ocean Conservancy https://issuu.com/oceanconservancy/docs/oc_rfts_v11_single, 2011. (hentet 6. Mai 2020)

[48] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., og Rabinovich A. *Going deeper with convolutions*. Google Inc., University of Michigan, University of North Carolina, Chapel Hill, arXiv, 2014.

[49] Sæther B.S. *Effekter av oppdrett på vill marin fisk*. Nofima, https://nofima.no/nyhet/2017/09/effekter-av-oppdrett-pa-vill-marin-fisk/, 2017. (hentet 24. April 2020)

[50] The Apache Software Foundation *Apache License, Version 2.0*. The Apache Software Foundation https://www.apache.org/licenses/LICENSE-2.0, 2004. (hentet 5. Mai 2020)

[51] Trana K., Klausen D. H., og Bolstad J. *Denne torsken er full av fôr fra oppdrettsnæringa*. NRK, https://www.nrk.no/trondelag/fiskere-har-solgt-torsk-som-er-full-av-lakselusmiddel-_-vet-ikke-om-det-er-farlig-1.14742706, 2019. (hentet 24. April 2020)

[52] Wada K. *labelme: Image Polygonal Annotation with Python*. Github https://github.com/wkentaro/labelme, 2016. (hentet 6. Mai 2020)

[53] Wu Y., Kirillov A., Massa F., Lo W.Y., og Girshick R. *Detectron2*. Github https://github.com/facebookresearch/detectron2, 2020. (hentet 6. Mai 2020)

[54] Xu W., og Matzner S. *Underwater Fish Detection using Deep Learning for Water Power Applications*. Pacific Northwest National Laboratory, 2018.

Kildekode

train.py
Denne koden er til å trene en RetinaNet-modell med Detectron2

```
import torch
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import cv2
import random
import os
import matplotlib.pyplot as plt

# model_zoo has a lots of pre-trained model
from detectron2 import model_zoo


# DefaultTrainer is a class for training object detector
from detectron2.engine import DefaultTrainer
# DefaultPredictor is class for inference
from detectron2.engine import DefaultPredictor

# detectron2 has its configuration format
from detectron2.config import get_cfg
# detectron2 has implemented Visualizer of object detection
from detectron2.utils.visualizer import Visualizer

# from DatasetCatalog, detectron2 gets dataset and from MetadatCatalog it
# gets metadata of the dataset
from detectron2.data import DatasetCatalog, MetadataCatalog

# BoxMode support bounding boxes in different format
from detectron2.structures import BoxMode

# COCOEvaluator based on COCO evaluation metric, inference_on_dataset is used for
# evaluation for a given metric
from detectron2.evaluation import COCOEvaluator, inference_on_dataset

# build_detection_test_loader, used to create test loader for evaluation
from detectron2.data import build_detection_test_loader

data_root = 'data'
```

```
train_txt = 'fish_train.txt'
test_txt  = 'fish_test.txt'

train_data_name = 'fish_train'
test_data_name  = 'fish_test'

thing_classes = ['atlantic_cod', 'saithe']

output_dir = 'outputs'

def count_lines(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

train_img_count = count_lines(os.path.join(data_root, train_txt))

# register train data
DatasetCatalog.register(name=train_data_name,
                        func=lambda: get_fish_dicts(data_root, train_txt))
train_metadata = MetadataCatalog.get(train_data_name).set(thing_classes=thing_classes)

# register test data
DatasetCatalog.register(name=test_data_name,
                        func=lambda: get_fish_dicts(data_root, test_txt))
test_metadata = MetadataCatalog.get(test_data_name).set(thing_classes=thing_classes)

# default configuration
cfg = get_cfg()

# update configuration with RetinaNet configuration
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5

# We have registered the train and test data set with name fish_train and fish_test.
# Let's replace the detectron2 default train dataset with our train dataset.
cfg.DATASETS.TRAIN = (train_data_name,)

# No metric implemented for the test dataset, we will have to update
# cfg.DATASET.TEST with empty tuple
cfg.DATASETS.TEST = ()

# data loader configuration
cfg.DATALOADER.NUM_WORKERS = 4
```

```
# Update model URL in detectron2 config file
cfg.MODEL.WEIGHTS=model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_3x.yaml")
# batch size
cfg.SOLVER.IMS_PER_BATCH = 4

# choose a good learning rate
cfg.SOLVER.BASE_LR = 0.001

# We need to specify the number of iteration for training in
# detectron2, not the number of epochs.
# lets convert number of epoch to number or iteration (max iteration)

epoch = 1000
max_iter = int(epoch * train_img_count / cfg.SOLVER.IMS_PER_BATCH)
max_iter = 500

cfg.SOLVER.MAX_ITER = max_iter

# number of output class
cfg.MODEL.RETINANET.NUM_CLASSES = len(thing_classes)

# update create ouptput directory
cfg.OUTPUT_DIR = output_dir
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
# Create a trainer instance with the configuration.
trainer = DefaultTrainer(cfg)

# if rseume=False, because we don't have trained model yet. It will
# download model from model url and load it
trainer.resume_or_load(resume=False)

# start training
trainer.train()
```

inference.py
Denne koden er til å lage en video ved å gjøre inferens med en RetinaNet-modell

```
def video_read_write(video_path):
    """
    Read video frames one-by-one, flip it, and write in the other video.
    video_path (str): path/to/video
    """
    video = cv2.VideoCapture(video_path)

    # Check if camera opened successfully
    if not video.isOpened():
        print("Error opening video file")
        return

    # create video writer
    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frames_per_second = video.get(cv2.CAP_PROP_FPS)
    num_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))

    isStreamOpen = False
    while video.isOpened():
        ret, frame = video.read()

        if ret:
            outputs = predictor(frame)

            v = Visualizer(frame[:, :, ::-1],
                           metadata=test_metadata,
                           scale=0.8
            )

            instances = outputs["instances"].to("cpu")

            v = v.draw_instance_predictions(instances)

            plt.imsave('outputs/frame_intermediate.png', v.get_image())

            if isStreamOpen == False:
                img = cv2.imread('outputs/frame_intermediate.png')
                height, width, layers = img.shape
                size = (width,height)
                out = cv2.VideoWriter('out.avi',cv2.VideoWriter_fourcc(*'DIVX'),
                                      frames_per_second, size)
                isStreamOpen = True
```

```
            img = cv2.imread('outputs/frame_intermediate.png')

            cv2.putText(img, 'num instances: ' + str(len(instances)), (5,100),
                        cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 2, cv2.LINE_AA)
            out.write(img)

            print ('num instances: ' + str(len(instances)))
        else:
            break

    out.release()
    video.release()

    return

cfg.DATASETS.TEST = (test_data_name,)

# create a predictor instance with the configuration (it has our fine-tuned model)
# this predictor does prdiction on a single image
predictor = DefaultPredictor(cfg)

# create directory for evaluation
eval_dir = os.path.join(cfg.OUTPUT_DIR, 'coco_eval')
os.makedirs(eval_dir, exist_ok=True)

# create evaluator instance with coco evaluator
evaluator = COCOEvaluator(dataset_name=test_data_name,
                          cfg=cfg,
                          distributed=False,
                          output_dir=eval_dir)

# create validation data loader
val_loader = build_detection_test_loader(cfg, test_data_name)

# start validation
inference_on_dataset(trainer.model, val_loader, evaluator)

# Run inference on video
video_read_write('in.mp4')
```

main.cpp
Denne koden er programmet som bruker en YOLOv4 modell til å telle torsk og sei, og lage en t

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/dnn.hpp>
//#include <opencv2/tracking.hpp>
#include <fstream>
//#include "matplotlibcpp.h"
#include <chrono>
#include <ctime>

using namespace std;
//using namespace matplotlibcpp;
using namespace cv::dnn;
using namespace cv;

// globals
float objectnessThreshold; // Objectness threshold
float confThreshold; // Confidence threshold
float nmsThreshold;  // Non-maximum suppression threshold
int inpWidth;  // Width of network's input image
int inpHeight; // Height of network's input image
vector<string> classes;

vector<Rect2d> bboxes;
int codQuantity, saitheQuantity;

string objectLabel;

const char *WINDOW_TITLE = "Press ESC to quit";

// Get the names of the output layers
auto getOutputsNames(const Net& net)
{
    static vector<String> names;
    if (names.empty())
    {
        // Get the indices of the output layers, i.e. the layers with unconnected outputs
        vector<int> outLayers = net.getUnconnectedOutLayers();

        // Get the names of all the layers in the network
        vector<String> layersNames = net.getLayerNames();

        // Get the names of the output layers in names
        names.resize(outLayers.size());
        for (size_t i = 0; i < outLayers.size(); ++i)
            names[i] = layersNames[outLayers[i] - 1];
```

```
    }
    return names;
}

// Draw the predicted bounding box
void drawPred(
        int classId,
        float conf,
        int left,
        int top,
        int right,
        int bottom,
        Mat& frame)
{
    // Draw a rectangle displaying the bounding box
    rectangle(frame, Point(left, top), Point(right, bottom), Scalar(255, 178, 50), 3);

    // Get the label for the class name and its confidence
    string label = format("%.1f", conf * 100);
    if (!classes.empty())
    {
        CV_Assert(classId < (int)classes.size());
        label = classes[classId] + " " + label + "\%";
    }

    // Display the label at the top of the bounding box
    int baseLine;
    Size labelSize = getTextSize(label, FONT_HERSHEY_SIMPLEX, 0.5, 1, &baseLine);
    top = max(top, labelSize.height);
    rectangle(frame,
                        Point(left, top - round(1.5*labelSize.height)),
                    Point(left + round(1.5*labelSize.width),
                    top + baseLine),
                    Scalar(255, 255, 255),
                    FILLED);
    putText(frame, label, Point(left, top), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0,0,0),1);

    objectLabel = label;
}

// Remove the bounding boxes with low confidence using non-maxima suppression
void postprocess(Mat& frame, const vector<Mat>& outs)
{
    vector<int> classIds;
    vector<float> confidences;
    vector<Rect> boxes;
```

```
codQuantity = 0;
saitheQuantity = 0;

for (size_t i = 0; i < outs.size(); ++i)
{
    // Scan through all the bounding boxes output from the network and keep only the
    // ones with high confidence scores. Assign the box's class label as the class
    // with the highest score for the box.
    float* data = (float*)outs[i].data;
    for (int j = 0; j < outs[i].rows; ++j, data += outs[i].cols)
    {
        Mat scores = outs[i].row(j).colRange(5, outs[i].cols);
        Point classIdPoint;
        double confidence;
        // Get the value and location of the maximum score
        minMaxLoc(scores, 0, &confidence, 0, &classIdPoint);

        if (confidence > confThreshold)
        {
            int centerX = (int)(data[0] * frame.cols);
            int centerY = (int)(data[1] * frame.rows);
            int width = (int)(data[2] * frame.cols);
            int height = (int)(data[3] * frame.rows);
            int left = centerX - width / 2;
            int top = centerY - height / 2;

            classIds.push_back(classIdPoint.x);
            confidences.push_back((float)confidence);
            boxes.push_back(Rect(left, top, width, height));
        }
    }
}

// Perform non maximum suppression to eliminate redundant overlapping boxes with
// lower confidences
vector<int> indices;
NMSBoxes(boxes, confidences, confThreshold, nmsThreshold, indices);
for (size_t i = 0; i < indices.size(); ++i)
{
    int idx = indices[i];
    Rect box = boxes[idx];
    drawPred(classIds[idx], confidences[idx], box.x, box.y,
             box.x + box.width, box.y + box.height, frame);

             // Count fish
             if (classes[classIds[idx]] == "atlantic cod") {
```

```
                codQuantity++;
            }
            else if (classes[classIds[idx]] == "saithe")
            {
                saitheQuantity++;
            }
    }
}

int main(int argumentQuantity, char *arguments[])
{
    // Configuration for log file
    string filename = "log.csv";
    bool newFile = true;

    // Check if log file exists
    ifstream ifile(filename);
    if (ifile)
    {
        newFile = false;
    }

    // Open log file
    ofstream logFile;
    logFile.open(filename, std::ios_base::app);

    // Write what you will find in the log file on the first line
    // if it does not already exist
    if (newFile)
    {
        logFile << "atlantic_cod_quantity,saithe_quantity,datetime" << endl;
    }

    // Open video file
    VideoCapture video;

    // Print usage as a warning
    clog << "Usage: ./OBJECT_DETECTOR <videoPath>" << endl;

    // Use webcam or filepath from command line
    if (argumentQuantity > 1)
    {
        string videoPath = arguments[1];
        video = VideoCapture(videoPath);
    }
    else
```

```
{
    clog << "No video path given. Using camera 0" << endl;
    video = VideoCapture(0);
}

if (video.isOpened() == false)
{
    cerr << "Failed to open video stream" << endl;
    return -1;
}

// Frame matrix
Mat frame;

// Initialize the parameters
int inpWidth = 416;  // Width of network's input image
int inpHeight = 416; // Height of network's input image

// Give the configuration and weight files for the model
String modelConfiguration = "data/models/yolov-obj.cfg";
String modelWeights = "data/models/yolo.weights";

// Check if model exists in data folder
ifile = ifstream(modelConfiguration);
if (!ifile)
{
    cerr << "YOLOv4 model could not be found." << endl;
    return -1;
}

// Load the network
Net net = readNetFromDarknet(modelConfiguration, modelWeights);

// Load names of classes
string classesFile = "data/models/obj.names";
ifstream ifs(classesFile.c_str());
string line;
while (getline(ifs, line)) classes.push_back(line);

// Configure user interface
namedWindow(WINDOW_TITLE, WINDOW_AUTOSIZE);

// Main loop
bool isAlive = true;
const int ESCAPE_KEYCODE = 27;
const int DELAY_MILLISECONDS = 1;
```

```
int key;

while(isAlive)
{
    key = waitKey(DELAY_MILLISECONDS);

    if (key == ESCAPE_KEYCODE)
    {
        isAlive = false;
        break;
    }

    double timer = (double)getTickCount();

    video >> frame;

    if (frame.empty())
    {
        isAlive = false;
        break;
    }

    // Create a 4D blob from a frame.
    Mat blob;
    blobFromImage(frame,
                  blob,
                  1/255.0,
                  Size(inpWidth, inpHeight),
                  Scalar(0,0,0),
                  true,
                  false);

    //Sets the input to the network
    net.setInput(blob);

    // Runs the forward pass to get output of the output layers
    vector<Mat> outs;
    net.forward(outs, getOutputsNames(net));

    // Remove the bounding boxes with low confidence
    postprocess(frame, outs);

    // Put efficiency information.
    // The function getPerfProfile returns the overall time for inference(t)
    // and the timings for each of the layers(in layersTimes)
```

```
        vector<double> layersTimes;
        double freq = getTickFrequency() / 1000;
        double t = net.getPerfProfile(layersTimes) / freq;
        string label = format("Inference time for a frame : %.2f ms", t);
        putText(frame, label, Point(0, 15), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255));

        float fps = getTickFrequency() / ((double)getTickCount() - timer);

        putText(frame,
                            "FPS: " + std::to_string(int(fps)),
                    Point(100,50),
                    FONT_HERSHEY_SIMPLEX, 0.75,
                    Scalar(50,170,50), 2);
        putText(frame,
                            "Atlantic cod quantity: " + std::to_string(codQuantity),
                    Point(100,100),
                    FONT_HERSHEY_SIMPLEX,
                    0.75,
                    Scalar(50,170,50),
                    2);
        putText(frame,
                            "Saithe quantity: " + std::to_string(saitheQuantity),
                    Point(100,130),
                    FONT_HERSHEY_SIMPLEX,
                    0.75,
                    Scalar(50,170,50),
                    2);

        imshow(WINDOW_TITLE, frame);

        // Get the time
        auto end = std::chrono::system_clock::now();
        std::time_t end_time = std::chrono::system_clock::to_time_t(end);

        // Remove the newline after time
        // which is typically included from the C library call
        char *time = ctime(&end_time);
        if (time[strlen(time)-1] == '\n') time[strlen(time)-1] = '\0';

        // Log cod and saithe quantities to csv file
        if (codQuantity > 0 || saitheQuantity > 0)
        {
        logFile << codQuantity << "," << saitheQuantity << ",\"" << time << "\"" << endl;
        cout << codQuantity << "," << saitheQuantity << ",\"" << time << "\"" << endl;
        }
    }
```

```
    logFile.close();

    video.release();

    destroyAllWindows();

    return 0;
}
```