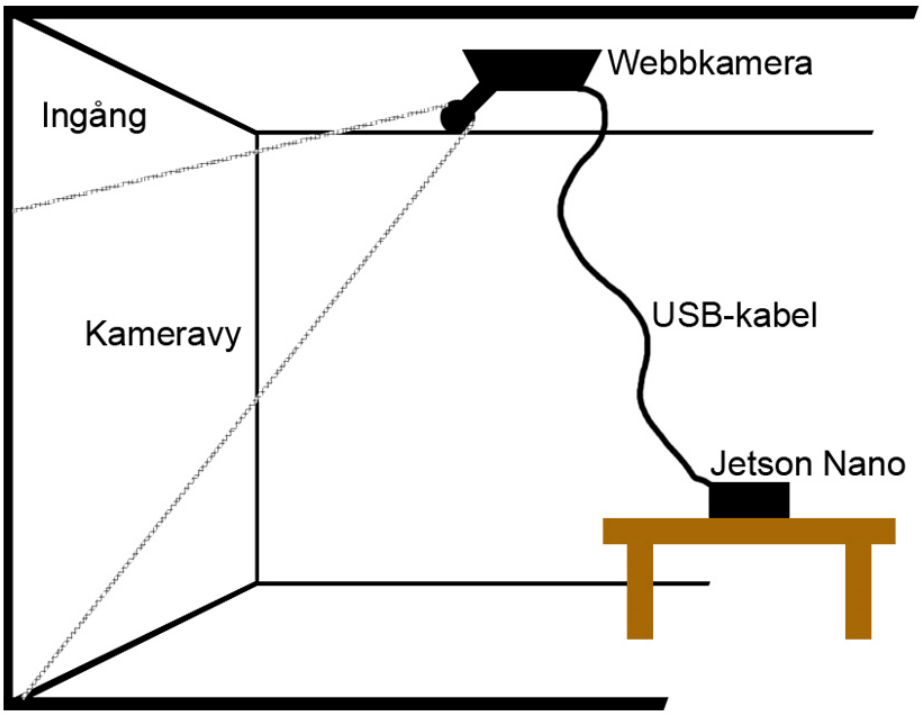


Realtids människoräknare

Examensarbete inom data- och informationsteknik

ANTHON LENANDER
KONRAD REJ

Institutionen för data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA & GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022

Examensarbete 2022

Realtids människoräknare

Anthon Lenander
Konrad Rej

CHALMERS

Institutionen för data- och informationsteknik
Chalmers tekniska högskola & Göteborgs Universitet
Göteborg, Sverige 2022

Realtids människoräknare
Anthon Lenander, Konrad Rej

Handledare: Jonas Duregård, Institutionen för data- och informationsteknik
Examinator: Lars Svensson, Institutionen för data- och informationsteknik

Examensarbete 2022
Institutionen för data- och informationsteknik
Chalmers tekniska högskola & Göteborgs universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: Visualisering av hur datainsamlingssystemet installerades.

Skriven i LaTeX
Utskriven av Chalmers Reproservice
Göteborg, Sverige 2022

Realtids människoräknare
Examensarbete inom data- och informationsteknik
ANTHON LENANDER, KONRAD REJ
Institutionen för data- och informationsteknik
Chalmers tekniska högskola & Göteborgs universitet

Abstract

The Covid-19 pandemic forced many employees from around the world to exchange their ordinary workplaces with solutions at home. As more and more employees return to the office once again, many employers are looking for solutions to keep track of the number of office workers currently in office. The objective of this study is to develop a camera-based data collection system to monitor the amount of people entering and exiting an office. The people counting is done with the use of object detection, which is why different object detection algorithms are also explored and compared. The collected data is then presented on an associated website through a developed API.

Sammanfattning

Covid-19 pandemin tvingade många anställda runtom i världen att byta ut sina vanliga arbetsplatser mot olika hemmalösningar. I och med att fler anställda nu återvänder till kontoret letar därför många arbetsgivare efter lösningar för att hålla koll på antalet anställda som befinner sig på kontoret. Målet med den här studien är därför att utveckla ett kamera-baserat datainsamlingssystem för att övervaka antalet människor som går in och ut ur ett kontor. Människoräkningen görs med hjälp av objektdetektion, vilket är varför olika algoritmer för objektdetektion utforskas och jämförs. Insamlad data visas sedan upp på en tillhörande hemsida med hjälp av en utvecklad API.

Nyckelord: AI, Objektdetektering, Faster R-CNN, MobileNet-SSD, YOLOv5, Amazon Web Services.

Förord

Vi skulle vilja tacka företaget Knightec för att de givit oss möjligheten att skriva vårt examensarbete hos dem. Tack till Anna Nordlander som visade oss runt i kontoret och hjälpte oss med nyckel-taggar. Ett speciellt tack går till Alex Darborg och Terje Engelbertsen som givit oss hjälp genom examensarbetets gång i form av frekvent feedback, idéer och läsning av rapporten. Vi vill även tacka Joakim Brandt som varit snäll och korrekturläst rapporten.

Ett stort tack går även till vår handledare Jonas Duregård på Chalmers Tekniska Högskola, som givit kontinuerlig feedback gällande rapporten och alltid varit glad över att svara på frågor.

Anthon Lenanader, Konrad Rej, Göteborg, Juni 2022

Förkortningar

Nedan följer en lista av förkortningar som har använts genomgående i rapporten, listade i alfabetisk ordning:

AI	Artificiell Intelligens
API	Application Programming Interface
AWS	Amazon Web Services
CNN	Convolutional Neural Network
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
FPS	Frames Per Second
GPU	Graphical Processing Unit
HTTP	Hypertext Transfer Protocol
OS	Operativ System
RAM	Random-Access Memory
SSD	Single Shot Detection

Innehåll

Figurlista

Tabellista

1
Introduktion

Detta kapitel kommer förklara bakgrunden till varför arbetet utförs samt dess syfte. Projektets mål och avgränsningar kommer också diskuteras här.

1.1 Inledning och bakgrund

Covid-19 pandemin har tvingat många kontorsarbetare världen över att byta ut sina vanliga arbetsplatser mot olika typer av hemmalösningar. Mer och mer anställda återvänder nu till sina vanliga arbetsplatser, och många arbetsgivare letar därför efter olika sätt att hålla koll på antalet anställda som befinner sig på sina kontor. Detta för att möjliggöra för anställda att hålla avstånd enligt restriktioner under pandemin. Efter pandemin kan ett sådant sätt även möjliggöra för anställda att se ifall det finns plats på kontoret i förtid.

1.2 Syfte

Syftet med examensarbetet är att undersöka och jämföra ett flertal AI baserade algoritmer utifrån deras prestanda och noggranhet. Utifrån detta ska ett kamerabaserat system för detektering av in- och utgående människotrafik utvecklas i syfte att kunna visa antal personer på kontoret. Datan kommer tillgängliggöras genom Amazon Web Services och en exempelhemsida kommer utvecklas.

1.3 Mål

Målet med examensarbetet är att utveckla ett sätt för arbetsgivare att hålla koll på antalet personer som befinner sig på sina kontor. Detta mål kan delas upp i följande delmål:

- Utveckla ett kamerabaserat system för mätning av antal personer som går in och ut ur ett kontor.
- Undersöka och jämföra prestanda av olika algoritmer för detektion av persontrafik.
- Tillgängliggöra insamlad data genom Amazon Web Services för att sedan undersöka hur lätt det är att integrera datainsamlingssystemet i webbapplikationen.

1.4 Avgränsningar

Ett beslut har tagits angående att ett datainsamlingssystem som involverar sensorer, både var för sig samt i kombination med en kamera ej kommer att undersökas. Systemets konstruktion kommer vara begränsat till att enbart använda sig av mikrodatorn Jetson Nano samt en webbkamera, då företaget begärde det.

2
Teknisk bakgrund

Följande kapitel har som syfte att förklara metoder och teknologier som kommer att användas under projekets gång.

2.1 Ubuntu - Ett Linux-baserat öppen källkod OS

Ubuntu är enligt utvecklarna [1] ett komplett Linux operativsystem som är byggt på vissa idéer. En är att att mjukvara skall finnas tillgänglig gratis, en annan att mjukvaruverktyg skall vara brukbara för människor i deras modersmål och trots funktionshinder. Utöver detta skall en kund även kunna anpassa produkten för eget bruk. Operativsystemet finns tillgängligt för bland andra ARMv8-processorn, som råkar vara samma processorarkitektur som mikrodatorn Jetson Nano använder sig av.

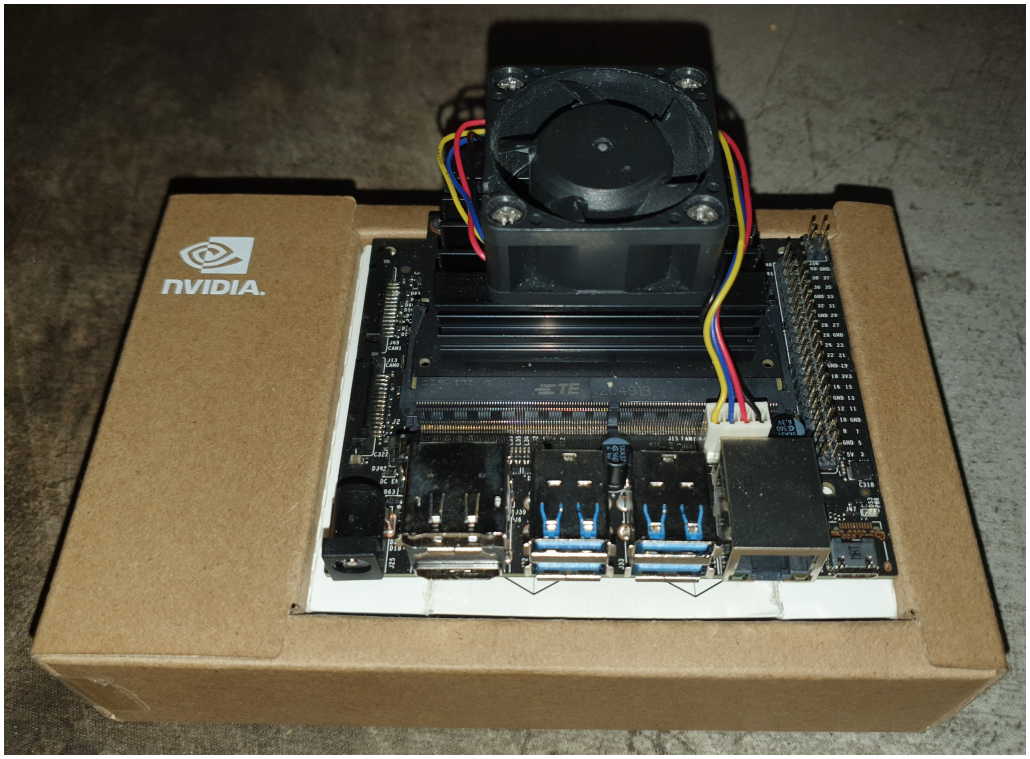
2.2 Jetson Nano - En mikrodator

Mikrodatorn Jetson Nano är enligt [2] en kompakt, men ändå kraftfull dator som tillåter körning av flera neurala nätverk parallellt vid användning i applikationer som bildigenkänning, detektion av objekt, segmentering och taligenkänning. För att strömförsörja mikrodatorn kopplas Jetson Nano in i vägguttag med hjälp av ett nätaggregat på 12V á 5A. I tabell 2.1 syns en mer noggrann teknisk specifikation utav mikrodatorn.

Teknisk specifikation	
Term	**Specifikation**
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Minne	4 GB 64-bit LPDDR4 25.6 GB/s
Lagring	microSD (ej inkluderad)
Anslutningar	Gigabit Ethernet & M.2 Key E
Display	HDMI & DisplayPort
USB	4x USB 3.0 & 1x USB 2.0 Micro-B

Tabell 2.1: Teknisk specifikation av mikrodatorn Jetson Nano

I tabell 2.1 kan ses att mikrodatorns processor är en Quad-core ARM A57 @ 1.43 GHz [2], vilken enligt [3] implementerar ARMv8-A arkitekturen. Detta möjliggör därför för installation av och mjukvaruutveckling på operativsystemet Ubuntu vid användning av mikrodatorn. Figur 2.1 nedan illustrerar mikrodatorn Jetson Nano.



Figur 2.1: Mikrodatorn Jetson Nano med tillhörande fläkt

2.3 OpenCV - Open Source Computer Vision

OpenCV är enligt [4] ett bibliotek med öppen källkod för *Computer Vision*, och innehåller över 2500 olika optimiserade algoritmer. Dessa algoritmer går att använda för uppgifter som till exempel detektering av objekt eller ansiktsigenkänning. Biblioteket har även stöd för programmeringsspråk som C++, Python, Java m.m.

2.4 Objektdetektering - Att hitta objekt i en bild

Tre algoritmer för objektdetektering används i projektet, vilka kommer att beskrivas. Till att börja med kommer *MobileNet-SSD* att diskuteras. Sedan kommer *YOLO* att utforskas. Därefter kommer även *Faster R-CNN* att tas upp.

2.4.1 MobileNet-SSD

Enligt [5] så är *MobileNet-SSD* en objektdetekteringsmodell som kombinerar två olika algoritmer för att detektera objekt. Objektdetekteringsmodellen använder sig

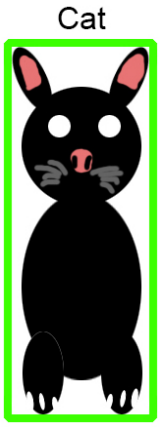
av både MobileNet och SSD för att dels lokalisera objekt i bild, men även klassifiera de lokaliserade objekten.

MobileNet är enligt [6] ett konvolutionerande neuralt nätverk som här fungerar som ryggrad i objektdetekteringsmodellen. Enligt [5] så används denna ryggrad för att extrahera så kallade *features* i form av en *feature map*, som SSD sedan kan använda sig av för att beräkna *Bounding Boxes*.

SSD, som är en förkortning för *Single Shot Detector*, består av 6 lager som enligt [5] och [7] räknar ut *Bounding Boxes* i minskande bildstorlek för varje lager. Dessa räknas ut med hjälp av den *feature map* som algoritmen fått av objektdetekteringsmodellens ryggrad MobileNet.

I och med att algoritmen använder sig av SSD istället för andra tekniker som tillexempel regionsförslag, som diskuteras i sektion 2.4.3, så kan algoritmen även användas på mobila enheter. Detta eftersom att körningstiden jämfört med till exempel regionsförslag är mycket bättre.

Algoritmen producerar en hel del *Bounding Boxes*, några av vilka kan tillhöra samma objekt. För att reducera antalet *Bounding Boxes* använder sig algoritmen av en metod som kallas för *Non-Maxima Suppression*, vars uppgift är att hitta *Bounding Boxes* som tillhör samma objekt, och slå ihop de till en *Bounding Box*. Begreppet *Bounding Box* översätts bäst till svenska som *minsta möjliga rektangel som kan omsluta ett objekt*. Exempel på en *Bounding Box* kan ses i figur 2.2 nedan.



Figur 2.2: Exempel på begreppet *Bounding Box*

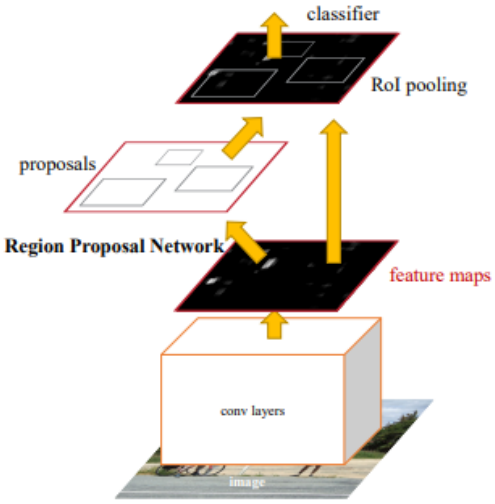
2.4.2 YOLO - You Only Look Once

YOLO, som står för *You Only Look Once*, är en objektdetekteringsalgoritm som bygger på fullt konvolutionerande neurala nätverk [8]. Sammanfattat kort så skickar man in en bild i nätverket, som skapar ett rutnät, och för varje cell i rutnätet försöker algoritmen gissa vad som finns där. Sedan returneras en lista av olika gissningar. Den största fördelen med algoritmen är att den är väldigt snabb jämfört med andra

algoritmer; i utvecklarnas tester så kördes *YOLO* algoritmen avsevärt snabbare än de flesta andra algoritmer som de testade [8]. Dessutom så utvecklade de även *Fast YOLO* algoritmen som mer än tredubblade hastigheten men tappade en del av noggrannheten från *YOLO*. Utöver detta så verkar algoritmen även förstå generella representationer av objekt, vilket betyder att algoritmen efter att ha blivit tränad på bilder från den riktiga världen fortfarande fungerar väl när den används på till exempel konstverk.

2.4.3 Faster R-CNN

Objektdetekteringsalgoritmen *Faster R-CNN* är enligt forskarna som skrev artikel [9] en snabbare och förbättrad version av algoritmen *Fast R-CNN* [10]. De beskriver att regionsförslag *(Region Proposals)* ofta blir en flaskhals beräkningsmässigt. Med regionsförslag menas alltså förslag från en algoritm kring olika regioner i en bild som kan innehålla objekt. I deras arbete har de istället utvecklat ett regionsförslagsnätverk, som kan ses i figur 2.3. Utvecklingen av detta nätverket har gjort att regionsförslagen nästintill blivit kostnadsfria beräkningsmässigt.



Figur 2.3: *Faster R-CNN's* uppbyggnad

Dessa regionsförslag används sedan i kombination med de *feature maps* som regionsförslagsnätverket fick av algoritmen, för att klassifiera objekten i bild. Denna klassificeringen görs med hjälp av algoritmen *Fast R-CNN's* [10] objektdetekteringsmodul. Vad som precis beskrivits kan även ses och kanske bättre förstås genom att titta på figur 2.3.

2.5 Objektspårning

Objektspårning innebär precis som det låter att spåra objekt mellan bilder. Detta kan göras genom att gå igenom en video bild för bild eller kontinuerligt läsa in bild från en kamera. En ideal objektspårningsalgoritm är enligt [11] en algoritm som enbart kräver objektdetektering en gång, som är snabb vad gäller körningstid och

som kan hantera när objekt hamnar ur bild. Den skall även vara robust mot ocklusion, det vill säga fungera bra även i fall där objekt i bild delvis är blockerade. Även objekt som dyker upp i bild igen efter att ha försvunnit skall kunna hanteras. Vad som precis beskrivits är dock bara beskrivningen av en idéel objektspårningsalgoritm. De flesta objektspårningsalgoritmerna är dessvärre mycket bra i avseende på några utav ovannämnda saker och mindre bra på andra. Nedan diskuteras två objektspårningsalgoritmer som kommer att användas i kombination.

2.5.1 KCF - Kernalized Correlation Filters

En forskningsgrupp från Linköpings Universitet beskriver i sin rapport [12] en metod för att kunna spåra objekt i en bild. Denna metoden har sedan implementerats i dlib som är ett Python och C++ bibliotek för bland annat bildbehandling [13]. Genom att ge algoritmen *bounding boxes*, går metoden sedan kortfattat ut på att försöka spåra objekten i boxarna algoritmen givits, bild för bild. Algoritmen lär sig ett *discriminative correlation filter* som den sedan använder sig av för att lokalisera objektet i framtida bilder [12].

2.5.2 Centroid Tracking

Objektspårningsalgoritmen mittpunktsspårning, kanske mer känd på engelska som *Centroid Tracking*, är enligt [11] en algoritm som ofta kombineras med en objektdetekteringsalgoritm. Algoritmen tar in *bounding boxes*, eller rektanglar av olika storlekar från en objektdetekteringsalgoritm för att sedan räkna ut mittpunkten på dessa rektanglar. Därefter räknar algoritmen ut det euklidiska avståndet mellan gamla och nya punkter. Detta görs för att kunna koppla samman punkter från en bild till en annan, med antagandet att om det euklidiska avståndet mellan två punkter är mindre än det euklidiska avståndet mellan alla andra punkter, så är det också samma objekt [11]. Eftersom en punkt på skärmen kan anses vara 2-dimensionell, så används här alltså det euklidiska avståndet i 2 dimensioner. Detta avstånd kan skrivas som:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2},$$

där p och q är två punkter i xy-planet.

2.6 Amazon Web Services - En del av molnet

Amazon Web Services (AWS) [14] är en molntjänstleverantör som erbjuder ett brett utbud av tjänster. Dessa tjänster inkluderar applikations-hosting, hemsidor, databaser, backups och lagring med mera. Dessa tjänster underlättar för andra företag genom att abstrahera bort infrastrukturen bakom dem. Dessutom ersätts den stora kapitalkostnaden som skulle krävas för att bygga ut ett företags egna infrastruktur med låga månadskostnader för endast de resurser som har använts [14].

2.6.1 API Gateway

API Gateway är en AWS tjänst som tillåter skapande och hantering av REST, HTTP eller WebSocket API:er [15]. API Gateway kan kopplas till andra AWS tjänster så som Lambda, DynamoDB med mera och utföra olika åtgärder i dessa. Det tillåter även tillgångshantering genom exempelvis API nycklar men även mer avancerade metoder som erbjuds genom andra AWS tjänster [15].

2.6.2 DynamoDB - En dynamisk databas

DynamoDB är en tjänst som erbjuds av AWS och erbjuder en snabb och pålitlig NoSQL databas [16]. DynamoDB tjänsten hanterar diverse funktioner så som lastbalansering, tillgänglighet, replikering med mera. Detta leder till att användarna behöver endast fokusera på datan som lagras vilket underlättar utveckling av programvara som använder databasen [16].

2.6.3 Lambda - Serveroberoende funktioner

AWS Lambda är en så kallad *function as a service* (FaaS) tjänst. FaaS är en del av ett bredare koncept kallat *serverless* som i grunden innebär att man kör server baserad kod på molnservrar istället för att hantera sina egna servrar [17]. Detta innebär att Lambda tillåter användaren att skapa individuella funktioner som körs på AWS servrar. Lambda funktioner drivs av event som kan komma från en stor mängd AWS tjänster, exempelvis så kan Lambda kopplas till S3 så att en funktion körs när en ny fil laddas upp [18].

2.6.4 S3 - Molnbaserad datalagring

Amazon S3 är en molntjänst som erbjuder en mängd olika typer av objekt lagring [19]. Dessa lagringstyper varierar i hastighet samt kostnad för lagring, hämtning och uppladdning. Utöver detta så erbjuder S3 tjänster gällande datahantering så som replikering, objektlåsning, tillgångshantering, databehandling med mera som leder till att S3 är en väldigt mångsidig lagringsplattform [20].

2.7 Serverless Framework

Serverless Framework är ett ramverk utvecklat av Serverless Inc organisationen som är till för att förenkla byggandet av tjänster som använder sig av *serverless* konceptet [21]. Serverless Framework har stöd för en stor mängd tjänster som erbjuds av diverse molntjänstleverantörer såsom Amazon Web Services, Microsoft Azure, Google Cloud Platform med flera [22]. Ramverket stödjer ett flertal språk och verktyg som exempelvis Node.js, Python och Go. Utöver de tjänsterna som officiellt stöds så kan ramverket få stöd för ny funktionalitet genom tillägg [21].

3
Metod

I följande kapitel förklaras de olika delarna av arbetet som kommer att utföras för att nå målen från kapitel 1.3.

3.1 Utveckling av datainsamlingssystemet

För att kunna evaluera de olika objektdetekteringsalgoritmerna kommer en grund för datainsamlingssystemet att utvecklas som sedan kan byggas vidare på. Efter begäran från företaget kommer en Jetson Nano mikrodator att användas för att köra programvaran. Den utvecklade programkoden kommer köras i operativsystemet Ubuntu. Insamlingssystemet kommer behöva en datakälla; för detta kommer en USB-baserad webbkamera att anslutas till systemet. Detta kommer göra det möjligt för programkoden att nå webbkameran och hämta videodatan som kommer användas i objektdetekteringsalgoritmerna.

Objektdetekteringen är dock väldigt kostsamt beräkningsmässigt. För att kompensera för detta kommer objektdetekteringen enbart köras en gång per sekund. Resterande tid kommer objektspårningsalgoritmer istället att användas, vilka är mycket mindre kostsamma beräkningsmässigt.

Valet av objektspårningsalgoritmer kommer inte att undersökas på samma sätt som valet av objektdetektionsalgoritmen. Detta eftersom att objektspårningsalgoritmerna inte är såpass kostsamma beräkningsmässigt, och därför spelar det ingen större roll vilken/vilka som används i realtidssyfte. Algoritmerna *Kernelized Correlation Filter* och *Centroid Tracking* kommer att användas i kombination. Detta för att KCF kommer att ge *Centroid Tracking* algoritmen bounding boxes som *Centroid Tracking* algoritmen sedan kan göra om till mittpunkter och försöka koppla samman till punkter från tidigare bilder. På så sätt kan vi hålla koll på objekten i bild var för sig, och räkna dem när de rört sig förbi en viss punkt i bild.

3.2 Val av objektdetekteringsalgoritmer

Vilken objektdetekteringsalgoritm som används påverkar noggrannheten av hela systemet i sig. Dels på grund utav algoritmens noggrannhet, men även på grund av dess körningstid. Därför skall flera olika algoritmer för objektdetektering undersökas för att se vilka algoritmer som passar bäst för en uppgift som denna. Det som kommer avgöra valet mellan de olika algoritmerna är dels hur snabba de är, eftersom

att de skall köras i realtid, men också hur noggranna de är.

För att undersöka och jämföra prestanda av olika algoritmer för detektion av persontrafik, måste ett urval av objektdetekteringsalgoritmer också göras. För att hitta rimliga algoritmer att undersöka genomfördes en informationssökning där det visade sig att tre av de bästa algoritmerna för objektdetektion var *Faster R-CNN*, *MobileNet-SSD* och *YOLOv5* [23][24][25]. Därav valdes de tre algoritmerna.

3.3 Evaluering av datainsamlingssystemet

För att kunna evaluera datainsamlingssystemet behöver en evalueringsmetod fastställas. I och med att algoritmerna som kommer att användas redan är förtränade, så går det inte att jämföra dem i termer av standardiserade evalueringsmetoder inom maskininlärning. Istället kommer testvideor att användas.

Antalet personer som går förbi en linje placerad i mitten av bilden i testvideon kommer alltså först att räknas manuellt. Neråt representerar här ingång och uppåt representerar utgång. Här kanske är viktigt att nämna att det finns en mänsklig faktor inblandad i räknandet av personer. För att elaborera så kan det såklart bli rätt uttråkande att sitta och räkna människor. Detta på grund utav att den första videon enligt utförd räkning innehåller 52 personer sammanlagt, och den andra videon 105. Dessutom kan en människa missa att räkna en viss person som datorn istället detekterar.

Syftet med att testa algoritmerna mot två olika videor med ökande antal personer i varje video, är för att se ifall prestandan av systemet påverkas märkbart ju längre systemet får stå på. Tanken är ju att systemet skall vara igång under längre tid. Testvideorna som nämnts ser ut som i figur 3.1.

Figur 3.1: En bild från en av testvideorna som använts med mittlinjen placerad

Algoritmerna kommer var för sig att testas mot videorna. Detta för att se hur många av antalet personer som passerar mittlinjen i videorna som har räknats av algoritmen. På detta sätt kan sedan ses hur många personer algoritmen räknar kontra hur många personer som faktiskt passerade mittlinjen. Med hjälp av denna procenten kan de olika algoritmerna sedan jämföras. Metoden ger även information om vad algoritmerna är bra/mindre bra på. Till exempel kanske en algoritm är jättebra på att detektera människor som går in i en byggnad, men mindre bra på att detektera människor som går ut ur en byggnad.

En annan viktig aspekt gällande objektdetekteringsalgoritmer i realtidssammanhang är så klart körningstiden. Algoritmen måste vara tillräckligt snabb för att kunna användas i ett realtidssyfte, vilket ger ännu en parameter att jämföra algoritmerna med.

3.4 Tillgängliggörande av insamlad data

Den här rapporten kommer fokusera på två huvudsakliga sätt för tillgängliggörande av den insamlade datan. Det ena sättet innebär att man tillhandahåller en enkel API med en så kallad ändpunkt som är en webbadress som vid begäran skickar tillbaks (returnerar) den insamlade rådatan, alltså en lista av insamlade event. Det andra sättet innebär en mer avancerade API som tillhandahåller flera ändpunkter med bearbetade versioner av rådatan. För att kunna undersöka hur lätt integreringen mellan datainsamlingssystemet och en webbapplikation är kommer en blandning av dessa metoder att användas. Rådatan kommer finnas tillgängliga och ett par mer avancerade ändpunkter kommer skapas för att kunna jämföra svårigheter mellan de två tillvägagångssätten. För att utveckla ändpunkterna kommer AWS tjänsterna API Gateway, DynamoDB samt Lambda att användas.

3.4.1 Utveckling av hemsidan

För att kunna testa integrationssvårigheten kommer en hemsida att utvecklas. Hemsidan kommer visa olika statistik som kan extrapoleras utifrån rådatan så som nuvarande antal personer på kontoret, snitt antal per vecka, antal per specifik dag i X antal veckor med mera. Sidan kommer byggas med AWS tjänsterna API Gateway, Lambda och S3. Dessa tjänster kommer konfigueras genom Serverless Framework och konfigurationen kommer att utgå ifrån mallen som hittas i [26]. Sidan kommer skrivas huvudsakligen med Typescript och biblioteket React kommer att användas.

4

Resultat

Detta kapitel har som syfte att presentera resultaten av målsättningarna för arbetet. Först presenteras den utvecklade API:n följt av datainsamlingssystemets konstruktion. Därefter visas resultatet av jämförelser mellan de olika algoritmerna för objektdetektion. Sedan redogörs integreringen mellan webbapplikationen och API:n.

4.1 Data API

Totalt implementerades fyra ändpunkter där en av dessa var en enkel ändpunkt som förmedlade rådatan och resterade tre ändpunkter erbjöd bearbetad data istället. Alla ändpunkter har stöd för OPTIONS metoden som används av webbläsaren för att få information gällande tillåtna metoder, HTTP headers samt tillåtna ursprung för begäran till den använda adressen. Alla ändpunkters GET förfrågningar kräver att parametern *company* ska skickas med för att filtrera datan efter företag.

Ändpunkten i tabell 4.1 implementerades som den enkla ändpunkten, alltså ändpunkten som returnerar rådatan utan bearbetning. När en GET förfrågan görs till ändpunkten returneras en lista med alla event. Om de valfria parametrarna *end_date* och *start_date* används med formatet YYYY-MM-DD så returneras event som ligger mellan dessa två datum (inkluderande). Dessa event kan filtreras ytterligare genom att förse *weekday*-parametern med engelska dagnamnet i gemener vilket resulterar i att resultatet endast innehåller event som hände på matchande veckodag i det givna datumintervallet. För att *weekday* parametern ska användas måste *start_date* och *end_date* parametrarna användas.

Ändpunkten /			
HTTP Metod	**Auktorisation**	**Parametrar**	**Body (JSON)**
DELETE	API Nyckel	-	event_id company
GET	-	company start_date (valfri) end_date (valfri) weekday (valfri)	-
OPTIONS	-	-	-
POST	API Nyckel	-	event_id company total_in total_out

Tabell 4.1: Tillgängliga HTTP metoder samt auktorisation metod och parametrar för respketive metod för ändpunkten /

Ändpunkten i 4.1 används även för datainmatning samt borttagning, i dessa syften implementerades även stöd för HTTP metoderna DELETE och POST. För inmatning används POST metoden, den har inga parametrar men kräver en JSON formatterad *body* med *event_id* i form av en siffra, *company* i form av en sträng med företagsnamnet, *total_in* i form av en siffra samt *total_out* i form av en siffra. *Total_in* och *total_out* variablerna bör hålla antalet personer som har gått in respektive ut sen insamlingssystemet sist kontaktade API:n. Alla dessa fält är obligatoriska och behövs för att ett nytt event ska läggas till i databasen. Bortaggnings metoden (DELETE) fungerar som inmattningsmetoden men har endast *event_id* och *company* fälten i *body* med samma innehåll som beskrivet för POST.

Den första bearbetande ändpunkten syns i tabell 4.2. Ändpunkten implementerar GET metoden och vid förfrågan måste parametern *date* skickas med och dess format bör vara YYYY-MM-DD. Datan som returneras är det totala antalet in- och utgående personer under dagen som specificeras av parametern.

Ändpunkten /daily-total		
HTTP Metod	**Auktorisation**	**Parametrar**
GET	-	company date
OPTIONS	-	-

Tabell 4.2: Tillgängliga HTTP metoder samt auktorisation metod och parametrar för respketive metod för ändpunkten /daily-total

Nästa avancerade ändpunkten finns i tabell 4.3, den tillhandahåller en stor mängd bearbetad data relaterade till en specifik vecka och år. Vid en GET förfrågan finns

det två obligatoriska parametrar för att specificera veckan samt året. Veckan specificeras av parametern *week_number* och innehåller veckans nummer och året specificeras av parametern *year* i formattet YYYY. Utifrån dessa två parametrar returnerar ändpunkten veckans totala antal personer, total antal personer per dag och snitt antalet per dag samt att den specificerar start och slut datumet, veckonummret och året som användes.

Ändpunkten /weekly-data		
HTTP Metod	**Auktorisation**	**Parametrar**
GET	-	company week_number year
OPTIONS	-	-

Tabell 4.3: Tillgängliga HTTP metoder samt auktorisation metod och parametrar för respketive metod för ändpunkten /weekly-data

Sista implementerade ändpunkten syns i tabell 4.4 och tillhandahåller genom GET förfrågningar genomsnitt antal personer per veckodag för de senaste veckorna. Mängden senaste veckor specificeras av parametern *amount* i form av en siffra, veckorna räknas från och med veckan innan nuvarande vecka.

Ändpunkten /weekly-data/last		
HTTP Metod	**Auktorisation**	**Parametrar**
GET	-	company amount
OPTIONS	-	-

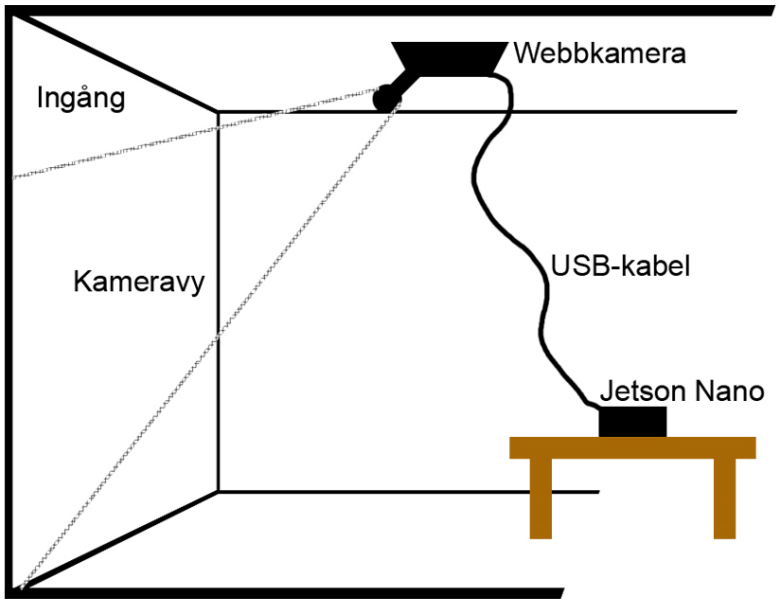
Tabell 4.4: Tillgängliga HTTP metoder samt auktorisation metod och parametrar för respketive metod för ändpunkten /weekly-data/last

4.2 Datainsamlingssystemet

I denna sektion presenteras olika resultat relaterade till datainsamlingssystemet. Först presenteras själva datainsamlingssystemet, och därefter listas resultat från jämförelserna av de olika algoritmerna för objektdetektion.

4.2.1 Det kamerabaserade systemet

För att samla in data installerades operativsystemet Ubuntu på mikrodatorn Jetson Nano för att kunna köra programkod. Sedan kopplades en webbkamera in i mikrodatorn. Webbkameran monterades upp i taket så att en dörröppning eller motsvarande syns i bild, likt figur 4.1.



Figur 4.1: Hur datainsamlingssystemet är uppsatt

Systemet läser in bilder en efter en från kameran och en gång per sekund utför objektdetektion på en av bilderna. Resterande tid utför systemet istället objektspråning för att dra ner på den beräkningsmässiga kostnaden. Under objektspårningsfasen kollar även systemet för varje objekt ifall objektet rört sig över en viss linje som placerats i mitten av bilden (dörröppningen) för enkelhetens skull. Systemet håller koll på hur många personen som rört sig över linjen med hjälp av variablerna *total_in* och *total_out*; dessa variabler inkrementeras då någon har gått in respektive ut.

När en person har rört sig över mittlinjen så skapar programmet en ny tråd som sätts i vila en viss tid för att sedan skicka en förfrågan till ändpunkten / som ses i tabell 4.1 via HTTP metoden POST. Bland parametrarna för HTTP metoden POST i ändpunkten / finns *event_id* och *company*. Parametern *company* sätts manuellt i

koden, och *event_id* räknas ut genom att hämta all data för företaget *company* via HTTP metoden GET i ändpunkten /, och låta *event_id* vara längden på den listan. När förfrågan är färdig nollställs variablerna *total_in* och *total_out* i systemets programkod.

4.2.2 Jämförelse av olika algoritmer för objektdetektion

I tabell 4.5 jämförs körningstiden av de tre algoritmerna som valdes att testas för objektdetektering. Med körningstiden menas alltså hur lång tid det tar att detektera objekt i en bild.

Körningstid			
	Faster R-CNN	**MobileNet-SSD**	**YOLOv5**
CPU	~47600ms	~230ms	~1800ms
GPU	-	~25ms	~240ms

Tabell 4.5: Jämförelse av körningstid

I tabell 4.6 jämförs den genomsnittliga FPS:n av de tre algoritmerna som valdes att testas för objektdetektering. Den genomsnittliga FPS:n mättes genom att köra algoritmerna var för sig på en testvideo och låta ett bibliotek i Python räkna ut FPS:n.

Genomsnittlig FPS			
	Faster R-CNN	**MobileNet-SSD**	**YOLOv5**
CPU	-	3.71	-
GPU	-	4.18	2.15

Tabell 4.6: Jämförelse av genomsnittlig FPS för algoritmerna på CPU och GPU

I tabellerna 4.7, 4.8, 4.9 och 4.10 jämförs noggrannheten av de tre algoritmerna som valdes att testas för objektdetektering. Noggrannheten uppmättes genom att räkna antalet personer i två olika testvideor, som passerade en linje placerad i mitten av bilden. Sedan fick varje algoritm försöka räkna antalet personer i testvideorna. Detta för att se hur många procent av antalet personer som gick in, ut och totalt som algoritmerna räknade.

Noggrannhet (CPU) 52 totalt, 37 in, 15 ut			
	Faster R-CNN	**MobileNet-SSD**	**YOLOv5**
% av antal in	-	59% (22 av 37)	0% (0 av 37)
% av antal ut	-	93% (14 av 15)	0% (0 av 15)
% totalt	-	69% (36 av 52)	0% (0 av 52)

Tabell 4.7: Jämförelse av algoritmernas noggrannhet på CPU:n för testvideon med 52 människor

Noggrannhet (GPU) 52 totalt, 37 in, 15 ut			
	Faster R-CNN	**MobileNet-SSD**	**YOLOv5**
% av antal in	-	54% (20 av 37)	81% (30 av 37)
% av antal ut	-	100% (15 av 15)	93% (14 av 15)
% totalt	-	69% (35 av 51)	86% (44 av 51)

Tabell 4.8: Jämförelse av algoritmernas noggrannhet på GPU:n för testvideon med 52 människor

Noggrannhet (CPU) 105 totalt, 71 in, 34 ut			
	Faster R-CNN	**MobileNet-SSD**	**YOLOv5**
% av antal in	-	56% (40 av 71)	0% (0 av 71)
% av antal ut	-	68% (23 av 34)	0% (0 av 34)
% totalt	-	60% (63 av 105)	0% (0 av 105)

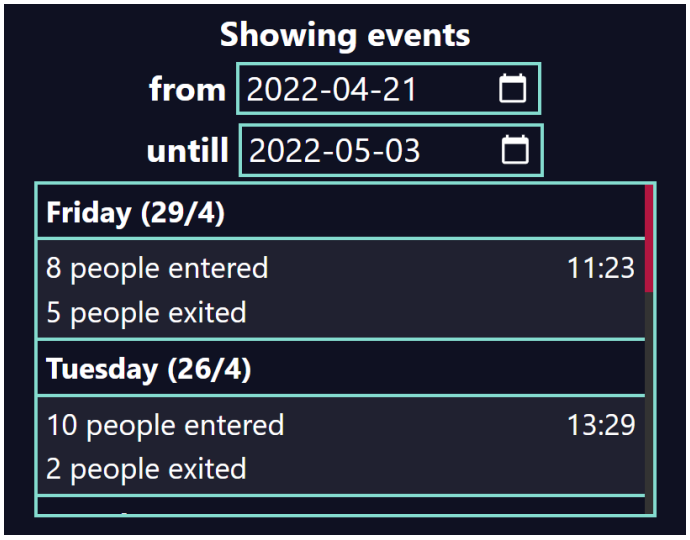
Tabell 4.9: Jämförelse av algoritmernas noggrannhet på CPU:n för testvideon med 105 människor

Noggrannhet (GPU) 105 totalt, 71 in, 34 ut			
	Faster R-CNN	**MobileNet-SSD**	**YOLOv5**
% av antal in	-	54% (38 av 71)	83% (59 av 71)
% av antal ut	-	68% (23 av 34)	97% (33 av 34)
% totalt	-	58% (61 av 105)	88% (92 av 105)

Tabell 4.10: Jämförelse av algoritmernas noggrannhet på GPU:n för testvideon med 105 människor

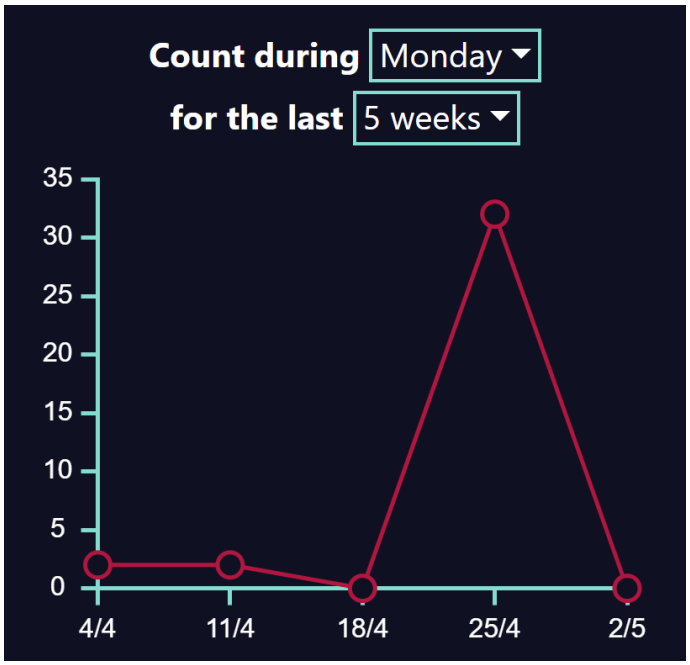
4.3 Webbapplikation och API integrering

Hemsidan använder alla de implementerade ändpunkterna för att hämta data. Den enkla ändpunkten från tabell 4.1 används för att hämta data för eventlistan i figur 4.2. En GET förfrågan med *company*, *start_date* och *end_date* parametrarna skickas till ändpunkten och det returnerade resultatet sorteras och omarbetas för att kunna visa datan i den implementerade eventlistan.



Figur 4.2: Eventlista i webbapplikation, se appendix B för hela sidan

Ändpunkten används också för att populera linje grafen i figur 4.3. En GET förfrågan med parametrarna *company*, *start_date*, *end_date* samt *weekday* görs och den returnerar en lista på event hos de angivna företaget mellan de två datum för dagar som matchar den angivna veckodagen. Sedan bearbetas datan genom att summera varje enskild dags event för att sedan kunna visa det totala antalet personer som gick in under den dagen i linje grafen.

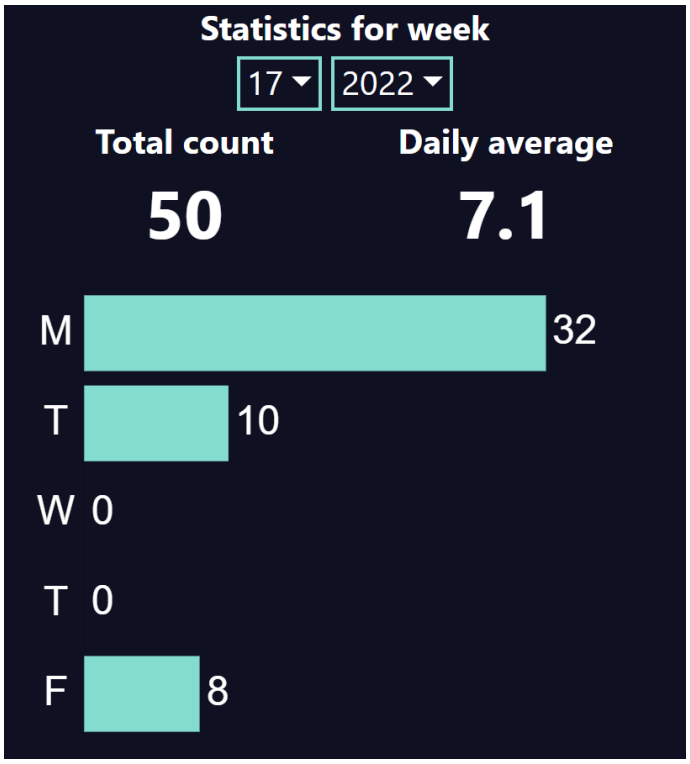


Figur 4.3: Linjegraf i webbapplikation, se appendix B för hela sidan

För att implementera nuvarande antal personer i kontoret som syns i figur 4.4 används ändpunkten i tabell 4.2 för att hämta dagens totala antal in- och utgående personer. I förfrågan skickas *company* och *date* parametrarna med där datum är dagens datum. Antalet utåtgående personer subtraheras sedan från antalet inåtgående personer för att få nuvarande antal personer i kontoret.

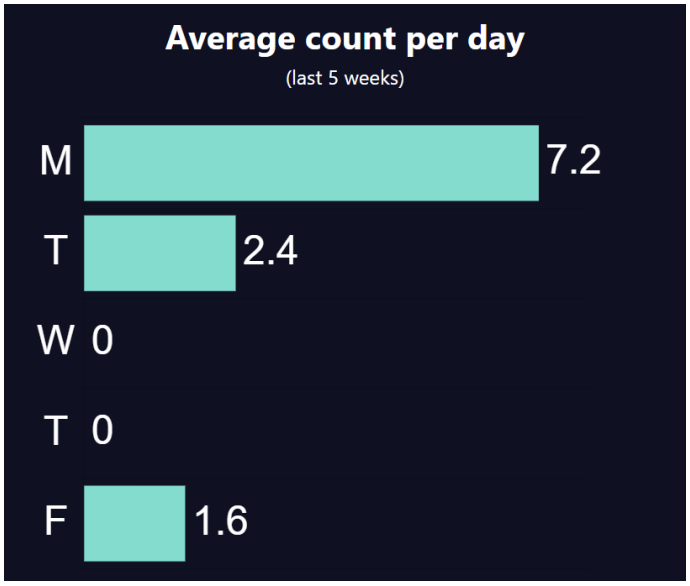
Figur 4.4: Nuvarande antal i webbapplikation, se appendix B för hela sidan

Datan för implementeringen av veckans statistik kommer ifrån ändpunkten i tabell 4.3 och informationen i webbapplikationen syns i figur 4.5. I förfrågan skickas *company*, *year* samt *week_number*, datan bearbetas inte utan visas direkt.



Figur 4.5: Veckans statistik i webbapplikation, se appendix B för hela sidan

Ändpunkten i tabell 4.4 används i figur 4.6 för att visa genomsnittligt antal personer per dag under de senaste 5 veckorna. *company* och *amount* parametern skickas med i GET förfrågan för att specificera företaget samt att snittet ska vara för 5 veckor. Även här bearbetas datan inte utan visas direkt.



Figur 4.6: Per veckodag genomsnitt i webbapplikation, se appendix B för hela sidan

5
Diskussion

I det här kapitlet kommer resultaten som presenterades i kapitel 4 att diskuteras. Detta görs dels i syfte att jämföra körningstiden för varje algoritm, men även att jämföra genomsnittlig FPS och noggrannhet. Även utveckling och integration av AWS API:n samt etik och hållbarhet kommer att diskuteras. Slutligen kommer framtida arbete att gås igenom. Flera problem som stöttes på längs med vägen samt potentiella lösningar till dessa kommer att tas upp allt eftersom. Kommer även kort nämnas angående tidsplaneringen.

5.1 Datainsamlingssystemet

I den här sektionen kommer resultatet av utvecklingen av datainsamlingssystemet att diskuteras.

5.1.1 Vilotiden innan kontakt med AWS

Varför det valdes att låta systemet sättas i vila en viss tid innan kontakt med ändpunkten / är för att eventlistan som kan ses i figur 4.2 inte skall bli överfylld med enstaka event. Istället gjordes beslutet att klumpa ihop event inom en viss tidsram. Detta drar även ner kostnaderna för användandet av olika AWS tjänster.

5.1.2 Alternativa sätt att räkna människor på

På grund utav att valet gjordes att hänga webbkameran i taket, blev just människoräkningen rätt limiterad. Så systemet är implementerat försöker systemet att se ifall ett detekterat objekt rört sig förbi en linje placerad i mitten av bilden. Därav måste kameran justeras så att mittlinjen infaller på en dörrtröskel eller dylikt.

Andra sätt att räkna människor på skulle kunna vara att låta användaren av systemet specificera linjen själv. Sedan kan x-koordinaten av mittpunkten för det detekterade objektet användas för att räkna ut och jämföra y-koordinaten på linjen vid objektets x-koordinat med objektets y-koordinat. På så sätt kan ses ifall objektet rört sig in respektive ut. Ett helt annat angreppssätt skulle kunna vara att markera dörröppningen som en *Bounding Box*, och sedan jämföra det detekterade objektets *Bounding Box* mot dörrens. Ifall objektets *Bounding Box* ökar i area/höjd/bred och ifall denna jämförbara egenskap är större än dörrens, då har ett objekt gått in och vice versa.

5.2 Körningstiden av algoritmerna

I den här sektionen kommer resultatet av testerna kring körningstiden av algoritmerna att diskuteras.

5.2.1 YOLOv5

I tabell 4.5 under sektion 4.2.2 kan ses att körningstiden för *YOLOv5* per bild när algoritmen körs på mikrodatorn Jetson Nano's CPU är upp emot 1,8s. Detta är inte ett bra resultat i realtidssyfte då systemet inte ens klarar att processa en bild per sekund. Så som systemet är implementerat i nuläget så körs objektdetekteringen en gång per sekund, men eftersom att objektdetekteringen alltid tar mer än en sekund så kommer systemet aldrig till objektspårningsfasen. Detta i sin tur betyder att systemet aldrig räknar hur många människor som rört sig in respektive ut ur en byggnad, eftersom att detta görs i just objektspårningsfasen.

En annan notering är att det tog allt för lång tid för systemet att köra igenom en av testvideorna, på grund utav tiden det tog för varje enskild objektdetektering. Denna fakta var något som redan kändes till, men beslutet togs att ändå låta systemet stå på under två nätter, en natt per testvideo. Detta skulle i ett realtidssyfte istället innebära att systemet processar en ny bild varje ~1,8s, så egentligen spelar det ingen större roll att det tog så lång tid det tog. Systemet skulle dock vara mindre noggrannt eftersom att det missar en hel del bilder, varav ett fåtal utav dessa kanske innehåller en människa, på de ~1,8s det tar för systemet att processa bara en bild.

I tabell 4.5 under sektion 4.2.2 kan ses att körningstiden för *YOLOv5* per bild när den körs på GPU:n var betydligt bättre. Körningstiden var alltså endast ~240ms istället för ~1.8s. En noterbar sak som gäller både för när algoritmen kördes på CPU:n och GPU:n var att körningstiden första gången algoritmen utförde en objektdetektering var betydligt längre. Första gången algoritmen kördes på GPU:n var körningstiden ~8s, medans den som redan diskuterats enbart var ~240ms under följande objektdetekteringar.

5.2.2 Faster R-CNN

I tabell 4.5 under sektion 4.2.2 kan ses att samma sak som gäller för *YOLOv5* även gäller för *Faster R-CNN* algoritmen då den körs på CPU:n. Problemet är dock ännu större här eftersom att körningstiden för objektdetektering på enbart en bild tog ~48s. Ännu värre än resultatet då *YOLOv5* kördes på CPU:n är att algoritmen krävde mer minne än systemet kunde tillgängliggöra. Detta ledde sedan till att processen dödades då tillräcklig allokering av minne misslyckades. För att förtydliga så dök inget felmeddelande upp gällande hur mycket minne algoritmen behövde, men som kan ses i tabell 2.1 så har mikrodatorn väldigt limiterat minne. Resultatet av att köra *Faster R-CNN* algoritmen på CPU:n var alltså katastrofalt. Ännu värre var resultatet då algoritmen kördes på GPU:n, vilket diskuteras mer noggrannt i sektion 5.6.

5.2.3 MobileNet-SSD

I tabell 4.5 under sektion 4.2.2 kan ses att körningstiden för objektdetektering på CPU:n då algoritmen *MobileNet-SSD* användes var ~230ms. Problemet som diskuterades i subsektionerna 5.2.1 och 5.2.2 gällande körning på CPU:n påverkade alltså inte körningen utav algoritmen *MobileNet-SSD*. Algoritmen presterade ännu bättre när den kördes på GPU:n, där körningstiden enbart var ~25ms. Detta är ett resultat som ökar hastigheten per bild med nästan hela 10 gånger jämfört med körning på CPU:n.

5.3 Algoritmernas genomsnittliga FPS

I den här sektionen kommer resultatet av testerna kring algoritmernas genomsnittliga FPS att diskuteras.

5.3.1 YOLOv5

I tabell 4.6 under sektion 4.2.2 kan ses att den genomsnittliga FPS:n för algoritmen *YOLOv5* när den kördes på mikrodatorns CPU var struken. Detta eftersom att FPS:n på grund utav den långa körningstiden per objektdetektering nästintill var försumbar. Kollar man istället på FPS:en vid körning på GPU:n så ser vi ett något högre värde på 2.15. Här är värt att notera att testerna som utfördes innehöll en hel del människor i bild jämfört med hur systemet egentligen kommer fungera, nämligen i en entré till ett kontor. Där kan förväntas att människor kommer in en och en, och oftast inte 50+ människor på led. Utöver detta var testvideorna rätt så långa. Den ena var ~5min lång, och den andra ~11min lång. Ifall testvideor med färre människor i bild samtidigt hade använts så hade den genomsnittliga FPS:en antagligen varit mycket högre. Detta gäller för samtliga algoritmer som testats.

5.3.2 MobileNet-SSD

I tabell 4.6 under sektion 4.2.2 kan ses att den genomsnittliga FPS:en för algoritmen *MobileNet-SSD* när den kördes på Jetson Nano:s CPU var något högre än för *YOLOv5*, och hamnade på 3.71. Som kanske förväntat så ökade FPS:en då algoritmen istället kördes på GPU:n, där resultatet istället blev 4.18. Detta är inte en jättestor skillnad jämfört med körning på CPU:n, men ändå en liten förbättring.

5.3.3 Faster R-CNN

Dels på grund utav vad som diskuterades i sektion 5.2.2 men även vad som senare diskuteras i sektion 5.6, saknas data för algoritmen *Faster R-CNN's* genomsnittliga FPS vid körning på både CPU:n och GPU:n.

5.4 Algoritmernas noggrannhet

I den här sektionen kommer resultatet av testerna kring noggrannheten av algoritmerna att diskuteras.

5.4.1 YOLOv5

I tabell 4.7 och 4.9 under sektion 4.2.2 kan ses att noggrannheten för algoritmen *YOLOv5* när den kördes på mikrodatorn Jetson Nano's CPU var 0% för alla fält. Resultatet 0% gäller här alltså både för testvideon med 52 och 105 människor totalt. Detta beror på vad som diskuterades i 5.2.1, nämligen att objektdetektering i koden görs varje sekund, annars körs objektspårningskoden som även har som uppgift att räkna objekten. Eftersom objektdetekteringen tar mer än en sekund körs aldrig objektspårningen dock, och därför räknades inga människor vid körning på CPU:n.

I tabell 4.8 och 4.10 under sektion 4.2.2 ses dock att noggrannheten för algoritmen *YOLOv5* vid körning på GPU:n var betydligt bättre än vid körning på CPU:n. Vid test med testvideon innehållande totalt 52 människor lyckades algoritmen räkna 81% av ingående trafik, 93% av utgående trafik, och 86% av totalt antal människor i bild. När algoritmen testades mot testvideon innehållande 105 människor totalt lyckades algoritmen räkna 83% av ingående trafik, 97% av utgående trafik, och 88% av totalt antal människor i bild. Resultatet visar här alltså på att algoritmens noggrannhet inte sjunker något märkbart ju fler människor som är i bild samt ju längre systemet låts stå på.

Inga bra jämförelser kan göras här mellan körning på CPU:n och GPU:n dock, med tanke på vad som diskuterades i subsektion 5.2.1.

5.4.2 MobileNet-SSD

I tabell 4.7 och 4.9 under sektion 4.2.2 kan ses att noggrannheten för algoritmen *MobileNet-SSD* när den kördes mot testvideon med totalt 52 människor på mikrodatorn Jetson Nano's CPU var 59% för ingående trafik, 93% för utgående trafik, och 69% totalt. När algoritmen istället kördes mot testvideon med totalt 105 människor i lyckades den räkna 56% av ingående trafik, 68% av utgående trafik och 69% totalt. Vi ser alltså att den totala noggrannheten gick ner med hela 9 procentenheter ju fler människor som var i bild totalt när algoritmen kördes på CPU:n.

I tabell 4.8 och 4.10 under sektion 4.2.2 ses att noggrannheten för algoritmen *MobileNet-SSD* vid körning på GPU:n var ungefär densamma som vid körning på CPU:n. När algoritmen testades mot testvideon innehållande 52 personer lyckades den räkna 54% av ingående trafik, 100% av utgående trafik och 69% totalt. Vid tester mot testvideon innehållande 105 personer sjönk dock noggrannheten. Här lyckades algoritmen enbart räkna 54% av ingående trafik, 68% av utgående trafik och 58% totalt. Vi ser alltså att den totala noggrannheten gick ner med hela 11 procentenheter ju fler människor som var i bild totalt när algoritmen kördes på GPU:n.

Noggrannheten var mer eller mindre densamma då man jämför körning av algoritmen på CPU:n och GPU:n för de båda testvideorna. Vi ser även rent allmänt att körningen av algoritmen på CPU:n och GPU:n följer samma mönster, där noggrannheten sjunker med ~10 procentenheter mellan de olika testvideorna. Alltså pekar detta resultat på att systemets noggrannhet vid körning av *MobileNet-SSD* bara blir sämre och sämre ju fler människor som är i bild samt ju längre systemet är på. Dock bör man kanske överväga vad som diskuterades emot slutet av subsektion 5.3.1, nämligen att när systemet väl sätts i bruk så kommer det antagligen bara vara en människa i bild åt gången. Detta i och med att systemet kommer att sitta vid en entré till ett kontor.

5.4.3 Faster R-CNN

Dels på grund utav vad som diskuterades i sektion 5.2.2 men även vad som senare diskuteras i sektion 5.6, saknas data för algoritmen *Faster R-CNN's* noggrannhet vid körning på både CPU:n och GPU:n.

5.5 Noggrannhetsskillnad mellan CPU och GPU

Förväntat är väl att noggrannheten borde vara densamma både då algoritmerna körs på CPU:n och GPU:n. I sektion 4.2.2 kan dock ses att så inte är fallet. Detta kan bero på att objektdetekteringen körs varje sekund istället för till exempel var 30:e bild. Eftersom körningstiden av objektdetekteringen är snabbare då algoritmerna körs på GPU:n, så kan det vara så att objektdetekteringen körts på olika bilder jämfört med på CPU:n. Därmed kan fler personer möjligtvis hittats, vilka givits till objektspårningen som sedan räknat de.

5.6 Angående körning av Faster R-CNN på GPU:n

Som kan ses i tabellerna 4.5, 4.6, 4.7, 4.8, 4.9 och 4.10 så saknas en hel del data vad gäller algoritmen *Faster R-CNN*. Problemet som uppstod vid körning på GPU:n var att när programkoden kördes, så frös programmet i ~2 minuter. Efter detta dödades programmet med felkoden *Killed*. Ingen bra information ges alltså om varför det inte fungerar som tänkt.

Mikrodatorn Jetson Nano är elektrostatiskt känsligt. Detta i kombination med att det är allmänt jobbigt att hela tiden på grund av den elektrostatiska känsligheten behöva montera upp och demontera Jetson Nano:n gjorde att en annan lösning blev mer attraktiv. För att underlätta utvecklingen har alltså all programkod skrivits på en Windows-maskin med bättre specifikationer än Jetson Nano:n. Koden har fungerat på Windows-maskinen, kanske enbart på grund utav att den haft bättre specifikationer rent allmänt. En annan sak som kan ha skiljt sig mellan Windows-maskinen och Jetson Nano:n är vilka versioner av Python bibliotek som använts.

5.7 Utveckling och integration av AWS API

Utvecklingen av data API:n gick snabbt och problemfritt. I nuvarande implementation av API:n används en DynamoDB tabell för att lagra data, separata Lambda funktioner för varje ändpunkt som bearbetar och returnerar resultatet samt API Gateway som utifrån den kallade ändpunkten startar rätt Lambda funktion. Den enkla ändpunkten som nämns i tabell 4.1 tog lite längre tid att implementera än förväntat. Även fast själva GET metoden är enkel att implementera då den endast hämtar data från en databas och möjligtvis filtrerar resultatet och sedan returnerar det så tog det en del tid att också implementera POST och DELETE metoderna. De resterande ändpunkterna tog lite mer tid per ändpunkt att implementera även fast de endast implementerade GET metoden. Detta beror på att de är de mer avancerade ändpunkterna och krävde extra bearbetning av datan innan den returnerades.

Integreringen utfördes snabbt och enkelt, som det nämns i sektion 4.3 så krävdes ingen eller minimal bearbetning av datan i webbapplikationen. Då den enkla ändpunkten endast användes för att hämta eventen i deras rå-format så märktes ingen större svårighet med den enkla ändpunkten. Hade de mer avancerade ändpunkterna inte implementerats så hade utvecklingen som lades på dessa ändpunkterna istället blivit lagd på bearbetning i webbapplikationen. Detta pekar till att arbetet kan skiftas mellan API utvecklaren samt de som använder API:n. I det här arbetet integreras endast en applikation samt att både API:n och API integreringen hanteras av samma utvecklingsgrupp vilket leder till att utvecklingen av en mer avancerad API inte hade någon större påverkan. Hade ett flertal applikationer integrerats så hade en avancerad API varit mer effektiv med ett antagande på att data bearbetningen för dessa applikationer är liknande om inte samma.

Enda problemet som stöttes på under integreringsfasen var att webbläsaren blockerade förfrågningar till API:n. Detta orskades av en säkerhetsfunktion som heter CORS, vilken blockerar vissa förfrågningar för att stoppa otillåten kod från att köras. För att tillåta dessa förfrågningar till API:n användes en funktion i API Gateway som automatiskt konfigurerar CORS relaterade inställningar. CORS konfigurationen skapade OPTIONS metoden på alla ändpunkter och konfigurerade dessa för att ge webbapplikationen tillgång. Utifrån detta kommer man fram till att svårighetsgraden för utveckling samt integrering av en API är huvudsakligen baserad på vilken data man utgår ifrån och vilka slutsatser man vill tillgängliggöra utifrån datan. Slutsatser som kräver mer bearbetning påverkar antagligen API utvecklingen eller integreringen beroende på om avancerade ändpunkter utvecklas för en enklare integrering eller ifall integreringen blir den avancerade delen med någon eller några få enkla ändpunkter.

5.8 Utvärdering av tidsplanering

Tidsplaneringen för det här projektet har följts, detta delvis då utvecklingen tog den förväntade tiden men även eftersom flera aktiviteter fick ett större tidsinterval än de krävde. Exempelvis så planerades fyra veckor per presentation vilket var mer tid än som kan rimligt behövas och som faktiskt krävdes. Planeringen gjordes på detta sätt eftersom exakta datum för dessa aktiviteter inte var kända när den gjordes.

5.9 Etik och hållbarhet

Ur ett etiskt perspektiv finns det inga särskilda aspekter som bör diskuteras. Detta då systemet endast dektererar människor och deras rörelseriktning och inte identifierar dem. Dessutom så analyseras kameradatan direkt i insamlingssystemet och tas bort direkt efter analys. Hade systemet identiferat enstaka människor så hade man behövt ta hänsyn till människors integritet men i nuläget så samlar det utvecklade systemet mindre information angående personerna i sig än vad säkerhetsbrickorna som öppnar dörren och är tilldelade specifika personer gör. Alla system som är uppkopplade mot internet kan dock bli hackade, alltså finns det en risk att en utomstående part får tillgång till liveströmmen från kameran. Därav bör man undersöka konfigurationer av brandväggar och andra säkerhetsfunktioner innan systemet används.

Ur ett hållbarhets-perspektiv kan man diskutera hur vettigt det är att ha en mikrodator och kamera startade konstant även fast det i princip inte kommer någon under nätter och helger. Att implementera någon sorts konfiguerbar start och stopp tid för systemet hade hjälpt spara energi vilket hade varit positivt för miljön. Även fast detta kan verka som en irrelevant mängd elektricitet som sparas per enhet så bör man ta hänsyn till att detta skalar med mängden enheter som används och kan snabbt bli en relevant mängd energi.

5.10 Framtida arbete

Att bygga systemet utifrån någon kombination av sensorer, exempelvis en rörelsesensor på varje sida av ingången skulle kunna vara en framtida utökning av arbetet för att jämföra det utvecklade kamerabaserade systemet med ett sensorbaserat system. Ett annat möjligt arbete är att undersöka fler eller modernare algoritmer för detekteringen av persontrafik. Alternativt kan samma algoritmer testas igen fast med nya modeller tränade specifikt för det här arbetet istället för förtränade modeller.

En upptäckt som är värd att ta med, ifall ett liknande projekt skulle utföras i framtiden, är att låta objektdetektionen köras varje x bild istället för varje x sekund, vilket reducerar körningstiden för segare objektdetekteringsalgoritmer drastiskt. Denna ändringen har en positiv påvärkan vid förinspelade videor men kan ha oförutsedda påverkningar på noggrannhet i realtidsscenarier där systemet hoppar

över enstaka bildrutor för att inte hamna efter. Därav bör alternativet undersökas vidare i dessa scenarier innan användning.

6
Slutsats

Ett av målen i sektion 1.3 var att utveckla ett kamerabaserat datainsamlingssystem för mätning av antal personer som går in och ut ur ett kontor. Målet nåddes genom att på mikrodatorn Jetson Nano installera operativsystemet Ubuntu, koppla en webbkamera till systemet samt köra utvecklad programkod på mikrodatorn. Webbkameran hängs sedan upp i taket vid en dörröppning eller motsvarande för att samla in data. In- och utträde detekteras i programkoden med hjälp av en linje placerad i mitten av bilden. Ifall en människa rör sig neråt och befinner sig under linjen så har en person gått in och vice versa. På detta sättet fungerar det kamerabaserade datainsamlingssystemet.

Målsättningen i sektion 1.3 angående att undersöka och jämföra prestanda av olika algoritmer för detektion av persontrafik på det utvecklade kamerabaserade systemet för mätning av persontrafik nådes också. Målet nåddes genom att producera två olika testvideor innehållande ökande antal människor. Sedan fick algoritmerna testas mot dessa videor för att se vilken algoritm som presterade bäst. Det gjordes jämförelser avseende på körningstid, det vill säga tiden det tar för objektdetektionen att behandla en bild, genomsnittlig FPS samt noggrannhet. Resultatet visade på att den mest noggranna algoritmen var *YOLOv5* när den kördes på mikrodatorns GPU. Behövs istället optimerad körningstid presterade algoritmen *MobileNet-SSD* bäst vid körning på GPU:n. Algoritmen *YOLOv5*'s körningstid är dock bra nog i realtidssyfte, och den är dessutom mest noggrann, vilket gjorde att just den algoritmen valdes för datainsamlingssystem.

Målet gällande tillgängliggörande av insamlade data genom Amazon Web Services för att sedan kunna undersöka hur lätt det är att integrera detta system i en webbapplikation nåddes. Detta utfördes genom att implementera en AWS baserad API med ett flertal avancerade ändpunkter samt en enkel, utveckla en webbapplikation samt att dessa två system integrerades med varandra. Integreringen visade sig vara lätt och snabb att göra vilket konstanterades var resultatet av att lägga tiden på implementeringen av mer avancerade ändpunkter.

Bibliografi

[1] (u.å.). ”Mission | To bring free software to the widest audience,” URL: https://ubuntu.com/community/mission (hämtad: 2022-02-21).

[2] (u.å.). ”NVIDIA Jetson Nano Developer Kit | NVIDIA Developer,” Nvidia, URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (hämtad: 2022-02-16).

[3] (u.å.). ”Cortex-A57,” URL: https://developer.arm.com/Processors/Cortex-A57 (hämtad: 2022-02-21).

[4] (u.å.). ”About - OpenCV,” URL: https://opencv.org/about/ (hämtad: 2022-02-16).

[5] D. Cochard. (2021). ”MobilenetSSD : A Machine Learning Model for Fast Object Detection,” URL: https://medium.com/axinc-ai/mobilenetssd-a-machine-learning-model-for-fast-object-detection-37352ce6da7d (hämtad: 2022-03-07).

[6] A. G. H. m. fl. (2017). ”MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” URL: https://arxiv.org/pdf/1704.04861 (hämtad: 2022-06-06).

[7] W. L. m. fl. (2016). ”SSD: Single Shot MultiBox Detector,” URL: https://arxiv.org/pdf/1512.02325 (hämtad: 2022-06-06).

[8] M. Chablani. (2017). ”YOLO — You only look once, real time object detection explained,” Towards Data Science, URL: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006 (hämtad: 2022-03-07).

[9] S. Ren, K. He, R. Girshick och J. Sun. (2016). ”Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” URL: https://arxiv.org/abs/1506.01497v3 (hämtad: 2022-05-08).

[10] R. Girshick. (2015). ”Fast R-CNN,” URL: https://arxiv.org/abs/1504.08083v2 (hämtad: 2022-05-08).

[11] A. Rosebrock. (2018). ”Simble object tracking with OpenCV - PyImageSearch,” PyImageSearch, URL: https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/ (hämtad: 2022-03-20).

[12] (2014). ”Accurate Scale Estimation for Robust Visual Tracking,” Linköping University, URL: http://www.cvl.isy.liu.se/research/objrec/visualtracking/scalvistrack/ScaleTracking_BMVC14.pdf (hämtad: 2022-04-05).

[13] (2022). ”dlib C++ Library,” dlib, URL: http://dlib.net/ (hämtad: 2022-04-05).

[14] (2022). ”About AWS,” Amazon Web Services, URL: https://aws.amazon.com/about-aws/ (hämtad: 2022-02-26).

[15] (2022). ”What is Amazon API Gateway? - Amazon API Gateway,” Amazon Web Services, URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html (hämtad: 2022-03-21).

[16] (2021). ”What Is Amazon DynamoDB? - Amazon DynamoDB,” Amazon Web Services, URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html (hämtad: 2022-02-26).

[17] M. Roberts, ”Serverless Architectures,” 2018. URL: https://martinfowler.com/articles/serverless.html#unpacking-faas (hämtad: 2022-03-11).

[18] (2022). ”What is AWS Lambda? - AWS Lambda,” Amazon Web Services, URL: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (hämtad: 2022-03-07).

[19] (2022). ”Cloud Object Storage - Amazon S3 - Amazon Web Services,” Amazon Web Services, URL: https://aws.amazon.com/s3/ (hämtad: 2022-03-08).

[20] (2021). ”What is Amazon S3? - Amazon Simple Storage Service,” Amazon Web Services, URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (hämtad: 2022-03-07).

[21] (2021). ”serverless/serverless: Serverless Framework – Build web, mobile and IoT applications with serverless architectures using AWS Lambda, Azure Functions, Google CloudFunctions & more! –,” Serverless Inc, URL: https://github.com/serverless/serverless (hämtad: 2022-03-21).

[22] (2022). ”Serverless - Infrastructure & Compute Providers,” Serverless Inc, URL: https://www.serverless.com/framework/docs/providers/ (hämtad: 2022-03-21).

[23] G. Boesch. (2022). ”Object Detection in 2022: The Definitive Guide,” URL: https://viso.ai/deep-learning/object-detection/ (hämtad: 2022-05-24).

[24] A. Choudhury. (2020). ”Top 8 Algorithms For Object Detection,” URL: https://analyticsindiamag.com/top-8-algorithms-for-object-detection/ (hämtad: 2022-05-24).

[25] A. Singh. (2021). ”Top 6 Object Detection Algorithms,” URL: https://medium.com/augmented-startups/top-6-object-detection-algorithms-b8e5c41b952f (hämtad: 2022-05-24).

[26] A. Rabold. (2022). ”arabold/serverless-react-boilerplate: React web application running on AWS Lambda using the Serverless Framework,” GitHub, URL: https://github.com/arabold/serverless-react-boilerplate (hämtad: 2022-02-11).

A

Objektdetektering med YOLOv5 i Python

```python
import numpy as np
import time
import dlib
import cv2


def format_for_yolov5_inference(frame):
    '''
        Method for formatting an input frame for
        Use in YOLOv5 inference. YOLOv5 accepts an
        image of size 300x300.
    '''

    # Get width and height from the input frame
    # and determine which of the two has the
    # highest value
    width, height, channels = frame.shape;
    maxOfWidthOrHeight = max(width, height);

    # Create a rectangle that is maxOfWidthOrHeight x maxOfWidthOrHeight
    # and set all pixel values to zero. Then put the original image
    # back between x=(0-width) and y=(0-height)
    result = np.zeros((maxOfWidthOrHeight, maxOfWidthOrHeight, 3), np.uint8);
    result[0:width, 0:height] = frame;

    return result;

def perform_yolov5_inference(input_image, net):
    '''
        Method for performing YOLOv5 inference. The input
        image is converted into a blob that is later input
        into the YOLOv5 network for inference. The inference
        time is measured and printed, and the method returns
        the predictions from the YOLOv5 inference.
```

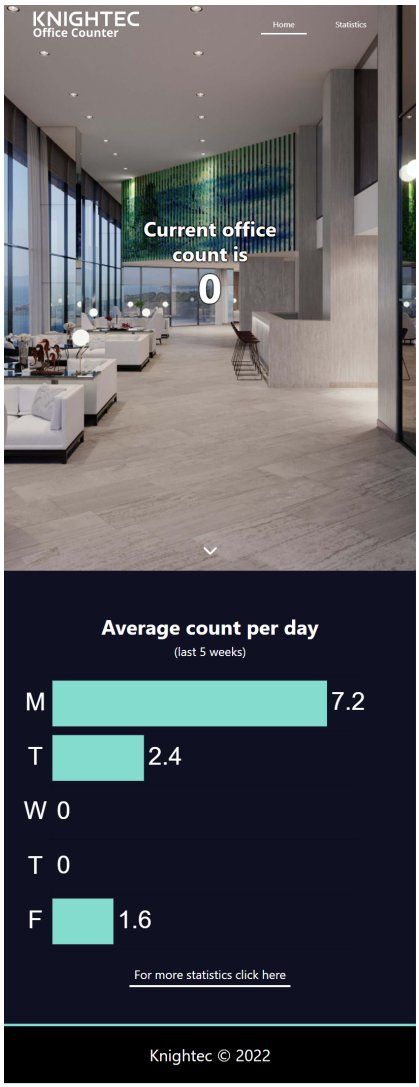
```
35      '''
36
37      # Convert our input image into a blob and set it as
38      # input to our YOLOv5 network.
39      blob = cv2.dnn.blobFromImage(input_image, 1 / 255.0, (640, 640), swapRB=True,
        ↪  crop=False);
40      net.setInput(blob);
41
42      # Perform YOLOv5 inference with input blob
43      # and measure inference time
44      before = time.perf_counter();
45      predictions = net.forward();
46      after = time.perf_counter();
47      print(f"[INFO]: YoloV5 inference time: {after - before}s");
48
49      return predictions;
50
51  def unwrap_yolov5_detections(input_image, predictions, confidence_threshold,
    ↪  nms_threshold):
52      '''
53          Method for unwrapping the predictions from the
54          YOLOv5 inference. Checks whether class confidence
55          is above set confidence threshold and performs
56          Non-Maxima Suppression.
57      '''
58      # Create empty lists for resulting bounding boxes,
59      # corresponding classId's and confidences
60      boxes = [];
61      classIDs = [];
62      confidences = [];
63
64      # Get total amount of predictions
65      num_rows = predictions.shape[0];
66
67      # Get input image dimensions and initiate
68      # two scale factor variables
69      image_width, image_height, channels = input_image.shape;
70      x_factor = image_width / 640;
71      y_factor = image_height / 640;
72
73      # Loop through all predictions row by row
74      for r in range(num_rows):
75          # Get prediction at row index r and extract its confidence score
76          row = predictions[r];
77          confidence = row[4];
78
```

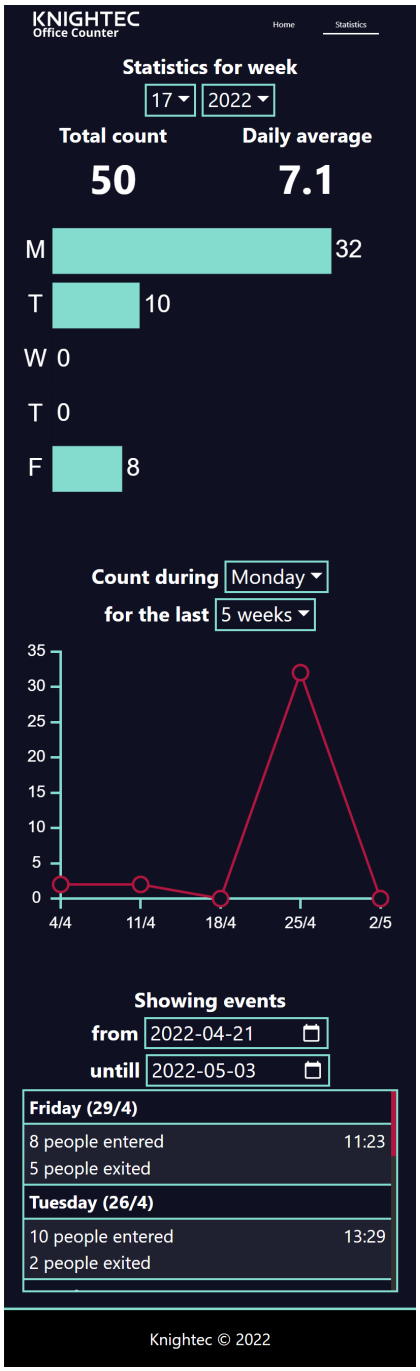
```
79          # Check if the confidence score is greater than the
80          # confidence threshold. This filters out weak detections.
81          if confidence > confidence_threshold:
82              # Extract the classID of the detection
83              class_scores = row[5:];
84              _, _, _, max_index = cv2.minMaxLoc(class_scores);
85              classID = max_index[1];
86
87              # Extract the bounding box coordinates of the detection and
88              # construct a box given said coordinates
89              (x, y, w, h) = (row[0].item(), row[1].item(), row[2].item(),
                ↪  row[3].item())
90              (startX, startY, endX, endY) = (int((x - 0.5 * w) * x_factor), int((y
                ↪  - 0.5 * h) * y_factor), int(w * x_factor), int(h * y_factor));
91              box = np.array([startX, startY, endX, endY]);
92
93              # Add variables to our lists
94              boxes.append(box);
95              classIDs.append(classID);
96              confidences.append(confidence);
97
98      # Perform non-maxima suppression, which removes overlapping
99      # bounding boxes representing the same object, and get the
100     # resulting indices in our lists.
101     indices = cv2.dnn.NMSBoxes(boxes, confidences, confidence_threshold,
        ↪  nms_threshold);
102
103     # Get the boxes, classID's and confidences
104     # after non-maxima suppression has been performed
105     # using the indices given by the function.
106     resulting_boxes = [];
107     resulting_classIDs = [];
108     resulting_confidences = [];
109
110     for i in indices:
111         resulting_boxes.append(boxes[i]);
112         resulting_classIDs.append(classIDs[i]);
113         resulting_confidences.append(confidences[i]);
114
115     return resulting_boxes, resulting_classIDs, resulting_confidences;
116
117 def process_yolov5_detection(boxes, classIDs, confidences, rgb_frame):
118     '''
119         Method for going through our unwrapped predictions
120         and checking if the detected object class corresponds
121         to that of a person. Then bounding box coordinates
```

```
122         are extracted and used to create a new kernalized
123         correlation filter object tracker and add it to our
124         list of object trackers.
125     '''
126     global CLASSES, trackers;
127 
128     for (box, classID, confidence) in zip(boxes, classIDs, confidences):
129         if CLASSES[classID] != "person":
130             continue;
131 
132         # Extract bounding box coordinates
133         (x, y) = (box[0], box[1]);
134         (w, h) = (box[2], box[3]);
135         (startX, startY, endX, endY) = (x, y, x + w, y + h);
136 
137         # Construct a dlib rectangle from bounding
138         # box coordinates and start a dlib
139         # correlation tracker.
140         tracker = dlib.correlation_tracker();
141         rect = dlib.rectangle(int(startX), int(startY), int(endX), int(endY));
142         tracker.start_track(rgb_frame, rect);
143 
144         # Add the tracker to our list of trackers
145         # so we can utilize it during skip frames
146         trackers.append(tracker);
147 
148 input_image = format_for_yolov5_inference(frame);
149 predictions = perform_yolov5_inference(input_image, net);
150 boxes, classIDs, confidences = unwrap_yolov5_detections(input_image,
    ↪  predictions[0], args["confidence"], args["threshold"]);
151 process_yolov5_detection(boxes, classIDs, confidences, rgb_frame);
```

B

Webbapplikationen





Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA & GÖTEBORGS UNIVERSITET
Göteborg, Sverige

CHALMERS