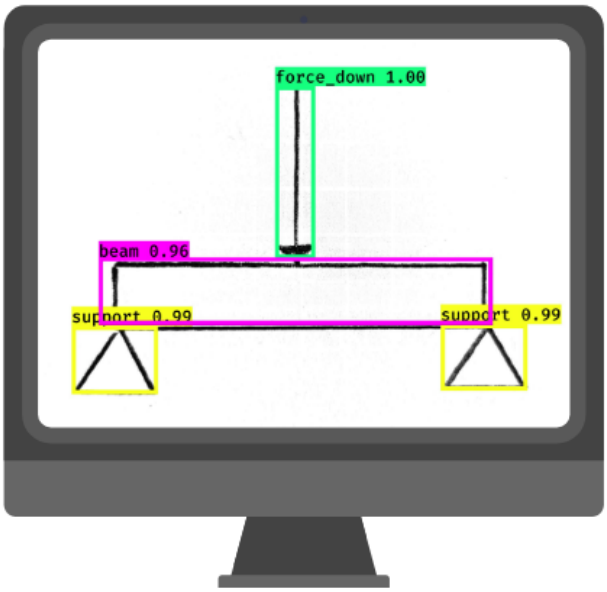
CHALMERS



Kan en dator lära sig hållfasthetslära?

Kandidatarbete i mekanik och maritima vetenskaper

Jonas Bohlin
Tobias Gabrielii
Jens Lundgren
Adam Oliv
Joakim Svensson
Benjamin Vinnerholt

INSTITUTIONEN FÖR
MEKANIK OCH MARITIMA VETENSKAPER

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

Kandidatarbete 2020:06

Kan en dator lära sig hållfasthetslära?

Kandidatarbete i mekanik och maritima vetenskaper

Jonas Bohlin
Tobias Gabrielii
Jens Lundgren
Adam Oliv
Joakim Svensson
Benjamin Vinnerholt

Institutionen för Mekanik och Maritima vetenskaper
Avdelning för Dynamik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2020

Kan en dator lära sig hållfasthetslära?
Objektdetektering tillämpad på hållfasthetslära
JONAS BOHLIN, TOBIAS GABRIELII, JENS LUNDGREN, ADAM OLIV, JOAKIM SVENSSON, BENJAMIN VINNERHOLT

Handledare: Jim Brouzoulis, Institutionen för mekanik och maritima vetenskaper
Examinator: Mikael Enelund, Institutionen för mekanik och maritima vetenskaper

Kandidatarbete 2020:06
Institutionen för mekanik och maritima vetenskaper
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telephone +46 31 772 1000

Omslag: Bild på en datorskärm som visar ett handritat balkproblem där komponenterna blivit detekterade.

Tryckeri /Institutionen för mekanik och maritima vetenskaper
Göteborg, Sverige 2020

Abstract

This report is written as part of a Bachelor's thesis at the institution for Mechanics and Maritime Sciences at Chalmers University of Technology during the spring of 2020.

When students first encounter solid mechanics it's not uncommon that their intuition fails them. Therefore a tool that visualises what occurs inside of, for example a beam, when different types of structural loads affect it could be helpful. Such a tool would also be useful in the construction industry since it could be used for calculating the thickness of structurally vital components of for example a house.

The purpose of this report is to examine the possibilities to implement machine learning and object detection to solve hand drawn solid mechanics problems. This was done by using the object detection algorithm YOLO to identify the components of the problem. In order to limit the project's scope, the output was reduced to the following quantities: beam deflection, normal force, shear force and bending moment. The calculations for these quantities were done using the finite element method. The final product is a computer program that, after some interaction with the user, can solve fundamental beam problems by computing the deflection, normal force, shear force and bending moment. In addition to diagrams of these quantities a computer generated model of the problem is provided. The largest obstacle that occured was to identify the different objects with high accuracy, despite many similarities in appearance. Because of this, some generalizations was made. For instance, the direction of moments was standardized and must therefore, in some cases, be adjusted by the user.

Sammanfattning

Denna rapport redogör för ett kandidatarbete vid institutionen för mekanik och maritima vetenskaper vid Chalmers tekniska högskola som utfördes våren 2020.

När studenter stöter på hållfasthetsproblem för första gången är det inte ovanligt att intuitionen sviker. Därför skulle det vara till stor hjälp om det fanns verktyg som enkelt kunde visualisera vad som sker till exempel inuti en balk när den utsätts för olika yttre belastningar. Ett sådant verktyg skulle inte enbart vara till hjälp för inlärningsprocessen utan skulle även gå att applicera i konstruktionsbranschen för att bland annat beräkna hur tjocka takbalkarna i ett hus behöver vara.

Denna rapport syftar till att undersöka möjligheterna till att implementera maskininlärning för att lära en dator att lösa handritade problem inom hållfasthetslära. För att läsa in problemet användes objektdetekteringsalgoritmen YOLO. Storheterna som programmet beräknar avgränsades till utböjning, normalkraft, snittmoment och tvärkraft. Beräkningen av utböjningen och de interna storheterna gjordes med hjälp av finita elementmetoden.

Slutprodukten är ett datorprogram som efter viss interaktion med användaren kan ta fram diagram över utböjningen, normalkraften, tvärkraften samt snittmomentet och därtill generera en datorritad bild av problemet. Den största svårigheten som stöttes på var att med stor noggrannhet lyckas identifiera alla enskilda objekt trots stora likheter i utseende. Detta ledde till att vissa generaliseringar fick göras, där bland annat riktningar på moment standardiseras och istället får justeras av användaren.

Förord

Rapporten är skriven som en del av kandidatarbetet MMSX20-20-15 under vårterminen 2020. Projektet är utfört av studenter inom Datateknik, Informationsteknik, Maskinteknik samt Teknisk fysik vid Chalmers Tekniska Högskola.

Vi vill rikta ett varmt tack till vår handledare Jim ”Jimbo” Brouzoulis som med sitt stöd och engagemang gett oss dem bästa möjliga förutsättningar att utföra projektet väl. Ett stort tack vill vi även rikta till vår examinator Mikael Enelund för stort intresse och bra feedback. Slutligen tackas Chalmers Centre for Computational Science and Engineering (C3SE) för möjligheten att använda deras datorkluster.

Göteborg, maj 2020
Jonas Bohlin, Tobias Gabrielii, Jens Lundgren, Adam Oliv, Joakim Svensson, Benjamin Vinnerholt

Terminologi

Datorkluster Ett datorskluster är en samling av noder bestående av CPUer och GPUer anpassade för att utföra tunga och krävande beräkningar, samt hantera stora mängder data.

Neuralt Nätverk Ett neuralt nätverk är en matematisk modell som kan användas för att klassificera data.

CNN Convolutional Neural Networks är en sorts djupt neuralt nätverk som innehåller ”convolutional layers” vilket är en typ av lager som är anpassat för att analysera bilder.

FEM Finita elementmetoden är en numerisk metod för att lösa partiella differentialekvationer inom exempelvis fysik.

YOLO YOLO-algoritmen är en objektdetekteringsalgoritm som används för att identifiera och lokalisera objekt i bilder.

Gömda lager Lager med noder som genomför beräkningar och skickar vidare sin utdata till ett annat lager med noder.

Epok Antal epoker som en modell tränas med bestämmer hur många gånger all träningsdata löps igenom.

Batch size Beskriver hur många bilder som utnyttjas vid varje iteration under träningen av en modell.

Kostnadsfunktion och kostnadsvärde Funktion som mäter hur bra en modell utfört en prediktion. Kostnadsfunktion ska minimeras när modellen tränas.

Python Ett objektorienterat programmeringsspråk som ofta används vid programmering inom maskininlärning.

Innehåll

1 Inledning

Djupinlärning är ett område inom maskininlärning där man genom att träna en algoritm lär en dator att lösa problem på egen hand. Metoder för djupinlärning och objektdetektering har utvecklats avsevärt de senaste åren vilket har öppnat upp möjligheter inom många nya områden. Exempelvis tillämpas objektdetektering i form av ansiktsigenkänning på offentliga platser i Kina för att bevaka och kartlägga befolkningens och specifika individers beteende [1] [2]. Ett annat tydligt exempel på ett användningsområde för objektdetektering är självkörande bilar som styrs med hjälp av djupinlärning [3].

Objektdetektering används även flitigt i sociala medier där bland annat Snapchat och Instagram erbjuder filter som kan ändra en persons utseende. Ytterligare ett exempel på dess användning är att de flesta ägare av smartphones kan låsa upp sin telefon genom att hålla upp den framför sitt ansikte.

Objektdetektering implementeras dessutom i appar som PhotoMath vilken löser matematiska ekvationer utifrån en hand- eller datorritad bild [4]. Appen ger både svaret på ekvationen och en lösningsgång.

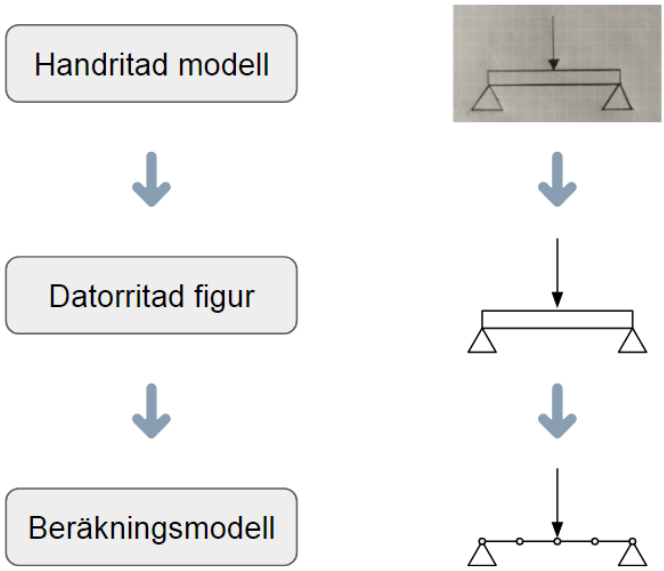
I dagsläget finns ingen motsvarighet till PhotoMath för ingenjörsmässiga ämnen som till exempel mekanik eller hållfasthetslära. I dessa ämnen är det ofta till stor hjälp att kunna visualisera de inblandade storheterna, då det intuitiva svaret inte alltid stämmer överens med det verkliga. Skulle det vara möjligt att med hjälp av objektdetektering utveckla ett program som visualiserar och löser problem av dessa slag? Detta projekt syftar till att utforska möjligheten till att applicera objektdetektering på problem inom hållfasthetslära och visualisera lösningarna till dem.

1.1 Syfte

Projektets syfte är att undersöka om datorbaserad objektdetektering kan användas för att identifiera handritade mekaniska strukturer, och överföra dessa till en numerisk modell av strukturerna. Vidare skall den numeriska modellen lösas automatiskt och leverera utdata enligt användarens önskemål.

1.2 Problembeskrivning

Uppgiften är att få en dator att lösa ett hållfasthetsproblem utifrån en handritad bild av den mekaniska strukturen inklusive laster och stöd. Denna uppgift går att dela upp i mindre uppgifter för att göra arbetsprocessen mer konkret, se Figur 1.



Figur 1: Figuren visar hur projektet är uppdelat i tre olika delar och i vilka steg arbetet kommer att genomföras.

Första deluppgiften är att träna en algoritm vars syfte är att känna igen standardiserade hållfasthetssymboler och balkproblem. Andra delen av uppgiften är att generera en datorritad bild av den handritade originalbilden. Därefter ska den datorritade bilden översättas till en numerisk modell. Slutligen ska relevanta storheter beräknas och visualiseras för användaren.

1.3 Intressenter

De intressenter som detta projekt berör är framförallt studenter, men även lärare och yrkesverksamma ingenjörer. Eftersom projektet syftar till att skapa ett program som löser grundläggande hållfasthetsproblem kan detta bli ett hjälpmedel för studenter i sitt lärande. Programmet kan även användas som ett visuellt verktyg som stöd till föreläsningar eller andra undervisningsformer och sammanhang.

Projektet bidrar även till forskning inom objektdetektering genom att applicera det på ett område som inte har gjorts tidigare. På så vis kan projektet ligga till grund för att ta över de grundläggande beräkningarna för till exempel arkitekter när de designar byggnader.

1.4 Avgränsningar

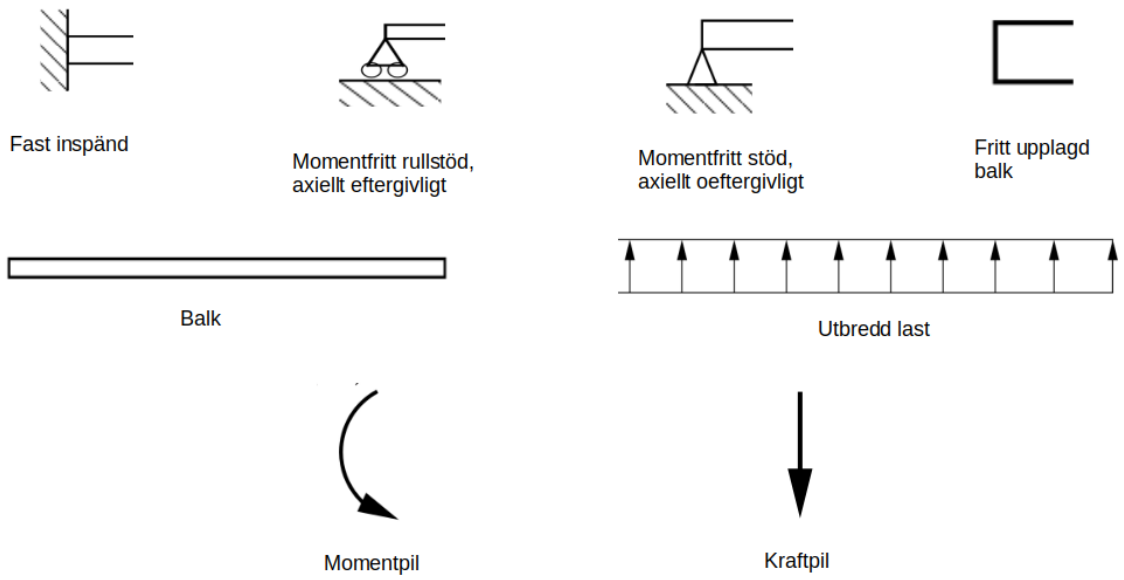
Detta projekt berör ett problem som är väldigt brett och därför var det viktigt med tydliga avgränsningar för att hålla projektet inom de givna tidsramarna.

Den tydligaste avgränsningen i detta projekt var att begränsa projektet till att enbart beröra balkproblem.

Objektdetektering avgränsades till att konstrueras utifrån existerande algoritmer och paket istället för att bygga en modell på egen hand. Algoritmen You Only Look Once

(YOLO) [5] användes och för att implementera YOLO utnyttjades det existerande projektet *TrainYourOwnYOLO* [6]. Detta projekt använder sig av de väletablerade paketen TensorFlow [7] och Keras [8] för maskininlärning i Python, vilket är ett objektorienterat programmeringsspråk [9]. Eftersom *TrainYourOwnYOLO* samt tillhörande paket är skrivna i Python skrevs hela programmet i Python.

För att ytterligare avgränsa omfattningen av projektet tränades modellen endast på de figurer, infästningar och laster som är vanligast förekommande inom strukturmekanik, vilket är den gren inom hållfasthetslära som behandlar balkstrukturen, se Figur 2.



Figur 2: Ett urval av standardiserade hållfasthetssymboler som är avgränsande för projektet.

Möjligheten att skriva ut magnitud på laster och moment i de handritade problemen uteslöts också. Anledningen till denna avgränsning var att modellen hade behövt koppla rätt detekterad siffra till rätt detekterad hållfasthetssymbol, vilket ansågs problematiskt och utanför projektets ramar. Istället ges möjlighet för användaren att ange magnitud på laster och moment innan beräkningsdelen av projektet.

Beräkningsmodellen begränsades till att endast använda finita elementmetoden (FEM) för att lösa balkproblemen, då det är ett standardverktyg för numeriska lösningar av hållfasthetsproblem.

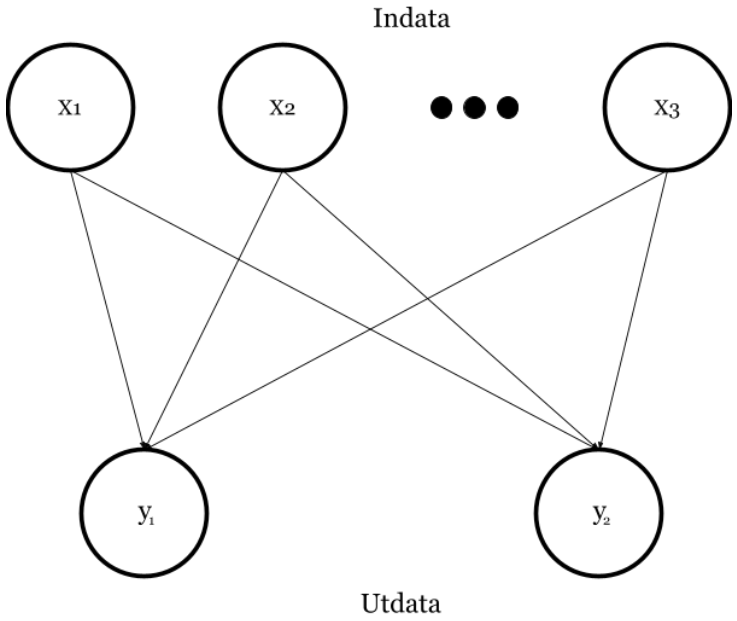
2 Teori

Följande kapitel kommer att behandla den teori som krävs för att genomföra projektet, det vill säga den grundläggande teorin inom maskin- och djupinlärning samt en kortfattad beskrivning av FEM.

2.1 Maskininlärning

Ett neuralt nätverk (NN) är inspirerat av hur hjärnor behandlar information genom neuroner och synapser [10]. Neuronerna i hjärnan är kopplade till varandra genom synapser och dessa påverkar om neuronerna aktiveras eller inte. Beroende på vilka neuroner som aktiveras fattar hjärnan olika beslut.

För att efterlikna detta kan artificiella neuroner och synapser programmeras. Synapserna representeras av viktade kopplingar mellan neuronerna och neuronerna representeras av en aktiveringsfunktion. De enklaste neurala nätverken, se Figur 3, består av två lager. I det första lagret består neuronerna av indatan och i det andra består neuronerna av utdatan.



Figur 3: Ett neuralt nätverk med endast indata- och utdatalager.

Maskininlärningsalgoritmerna som beskrivs senare i detta avsnitt är alla så kallade övervakade djupinlärningsalgoritmer, vilket betyder att ett neuralt nätverk tränas på indata-utdata-par. Att nätverket tränas på indata-utdata-par innebär att nätverket för en given indata vet vad utdatan ska vara och jämför sitt resultat med den inmatade ”sanningen” från paret.

En artificiell neuron, även kallad nod, se Figur 4, fungerar så att den tar indata i form av en vektor $\mathbf{x}$, där varje vektorkomponent x_i är utdatan från det tidigare lagrets i:te nod. Noden har även en vektor $\mathbf{w}$, där varje vektorkomponent w_i har till syfte att vikta

motsvarande indatakomponent x_i. Genom att vikta indatan på olika sätt förändras utdatan, vilket innebär att de olika komponenterna av indatan påverkar utdatan olika mycket.

Efter att indatan blivit viktad summeras den och det så kallade *aktiveringsvärdet*, b, adderas

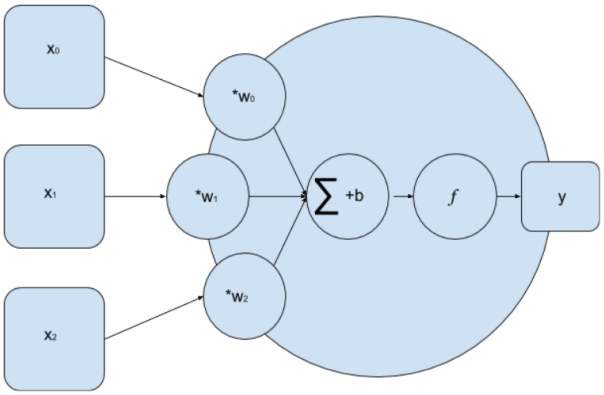
$$z = \mathbf{w}^\mathsf{T}\mathbf{x} + b \text{ där } b \in R.$$

Syftet med aktiveringsvärdet är att justera hur mycket varje individuell nod påverkar utdatan. För att en nod ska aktiveras måste z överstiga ett tröskelvärde som regleras av värdet på b. Slutligen appliceras en aktiveringsfunktion, f, på z. Aktiveringsfunktionens utdata blir neuronens utdata y

$$y(\mathbf{x}) = f(z).$$

Utdatan från noden skickas därefter vidare till nästa lager. När datan når det sista lagret samlas nodernas utdata i en vektor $\mathbf{y}$.

Parametrarna som går att justera i noden blir alltså viktvektorns alla komponenter och aktiveringsvärdet. När nätverket tränas är det dessa parametrar som justeras för att ge nätverket det önskade beteendet.

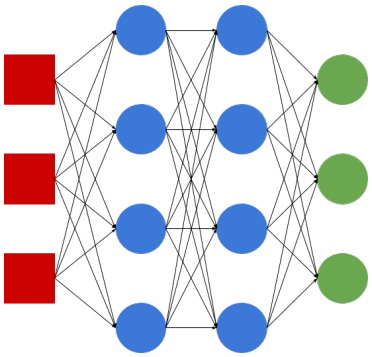


Figur 4: Figuren visar hur indatan går igenom en nod med utdatan y. Här är x_i nodens indata som multipliceras med w_i, vilket är nodens vikt för motsvarande indata. Den skalade indatan summeras sedan och ett aktiveringsvärde adderas och skickas till en aktiveringsfunktion f. Efter aktiveringsfunktionen fås utdatan y.

I Figur 5 visas en multilagersperceptron, vilket är ett nätverk som ses som grunden till modern objektdetektering [10]. Indatan består av värdet för alla pixlar i en bild. Pixelvärde kan referera till både gråskalevärde och RGB-värde. Efter att indatan lästs in propagerar den igenom ett antal ”gömda lager”. Nätverkets utdata består av en vektor $\mathbf{y}$ där varje element representerar sannolikheten att bilden tillhör en viss klass. Som tidigare nämnt i avsnittet behöver ett nätverk tränas för att kunna känna igen mönster i indatan. Detta görs genom att minimera en kostnadsfunktion C. Ett vanligt exempel på en kostnadsfunktion är

$$C(\mathbf{y}) = \|\mathbf{y} - \mathbf{y}_{\text{korrekt}}\|^2 \text{ där } y_i = P(\text{klass}_i)$$

och $\mathbf{y}_{\text{korrekt}}$ är en vektor där elementen motsvarar vilken klass bilden tillhör och $P(\text{klass}_i)$ är sannolikheten att y_i tillhör klass_i.

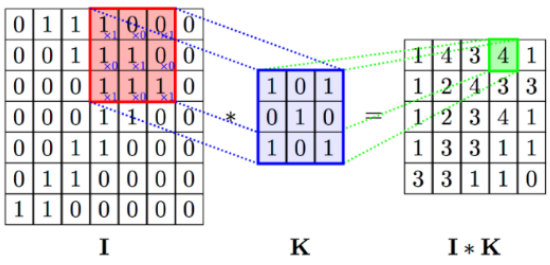


Figur 5: Figuren visar ett exempel på en multilagersperceptron med djup 2. Alla noder, det vill säga gröna och blåa cirklar i figuren, är noder av samma typ som noden i Figur 4 och de röda fyrkanterna representerar indatan.

2.2 Convolutional Neural Network

Olika neurala nätverk är anpassade för att hantera olika typer av data. Datan för en bild kan liknas vid en matris, där varje siffra i matrisen representerar ett pixelvärde. Datamängden för en bild blir därför mycket stor och ett vanligt NN, exempelvis ett sådant som är beskrivet i avsnitt 2.1, skulle kräva för mycket beräkningskraft för att vara användbara vid objektdetektering på högupplösta bilder i färg. Ett sätt att ta sig runt problemet med ökad beräkningsmängd är att använda sig av ett så kallat Convolutional Neural Network (CNN).

Ett CNN använder sig av olika filter som illustreras i Figur 6. Dessa filter är anpassade för att upptäcka specifika attribut i bilden, exempelvis kanter eller kurvor [10]. Ett filter är en mindre matris, till exempel 3×3. En attributskarta skapas genom att summera de elementvisa produkterna mellan filtret och regioner över bilden av samma storlek. I Figur 6 illustreras hur en attributskarta, längst till höger i figuren, skapas genom att summan av de elementvisa produkterna mellan filtret K och olika regioner av bilden samlas i en ny matris.



Figur 6: Ett filter, K, som appliceras på en matris, I, och den motsvarande attributskartan. Summan av de elementvisa produkterna mellan den röda regionen och filtret förs in i korresponderande ruta i attributskartan. Från [11].

Genom att applicera aktiveringsfunktioner på attributskartan fås en aktiveringskarta, vars syfte är att lyfta fram de tydliga attributen. För att ytterligare reducera behovet av beräkningskraft använder ett CNN sig av pooling layers som reducerar storleken hos attri-

butskartan genom att endast ha kvar de tydligaste attributen. Detta innebär i praktiken att aktiveringskartan delas in i regioner och endast det högsta värdet i varje region sparas.

De sista lagrena i ett CNN är fullt kopplade lager, det vill säga lager där varje nod i lagret är kopplad till alla noder i nästa lager. Syftet med de första lagrena är alltså att förbereda datan så att den beskriver bildens attribut, för att de fullt kopplade lagrena sedan ska kunna klassificera bilden genom att hitta kopplingar mellan klasserna och olika attribut.

2.3 Modeller för objektdetektering

Ett standard CNN uppskattar endast vilken klass bilden hör till, detta innebär att en bild endast kan tillhöra en klass. I de flesta scenarion där objektdetektering tillämpas är det dock av intresse att identifiera flera olika objekt i en bild. Verkligheten består inte av enskilda objekt utan är ett samspel mellan flera olika objekt bredvid varandra.

För att spåra objekt i en bild som innehåller flera objekt används så kallade objektdetekteringsalgoritmer. Syftet med dessa är att de både ska lokalisera och klassificera objekten. Spårningen visualiseras genom att programmet ritar en ruta runt objektet som den har spårat. Dessa rutor kallas för objektmarkeringar. I detta avsnitt kommer tre olika modeller för objektdetektering att presenteras samt deras för- och nackdelar.

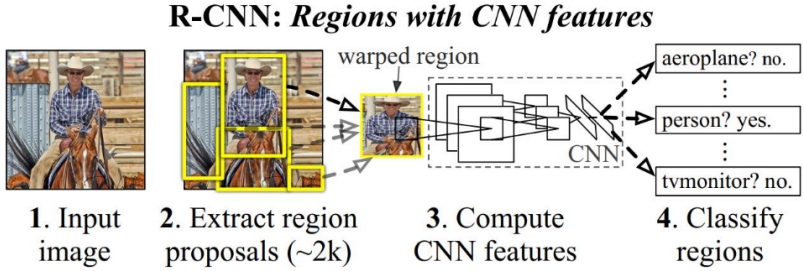
2.3.1 R-CNN

För att förstå hur de mer avancerade algoritmerna fungerar underlättar det att först förstå en algoritm vid namn R-CNN [12]. R-CNN står för Region-CNN och är en objektdetekteringsalgoritm i två steg, vilket innebär att algoritmen genomlöper bilden två gånger. För att klassificera objekten använder sig modellen av en algoritm som heter Support Vector Machine (SVM), för vidare läsning om SVM se [13]. R-CNN-modellen består av följande tre moduler:

1. Generera fönster/regioner som är kandidater för objektmarkeringar.

2. Tolka varje region med hjälp av ett traditionellt CNN och skapa attributskarta.

3. Ett lager av SVM. En SVM har tränats på varje klass och försöker klassificera bilden med hjälp av attributskartan.



Figur 7: Flöde över hur indatan (bilden) klassificeras genom de tre modulerna. Från [12]

Det finns olika tillvägagångssätt för att generera regionerna och det är viktigt att modulen inte genererar för många regioner, detta kan nämligen sakta ner hela processen avsevärt eftersom varje region ska analyseras. Modellen skalar sedan alla regioner till en fix storlek som passar det CNN som används så att nätverket kan bearbeta pixlarna i regionen. I originalarbetet [12] används fem convolutional layers och två fullt kopplade lager. Detta nätverk ger attributsvektorerna som skickas vidare till SVM-modulen. När klassificeringen är klar går sedan objektmarkeringen genom en linjär regressionsmodell för att successivt krympa rutan för att lokalisera objektet mer precist.

2.3.2 YOLO

YOLO har ett annorlunda angrepp på problemet jämfört med R-CNN. Som namnet, You Only Look Once, antyder genomlöper algoritmen bilden endast en gång. Detta görs genom att förena objektdetekteringens komponenter till ett nätverk.

Algoritmen börjar med att dela upp bilden i ett rutnät med $S \times S$ stycken celler (se Figur 8), där ett högre S ger en större noggrannhet. Om ett objekts mittpunkt befinner sig i en cell, så ska den cellen detektera objektet.

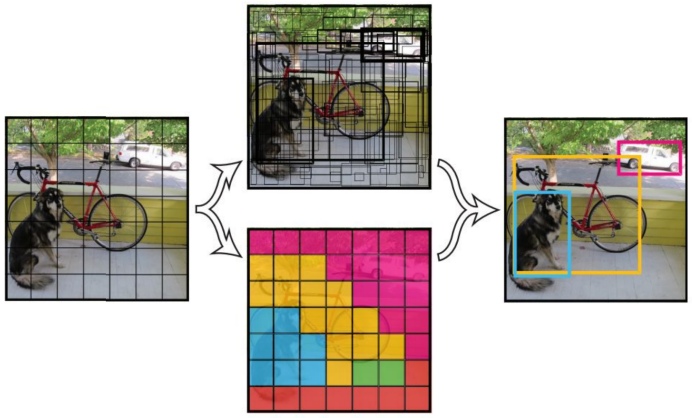
Varje cell förutspår B stycken objektmarkeringar med en konfidensvärde, denna poäng representerar hur säker den är på vad det är för objekt inuti objektmarkeringen och hur korrekt positionen av markeringen är. Konfidensvärdet ska vara noll om det inte finns något objekt i cellen, annars ska den vara proportionell mot hur mycket objektmarkeringen överlappar en objektmarkering som är helt korrekt.

Figur 8: Exempel på hur en bild delas in i celler med $S = 7$. Från [14].

Till varje objektmarkering finns det fem olika parametrar som nätverket har bestämt: x, y, w, h och k, där x- och y-koordinaterna beskriver förhållandet mellan objektmarkeringens mittpunkt och cellen. Parametrarna w och h beskriver objektmarkeringens bredd respektive höjd. Den sista parametern, k, är konfidensparametern.

För varje objektmarkering förutspår cellen också vilken klass objektet tillhör. Detta representeras i en vektor med längden C, där C är antalet klasser och varje siffra i vektorn representerar hur stor chans det är att det är just den klassen. Detta ger en siffra på både hur säkert nätverket är på att boxen är korrekt placerad samt på att rätt objekt har klassificerats.

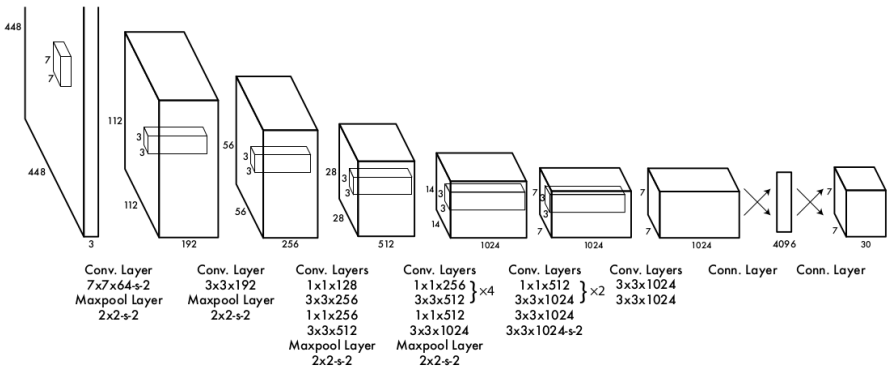
Det skapas totalt $S \times S \times B$ boxar när YOLO-algoritmen genomförs, men de flesta av dessa boxar har ett lågt konfidensvärde och tas därför bort. Trots att det skapas $S \times S \times B$ olika separata förutsägelser så körs de samtidigt i samma CNN, vilket gör algoritmen snabbare än R-CNN.



Figur 9: Bilden längst till vänster visar rutnätet som YOLO skapar, i mitten är lådorna med sannolikhet samt klassificering. Den sista bilden till höger visar algoritmens slutliga gissning. Från [14].

Med denna enstegsmetod är hela förutsägelsen representerad som en tensor med storleken $S \times S \times (B \times 5 + C)$, där 5 kommer från att boxarna har 5 parametrar. Problemet kan då hanteras som ett regressionsproblem, från pixeldata till klassannolikhet och koordinater för objektmarkeringar.

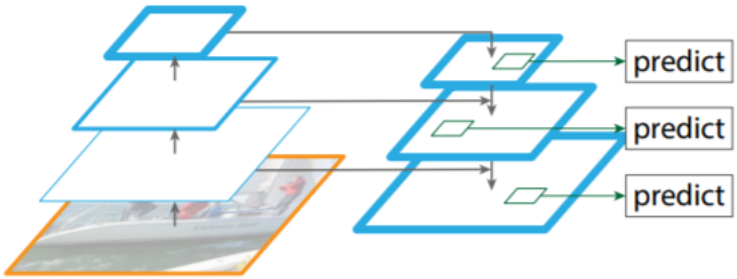
Nätverket för YOLO-algoritmen, se Figur 10, består av 24 stycken convolutional layers följt av 2 stycken fullt kopplade lager, vilka bestämmer konfidensvärde och koordinater för det detekterade objektet.



Figur 10: YOLO-nätverkets arkitektur. Nätverket består av 24 convolutional layers och 2 fullt kopplade lager. Från [14].

2.3.3 RetinaNet

En tredje modell som används för objektsklassificering är RetinaNet [15]. En av dess stora fördelar är att den är mycket bra på att detektera objekt med varierande skala. Detta beror på att RetinaNet använder sig av så kallade Feature Pyramid Networks (FPN). I ett FPN delas bilden upp i mindre bitar för att göra förutsägelser i olika skalor av bilden. Modellen blir således mindre känslig för förändringar i skala mellan träningsdata och bilden vars innehåll ska bestämmas [16]. Att dela upp bilden på detta sätt och testa varje lager i pyramiden i Figur 11 ger dock förhållandevis kostsamma beräkningar och gör RetinaNet långsammare än exempelvis YOLO [17].



Figur 11: Uppbyggnad av FPN. Till vänster delas bilden upp i mindre bitar av olika storlek som studeras individuellt. Därefter sammanfogas detta genom att resultatet för de olika bitarna kombineras för att kunna göra förutsägelser av vad bilden innehåller [18].

Vidare använder RetinaNet så kallad Focal Loss (FL). Den är dels simplare än kostnadsfunktionen för YOLO, och lägger samtidigt mer vikt vid svårklassificerade bilder än många andra simpla kostnadsfunktioner. Formeln för Focal Loss är

$$\mathrm{FL}(p_t) = -\alpha(1-p_t)^{\gamma}\log(p_t),$$

där α är en parameter som används för att kompensera för obalans mellan mängden träningsdata för varje klass, γ en parameter som styr hur mycket fokus som läggs vid svår- respektive lättklassificerade exempel och p_t något förenklat representerar sannolikheten att ett visst objekt tillhör en viss klass [19].

2.4 Finita elementmetoden

Finita elementmetoden är en numerisk metod för att numeriskt lösa partiella differentialekvationer. Metoden går att applicera på en mängd olika fysikaliska problem, däribland hållfasthetsrelaterade problem.

Kortfattat går metoden ut på att dela upp området som beräkningarna utförs på i finita element. Elementen kan variera i storlek och antal beroende på vilken noggrannhet som efterfrågas av beräkningen. Över varje element ansätts en lokal lösning till den styrande differentialevkationen, därefter söks den lösning som bäst approximerar den analytiska lösningen. För en utförligare beskrivning av finita elementmetoden se [20].

3 Metod

Det här avsnittet beskriver arbetsflödet under projektet, vilket var en iterativ process. Delprocessen med start från insamling av data till färdigtränat nätverk genomfördes flera gånger tills resultaten var tillfredsställande. Genom att dra lärdomar från varje försök gjordes förändringar i tillvägagångssättet mellan varje iteration. Då programmen för bilduppritning och beräkningar konstruerades parallellt behövde även dessa anpassas vid de olika iterationerna. I detta avsnitt beskrivs även paket, verktyg och metoder som använts samt motiveringar till varför de valdes.

3.1 Informationssökning

För att kunna skriva ett program för objektdetekteringen utfördes en litteraturstudie om objektdetektering och vilka algoritmer som skulle kunna uppfylla syftet. Studien grundade sig på artiklar och böcker tillgängliga via internet.

3.2 Insamling av träningsdata

För att kunna träna objektdetekteringsalgoritmen behövdes en stor mängd data att träna på. Datan bestod av handritade bilder, som ritades av samtliga medlemmar i projektgruppen för att öka variationen i datan. Det skapades två typer av bilder. Den första typen innehöll endast en symbol per bild och den andra var hela uppställningar av problem. För att träna algoritmen på bilderna behövde de individuella symbolerna markeras och klassificeras i bilderna. För att hitta relevanta symboler som utgör komponenterna i ett hållfasthetsproblem användes föreläsningsanteckningar från kursen MTM026 i hållfasthetslära vid Chalmers Tekniska Högskola [21].

3.3 Utveckling av programmet

Projektet delades upp i tre grenar som skulle utvecklas parallellt. De tre grenarna var objektdetektering, generering av datorritad bild och beräkningar. Innan arbetet påbörjades analyserades vilken indata och utdata underprogrammen behövde tillhandahållas och generera för att förenkla den slutliga sammansättningen.

3.3.1 Objektdetektering

För objektdetektering valdes algoritmen YOLO, dels för algoritmens prestanda samt för att den var enkel att implementera med hjälp av projektet *TrainYourOwnYOLO*. För att öka träningstakten användes C3SE:s datorkluster [22].

3.3.2 Bilduppritning

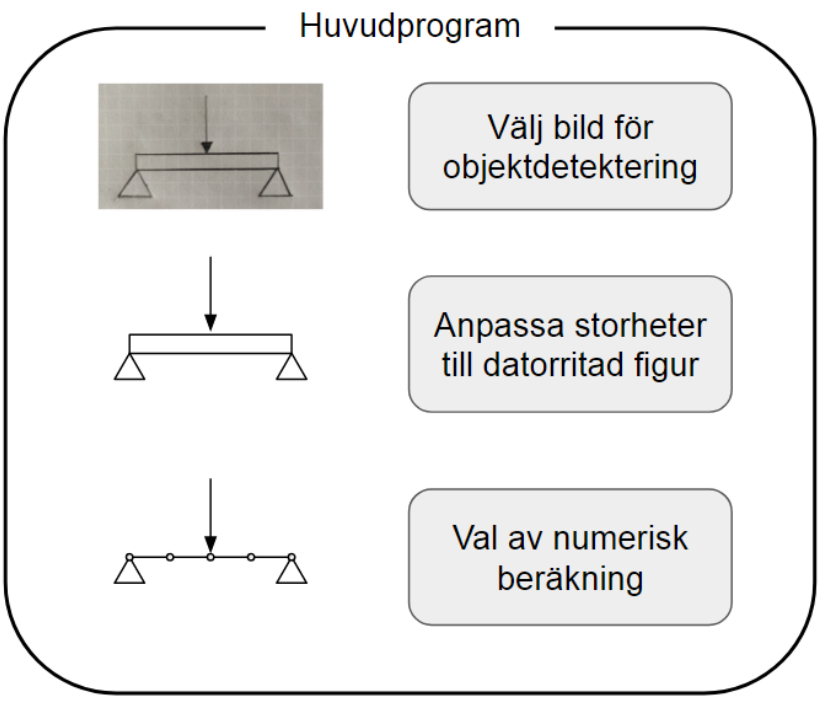
För att kompensera för den varierande storleken och positionen hos objektmarkeringarna implementerades funktioner för att säkerställa att objekten alltid låg kant i kant med objektens angreppspunkter. Detta är viktigt när balken diskretiseras i beräkningsmodulen. Uppritningen av balkproblemet använder paketet Tkinter som valdes då det är standardpaket för gränssnittsprogrammering i Python [23].

3.3.3 Beräkningar

För att göra beräkningar av hållfasthetsproblemet valdes FEM. Det valdes för att kunna ge numeriska lösningar samt möjlighet att plotta olika diagram, exempelvis tvärsnitt- och momentdiagram. FEM är även ett standardverktyg för numeriska lösningar av hållfasthetsproblem och programmet skrevs med hjälp av CALFEM [24].

3.3.4 Huvudprogram

För att samtliga delprogram skulle kunna användas tillsammans skapades ett huvudprogram vars syfte var att skicka datan mellan de olika delprogrammen samt skapa ett grafiskt användargränssnitt för val av bild. Detta möjliggör användande av alla delprogram utan programmeringkunskaper. I Figur 12 visas huvudprogrammets uppbyggnad.



Figur 12: Figuren visar hur huvudprogrammet är uppbyggt. Rutorna till höger i figuren motsvarar de funktioner användaren har tillgång till för att göra olika val i programmet.

4 Genomförande

I detta kapitel redogörs för hur projektet har genomförts. Det innefattar bland annat beskrivningar av de olika delprogrammen samt närmare beskrivningar av de iterationer som nämns i kapitel 3.

4.1 Objektdetektering

Följande avsnitt beskriver hur data samlades in och hur den användes för att träna ett neuralt nätverk att känna igen hållfasthetssymboler. Objektdetekteringen var en iterativ process och varje del genomfördes flera gånger tills programmet hade uppnått tillfredsställande resultat.

4.1.1 Implementering av objektdetekteringsalgoritm

Den första objektdetekteringsalgoritmen som implementerades var YOLO och det gjordes med hjälp av det befintliga projektet TrainYourOwnYOLO. För en detaljerad guide av hur projektet implementerades och det neurala nätverket tränades, se [6].

Till en början gjordes all träning av nätverket på projektgruppens egna datorer, vilket gav undermåliga resultat på grund av låg minneskapacitet. Det började därför arbetas parallellt med olika försök till att lösa problemen med objektdetektering. En av metoderna som testades var RetinaNet och den verkade mycket lovande.

Under denna period fick projektet även tillgång till C3SE:s datorkluster. Vid de första träningarna på klustret användes YOLO-algoritmen och då den gav utmärkta resultat valdes algoritmen att gå vidare med.

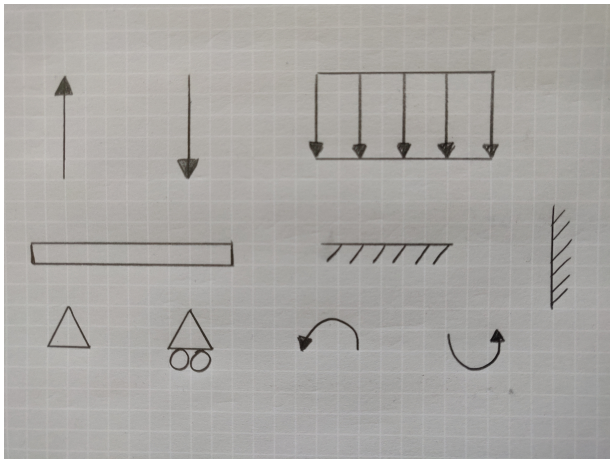
4.1.2 Insamling av träningsdata

För att enklast samla träningsdata på hållfasthetssymbolerna skapades datan från grunden genom att rita symbolerna på papper, fotografera de ritade symbolerna med mobilkameror och sedan föra över bilderna till en dator.

Vid första försöket att samla in träningsdata skapades bilder som innehöll en hållfasthetssymbol per bild. Tanken med detta var att undersöka algoritmens förmåga att känna igen symbolerna med väldigt få detaljer på bilderna. För att användare med olika handstil ska kunna använda programmet är det viktigt att algoritmen inte är överanpassad till perfekta symboler [25]. Under första försöket ritades därför symbolerna med varierande noggrannhet med hypotesen att inte överanpassa modellen till perfekta symboler.

Vid andra försöket ritades symbolerna så att det var flera symboler på samma bild, några bilder liknade ett uppställt hållfasthetsproblem och några bilder var flera fristående symboler uppritade bredvid varandra. Symbolerna ritades denna gång noggrannare än vid första försöket.

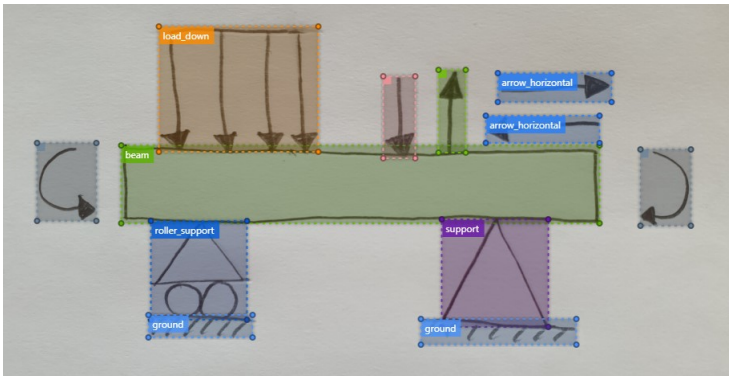
Vid det tredje försöket lades mer fokus på noggrannhet så att symbolerna skulle vara så lika varandra som möjligt. Detta gjordes med hypotesen om att tidigare träningars undermåliga resultat berodde på att variansen hos symbolerna inom samma klass var såpass stor att modellen hade svårt att finna likheter mellan dem. En stilmall skapades, se Figur 12, för att användas som stöd för att utseendet av de olika symbolerna skulle variera mindre.



Figur 13: Stilmallen som användes. Symbolerna som ritades skulle efterlikna symbolerna på bilden.

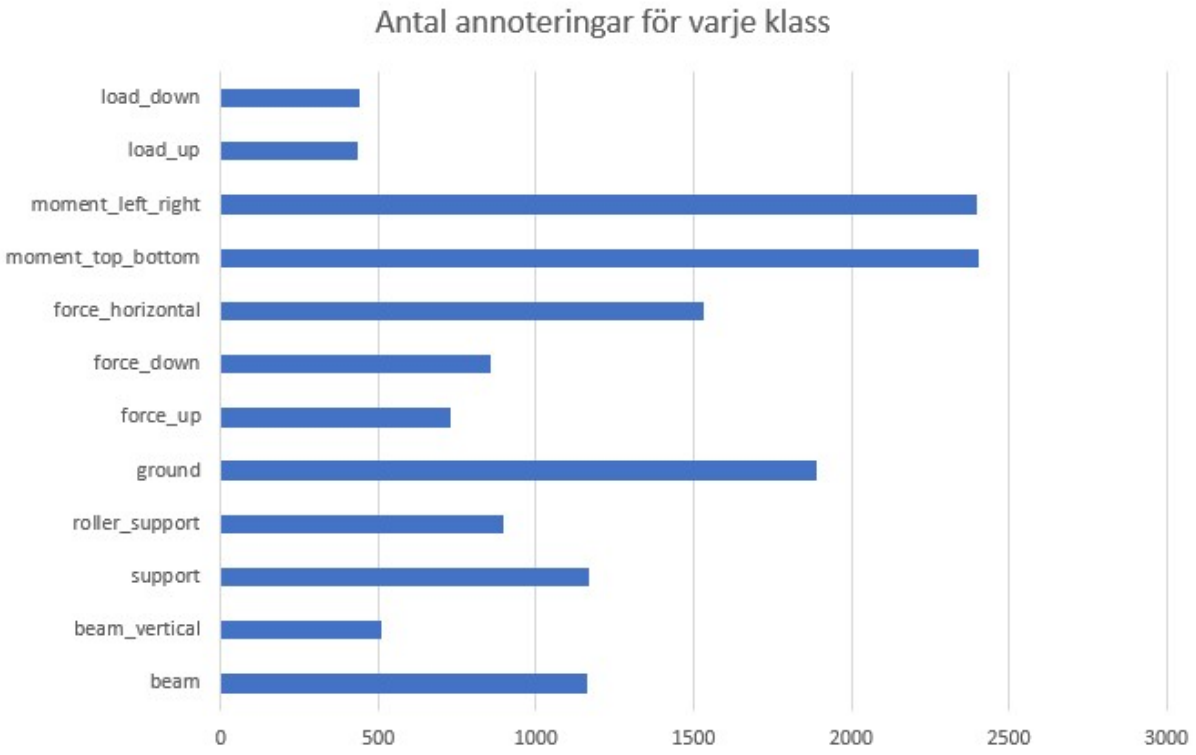
4.1.3 Annotering av träningsdata

Nästa steg var att annotera bilderna så att modellen fick information om position och klass på symbolerna i bilderna och därmed kunde träna på dem. Detta gjordes med hjälp av Microsofts Visual Object Tagging Tool (VoTT) [26]. Programmet tillät användaren att markera objekten med muspekaren och klassificera dem, se Figur 14 för ett exempel.



Figur 14: Skärmdump från annoteringsprogrammet VoTT. Markeringarna avgränsar komponenternas position, och komponenternas klass anges av etiketten på markeringen, exempelvis ”beam” för balk och ”roller_support” för rullstöd. Anledningen till att klassens namn inte visas på alla symboler är att namnet inte får plats horisontellt i markeringen.

Totalt annoterades 14420 symboler i 1127 bilder, och statistik för antal annoterade symboler för varje klass kan ses i Figur 15. Anledningen till att antal annoteringar för ”moment_left_right” och ”moment_top_bottom” är avsevärt högre än resterande klasser är att de slagits ihop av ett flertal klasser som användes tidigare i projektet.



Figur 15: Statistik för antal annoterade symboler för varje klass.

Från VoTT exporterades sedan en Comma Separated Values-fil (CSV) innehållande position och klass för alla annoterade symboler samt filsökväg för bilden. Denna konverterades sedan genom ett pythonskript till ett format anpassat för YOLO-algoritmen. I denna fil fanns all nödvändig information för att träna det neurala nätverket.

4.1.4 Träning av det neurala nätverket

Att träna nätverket visade sig vara en svårare uppgift än förväntat. Därför genomfördes ett flertal försök med olika dataset för att försöka lära nätverket att känna igen de utvalda symbolerna. Exempelvis ändrades klasserna hos de olika symbolerna flera gånger. Till en början klassificerades horisontella punktlaster som ”arrow_left” och ”arrow_right”, vilket sedan ersattes med ”force_horizontal” då modellen hade svårt att skilja på vänster- och högerpilar. Alla klasser i det slutgiltiga resultatet återfinns i Bilaga A.

Träningen utfördes på C3SE:s datorkluster. För ett detaljerat genomförande av proceduren att ansluta till klustret och använda deras resurser, se Bilaga B.

4.2 Bilduppritning

Bilduppritningsprogrammet utgörs av ett pythonskript som genererar datorritade bilder utifrån koordinaterna på de objektmarkeringar som genereras av objektdetekteringsmodellen. Då bilden skapats skickas alla objekt med respektive angreppspunkter vidare till beräkningsprogrammet. För det fullständiga programmet, se [27].

4.2.1 Tolkning av utdata från YOLO

Första steget i bilduppritningsprogrammet är att läsa in den CSV-fil som skapats från objektdetekteringen. Därefter löper skriptet igenom alla objekt och kontrollerar om några objekt överlappar varandra markant. Om så är fallet utförs en åtgärd beroende på vilka objekt som överlappar. Om exempelvis en punktlast identifieras inuti en utbredd last tas punktlasten bort och om två objekt av samma typ identifieras på ungefär samma position tas objektet med lägst konfidensvärde bort.

4.2.2 Uppritning av bild

När indatan förberetts enligt ovan ritas alla balkar som identifierats upp utifrån koordinaterna från CSV-filen. Balkarna delas då upp i 13 jämnt fördelade punkter på vardera av dess långa sidor, och en punkt i mitten på varje ände. Detta illustreras i Figur 16. Att just dela in balkens sidor i 13 punkter valdes då detta möjliggör placering på många av de angreppspunkter som bedömdes vanligast för grundläggande hållfasthetsproblem ($\frac{L}{2}$, $\frac{L}{3}$, $\frac{L}{4}$, $\frac{L}{6}$ med flera, där L är balkens totala längd).

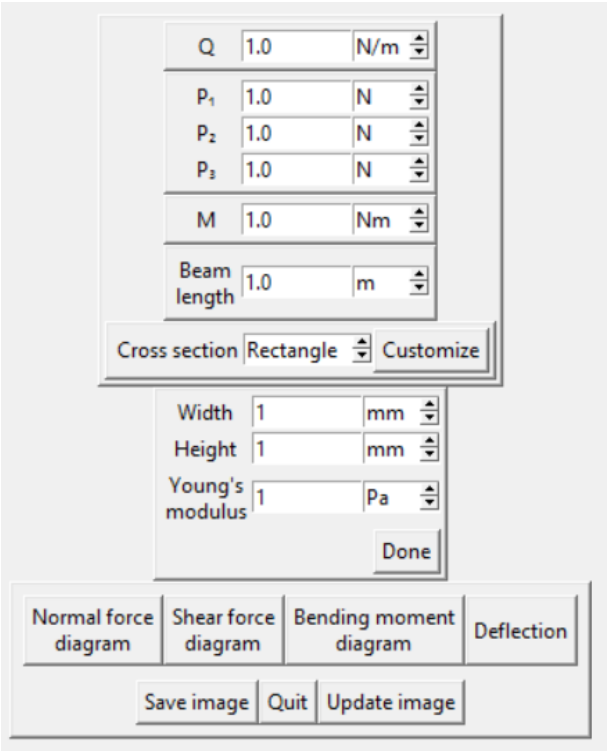
Figur 16: Schematisk bild över de totalt 28 punkter på en balk varvid ett objekt kan verka.

För de resterande objekten används sedan en funktion som utifrån de koordinater som erhållits från objektdetekteringen letar upp närmsta relevanta punkt på balken. Med relevant punkt menas en av de 28 punkterna i Figur 16 där objektet skulle kunna verka. En utbredd last som verkar i vertikal-led kan till exempel inte placeras i någon av punkterna i mitten på kortsidorna av balken i Figur 16. Objektets angreppspunkt flyttas därefter till den punkten. Alla objekt som har en magnitud (moment och laster) eller längd (balkar) tilldelas sedan en etikett och alla objekt ritas ut.

För att skapa en mer tilltalande visualisering av uppställningen används vissa standardstorlekar för de olika komponenterna, balkarna undantaget. Detta är speciellt relevant då objektmarkeringarna som genereras av YOLO-algoritmen inte är perfekta. Till exempel sätts alla punktlasters längd till 20% av den tillhörande balkens längd.

4.2.3 Anpassning av storheter

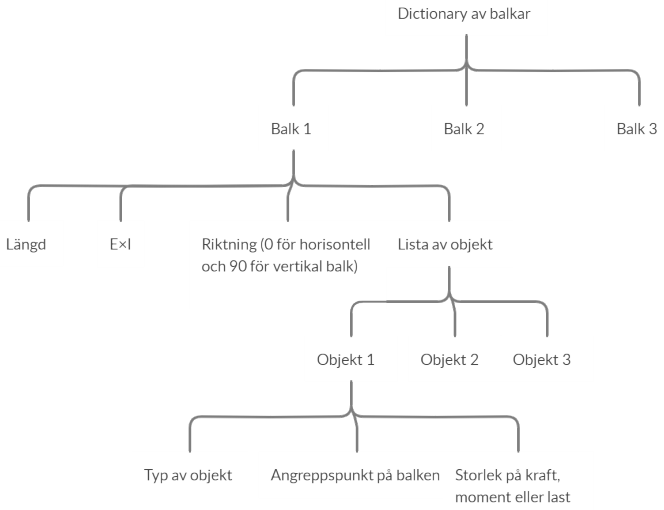
Efter att problemuppställningen ritats upp möts användaren av menyn i Figur 17, där objektens magnitud kan anges. Användaren kan även ange balkens tvärsnitt och elasticitetsmodul för att beräkna böjstyvheten. Menyn ger användaren tillgång till två ytterliggare funktioner: en sparfunktion och en justeringsfunktion. Sparfunktionen låter användaren välja en mapp i datorns lokala filsystem där den aktuella bilden sparas som en png-fil och kallas genom att trycka på ”Save image”-knappen. Justeringsfunktionen kallas då ett diagram ska tas fram eller då ”Update image”-knappen aktiveras. När den kallas ändras objektens magnitud till de ifyllda värdena, och om respektive fält innehåller ett negativt värde vänds även moment och laster.



Figur 17: Ett exempel på hur menyn kan se ut. I det här fallet verkar en utbredd last, tre punktlaster och ett böjmoment på balken. I figuren syns fältens standardvärden och standardenheter. För att ta fram de önskade diagrammen används knapparna på raden näst längst ner.

4.2.4 Utdata till beräkningsprogrammet

Då användaren trycker på en av diagramknapparna läggs objekten, deras magnitud samt vid vilka punkter de verkar, till i en lista tillhörande balken de verkar på. Varje balk, inklusive balklängd, riktning på balken, den uträknade böjstyvheten och tillhörande lista på objekt, läggs sedan till i en Python Dictionary som ligger till grund för arbetets nästa del, FE-beräkningarna. För visualisering av utdatan till FE-beräkningarna, se figur 18.



Figur 18: Schematisk bild över utdata till FE-beräkningarna. Utdatan är uppbyggd som en Python Dictionary med varje enskild balk som nyckel och en Tuple av balklängd, böjstyvhet ($E \times I$), orientering och en lista av alla tillhörande objekt som värde. Varje element i listan är i sin tur en Tuple som innehåller typ av objekt, angreppspunkt på balken samt storlek på moment eller laster. Den sistnämnda komponenten definieras som positiv åt höger respektive uppåt i bilden och sätts till ”None” för de objekt där storlek ej är relevant.

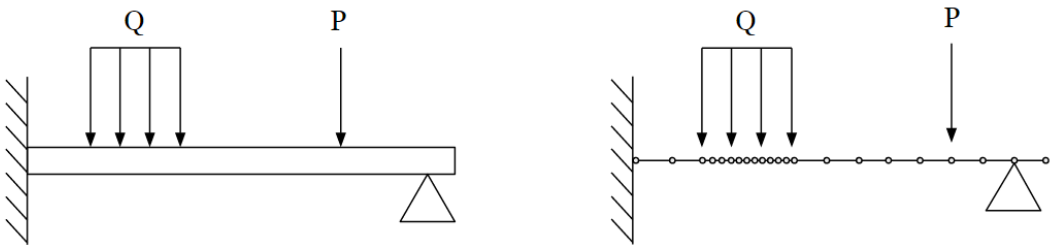
4.3 Beräkningar

Detta avsnitt avser att förklara hur FEM-programmet skapades (för att se koden i sin helhet, se [27]).

Det första programmet gör är att den sorterar indatan i tre olika listor. Den första listan innehåller alla punktlaster och moment med tillhörande koordinater, den andra innehåller alla stöd och dess koordinater och den tredje innehåller alla utbredda laster och dess koordinater. Listorna används sedan som input till en klass, Beam, som utför alla beräkningar. För att utforma klassen Beam utnyttjades biblioteket CALFEM, speciellt utnyttjas följande funktioner:

- beam2e - tar fram elementstyvhetsmatrisen och elementlastvektorn.
- assem - assemblerar elementstyvhetsmatriserna till den globala styvhetsmatrisen.
- solveq - löser ut förskjutningarna i balken, genom att lösa finita element-ekvationerna.
- extract - tar fram elementförskjutningarna.
- beam2s - räknar ut snittkrafterna i balken.

För att dela upp balken i element utgår programmet från var det finns punktlaster, stöd och utbredda laster då detta påverkar hur balken kan delas upp. Programmet har finare fördelning av finita element under de utbredda lasterna, detta för att få bättre precision i beräkningen av dessa delar. För att se hur balken delas upp i finita element se Figur 19.



Figur 19: Figuren visar ett exempel på hur indelningen av finita element går till. Till vänster i figuren visas ett exempel som är fast inspänt med en utbredd last Q, en punktlast P och ett stöd. Till höger i figuren visas FE-modellen med beräkningsnätet visualiserat. Notera hur indelningen är mycket finare under den utbredda lasten jämfört med övriga delar av balken. Punktlasten P och stödet ligger också båda på varsin nod.

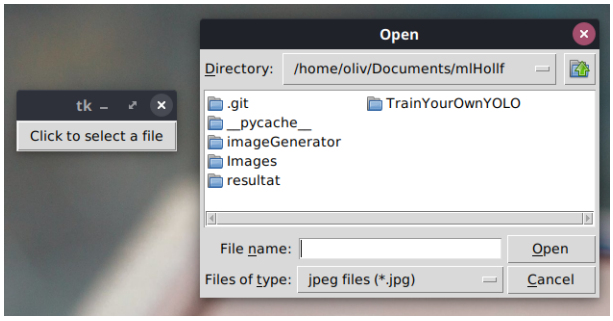
Det är nödvändigt att punktlaster och stöd placeras på noder i balken då FE-diskretiseringen genereras. En annan viktig sak att ta hänsyn till är att elementsyvhetsmatrisen ska beräknas annorlunda beroende på om balkelementet belastas av en utbredd last eller inte. Koordinatsystemet är definierat med x-led längs med balken i längdriktningen och y-led är vertikalt, dvs normalt mot x-axeln.

Utdatan från programmet är enligt avsnitt 1.4 begränsade till att enbart beröra moment-, tvärkrafts- och normalkraftsdiagram samt utböjningen i balken. Genom att skriva separata metoder för varje typ av utdata kan storheterna visualiseras i var sitt diagram. Diagrammen visar en översiktlig bild över hur punktlaster, moment och utböjningen varierar längs med balken.

4.4 Huvudprogram

För att kunna använda resultatet av nätverket och FE-beräkningarna rent praktiskt skapades det ett huvudprogram. Detta gjordes genom att skriva ett program som exekverade de olika delprogrammen med vissa ändringar i indatan mellan anropen.

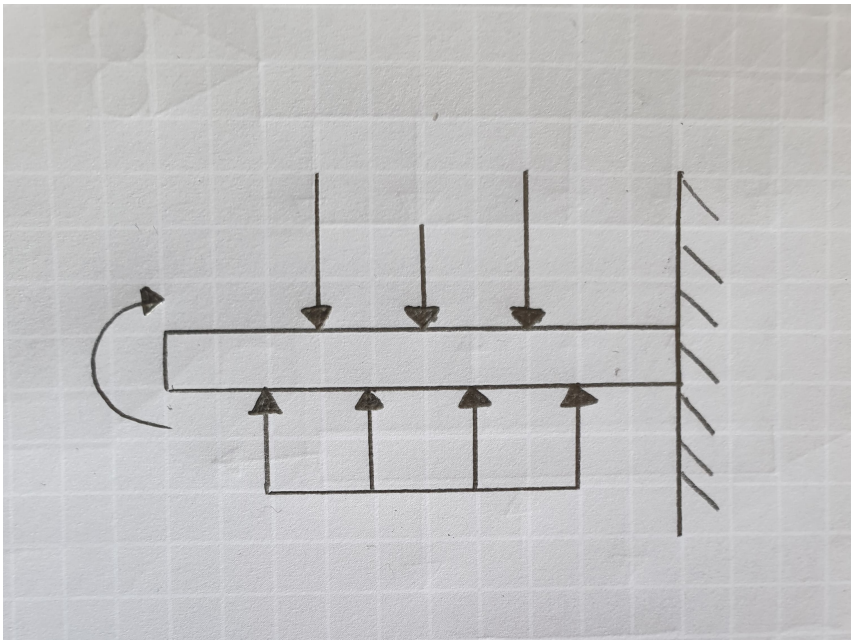
När huvudprogrammet körs ges användaren ett fönster för att välja bild, se Figur 20. Bilden kopieras sedan in i en utvald mapp för att köra objektdetektering. Därefter anropas objektdetekteringsalgoritmen och en CSV-fil med objektmarkeringar genereras. Därefter anropas bilduppritningsprogrammet för uppritning, slutligen anropas beräkningsprogrammet och förser användaren med de önskade diagrammen.



Figur 20: Figuren visar det gränssnitt som användaren presenteras vid körning av huvudprogrammet. Utseendet kan variera beroende på vilket operativsystem som körs.

5 Resultat

I detta avsnitt presenteras de resultat som erhölls under projektets genomförande. Resultaten illustreras genom ett exempel där de huvudsakliga momenten, från indata i form av en bild av ett problem till utdata i form av en datorritad bild av problemet, en deformationsfigur samt normalkrafts-, tvärkrafts- och momentdiagram, visas. Detta görs med bilden i Figur 21 som utgångspunkt.



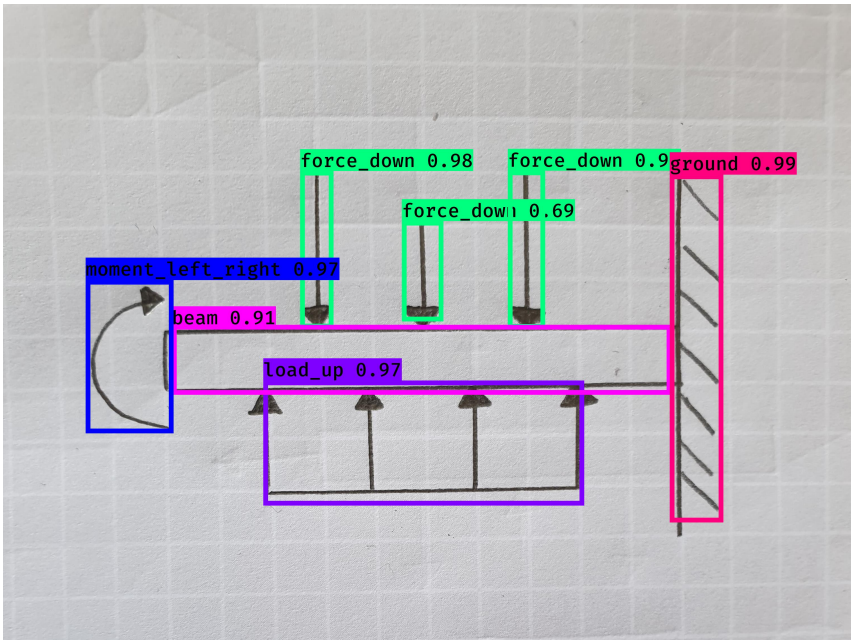
Figur 21: Exempelbild vars väg genom programmet kommer presenteras.

5.1 Objektdetektering

Det första steget i programmet är att identifiera de olika symbolerna i bilden. I det här avsnittet redogörs resultaten från träningen av modellen på hållfasthetssymboler.

5.1.1 Träning av det neurala nätverket

Träningen av det neurala nätverket avslutades automatiskt i förtid (med ett kostnadsvärde kring 38) efter 123 epoker då inlärningen hade planat ut. Anledningen till detta är att modellen inte lyckades minska kostnadsvärdet ytterliggare och därmed inte kunde lära sig mer. Modellen identifierar de flesta symbolerna korrekt och med hög konfidens, se Figur 22 för ett exempel, och Bilaga D för ett större urval.



Figur 22: Figuren visar de objekt som modellen identifierade i Figur 21. Modellen identifierar alla symboler korrekt och med hög konfidens på merparten av symbolerna. Exempelvis är modellen 91 % säker på att objektmarkeringen i mitten är innehåller en balk, vilket det är.

5.1.2 Utdata till bilduppritningsprogrammet

Utdatan från objektdetekteringen består av en CSV-fil innehållande koordinater för hörnen av objektmarkeringen och klass för symbolen för varje identifiering modellen gjort i bilden. Se Figur 23 för CSV-filen som representerar identifieringarna från Figur 22. Exempelvis representerar tredje raden nerifrån i Figur 23 markeringen kring balken i Figur 22, där en smal horisontell markering med 95 % konfidens kan ses.

image	image_pat	xmin	ymin	xmax	ymax	label	confidenc	x_size	y_size
91982784_	/local/tmp	1808	567	1925	1178	10	0.717519	3024	4032
91982784_	/local/tmp	1053	621	1177	1135	10	0.799967	3024	4032
91982784_	/local/tmp	1424	775	1572	1163	10	0.97844	3024	4032
91982784_	/local/tmp	826	1393	2228	1739	6	0.559441	3024	4032
91982784_	/local/tmp	624	1135	2355	1377	5	0.950905	3024	4032
91982784_	/local/tmp	2376	600	2550	1902	4	0.983016	3024	4032
91982784_	/local/tmp	306	989	624	1530	2	0.985935	3024	4032

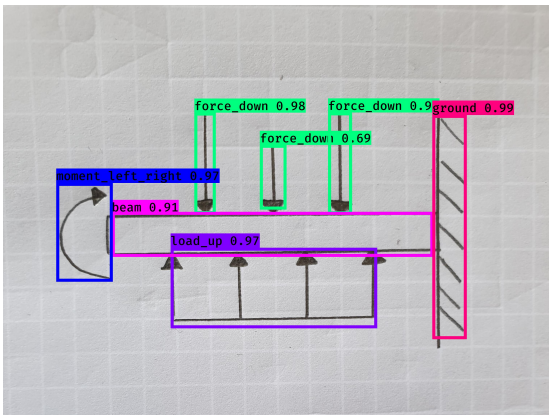
Figur 23: Figuren visar innehållet från CSV-filen som genererades utifrån bilden som visas i figur 21. Filen innehåller koordinater för objektmarkeringarnas hörnpunkter, säkerheten av klassificeringen, samt storlek för originalbildens bredd och höjd, x_size och y_size, i antal pixlar. Kolonnen ”label” indikerar vilken klass modellen identifierat markeringen som, där varje siffra motsvarar en klass.

5.2 Bilduppritningsprogram

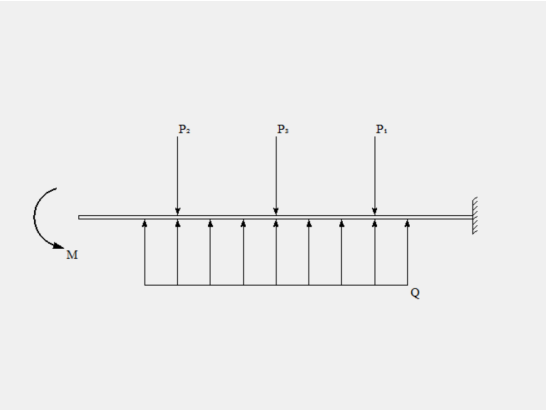
Utifrån CSV-filen från objektdetekteringen startas bilduppritningsprogrammet. Nedan följer ett exempel på de olika steg som användaren tas genom innan FE-beräkningarna utförs. Eftersom CSV-filen i Figur 23 och bilden i Figur 22 innehåller samma information används bilden för att enklare kunna visualisera processen, men i programmet är det istället CSV-filen som används.

5.2.1 Tolkning av utdata från YOLO

I Figur 24 visas ett exempel på hur YOLO-programmets utdata tolkas och ritas upp. Notera att alla objekt ritats upp korrekt förutom momentpilen på balkens vänstra sida. Detta beror på att alla momentpilar på vänstersidan av en balk sätts till motsols då ingen riktning fås från objektdetekteringen.



(a) Utdata från objektdetektering på Figur 21.

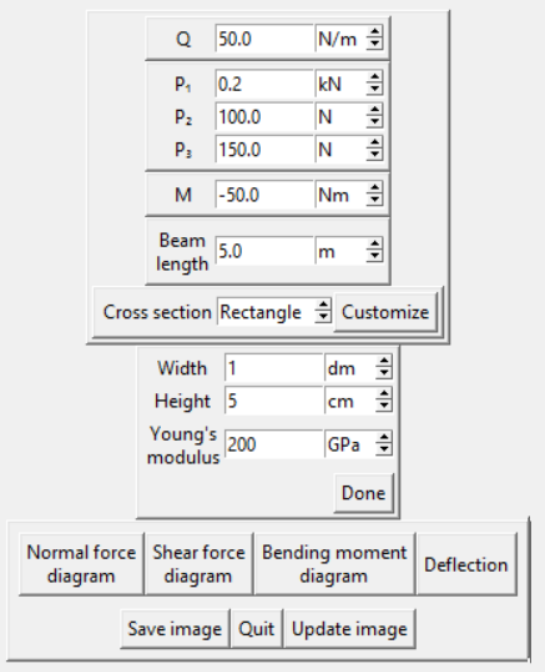


(b) Datorritad bild som genererats ur bilden till vänster.

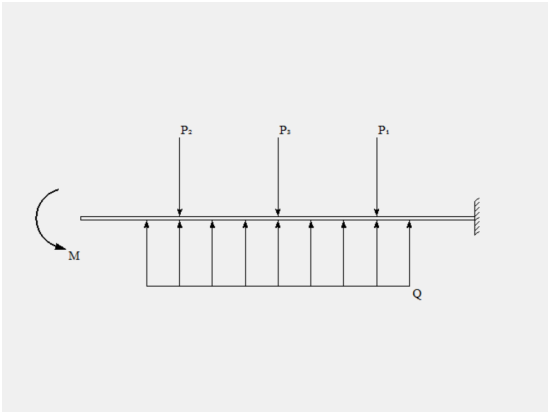
Figur 24: Ur objektmarkeringarna från bilden i (a) genererades bilden i (b).

5.2.2 Anpassning av storheter

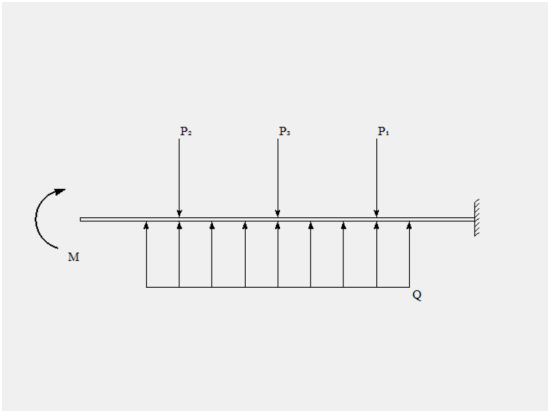
Förutom balkmodellen i Figur 24b förses användaren även med fälten i Figur 25a där storleken på moment och laster, samt längd och tvärsnitt på balkarna kan ställas in. Om ett negativt värde för moment eller laster anges ändras riktningen på objektet i bilden. Detta kan till exempel användas för att korrigera för fel i identifieringen, likt exemplet i Figur 24. Genom att fylla i ett negativt tal i fältet för M, i Figur 25a, ändras momentets riktning. Jämför Figur 25b med Figur 25c.



(a) Fälten där användaren kan anpassa storheterna.



(b) Bilden som genererats ur objektmarkeringarna från YOLO.

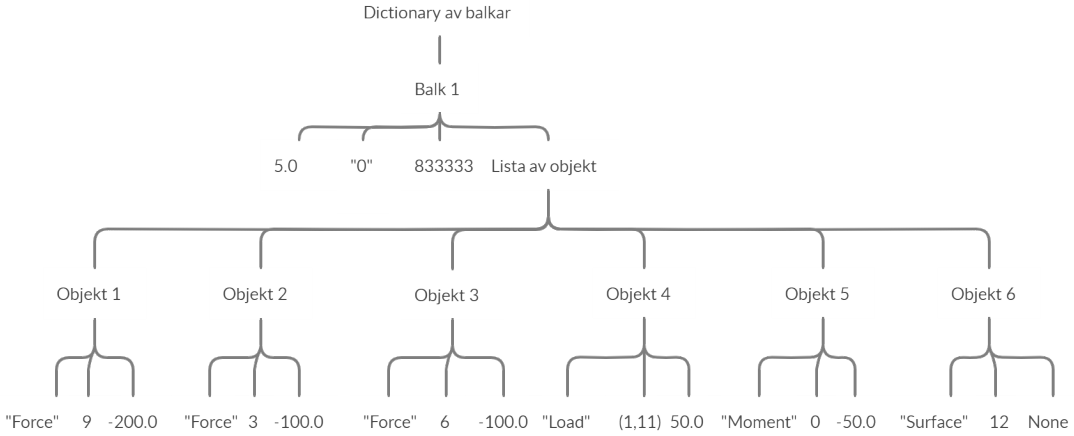


(c) Korrigerad version av (b) som erhålls med inställningarna i (a).

Figur 25: Genom att fylla i ett negativt tal i fältet för M i (a) ändras riktningen på momentet och bilden (b) ändras därmed till (c).

5.2.3 Utdata till FE-programmet

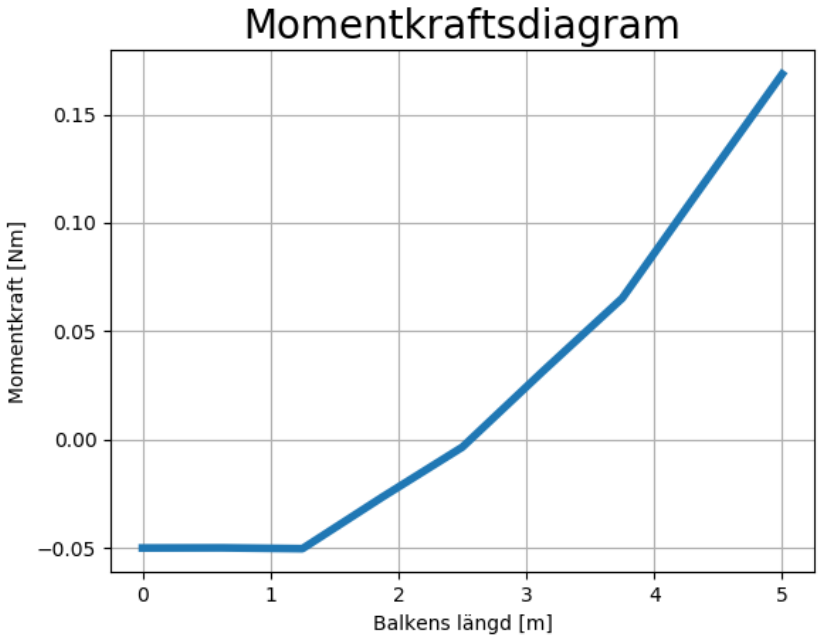
Bilduppritningsprogrammet har också till uppgift att tillhandahålla nödvändig information för att kunna skapa en FE-modell. Genom att fylla i fälten enligt Figur 25a och trycka på någon av knapparna för de olika diagrammen erhålls en Python Dictionary enligt Figur 26. Uppbyggnaden av detta beskrivs närmare i Figur 18.



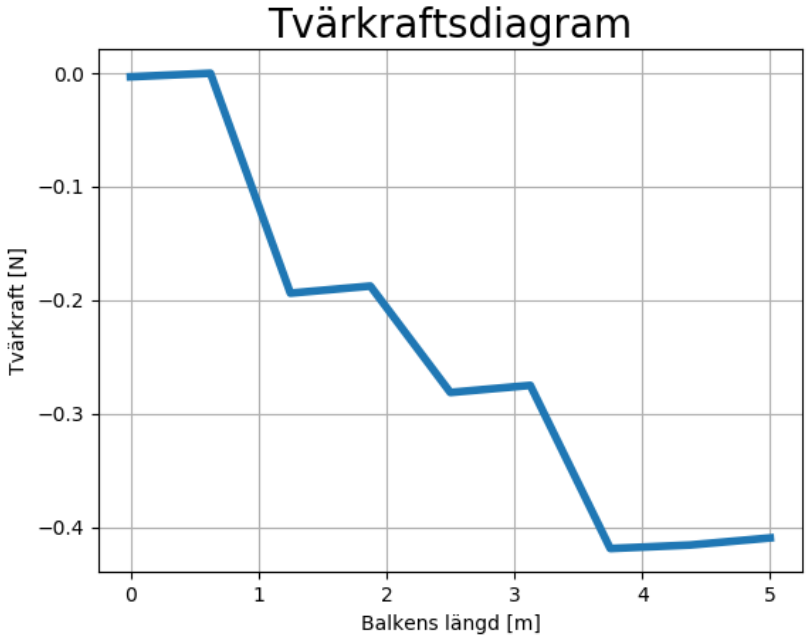
Figur 26: Schematisk bild över utdata från bilduppritningsprogrammet med inställningarna i Figur 25a.

5.3 Beräkningsprogram

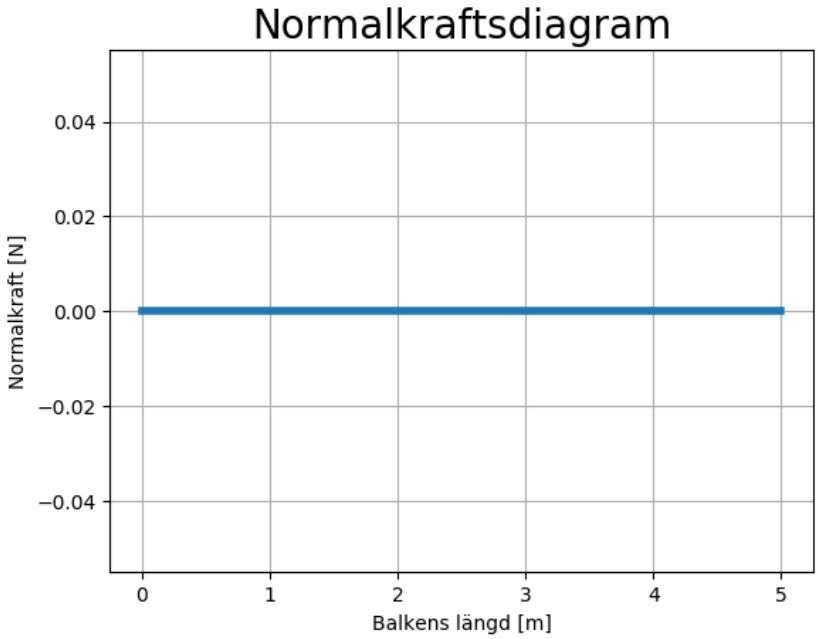
Beräkningsprogrammet ställer upp ett ekvationssystem utifrån FE-diskretisering för balken som ges i Figur 21. Programmet plottar, med hjälp av de laster och andra storheter som är givna i Figur 25a, fyra olika grafer för balken: moment-, tvärkrafts- och normalkraftsdiagram samt en deformationsfigur som visar balkens utböjning. Diagrammen bygger på balkens snittkrafter i noderna plottade mot nodernas koordinater längs med balken. I Figur 27 till 30 visas alla framtagna diagram för balken.



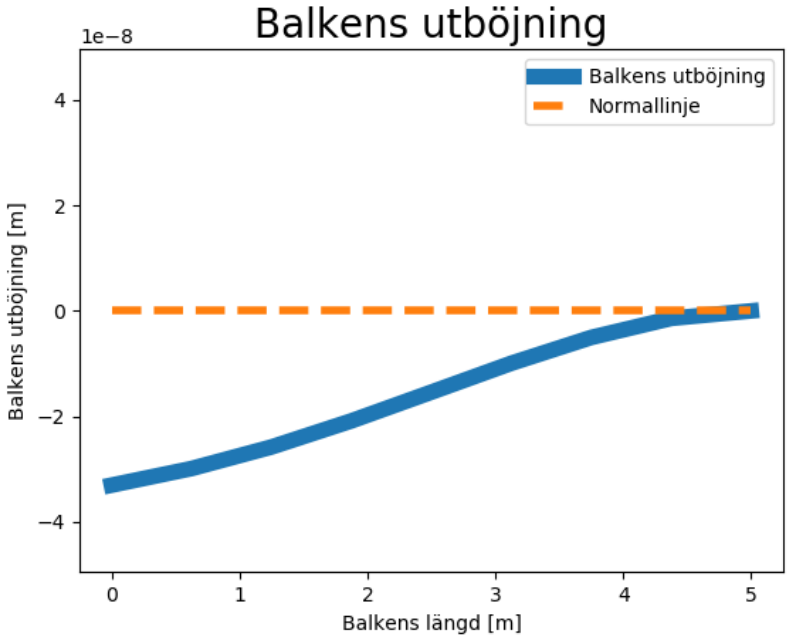
Figur 27: Momentkraftsdiagram för balken. Värdena på y-axeln visar momenten i balken med enheten Newtonmeter. Momentets förändring har approximerats till en linjär kurva i varje element. Ur grafen framgår att momentet är som högst i balkens högra ände, det vill säga vid den fasta inspänningen.



Figur 28: Tvärkraftsdiagram för balken. Värdena på y-axeln visar värdet för tvärkrafterna i balken med enheten Newton. Kurvan stämmer bäst överens med verkligheten i noderna.



Figur 29: Normalkraftsdiagram för balken. Värdena på y-axeln visar normalkrafterna i balken med enheten Newton. Eftersom balken inte utsätts för några axialkrafter uppstår inga normalkrafter.



Figur 30: Deformationsdiagram för balken. På y-axeln visas värdet för balkens utböjning i enheten meter. Syftet med den streckade linjen är att tydligare visualisera hur balkens utböjning skiljer sig mot centrumlinjen. Notera att värdena på y-axeln är i storleksordningen 10^{-8}.

6 Diskussion

I detta avsnitt diskuteras svårigheter som uppkommit under projektets gång och vissa val som gjorts förklaras närmare. Dessutom diskuteras de erhållna resultaten och vidareutvecklingar som skulle kunna göras i eventuella framtida projekt föreslås.

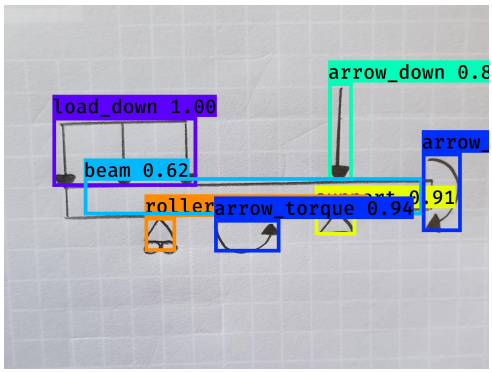
6.1 Objektdetektering

Detta delavsnitt berör resultaten från objektdetekteringen.

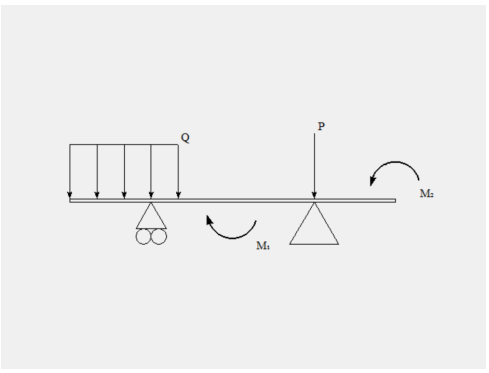
6.1.1 Generalisering av objekt

Vid träningen av det neurala nätverket uppstod flera problem, exempelvis blandades riktningar på horisontella pilar och moment ihop. Med anledning av dessa svårigheter slogs höger- och vänsterpilar ihop till att bara identifieras som horisontella pilar, och alla former av medsols och motsols moment slogs ihop till två olika objekt: ”moment_top_bottom” (moment som verkar på ovan- eller undersidan av en balk) och ”moment_left_right” (moment som verkar på vänster- eller högersidan av en balk). Generaliseringen bedömdes vara lönsam då modellen hade mycket svårt att skilja på riktningarna hos dessa objekt. Att ha kvar dem som distinkta objekt hade alltså resulterat i en nästintill slumpmässig riktning, men med betydligt lägre konfidens än vad som erhölls då de slagits ihop. Då möjligheten finns att i efterhand ändra objektens riktning ansågs sammanslagningarna inte heller vara en alltför stor begränsning.

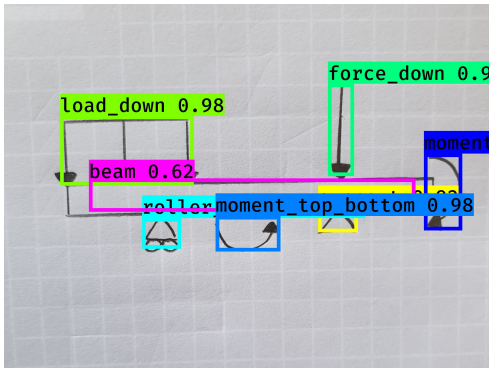
Anledningen till att alla moment inte slogs ihop till samma objekt var att då detta testades riskerade till exempel ett moment som ritades på högra kortsidan av balken att flyttas till ovansidan (se Figur 31). Detta påverkar visserligen inte beräkningarna eftersom det högra momentet i Figur 31 i båda fallen placeras på samma nod i FE-beräkningarna, men gör den datorritade bilden felaktig.



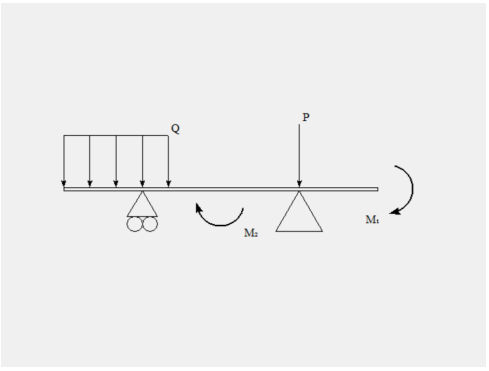
(a) Utdata från YOLO då alla moment slagits ihop i ”arrow_torque”.



(b) Bild genererad från (a). Notera att momentet till höger placerats felaktigt.



(c) Utdata från YOLO då momenten delas upp i ”moment_top_bottom” och ”moment_left_right”.



(d) Bild genererad från (c). Notera att placeringen av momentet till höger stämmer överens med (c).

Figur 31: Då alla moment behandlas som samma objekt placeras för bilden som visas i exemplet ovan det högra momentet ut felaktigt. Då momenten istället delas upp i två typer rättas felet till. Mellan de två iterationerna ändrades namnen på klasserna för punktlaster från ”arrow” till ”force” för att stämma bättre överens med övriga klassnamn, men de beskriver samma objekt.

6.1.2 Träningsdata

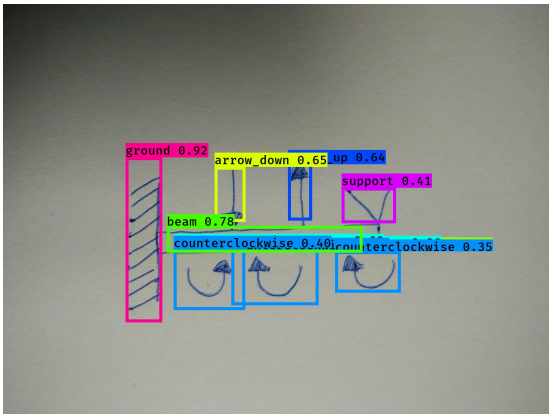
Bilderna som användes som träningsdata till modellen ritades för hand av sex olika personer. Modellen har alltså tränats på ett relativt smalt spektrum av handstilar, vilket skulle kunna bli ett problem om användarens handstil skiljer sig från träningsdatan. För att kontrollera att detta inte hade en betydande påverkan gjordes ett flertal tester på bilder ritade av en utomstående person. Ett exempel på detta är bilden som används för att presentera resultatet i avsnitt 5 (se Figur 21). Alla objekt identifierades och ritades upp korrekt, i de flesta fall med konfidens över 90 %. Det smala spektrumet av handstilar i träningsdatan kan därför anses ha fungerat bra, även om träningsdata med ytterligare handstilar förmodligen kunnat förbättra resultatet.

Insamling av träningsdata var ett tidskrävande arbete och för att undvika att behöva göra detta mer än nödvändigt går det att använda program som genererar mer data utifrån den befintliga datan. Med ett program som manipulerar bilderna genom att exempelvis

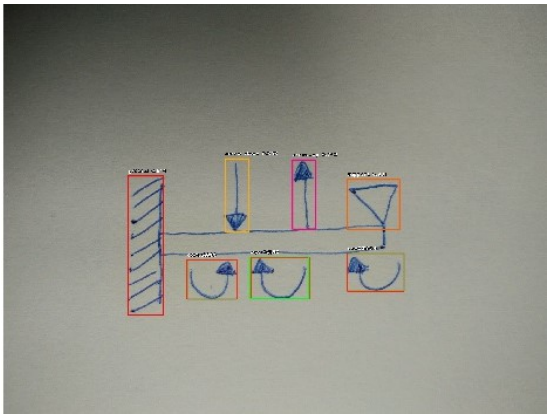
vrida eller spegla dem och samtidigt flyttar med markeringarna utökas datamängden avsevärt. Viktigt att tänka på är att inte förändra datan så mycket att symbolerna får en ny innebörd, exempelvis om en pil roteras för mycket och byter riktning.

6.1.3 RetinaNet vs YOLO

Som nämndes i avsnitt 4.1.1 gjordes även vissa försök att använda RetinaNet istället för YOLO. Träningen med RetinaNet tog förhållandevis lång tid och YOLO-träningen, som pågick parallellt, gav betydligt bättre resultat (se Figur 32 för exempel). Därför gjordes ingen noggrann jämförelse mellan de båda modellerna och utforskandet av RetinaNet avslutades. En rigorös jämförelse hade krävt fler träningar med de båda modellerna med bland annat samma batch size och antal epoker. Dessutom är kostnadsfunktionerna för de båda modellerna inte tydligt jämförbara vilket hade komplicerat jämförelsen ytterligare.



(a) Resultat från YOLO efter fyra epoker.



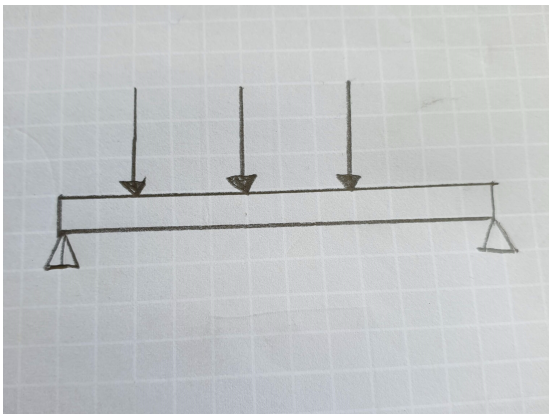
(b) Resultat från RetinaNet efter 16 epoker. Texten tillhörande objektmarkeringarna för RetinaNet är otydlig, men markeringarna förklaras närmare nedan.

Figur 32: Figurerna visar hur försöken med både YOLO och RetinaNet markerade de flesta komponenterna, med undantag för RetinaNet som ej lyckades detektera balken. Detta visade sig vara ett återkommande problem för RetinaNet. Momentpilarna var dubbelt markerade som medurs och moturs i både försöket med YOLO och RetinaNet.

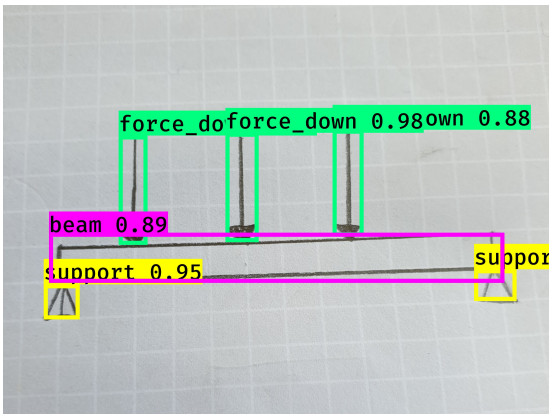
Då det inte låg inom projektets mål att utvärdera prestandan hos de olika metoderna för objektdetektering prioriterades inte detta. Eftersom ingen grundlig jämförelse gjordes bör därför alternativet att i framtiden använda RetinaNet för liknande projekt inte uteslutas.

6.2 Bilduppritning

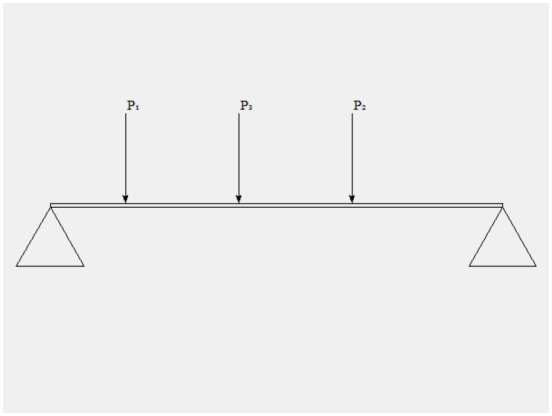
Bilduppritningsprogrammet kan i stora drag anses fungera som väntat, men det finns även vissa begränsningar. Till exempel kan de olika objekten endast placeras på vissa diskreta punkter på en balk enligt vad som beskrivs i avsnitt 4.2. Detta är en medveten begränsning som skapades för att komponenterna, i största möjliga grad, skulle placeras ut där de är avsedda att vara, vilket illustreras i Figur 33. Programmet är dock uppbyggt så att antalet angreppspunkter kan ökas om det i framtiden tas fram ett program som noggrannare kan lokalisera de olika komponenterna.



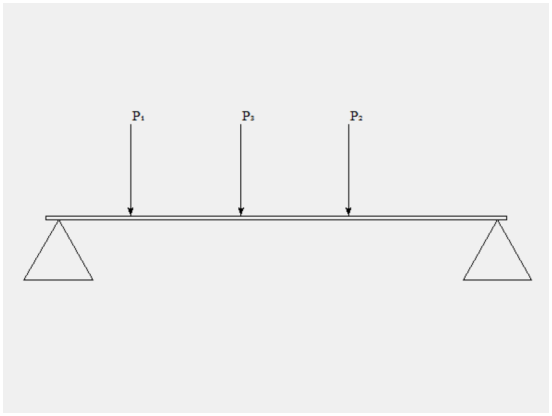
(a) Exempelbild för att påvisa inverkan av antalet element balken delas upp i.



(b) Utdata då YOLO appliceras på (a). Notera att mittpunkten på objektmarkeringarna för de båda stöden ligger en liten bit in från änden på balkens objektmarkering.



(c) Tolkning av bilden ovan med balken indelad i 12 element.



(d) Tolkning av bilden ovan med balken indelad i 999 element.

Figur 33: I (a) framgår det att användaren avsåg att rita de båda stöden på balkens ändar. Då balken delas in i 12 element korrigeras stödens koordinater så att de ritas upp längst ut på balken, se (c). När balken istället delas in i 999 element placeras stöden en bit in på balken, se (d).

Enligt vad som beskrivits i avsnitt 6.1.1 görs ingen skillnad i objektdetekteringen mellan pilar åt höger och vänster, respektive mellan momentpilar med- och motsols. För att kompensera för denna begränsning och vissa fel i objektdetekteringen implementerades vissa funktioner. Riktningen på alla horisontella pilar sätts till att peka mot balken eftersom det ansågs vara vanligare med tryckkrafter än dragkrafter inom grundläggande balkproblem. Moment sätts med riktning medurs på en balks högra sida och moturs på dess vänstra sida eftersom även detta ansågs vanligare. På balkens under- respektive ovansida sätts ett moments riktning till med- respektive motsols. Vidare ges möjligheten att vända riktningen på alla laster och moment genom att sätta negativ magnitud. Detta gör det dels möjligt att kompensera för att horisontella pilars och momentpilars riktningar inte bestäms vid objektdetekteringen, men löser också problemet om vertikala laster detekteras med motsatt riktning.

6.3 Vidareutveckling

Under projektets gång har stora möjligheter att förbättra och vidareutveckla programmet uppdagats.

6.3.1 Förbättringar och utökningar av programmet

Genom att samla in mer data från en större bredd av personer och testa mer utvecklade detektionsmodeller skulle objektdetekteringsmodulens användbarhet kunna utökas. Om den skulle kunna skilja på riktningarna hos de horisontella pilarna och momentpilarna skulle programmet bli smidigare att använda, då användaren inte hade behövt ändra riktningarna på objekten. Om noggrannheten i objektdetekteringen ökade skulle även mer komplicerade hållfasthetsproblem bli möjliga att lösa med godtagbar precision. Vidare kan fler symboler introduceras för att möjliggöra identifiering av exempelvis fackverk och problem med fjäderinfästningar, vilket skulle leda till ett bredare användningsområde för programmet.

Ett ytterligare sätt att bredda programmets användningsområde är att införa pilar i andra vinklar än horisontella och vertikala. Detta skulle antingen kunna implementeras som en del av objektdetekteringen eller som en extra inställning i bilduppritningsprogrammets användargränssnitt.

En möjlig vidareutveckling av programmet är att inkludera möjligheten att ange storleken på moment och laster direkt i bilden genom att skriva ut storlekarna bredvid respektive objekt. För att implementera detta skulle högre krav ställas på bildritningsprogrammet, där rätt siffror skulle behöva kopplas ihop med rätt objekt.

För att göra programmet mindre känsligt för fel i objektdetekteringen skulle funktioner för att lägga till, flytta och ta bort komponenter även kunna skapas. Med fler funktioner av denna typ spelar visserligen objektdetekteringen en mindre roll och det kan då diskuteras om syftet med projektet frångås. En fördel med dessa typer av funktioner skulle dock vara att möjligheten ges att experimentera med hur förflyttandet av olika komponenter påverkar de beräknade storheterna, utan att behöva rita om bilden för hand.

6.3.2 Maskininlärning och hållfasthetslära

I det här projektet har FEM använts för att beräkna de olika snittkrafterna i balken. För att räkna på balkar är det dock vanligt att istället göra en analytisk lösning eftersom balken kan beskrivas med en ordinär differentialekvation (ODE). Genom att ställa upp en differentialekvation som beskriver balken behöver den dock inte lösas analytiskt utan det går faktiskt att lösa den med ett neuralt nätverk.[28] Det är alltså möjligt att använda maskininlärningsalgoritmer till att även utföra beräkningarna på balken. För att beskriva processen översiktligt sätts ett nätverk som representerar lösningen upp, det vill säga den sökta ekvationen. Kostnadsfunktionen bygger sedan på differentialekvationen, randvillkoren och begynnelsevillkoren.

Ytterligare funktioner som skulle kunna introducera med hjälp av objektdetektering är

beräkningen av yttröghetsmoment och masscentrum för balkens tvärsnitt. Genom att låta ett NN analysera en bild på tvärsnittet skulle programmet kunna räkna ut dessa parametrar.

Ett annat område för vidareutveckling inom hållfasthetsberäkningarna är knäckningsanalys. Genom att implementera funktioner för beräkning av knäckning i beräkningsprogrammet kan exempelvis kritiska laster räknas ut. Framtida förbättringar av projektet kan därför implementera knäckningsanalys och därmed göra programmet mer komplett.

6.3.3 Implementerande av mobilapplikation

För att göra programmet mer tillgängligt och användbart skulle det i framtiden vara möjligt att implementera programmet som en mobilapplikation. En sådan applikation skulle kunna innebära att användaren endast behöver hålla kameran över en handritad skiss för att tillhandahållas den datorritade bilden samt diagram över snittkrafter och -moment. Med förbättringarna som beskrevs i avsnitt 6.3.1 skull en mobilapplikation av denna typ kunna bli ett mycket användbart hjälpmedel vid studier inom hållfasthetslära. Ett eventuellt problem skulle kunna vara att programmet tar en hel del utrymme. De tränade vikterna tar till exempel 236 MB minne, vilket kan vara problematiskt vid implementation av en applikation.

Referenser

[1] C. Buckley och P. Mozur. (maj 2019). How China Uses High-Tech Surveillance to Subdue Minorities, URL: https://www.nytimes.com/2019/05/22/world/asia/china-surveillance-xinjiang.html.

[2] A. Qin. (jan. 2020). Chinese City Uses Facial Recognition to Shame Pajama Wearers, URL: https://www.nytimes.com/2020/01/21/business/china-pajamas-facial-recognition.html.

[3] S. Grigorescu, B. Trasnea, T. Cocias och G. Macesanu. A Survey of Deep Learning Techniques for Autonomous Driving, URL: https://arxiv.org/abs/1910.07738 (hämtad 13 maj 2020).

[4] M. Elliott. (okt. 2014). Use your phone's camera to solve equations with PhotoMath, URL: https://www.cnet.com/news/use-your-phones-camera-to-solve-equations-with-photomath/.

[5] J. C. Redmon. YOLO: Real-Time Object Detection, URL: https://pjreddie.com/darknet/yolo/ (hämtad 5 maj 2020).

[6] AntonMu. TrainYourOwnYOLO, URL: https://github.com/AntonMu/TrainYourOwnYOLO.

[7] Tensorflow. Tensorflow, URL: https://www.tensorflow.org/ (hämtad 13 maj 2020).

[8] Keras. Keras, URL: https://keras.io/ (hämtad 13 maj 2020).

[9] Python. (sept. 2018). The Python Wiki, URL: https://wiki.python.org/moin/.

[10] E. A. B. Planche, *Hands-On Computer Vision with TensorFlow 2*. Packt Publishing, 2019, ISBN: 978-1-78883-064-5.

[11] O'Reilly. Convolutional neural networks, URL: https://www.oreilly.com/library/view/learn-unity-ml-agents/9781789138139/16671cc8-0aff-433a-878c-7430be8b9aa1.xhtml (hämtad 13 maj 2020).

[12] R. Girshick, J. Donahue, T. Darrell och J. Malik. (okt. 2014). Rich feature hierarchies for accurate object detection and semantic segmentation, URL: https://arxiv.org/pdf/1311.2524 (hämtad 13 maj 2020).

[13] R. Gandhi. Support Vector Machine, URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (hämtad 14 maj 2020).

[14] J. Redmon, S. Divvala, R. Girshick och A. Farhadi. (maj 2016). Rich feature hierarchies for accurate object detection and semantic segmentation, URL: https://arxiv.org/pdf/1506.02640 (hämtad 13 maj 2020).

[15] Esri. How RetinaNet works, URL: https://developers.arcgis.com/python/guide/how-retinanet-works/ (hämtad 11 april 2020).

[16] P. Jay. (mars 2018). The intuition behind RetinaNet, URL: https://medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d (hämtad 25 april 2020).

[17] M. Han. (mars 2018). The YOLOv3 Object Detection Network Is Fast!, URL: https://syncedreview.com/018/03/27/the-yolov3-object-detection-network-is-fast (hämtad 25 april 2020).

[18] J. Hui. (mars 2018). What do we learn from single shot object detectors (SSD, YOLOv3), FPN Focal loss (RetinaNet)?, URL: https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d (hämtad 25 april 2020).

[19] S.-H. Tsang. Review: RetinaNet - Focal Loss (Object Detection), URL: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4 (hämtad 7 maj 2020).

[20] N. Ottosen, *Introduction to the Finite Element Method*. Pearson, 1992, ISBN: 9780134738772.

[21] M. Enelund. Course in solid mechanics, URL: https://student.portal.chalmers.se/sv/chalmersstudier/programinformation/sidor/sokprogramutbudet.aspx?course_id=23581&parsergrp=2 (hämtad 14 maj 2020).

[22] C3SE. C3SE - Chalmers Center for Computational Science and Engineering, URL: https://www.c3se.chalmers.se/ (hämtad 29 april 2020).

[23] Python. (dec. 2019). TkInter, URL: https://wiki.python.org/moin/TkInter (hämtad 10 mars 2020).

[24] A. Ottosson. CALFEM for Python, URL: http://www.byggmek.lth.se/fileadmin/byggnadsmekanik/publications/tvsm5000/web5167.pdf (hämtad 14 maj 2020).

[25] A. Al-Masri. What are overfitting, URL: https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690 (hämtad 14 maj 2020).

[26] Microsoft. VoTT, URL: https://github.com/microsoft/VoTT.

[27] A. Oliv. Projektets GitHub, URL: https://github.com/Olivolja/mlHollf (hämtad 14 maj 2020).

[28] D. Ferreira. How to solve ODE with a neural network, URL: https://towardsdatascience.com/how-to-solve-an-ode-with-a-neural-network-917d11918932 (hämtad 14 maj 2020).

[29] SNIC. SUPR - SNIC User and Project Repository, URL: https://supr.snic.se/ (hämtad 29 april 2020).

[30] C. IT-Office. VPN - remote network access, URL: https://it.portal.chalmers.se/itportal/NonCDAWindows/VPNGeneral (hämtad 29 april 2020).

[31] T. Ylonen. SSH - Secure Login Connections over the Internet., URL: https://www.ssh.com/ssh/protocol/ (hämtad 29 april 2020).

[32] PuTTY. PuTTY, URL: https://www.putty.org/ (hämtad 29 april 2020).

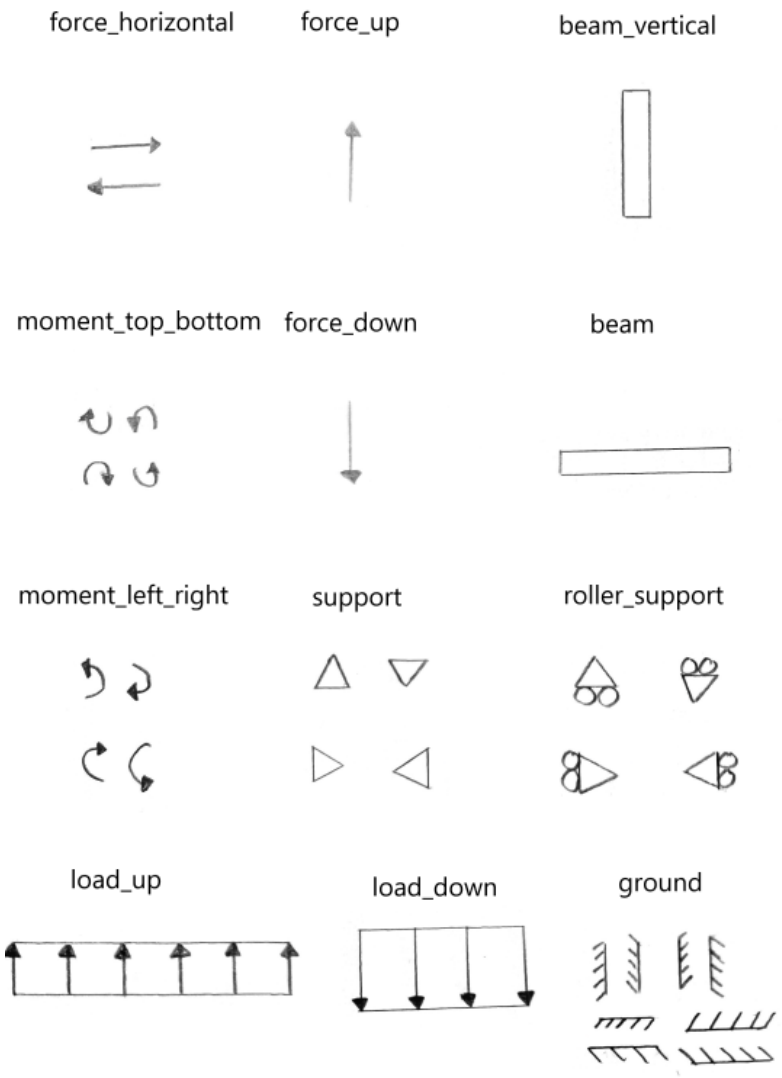
[33] M. Wilson. (dec. 2019). SCP – What is Secure Copy Protocol – Definition Example, URL: https://www.pcwdld.com/what-is-scp (hämtad 29 april 2020).

[34] WinSCP. WinSCP, URL: https://winscp.net/ (hämtad 29 april 2020).

[35] C3SE. Getting started, URL: https://www.c3se.chalmers.se/documentation/getting_started/ (hämtad 7 maj 2020).

Bilagor

A Klasser för objektdetektering



Figur 34: Ovan syns de olika klasser av objekt som algoritmen tränats på. Här ses till exempel att ”moment_top_bottom” innefattar både med- och moturs moment som skulle kunna placeras på ovan- eller undersidan av en balk.

B Träning på C3SEs datorkluster

För att påskynda träningsprocessen användes datorklustret hos Chalmers Centre for Computational Science and Engineering (C3SE) [22]. Detta minskade tidsåtgången från 40 timmar på persondator till 10 timmar på klustret. I denna bilaga redogörs hur detta möjliggjordes.

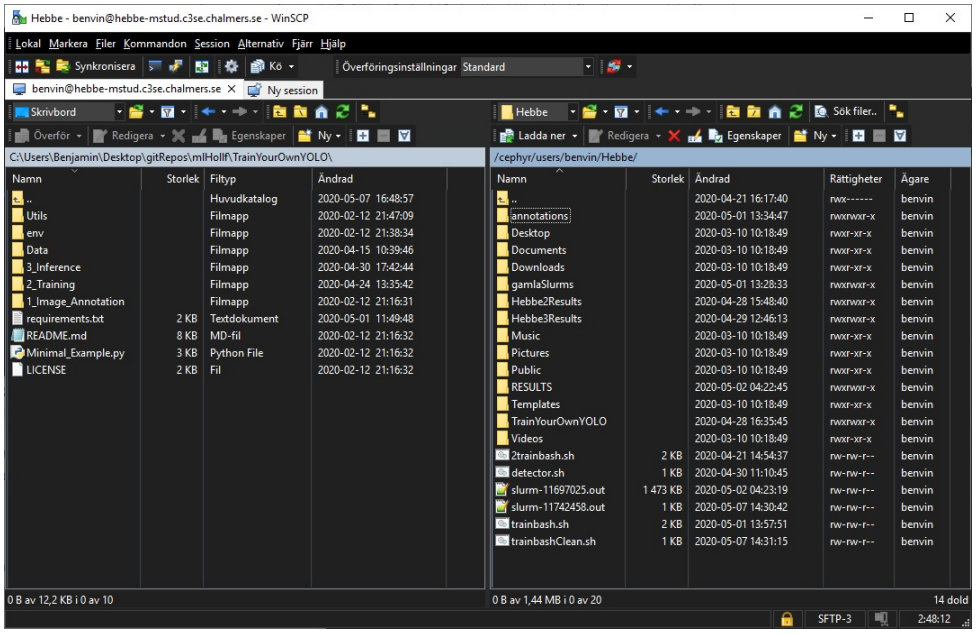
B.1 Att få tillgång

För att få tillgång till klustret krävs ett projekt och medlemskap på SNIC [29]. Där kan man monitorera sina allokerade resurser, som mäts i core-timmar.

För att nå klustret ansluter man till dess loginnod. Noden kan endast nås genom Chalmers nätverk, vilket finns tillgängligt i Chalmers anläggningar alternativt genom Chalmers VPN [30]. Anslutningen upprättas genom Secure Shell (SSH), vilket är ett protokoll för att ansluta sig säkert mot andra datorer över internet [31]. I projektet användes programvaran PuTTY [32] för SSH-anslutningen till noden.

B.2 Filhantering

När anslutningen var upprättad användes Secure Copy Protocol (SCP) [33], vilket är ett protokoll för säker överföring av filer mellan datorer över internet, för att kopiera över filer som skulle användas till loginnoden. Mer specifikt användes programvaran WinSCP [34], i vilken man på samma sätt som PuTTY använder sig av SSH för att få tillgång till loginnoden. Efter anslutning visar WinSCP upp ett tvådelat fönster, med ditt egna filsystem till vänster och loginnodens filsystem till höger, se Figur 35.



Figur 35: Vyn i WinSCP när anslutning upprättats till loginnoden. Filer kan nu flyttas mellan de olika filsystemen genom att markera dem och dra dem över till andra sidan med muspekaren.

B.3 Jobbskript

På loginnoden kan terminalen användas precis som vanligt genom PuTTY. Träningen körs dock inte på loginnoden, utan på datorklustret, där terminalen inte kan användas. Istället skickar man in ett ”jobbskript”. Jobbskriptet är ett bashskript, vilket består av kommandon som sekventiellt exekveras precis som de skulle göras i terminalen. Hela skriptet som användes i projektet återfinns kommenterat i Bilaga C. När jobbskriptet är färdig skickas det in till datorklustret genom kommandot *sbatch <jobbskriptetsNamn>* i terminalen. Ett jobbskript med namnet jimsSkript skickas alltså in genom kommandot *sbatch jimsSkript*.

För hjälp angående skrivande av jobbskript eller användning av klustret generellt, se C3SEs guide på deras hemsida [35].

C Jobbskript

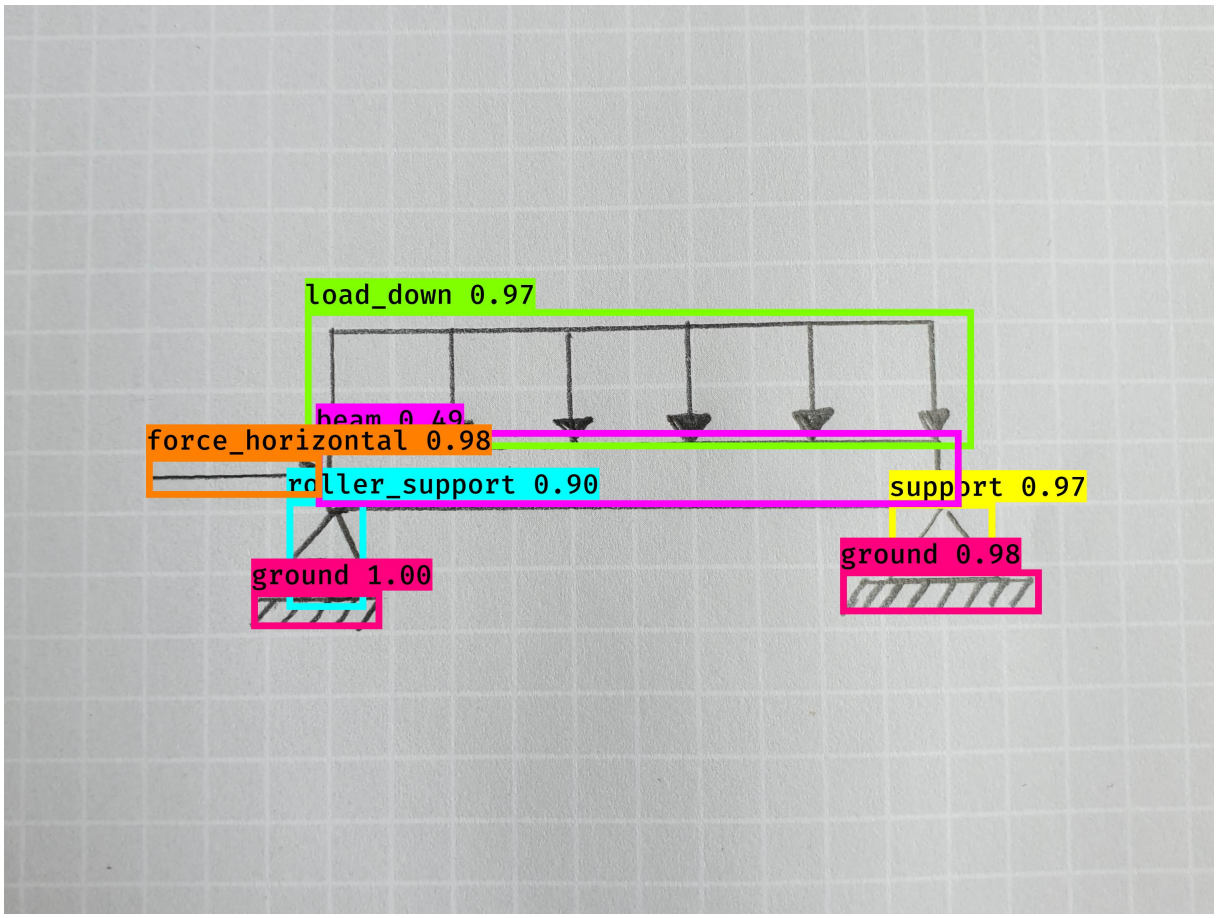
Kodstycke 1: Kommenterad version av jobbskriptet som användes i projektet. Röd text är klusterspecifika kommandon för att sätta parametrar kring utförandet av jobbet, blå text är vanliga unixkommandon och grå text är kommentarer. Den grå texten är alltså inte del av skriptet utan endast där för att förklara innebörden av alla kommandon.

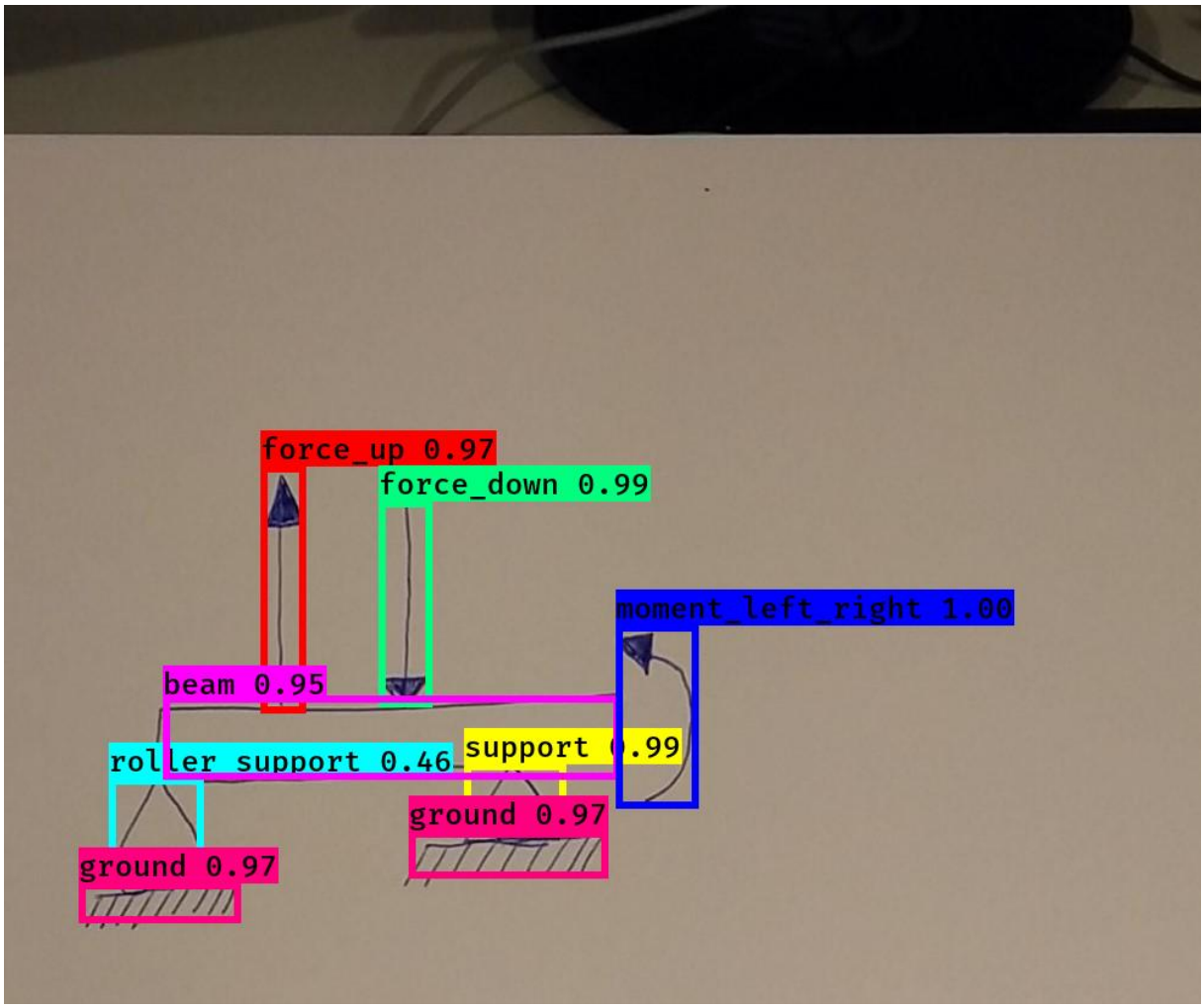
```
#!/usr/bin/env bash # Kallas shebang - markerar att detta är ett
    bashskript
#SBATCH -A C3SE2020-2-6 # Anger vilket projekt hos C3SE
   jobbskriptet tillhör
#SBATCH -p mstud # Anger vilken partition av klustrets som
   jobbet ska köras på, i detta fall mstud
#SBATCH -t 1-23:59:00 # Anger maximal tidsåtgång för jobbet, kör
    jobbet så länge så avbryts det
#SBATCH -n 20 # Anger antal noder som ska användas till jobbet
#SBATCH -mail-user=benvin@student.chalmers.se -mail-type=end # Anger
   att ett mail ska skickas när jobbet är slut

echo "Job starting!" # Skriver ut Job starting! i loggen

cp -r TrainYourOwnYOLO $TMPDIR # Kopierar mappen
   TrainYourOwnYOLO till klustrets lagring
cd $TMPDIR # Byter mapp till klustrets root-mapp

module load intel # Laddar in modulen intel, används för att
   kunna nå Python-modulen
module load Python # Laddar in Python
virtualenv jimsenv # Skapar en "virtual environment" vid namn
    jimsenv
source jimsenv/bin/activate # Aktiverar virtual environmenten

cd TrainYourOwnYOLO/ # Byter till mappen TrainYourOwnYOLO
pip install -r requirements.txt # Installerar alla nödvändiga
    Python-paket genom paketinstalleraren pip

cd 2_Training/ # Byter till mappen 2_Training
python3 Train_YOLO.py # Kör scriptet Train_YOLO.py, här sker
   träningen av nätverket

cd ../3_Inference # Byter mapp till mappen 3_Inference
python3 Detector.py # Kör scriptet Detector.py, här testas nä
   tverket på att känna igen hållfasthetssymboler i en mängd
   bilder

cd ../Data # Byter till mappen ../Data
cp -r Model_Weights $SLURM_SUBMIT_DIR/RESULTS # Kopierar över
```

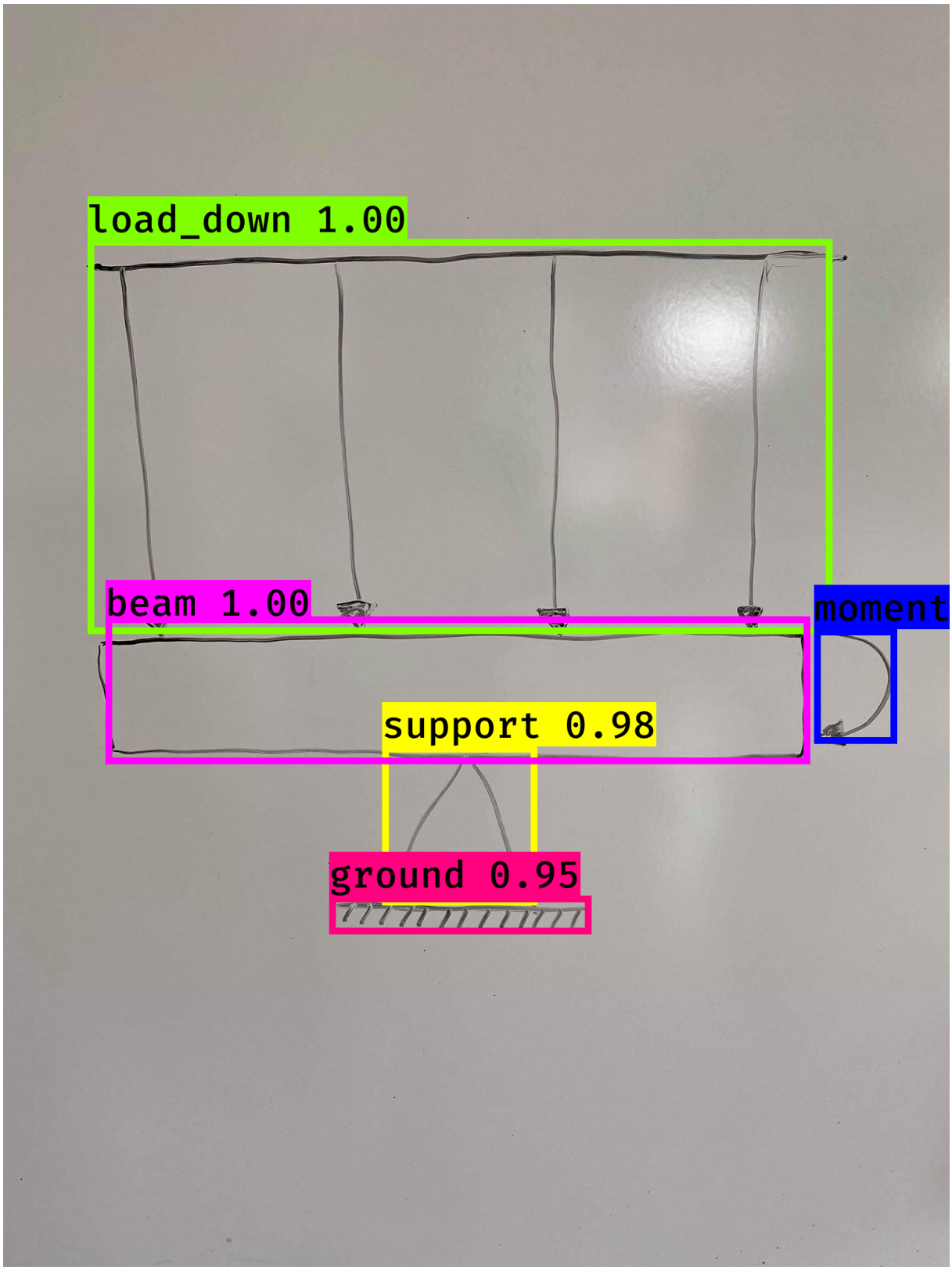
```
    vikterna från det tränade nätverket till loginnodens
   lagring

cd Source_Images # Byter till mappen Source_Images
cp -r Test_Image_Detection_Results $SLURM_SUBMIT_DIR/RESULTS
   # Kopierar över resultatet av objektdetekteringstestet
   till loginnodens lagring

echo "Job ending!" # Skriver ut Job ending! i loggen
```

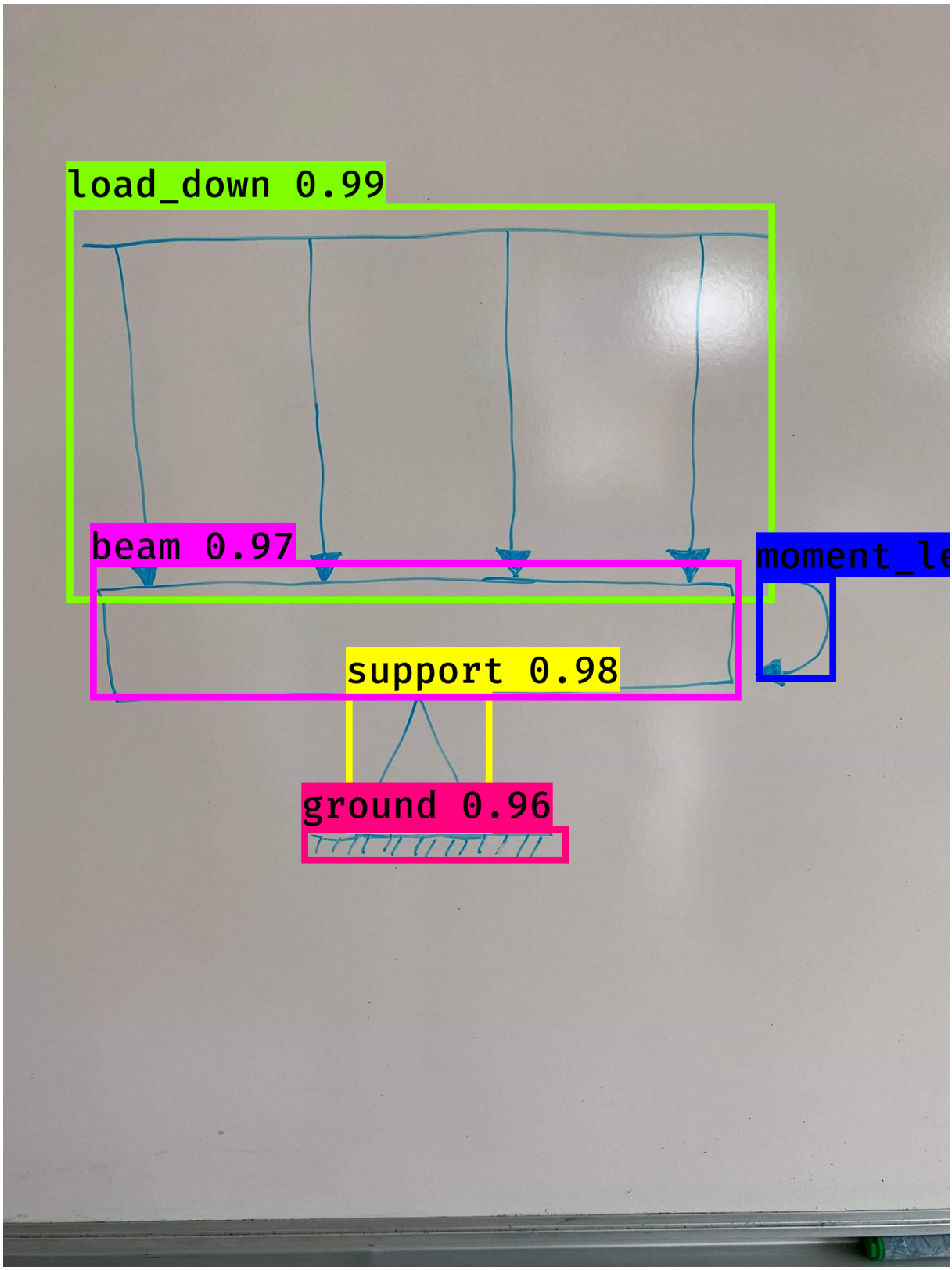
D Urval av resultat från objektdetektering

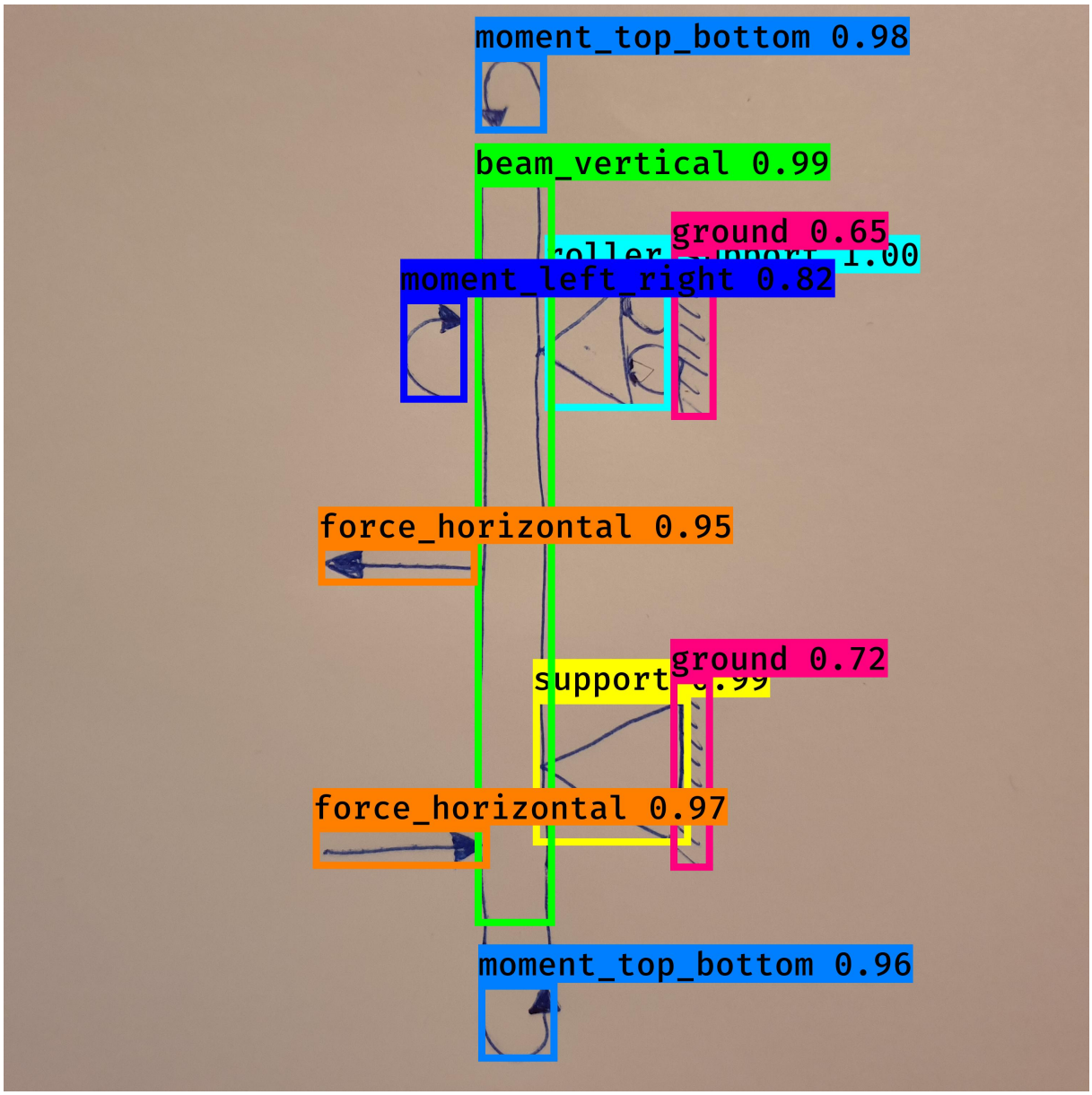
Här presenteras ett urval från testningen av objektdetekteringsmodellen. Bilderna är ritade av fyra olika personer, med olika slags pennor, på olika underlag, och bilderna är tagna med olika upplösningar och i olika ljus.

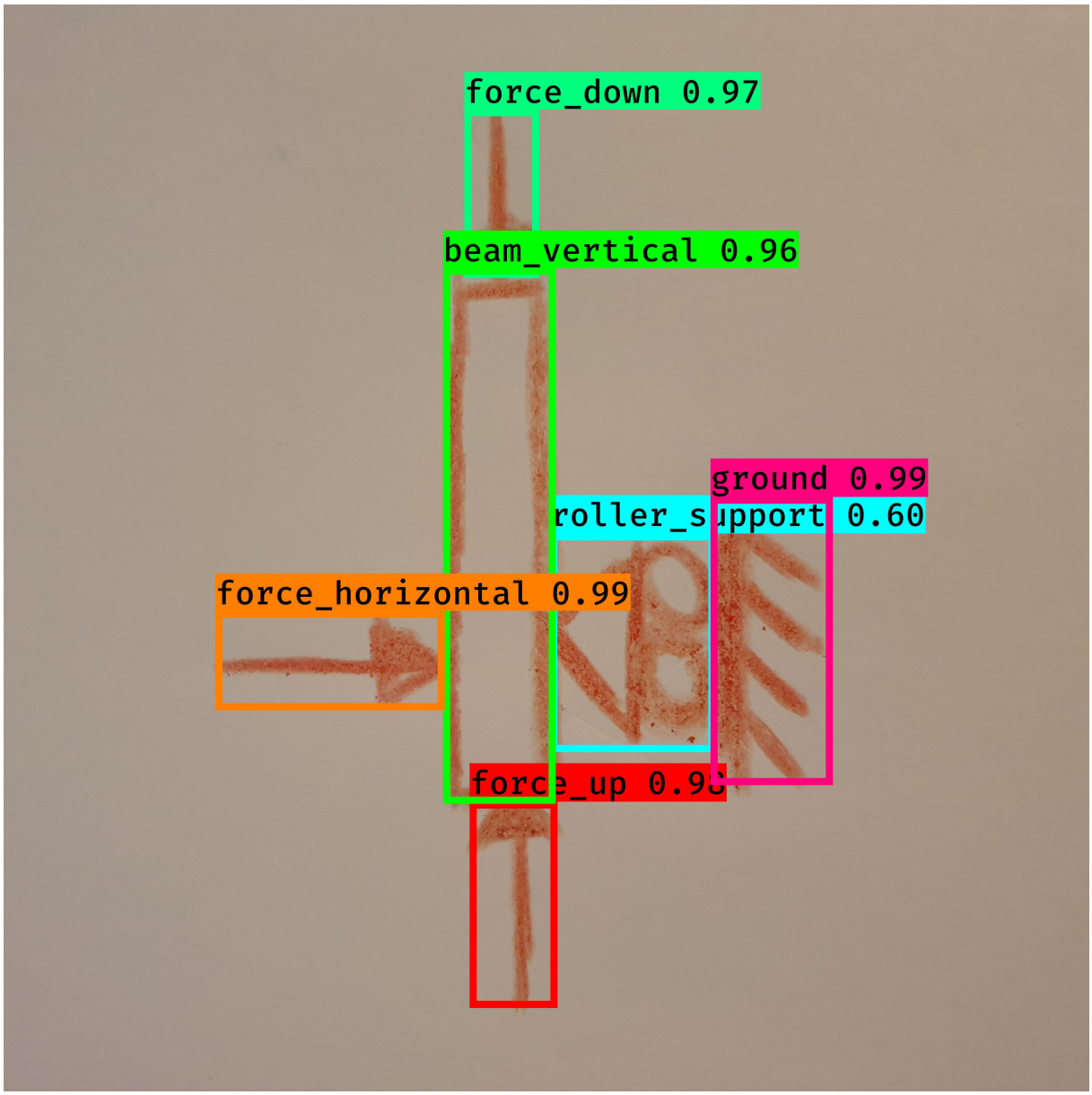



force_up 0.97
force_down 0.99
moment_left_right 1.00
beam 0.95
roller support 0.46
support
ground 0.97
ground 0.97


load_down 1.00
beam 1.00
moment
support 0.98
ground 0.95


load_down 0.99
beam 0.97
moment_le
support 0.98
ground 0.96


moment_top_bottom 0.98
beam_vertical 0.99
ground 0.65
moment_left_right 0.82
force_horizontal 0.95
ground 0.72
force_horizontal 0.97
moment_top_bottom 0.96


force_down 0.97
beam_vertical 0.96
ground 0.99
roller_support 0.60
force_horizontal 0.99
force_up 0.98