Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vårsemesteret 2023 Åpen
Forfatter(e): Benjamin Andre Scheiene Myklebust, Steffen Larsen Anthonsen	
Fagansvarlig: Ferhat Özgur Catak Veileder(e): Ferhat Özgur Catak	
Tittel på bacheloroppgaven: DDoS angrep generering for å utmatte GPU/CPU ressurser til IoT baserte AI applikasjoner Engelsk tittel: DDoS attack generation to exhaust the GPU/CPU resources of IoT based AI applications	
Studiepoeng: 20	
Emneord: Dyplæring, DDoS	Sidetall: 50 + vedlegg/annet: 2 Stavanger 15. mai 2023

Innhold

Sammendrag

I vår studie sikter vi på å teste dyktigheten til nettverksangrep deteksjonssystemer i Internet of Things enheter og deres egenskap til å motstå belastningsangrep på prosessorer gjennom et åpenkildet evolusjonær utviklingsprogram kalt NSGA-II. Dette er en nyoppdaget «hackingmetode» som i dag generelt blir kalt DDoS AI angrep. Vi studerer muligheten for at det ikke krever komplisert utstyr for at slike angrep skal kunne bli utført, og om offentlig tilgjengelige programvarer og algoritmer kan ha like stor effekt som de komplekse algoritmene som har blitt utviklet for de få studiene innen dette emnet. Motivasjonen bak denne utforskningen er den økende veksten i dyplæringsteknologi samt mangel i cybersikkerhetstjenester.

Ved et DDoS AI angrep gis et datainput til en maskinlæringsmodell (ML) med hensikt til å få modellen til å sløse bort så mye prosesseringskraft som mulig. Disse angrepene er malisiøse forsøk på å villede og forvirre maskinlæringsmodeller. En angriper kan f.eks. endre et bilde på en måte som gjør at et datasynssystem ikke greier å gjette hva det er meningen bildet skal representere, og dermed tvinger datasynsystemet til å sløse så mye prosesseringskraft som mulig for å prøve å gjette seg fram til riktig svar.

Ut i fra våre resultater observerer vi at angrepsmetodens effektivitet varierer bemerkelig på hva type datasett og modell NSGA algoritmen blir utsatt for. Både vår testmodell og vår hovedmodell ville eventuelt nå en grense hvor den ikke sløste bort flere nevroner eller prosesseringskraft uansett hvor mye NSGA fortsatte å manipulere og finjustere inndataen. Derimot hadde NSGA en bemerkelig effekt på prosesseringskraften til modellene. På den enkle testmodellen kunne prosesseringskraften øke med 43% gjennom datamanipulering, mens i vår komplekse datamodell fløt prosseseringsøkningen

rundt 57% etter datamanipulering.

Ut i fra våre resultater kom vi fram til at angrepsmetoden har en bemerkelig effekt på modellenes ytelse, og at det bør implementeres metode for å begrense hvor mange noder som kan bli aktivert etter hvor bra systemapparatet modellen kjører på er. Motstridelsesøving bør også bli tatt inn til konsiderasjon, hvor modellene øver på manipulert data for å kunne oppdage dem og ikke sløse bort prosesseringskraft på dem. Videre utforskning kreves for å forstå fullskalaen av hvor stor innflytelse disse angrepsmetodene kan ha.

Kapittel 1

Introduksjon

Det er få som ikke har hørt om maskinlæring den dag i dag. Maskinlæringsdebatten har tatt verden med storm, og har sakte trengt seg inn i hver side av våre hverdagsliv. Fra arbeidsoppgaver til moderne kunst er det ikke mulig å unngå maskinlæringens innflytelse.

Maskinlæring er et underfelt av kunstig intelligens, også kjent som AI («Artificial Intelligence»)[16]. AI takler spørsmålet om hvordan å skape datamaskiner som er egnet til intelligent oppførsel, mens maskinlæring (ML) går i dybde på algoritmer som kan danne forutsigelser utifra data. ML har sitt eget underfelt kalt «dyp læring» (Deep Learning, DL) som baserer seg på å danne algoritmer, som kan lære fra data på en lignende måte på hjernen sitt nervesystem[19]. DL algoritmer tar inn lassevis med data for å kunne skape nøyaktige forutsigelser og konklusjoner. DL algoritmer er i høy etterspørsel på grunn av dens punktlige nøyaktighet i å gjenkjenne mønstre, gjøre forutsigelser og dens egenskap til å løse problemer. På grunn av dette har AI feltet utvidet seg bemerkelig det siste tiåret, og teknologien fortsetter å revolusjoneres.

En lite omdiskutert side av maskinlæring er dens motstandsdyktighet til angrep og hva effekt disse angrepene kan ha på systemene programvaren er instalert på. Lite studier har blitt gjort innen dette emnet, og maskinlæringssikkerhet har ikke blitt betydelig omdiskutert før de siste årene nå som maskinlæringsprogrammer som ChatGPT, bildegjenkjenningsprogrammer

og kunstverksprogrammer skaper en betydelig innflytelse på verden. Hensikten med denne bacheloren er å utforske en nyoppdaget sårbarhet innen maskinlæring hvor AI programvaren kan være sårbar til DDoS lignende angrep, og dermed studere hvor stor innflytelse denne angrepsmetoden kan ha på maskinvarens prosessor.

Få antall studier har blitt gjort innen dette emnet, men deres resultater har slått alarm blant forskere innen maskinlæringsindustrien. Den populære studien «A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference» [35] er hva som har skapt mest støy blant forskere, og studerer akkuratt dette emnet. Studien utviklet et kompleks angrepsprogram kalt DeepSloth, med hensikt til å angripe små Internet of Things enheter med lav prosesseringskraft gjennom å forstyrre deres bildegjenkjenningsprogram med manipulerte bilder skapt av DeepSloth programmet. Studien kom fram til det sjokkerende resultatet at man kunne begrense en Internet of Things enhet prosesseringskraft med hele 90-100% gjennom deres DeepSloth program[35]. I vår studie sikter vi på å teste dyktigheten til nettverksangrep detektor Internet of Things enheter i deres egenskap til å motstå slike belastningsangrep på prosessorer gjennom et åpenkildet evolusjonær utviklingsprogram kalt NSGA-II. Vi studerer muligheten for at det ikke krever noe komplisert utstyr for at slike angrep skal kunne bli utført, og om offentlig tilgjengelige programvarer og algoritmer kan ha like stor effekt.

1.1 Bakgrunn og motivasjon

I moderne tid utgjør «hacking» og nettverksangrep noen av de mest betydelige truslene mot bedrifter og organisasjoner[24]. Disse angrepene kan ha ødeleggende konsekvenser for virksomhetenes økonomi og omdømme. Ifølge en artikkel fra Inc taper bedrifter over hele verden over 400 milliarder dollar hvert år til cyberkriminalitet[21]. I tillegg kan angrepene ha alvorlige implikasjoner for både kundenes og sivilbefolkningens sikkerhet. En av de mest alvorlige konsekvensene av «hacking» og nettverksangrep er økonomisk tap for bedrifter. En enkelt feil i sikkerhetssystemet kan gi en «hacker» evnen til å legge ned tjenester og koste bedriften tusenvis, om ikke millioner, i tapte inntekter[10]. I tillegg kan angrep påvirke bedriftens omdømme, noe som kan føre til varig skade på virksomheten. Bedrifter kan også risikere å måtte betale bøter og erstatning til kunder og myndigheter hvis de ikke har

tilstrekkelig beskyttelse mot angrep. En annen fare ved «hacking» og nettverksangrep er risikoen for at «hackere» får tilgang til kundenes personlige informasjon. Dette kan være alt fra kredittkortopplysninger til personnummer, og kan utgjøre en stor fare for kundenes privatliv og sikkerhet. Hvis en bedrifts kunder blir utsatt for svindel eller identitetstyveri, kan dette føre til at kundene mister tilliten til bedriften og unngår å gjøre forretninger med dem i fremtiden. Vi må også ta hensyn til trusselen som «hacking» og nettverksangrep utgjør mot infrastruktur[33]. Politisk motiverte «hackere» kan lemme kommunikasjons- og helseinfrastruktur, og skape panikk og helsefare blant sivilbefolkningen for å oppnå sine mål. Dette kan føre til kaos og uro i samfunnet, og i noen tilfeller kan det til og med true nasjonal sikkerhet.

Cybersikkerhet har blitt et kritisk problem de siste årene på grunn av det økende antallet nettangrep på enkeltpersoner, bedrifter og myndigheter. Cyberkriminalitet har blitt mer sofistikert, og «hackere» har blitt stadig flinkere til å utnytte sårbarheter i datasystemer og nettverk. Cybersikkerhet er et felt i stadig utvikling, og det krever konstant overvåkning og tilpasning for å ligge i forkant av nettkriminelle. En av de viktigste utfordringene cybersikkerhet står overfor i dag er omfanget og kompleksiteten av trusselandskapet. Cyberangrep kan komme fra en rekke kilder, inkludert kriminelle grupper, nasjonalstater og individuelle «hackere». Utvalget av angrepsvektorer er stort, og nettkriminelle innoverer hele tiden for å finne nye måter å bryte systemer og stjele data. En annen betydelig utfordring cybersikkerhet står overfor er mangelen på dyktige cybersikkerhetsfagfolk[3][39]. Etterspørselen etter fagfolk innen cybersikkerhet overgår raskt tilbudet, og dette har skapt et kompetansegap som er vanskelig å bygge bro over. Denne mangelen er spesielt akutt i utviklingsland, hvor mange organisasjoner mangler ressurser til å ansette eksperter på nettsikkerhet.

Bedrifter bruker tilsammen over 170 milliarder dollar i året til å minimere cyberangrep[41]. Cybersikkerhetstjenester vet hvordan å forsikre tjenester mot nesten alle anerkjente «hackingmetoder», og med riktige mitigeringsgrep kan de fleste institusjoner unngå kritiske angrep fullstendig. Derimot kan angrep siktet mot bedrifters etablerte AI programvarer utgjøre en helt ny og ubevist trussel innen cybersikkerhetsfeltet, og kan skape uforutsigbare trusler mot bedrifter og deres kunders sikkerhet[26].

På grunn av den økende sofistikering innen cyberkriminalitet og dets mangel på erfarne cybersikkerhetsfagfolk er vi motiverte til å videreutforske disse

truslene. I denne bacheloroppgaven velger vi å videreutforske en nyoppdaget cyberangrepsmetode for å «hacke» AI programmer og sette dem ut av bruk, og om man kan gjennom åpen-kildete programmer som NSGA gjøre angrep mot AI som angir en betydelig trussel for det moderne samfunn og dermed måle hvor sterk innflytelse et angrep mot AI kan ha. Vår oppgave sikter spesifikt på AI ment til å oppdage nettverskangrep, men våre resultater vil peke på om det finnes en generell sårbarhet i hvordan vi utvikler AI gjennom våre maskinlæringsprogramvarer.

1.2 Teori

1.2.1 AI, maskinlæring og artifisielle nevrale nettverk

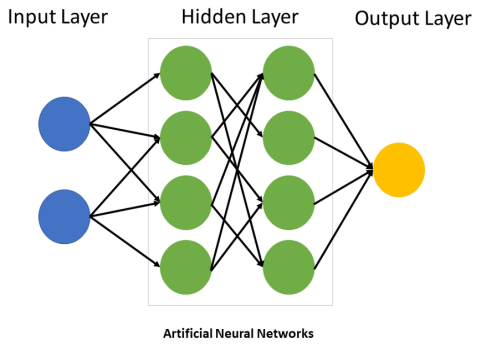
AI står for «artificial intelligence» eller kunstig intelligens, og det refererer til datamaskiners evne til å utføre oppgaver som normalt krever menneskelig intelligens, for eksempel gjenkjenning av bilder, talegjenkjenning, oversettelse av språk, planlegging og beslutningstaking[28]. Det finnes forskjellige tilnærminger til å implementere AI, og maskinlæring er en av de viktigste og mest populære metodene.

Maskinlæring er en gren av AI som innebærer å lære fra data, uten å bli programmert spesifikt for en bestemt oppgave. Det handler om å utvikle algoritmer som kan identifisere mønstre i dataene og bruke denne kunnskapen til å gjøre prediksjoner eller ta beslutninger på ny og ukjent data[16]. For eksempel kan en maskinlæringsalgoritme trenes på store mengder bilder av hunder og katter for å lære å skille mellom de to dyrene, og deretter brukes til å klassifisere nye bilder som enten hund eller katt. Maskinlæring er en viktig komponent i utviklingen av AI, fordi det gir datamaskiner muligheten til å lære og forbedre seg selv over tid. Det har ført til store fremskritt innen mange forskjellige områder, som talegjenkjenning og bildegjenkjenning.

Artifisielle nevrale nettverk (ANN) er et undersett av maskinlæring, og er konseptet dyplæring er bygget på. Et ANN gjør prediksjoner om data samt lærer fra tidligere opplevelser. Navnet og strukturen på ANNer er inspirert av menneskehjernens nevrale system og er ment å herme menneskehjernens tankegang[42].

1.2 Teori

Et ANN består av 3 sett med definerte «node-layers», ett «input layer», ett eller flere «hidden layers» og ett «output layer». Hver node, som vi ser på som et artifisielt nevron, er tilkoblet til hver andre node, og hver node inni «hidden layeret» er tildelt sin egen «vekt». ANN går gjennom 3 faser. Først blir AI modellen bygget, så trenes modellen på et treningsdatasett og tilslutt testes modellen på et testdatasett. Vekten til hver nevron blir justert i treningsfasen til det nevrale nettverket[7].



Figur 1.1: Eksempel av et nevralt nettverk sine «layers».

«Input layeret» er hvor data blir sendt inn i nettverket for å bli bearbeidet. Arbeidet blir gjort inni «hidden layeret», hvor nettverket lærer fra dataen som blir sendt inn og lærer seg å gjøre forutsigelser. Vektene inni «hidden layeret» bestemmer hvordan dataen fra «input layeret» er prosessert. Antall «hidden layers» vil somregel bli høyere jo mer kompleks dataen programmet er ment til å ta inn er. «Output layeret» er hvor forutsigelser blir bestemt og sendt til brukeren. Under trening justeres vektene på forbindelsene basert på feilen mellom den faktiske utgangen til nettverket og ønsket utgang. Denne prosessen, kjent som «backpropagation», lar nettverket lære å gjenkjenne mønstre og relasjoner i inndataene og lage spådommer eller klassifiseringer.

ANN-er har blitt brukt i et bredt spekter av applikasjoner, inkludert bilde- og talegjenkjenning, naturlig språkbehandling, autonome kjøretøy og robotikk.

1.2.2 Dyplæring

Dyplæring er en undergruppe av maskinlæring som involverer trening av kunstige nevrale nettverk med flere lag for å løse komplekse problemer som bildegjenkjenning, naturlig språkbehandling og talegjenkjenning[15]. Det er et aktivt område for forskning og utvikling, med pågående innsats for å forbedre ytelsen og effektiviteten til dyplæringsmodeller og bruke dem på nye og utfordrende problemer. Dyplæringsalgoritmer lærer å gjenkjenne mønstre og funksjoner i data ved å behandle store mengder informasjon gjennom lag med sammenkoblede noder, som er organisert i input-, skjulte og outputlag[19][28].

I dyp læring trenes det nevrale nettverket vanligvis ved hjelp av et stort datasett med merkede eksempler, for eksempel bilder med kjente kategorier eller tekster med kjente betydninger. Under treningsprosessen justerer nettverket vektene og skjevhetene til nodene for å minimere feilen mellom de forutsagte outputene og de faktiske outputene.

En av de viktigste fordelene med dyp læring er dens evne til å lære hierarkiske representasjoner av data, der de nedre lagene i nettverket lærer enkle funksjoner, for eksempel kanter eller hjørner i bilder, og de høyere lagene lærer mer komplekse funksjoner, for eksempel former eller gjenstander[40]. Denne hierarkiske representasjonen lar dyplæringsmodeller oppnå toppmoderne ytelse på et bredt spekter av oppgaver, som bildeklassifisering, objektgjenkjenning, talegjenkjenning og naturlig språkbehandling.

I moderne tider har det grodd en betydelig interesse for dyplæringsalgoritmer. En av hovedfaktorene som har ledet til dyplæringsalgoritmenes store suksess og ledet til dens breie integrering i ulike applikasjoner er deres evner til å lære fra ustrukturerte og umerkede data[19][28]. Dette er i motsetning til tradisjonell maskinlæringsmetoder, som krever at data er sterkt strukturert og merket for at en modell skal kunne lære av det. Dyplæringsmetoder kan lære av mindre strukturerte data, noe som gjør dem mye mer allsidige og kraftig. Nok en suksess som har ført til at Deep Learning-metoder er innarbeidet i ulike applikasjoner er evnen til Deep Learning-metoder å skalere til svært store datasett. Tradisjonelle maskinlæringsmetoder har en tendens til å bli overbelastet når de står overfor svært store datasett, men dyplæringsmetoder kan skaleres til disse datasettene relativt raskt. Dette er fordi Deep Learning-metoder kan parallelliseres på tvers av flere GPUer,

som lar dem trene på store datasett mye raskere enn tradisjonell maskinlæringsmetoder.

1.2.3 Dyplæring og cybersikkerhet

Det er umulig å ignorere dyplærings innflytelse på våre hverdagsliv den dag i dag. Dyplæring gror så raskt at det flyr over hodet på mange hvor bred implementasjonen av dyplæringsmodeller egentlig er. Den eksponensielle veksten av dyplæringsmodellers applikasjon inn til våre hverdagsliv gror bekymring for at modellene blir implementert raskere enn vår egenskap til å beskytte dem mot angrep [17]. Etter hvert som flere organisasjoner og enkeltpersoner bruker dyplæring for å lage og trene avanserte modeller, øker også de potensielle risikoene og sårbarhetene knyttet til disse teknologiene[13]. Det finnes flere grunner til disse bekymringene. Blant dem er deres økende kompleksitet. Dyplæringsmodeller er komplekse og vanskelige å forstå, noe som gjør det utfordrende å identifisere og adressere potensielle sårbarheter og sikkerhetsrisikoer. Også som tidligere nevnt er hastigheten på dyplæringsmodellers utvikling en bekymring. Utviklingshastigheten innen dyplæringsteknologi er mye raskere enn utviklingen av sikkerhetstiltak for å beskytte dem [5]. Som et resultat er det ofte et etterslep mellom fremveksten av nye sikkerhetstrusler og utviklingen av effektive mottiltak. Mangel på standardisering er også en av hovedbekymringene innen dyplæringssikkerhet. Mangelen på standardisering i utvikling og distribusjon av dyplæringsmodeller gjør det vanskelig å implementere konsistente sikkerhetstiltak på tvers av ulike applikasjoner og miljøer.

Den økende bruken av dyplæring i ulike applikasjoner har reist nye sikkerhetsutfordringer, og det er ennå ikke klart hvor godt rustet vi er til å møte dem. Noen av bekymringene knyttet til sikkerheten til dyp læring inkluderer:

Adversariale angrep: Dyp-læringsmodeller er sårbare for angrep som vil modifisere inndata på en måte som gjør at modellen feilaktig klassifiserer den. Adversariale angrep kan ha alvorlige konsekvenser i applikasjoner som autonome kjøretøy og medisinsk diagnostisering[5].

Personvern: Dyp-læringsmodeller kan lære å hente sensitiv informasjon fra data, som personlige identifikatorer, medisinsk informasjon eller finansielle

data. Å sikre personvernet til denne informasjonen er en betydelig utfordring.

Robusthet: Dyp-læringsmodeller kan være sårbare for feil eller anomalier i treningsdataene, som kan påvirke ytelsen deres i virkelige applikasjoner. Å sikre robustheten til disse modellene er en pågående utfordring.

Modelltyveri: Dyp-læringsmodeller kan være verdifulle immaterielle eiendeler, og det er en risiko for tyveri eller reverse engineering

Dette er noen få av de mest bevisste sikkerhetsrisikoene innen dyplæringsmodeller[44]. For å takle disse bekymringene utvikler forskere nye teknikker for å sikre dyp-læringsmodeller, som differensiell personvern, federert læring og sikker multiparty beregning. Imidlertid er det fortsatt mye arbeid som må gjøres for å sikre at dyplæring brukes sikkert og ansvarlig i årene som kommer.

1.2.4 Nevral nettverksarkitekturer

Nevrale nettverk har forskjellige typer arkitekturer fordi de er designet for å løse ulike typer problemer. Hver arkitektur har sine egne styrker og svakheter, som gjør dem bedre egnet for visse oppgaver enn andre. Denne delen går over hva nettverksarkitekturer som blir brukt til å utvikle våre modeller.

Feedforward nevrale nettverk

Et «feedforward» nevralt nettverk er en type kunstig nevralt nettverk som ofte brukes for overvåket læringsoppgaver som klassifisering og regresjon. I denne typen nettverk flyter informasjonen i en enkelt retning fra «input-layeret», gjennom en eller flere «hidden-layere» og til slutt til «output-layeret». Det er ingen tilbakemelding eller forbindelser mellom nevroner i samme lag. [37]

En multilayer-perceptron (MLP) er en mer kompleks form for «feedforward» nevralt nettverk som inkluderer flere «hidden layers»[6]. I en MLP består hvert «hidden layer» av en rekke prosesseringsenheter som utfører en vektet sum av sine innganger og anvender en aktiveringsfunksjon. Ut-

gangene fra hvert lag blir deretter sendt videre som innganger til neste lag til den endelige utgangen blir produsert. Arkitekturen til en MLP kan variere avhengig av det spesifikke problemet som blir adressert. Antall skjulte lag, antall prosesseringsenheter i hvert lag og valg av aktiveringsfunksjon kan alle ha betydelig innvirkning på ytelsen til nettverket. Trening av en MLP innebærer vanligvis å bruke en algoritme som «backpropagation» for å justere vektene og biasene i nettverket for å minimere en kostnadsfunksjon. Dette innebærer å beregne feilen mellom nettverkets utgang og ønsket utgang, og deretter propagere denne feilen tilbake gjennom nettverket for å justere vektene og biasene. Bruk av flere «hidden layers» gjør nettverket i stand til å lære komplekse funksjoner fra data, noe som gjør det til et effektivt maskinlæringsverktøy.

Convolutional nevrale nettverk

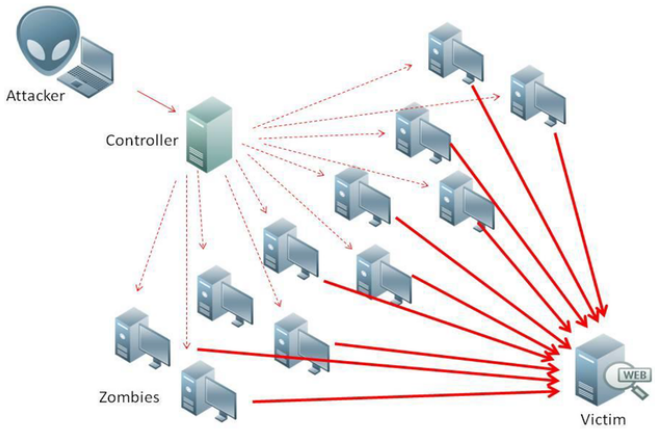
«Convolutional» nevrale nettverk (CNNs) er en type dyplæringsalgoritme som ofte brukes i bildegjenkjenning, objektdeteksjon og andre typer dataoppgaver som krever «syn». De er utformet for å etterligne måten hjernen vår bearbeider visuell informasjon på ved å bruke lag av sammenkoblede nevroner som kan lære å gjenkjenne mønstre i bilder. Den viktigste ideen bak CNNs er at de bruker «convolutional-layers» for å oppdage lokale mønstre eller funksjoner i inngangsbildet[27]. Et «convolutional-layer» består av et sett med filtre (også kjent som «kernels»), hvor hvert filter er liten i størrelse og påføres på inngangsbildet en om gangen. Hvert filter produserer et todimensjonalt funksjonskart ved å konvolvere den med inngangsbildet. Disse funksjonskartene fremhever tilstedeværelsen av visse mønstre eller funksjoner i inngangsbildet, som for eksempel kanter, hjørner eller klumper. Utgangen fra konvolusjonslaget sendes deretter gjennom en ikke-lineær aktiviseringsfunksjon, som for eksempel «Rectified Linear Unit» (ReLU), som introduserer ikke-linearitet i modellen og tillater den å lære mer komplekse funksjoner. Denne prosessen gjentas for flere konvolusjonslag, der hvert påfølgende lag bygger på funksjonene lært av de forrige lagene. Etter flere konvolusjonslag sendes utgangen gjennom ett eller flere fullstendig sammenkoblede lag, som utfører den endelige klassifiserings- eller regresjonsoppgaven. De fullstendig sammenkoblede lagene tar de høynivåfunksjonene som er lært av konvolusjonslagene, og bruker dem til å lage prediksjoner om inngangsbildet[38].

En av de viktigste fordelene med CNNs er deres evne til å lære funksjoner automatisk fra rådata, uten behov for manuell funksjonsteknikk. Dette gjør dem godt egnet for oppgaver som objektdeteksjon, der målet er å identifisere objekter av interesse i et bilde. Oppsummert er konvolusjonelle nevrale nettverk et kraftig verktøy for bildegjenkjenning og andre dataseeende oppgaver, og de har oppnådd state-of-the-art ytelse på en rekke benchmark-datasett.

1.2.5 DDoS

Distribuert tjenestenektangrep (DDoS) er en form for nettverksangrep som er utbredt og malisiøst. Det har til hensikt å gjøre en nettverkstjener utilgjengelig for legitim trafikk ved å overbelaste den med en stor mengde trafikk fra et nettverk av sammenkoblede datamaskiner.[25] Dette kan føre til at serveren mister evnen til å håndtere den vanlige nettverkstrafikken, noe som kan få konsekvenser for kundene som ønsker å bruke tjenester som serveren tilbyr.

En av de mest brukte teknikkene for å skape denne typen trafikk på en server er gjennom et SYN-flomangrep. Dette utnytter en svakhet i TCP-handshaking-protokollen, som er den prosessen som brukes for å opprette en tilkobling mellom to enheter.[30] Brukeren sender en SYN-pakke for å be om en tilkobling til serveren, og serveren svarer med en SYN/ACK-pakke for å bekrefte at tilkoblingen kan etableres. Deretter venter serveren på en ACK-pakke fra brukeren for å bekrefte tilkoblingen. Imidlertid kan en angriper utnytte denne prosessen ved å sende en stor mengde SYN-pakker og deretter aldri sende tilbake ACK-pakker. Dette vil føre til at serveren sløser bort mye prosesseringskraft på å vente på ACK-pakkene som aldri kommer. Ved å gjenta denne prosessen mange ganger, kan en angriper overbelaste serveren og gjøre den utilgjengelig for legitim trafikk. Ved å overbelaste en server med disse pakkene gjennom et sammenkoblet system av computere, kan du få en server til å sløse bort all sin prosesseringskraft slik at tjenester ikke kan bli servert til kunder.



Figur 1.2: Visualisert eksempel av et DDoS angrep. Den malisiøse aktøren har her erobret flere PC systemer han bruker som «zombier» til å angripe et bestemt offer.

Det er flere faktorer som kan bidra til en vellykket DDoS-angrep, inkludert antall angripende datamaskiner, kapasiteten til angriperens nettverk, serverens evne til å håndtere trafikk og nettverksinfrastrukturen som brukes. DDoS-angrep kan ha alvorlige konsekvenser for virksomheter, inkludert tap av inntekter, tap av kunder og et redusert omdømme. Tilstrekkelige sikkerhetstiltak blir satt på plass for å hindre slike angrep, inkludert brannmurer, innbruddsdeteksjonssystemer og nettrafikkanalyseverktøy.[34]

Angrepsmetoden utforsket i dette prosjektet er lignende denne form for nettverksangrep, som er grunnen til at angrepsmetoden blir tilgitt lignende terminologi. Istedenfor å overbelaste en server med nettverkspakker for å redusere dataens prosessorkraft, overbelaster vi heller et dyplæringsprogram med så kompleks data at programmet bruker opp all prosesseringskraft til å prøve å gjenkjenne hva som er i dataen.

1.2.6 Neural activation coverage

«Neural activation coverage» måler andel nevroner (noder) som blir aktivert i et nevralt nettverk[43]. Når et AI gjør en prediksjon gitt ett input vil hver nevron gi en verdi som output. Hvis nevronet sin output verdi er større enn

0 blir det regnet som aktivert. Hensikten med denne målingen er å utforske effektiviteten til et AI i å gjøre sine målinger, og hvor mye prosessering som kreves for å bearbeide dataen. Data som er enkel for AIet å predikere krever mindre nevroner enn kompleks data. I sammenheng betyr et høyt neural coverage at mer prosesseringskraft blir brukt, noe som vi videre kommer til å utnytte for å utføre angrepet vårt.

«Neural activation coverage» er en måling som brukes til å analysere nevrale nettverk. Det måler andelen nevroner (noder) som blir aktivert i et nevralt nettverk når en prediksjon blir gjort basert på et gitt input. Hvert nevron i nettverket gir en output-verdi når den mottar input, og hvis output-verdien er større enn 0, blir nevronet regnet som aktivert. Formålet med å måle «neural activation coverage» er å utforske effektiviteten til AI i å utføre sine oppgaver og for å finne ut hvor mye prosesseringskraft som kreves for å bearbeide dataen. Det er viktig å merke seg at en enkel og lite kompleks input vil kreve færre nevroner enn et komplekst input for å oppnå en høy dekning. Hvis det kreves flere nevroner for å oppnå en høy dekning, betyr det at mer prosesseringskraft blir brukt til å behandle inputen. Dette kan ha betydning i sammenheng med å utnytte nevrale nettverk for å utføre angrep[43].

I forbindelse med å utføre angrep kan en angriper utnytte en høy «neural activation coverage» for å manipulere inndataene slik at nettverket trenger å aktivere mange nevroner for å kunne utføre oppgaven. Dette kan føre til at nettverket bruker mye prosesseringskraft, noe som kan ha en negativ effekt på ytelsen og redusere effektiviteten til nettverket.

1.2.7 Motstridelses angrep

Ett av de mest vanlige typer angrep mot dyplæringsstrukturer er «Adversarial attacks», eller motstridelses angrep på norsk. Motstridelses angrep er en type angrep der en malisiøs «hacker» bevisst introduserer små, nøye utformede forstyrrelser til inndata, med mål om å få en dyplæringsmodell til å gjøre en feilklassifisering[20]. Motstridelses angrep kan være målrettede (dvs. angriperen har spesifikk kunnskap om målmodellen) eller umålrettede (dvs. angriperen ønsker ganske enkelt å forårsake feilklassifisering).

Motstridende angrep er bekymrende fordi de kan føre til at dyplæringsmo-

deller gir uriktige spådommer med potensielt alvorlige konsekvenser. For eksempel kan et motstridende angrep på en selvkjørende bils objektdeteksjonssystem føre til at bilen feilidentifiserer en fotgjenger som en lyktestolpe, noe som fører til en kollisjon.

Det er flere metoder for å forsvare seg mot motstridelses angrep, for eksempel motstridende trening, som innebærer å omskolere modellen på motstridende eksempler[9]; inndataforbehandling, som innebærer å filtrere ut motstridende forstyrrelser før dataene mates inn i modellen; og motstridende deteksjon, som innebærer å oppdage når en inngang kan være et motstridende eksempel og avvise den. Det er imidlertid ikke noe sikkert forsvar mot motstridelses angrep, og våpenkappløpet mellom «hackere» og sikkerhetsutviklere utvikler seg fremdeles[29].

1.2.8 DDoS AI angrep

Hensikten med denne bacheloren er å utforske den nyoppdagete form for DDoS lignende angrep mot AI. DDoS AI angrep bygger oppå den tidligere nevnte motstridelsesangrep metoden hvor vi forsiktig manipulerer data til å forstyrre en dyplæringsmodell, men hvor istedenfor å produsere data som «manipulerer» en modell til å utføre visse aksjoner velger vi heller å produsere data som sløser bort prosseserngskraft. Angrepet går ut på å skape en så høy «neural activation coverage» som mulig ved å produsere kompleks data gjennom NSGA som vil aktivere så mange neuroner i modellen som mulig, som i gjengjeld vil tvinge dyplæringsmodellen til å bruke mer prosesseringskraft. Målet er å utforske om det er mulig å redusere et systems prosesseringskraft gjennom å tilføre dyplæringsmodellen så mye kompleks data at prosseseringskraft ikke kan bli brukt til noe som helst annet, og dermed fornekte tjenester til andre brukere.

1.2.9 Internet of Things

IoT står for Internet of Things. Det refererer til det breie nettverket av sammenkoblede enheter og objekter som kommuniserer og utveksler data med hverandre over internett uten menneskelig innblanding[4]. Disse enhetene, ofte referert til som «smarte» enheter, er innebygd med sensorer, program-

vare og andre teknologier som lar dem samle inn og overføre data til andre enheter eller systemer.

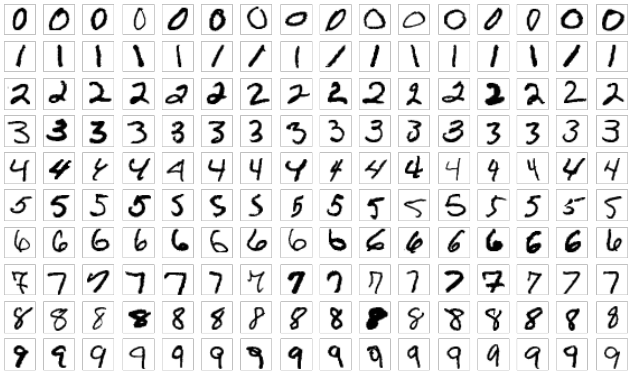
IoT enheter kan bli funnet i nesten hvert aspekt av våre hverdagsliv. I for eksempel industri kan du finne IoT enheter som sensorer og monitorer som brukes i fabrikker, varehus og logistikkoperasjoner for å samle inn data om utstyrshelse, lagernivåer, miljøforhold og andre driftsparametere for å optimalisere prosesser og forbedre effektiviteten[45]. I hjemme ditt kan man for eksempel finne enheter som smarte kjøleskap, smarte ovner, smarte vaskemaskiner og smarte oppvaskmaskiner som kan kan fjernstyres, overvåkes og optimaliseres for økt effektivitet og enkelt bruk. Enheter som Amazon Echo, Google Home eller Apple HomePod er stemmeaktiverte smarthøyttalere som bruker virtuelle assistenter som Amazons Alexa, Google Assistant eller Apples Siri for å kontrollere tilkoblede enheter, spille musikk, gi informasjon og utføre ulike oppgaver.

Dataene som samles inn fra disse IoT-enhetene kan analyseres for å få innsikt, automatisere prosesser og ta informerte beslutninger på ulike domener, noe som fører til forbedret effektivitet, enklere brukbarhet og produktivitet[4].

1.3 Tidligere arbeid

1.3.1 MNIST

MNIST er et standard datasett innen maskinlæring med 28x28 gråskala bilder av håndskrevne heltall fra 0 til 9. Hvert bilde er merket med et tall for å vise hvilket tall som faktisk er på bildet. Datasettet består av et treningssett og et testsett hvor treningssettet inneholder 60000 bilder, mens testsettet inneholder 10000 bilder [2]. MNIST gir en grunnlinje for testing av bildebehandlingssystemer og brukes ofte av dataanalytikere til å teste om nye rammeverk fungerer [1].



Figur 1.3: Eksempel av håndskrevne heltall fra MNIST datasettet

1.3.2 The TON IoT Datasets

The TON IoT Dataset er et nettverkstrafikk datasett produsert av USNW Sydney[31]. Datasettet produserer flere felt med standard nettverktrafikkinformasjon, som for eksempel en brukers «source/destination» ip adresse og hva type nettverkspakke det er. Datasettet inneholder også malisiøs nettverkstrafikk, og hensikten med dette datasettet er at man skal kunne utvikle en modell som greier å se forskjellen på trygg og malisiøs nettverkstrafikk utifra informasjonen providert i nettverkspakkene. Dataen er tatt ut fra målinger av flere forskjellige typer IoT enheter innen cybersikkerhet.

Datasettet inneholder flere forskjellige typer angrep som dyplæringsmodellene skal kunne oppdage. Disse angrepene er «scanning, dos, injection, ddos, password, xss, ransomware, backdoor, mitm», som alle er forkortelser for vanlige typer nettverksangrep. Ut i fra informasjonen i datapakkene som dyplæringsmodellen øver på, skal den kunne skille mellom hvilke pakker som er normale og hvilke pakker som er malisiøse.

	duration	src_bytes	dst_bytes	missed_bytes	src_pkts	src_ip_bytes	dst_pkts	dst_ip_bytes	dns_qclass	dns_qtype	dns_rcode	http_request_body_len	http_response_body_len	http_status_code	type
0	80549.530260	1762852	41933215	0	252181	14911156	2	236	0	0	0	0	0	0	0
1	0.000000	0	0	0	1	63	0	0	0	0	0	0	0	0	0
2	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0.000549	0	298	0	0	0	2	354	0	0	0	0	0	0	0
5	0.000000	0	0	0	1	63	0	0	0	0	0	0	0	0	0
6	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0.000000	0	0	0	1	63	0	0	0	0	0	0	0	0	0
11	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0.000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0.000000	0	0	0	1	63	0	0	0	0	0	0	0	0	0
14	0.000499	0	298	0	0	0	2	354	0	0	0	0	0	0	0

Figur 1.4: Eksempel av datasettet etter ferdigmodifisering med bare numeriske verdier

1.3.3 Keras

Keras er et åpenkildet programvare som providerer brukervennlig ANN verktøy i Python[11]. Keras tillater rask og effektiv eksperimentering med nevrale nettverk, og providerer grunnfunksjonene vi trenger til vår eksperimentering som «layers» og egenskapen til å bearbeide data. Datamodellene vi bruker i vår eksperimentering er bygget på Keras sine redskaper.

Vi bruker python pakken Keras for å implementere både testmodellen vi bruker for å utvikle vår NSGA algoritme, samtidig som vi bruker det til å produsere modellen til hovedprosjektet vårt. Keras providerer blant det breieste utvalget av funksjoner innen dyplæringsmodeller innen python, dermed vill vi gå over de funksjonene vi bruker slik at det er lettere å forstå utviklingen av vår modell[23].

Dense layer Dette er det vanligste layeret i nevrale nettverk. Det er et fullstendig sammenkobler «layer», hvor hvert nevron er sammenkoblet til alle andre nevroner i det forrige «layeret».

Parametere:

- `units`: Antall nevroner i «layeret»
- `activation`: Aktiveringsfunksjonen som skal brukes på utgangen av laget.
- `use_bias`: En boolsk verdi som bestemmer om «layeret» skal inkludere «bias terms»

Convolutional layer Dette laget brukes til bildebehandlingsoppgaver, der inngangen er et bilde eller et volum. Dette er viktig for MNIST bildeprosesseringsmodellen vår.

Parametere:

- `filters`: Antall filtere å bruke i «layeret»
- `kernel_size`: Størrelsen på filterene
- `strides`: Antall steg til konvolusjonsoperasjonen.
- `padding`: Den type «paddingen» eller belegg som skal brukes for konvolusjonsoperasjonen.
- `activation`: Aktiveringsfunksjonen som skal brukes på utgangen av laget.

Pooling layer Dette «layeret» brukes til å redusere de romlige dimensjonene til inputtet.

Parametere:

- `pool_size`: Størrelsen på «pooling» vinduet.
- `strides`: Fremskrittet i sammenslåingsoperasjonen.
- `padding`: Polstringen som skal brukes til sammenslåingsoperasjonen.

Recurrent layer Dette laget brukes til sekvensielle databehandlingsoppgaver, der input er en sekvens av data.

Parametere:

- `units`: Antall nevroner i laget.
- `activation`: Aktiveringsfunksjonen som skal brukes på outputen av «layeret».
- `return_sequences`: En boolsk verdi som indikerer om laget skal returnere hele sekvensen av utdata eller bare den siste outputen.

Dropout layer Dette laget brukes til å forhindre overtilpasning ved tilfeldig å slippe ut noen av nevronene under trening.

Parametere:

- `rate`: Brøkdelen av inputenhetene som skal bli droppet.

BatchNormalization layer Dette «layeret» er brukt til å normalisere outputen av det forrige «layeret».

Parametere:

- `momentum`: Momentumet som brukes for det glidende gjennomsnittet.
- `epsilon`: Verdien lagt til variansen for å unngå divisjon med null.

Flatten layer Dette laget brukes for å omforme utgangen fra det forrige laget til en flat vektor. Denne type «layer» har ingen ekstra konfigurerbare parametere

Dette er de mest brukte lagene i Keras, men det er mange andre lag tilgjengelig også. Når du oppretter et lag i Keras, kan du spesifisere parametrene for å tilpasse oppførselen til din spesifikke modellen.

Aktiveringsfunksjoner/ReLu aktivering

En ReLu («Rectified Linear Unit») aktiveringsfunksjon er en matematisk funksjon som brukes i kunstige nevrale nettverk. Det er den aller mest vanlige, og i fleste situasjoner beste, aktiveringsfunksjonen brukt inni et nevralt nettverks «hidden layer»[32]. Det er en veldig enkel funksjon som i hovedsak erstatter eventuelle negative verdier i inngangen med null, mens positive verdier forblir uendret.

For å si det enkelt, forestill deg at du har en haug med tall, noen positive og noen negative. Når du bruker ReLU-aktivering på disse tallene, blir eventuelle negative tall null, mens de positive tallene forblir de samme. Så hvis du har tallet -3, vil ReLU-aktivering gjøre det til 0, men hvis du har tallet 5, vil ReLU-aktivering la det være 5[18].

ReLU-aktivering brukes ofte i nevrale nettverk fordi det bidrar til å løse problemet med «forsvinnende gradienter», som kan oppstå ved bruk av andre aktiveringsfunksjoner. I hovedsak oppstår forsvinnende gradienter når gradienten (eller endringshastigheten) til aktiveringsfunksjonen blir veldig liten ettersom inngangen blir veldig stor eller veldig liten. Dette kan gjøre det vanskelig for nettverket å lære og komme med nøyaktige spådommer. ReLU-aktivering bidrar til å unngå dette problemet ved å sikre at gradienten alltid er enten 0 eller 1, noe som gjør det lettere for nettverket å lære og lage nøyaktige spådommer[22].

Etter testing med flere forskjellige typer aktiveringsfunksjoner har modellene våre blitt justert til å bare bruke ReLu aktivering i våre «hidden layers», etter dette providerte best resultat.

Modelltrening

Keras fit()-funksjonen brukes til å trene et nevralt nettverksmodell på et gitt datasett[11]. Den tar flere argumenter, inkludert treningsdata (x), måldata (y), batchstørrelse, antall epoker og valideringsdata (valgfritt).

Her er en oversikt over hvordan fit()-funksjonen fungerer:

1. Treningsdata og måldata blir sendt til modellen som inndataargumenter.

2. Dataene blir delt opp i bunter av størrelse som er spesifisert av batch_size-argumentet.

3. Modellens vekter blir oppdatert iterativt for det angitte antallet epoker. I hver epoke gjør modellen prediksjoner på treningsdataene og beregner tapfunksjonen basert på den predikerte utgangen og den faktiske utgangen. Optimalisereren brukes deretter til å oppdatere vektene til modellen for å minimere dette tapet.

4. Hvis validation_data-argumentet er angitt, evalueres modellens ytelse på denne datamengden ved slutten av hver epoke. Dette lar deg overvåke modellens fremgang og oppdage om den overtilpasser eller undertilpasser.

5. Etter at alle epoker er fullført, returnerer fit()-funksjonen et History-objekt som inneholder informasjon om treningsprosessen, for eksempel tap og nøyaktighet ved hver epoke.

Her er et eksempel på å bruke fit()-funksjonen til å trene et enkelt nevralt nettverk:

Kode 1.1: Eksempel av en dyplæringsmodell

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Trening av modellen
history = model.fit(x_train, y_train,
                    batch_size=32,
                    epochs=10,
                    validation_data=(x_val, y_val))
```

I dette eksemplet trener vi et enkelt nevralt nettverk med ett skjult lag og ett utgangslag. Vi bruker Adam-optimalisatoren, binær kryssentropitap og nøyaktighet som evalueringsmetrikker. Vi trener også modellen i 10 epoker med en batchstørrelse på 32 og validerer modellen på en valideringsmengde av `(x_val, y_val)`. Til slutt lagrer vi treningshistorikken i et History-objekt kalt `history`.

1.3.4 NSGA

NSGA («Nondominated Sorting Genetic Algorithm») er en populær multi-objektiv optimaliseringsalgoritme som ofte brukes til å gjøre evolusjonære beregninger. Den ble introdusert av Kalyanmoy Deb, en anerkjent forsker som arbeider for forskningsinstituttet IEEE innen evolusjonær beregning, i 2002 som en forlengelse av den klassiske genetiske algoritmen (GA) for å løse problemer med flere motstridende mål[14].

«Non-dominated Sorting Genetic Algorithm-II» (NSGA-II) er en type evolusjonær algoritme som brukes til å løse optimaliseringsproblemer. Evolusjonære algoritmer etterligner evolusjonsprosessen av naturlig utvalg som vi ser i naturen, hvor de mest spreke subjekter fortsetter å reprodusere. I vårt tilfelle fungerer NSGA på den måten med at programmet tar den type data som produserer høyest «neural activation coverage» og får den dataen til å videreprodusere flere av den type data med høyere «neural activation coverage». NSGA applikerer tilfeldig støy på datapakkene og reagerer på hva gjenkjenningsprogrammet ser på som kompleks, og videreutvikler disse datapakkene. Hva som skiller NSGA-II-algoritmen ut fra de forrige genetiske algoritmene er at den produserer «non-dominated» løsninger, som betyr den bare returnerer de resultatene som dominerer absolutt alle andre resultat. [14] Til innspill for resultatet faktoriseres f.eks befolkningsstørrelse, antall generasjoner, objektiv funksjon og nedre og øvre grenser av beslutningsvariabler.

For å implementere NSGA algoritmen i python bruker vi pythonpakken Pymoo sin NSGA2() funksjon[8]. Algoritmen tar inn flere parametere som styrer dens oppførsel og søkestrategi. Dermed er det viktig for oss å lage en oversikt over de viktigste parameterene vi bruker for å kunne forklare modellen vår.

pop_size Denne parameteren bestemmer størrelsen på populasjonen av løsninger som algoritmen opprettholder gjennom søkeprosessen. Det er i hovedsak antallet individer i populasjonen.

offsprings Denne parameteren kontrollerer antall nye løsninger, eller lettere sagt «barn», generert av de genetiske operatørene (kryss og mutasjon) i hver iterasjon. Det totale antallet nye løsninger som genereres i hver iterasjon vil være $pop\_size + offsprings$.

sampling Denne parameteren bestemmer prøvetakingsmetoden som brukes til å generere den første populasjonen av løsninger, eller lettere sagt bestemme hvilke data man skal begynne å pare med hverandre for å lage «barn». Standardverdien er «random», som betyr at løsninger genereres tilfeldig innenfor søkeområdets grenser. Andre alternativer inkluderer «latin_hypercube» og «real_random».

crossovers Denne parameteren bestemmer crossover-operatøren som brukes til å generere barn fra foreldreløsninger. Standardverdien er «simulert binær crossover» (SBX) operatør, men andre alternativer inkluderer «polynomial» og «uniform».

mutation : Denne parameteren bestemmer mutasjonsoperatøren som brukes til å modifisere avkomløsninger. Standardverdien er operatøren «polynom mutasjon», men andre alternativer inkluderer «bitflip» og «random».

1.3.5 Keract

Keract er et python-bibliotek laget av Philippe Remy i 2020 som finner aktiverings- og gradientverdiene for nevronene i hvert «layer» for en Tensorflow/Keras modell gitt noe inndata [36]. Biblioteket gjør at vi bare trenger å bruke en funksjon for å få aktiveringsverdiene som vi trenger til å kalkulere «neural activation coverage».

Kapittel 2

Prosess

2.1 Metode

Denne utforskingen sikter på å utvikle en dyplæringsmodell med egenskapen til å utvikle kompleks nettverkstrafikk dedikert til å forvirre dyplæringsmodeller som gjenkjenner nettverksangrep, med hensikt til å få modellen til å sløse bort så mye prosesseringskraft som mulig. For å utvikle denne modellen blir prosjektet delt opp i faser. Til å begynne utvikler vi en testmodell som vi kan bruke som grunnlag til å bygge vår NSGA-II algoritme. Vi starter med å lage et simpelt bildegjenkjenningsprogram ut i fra det tidligere nevnte MNIST bildedatasettet. Hensikten med dette er å teste ut NSGA algoritmens egenskap til å manipulere data og hvor stor innflytelse dette har på en simpel modells prosesseringskraft, som vi deretter kan bruke som grunnlag for når vi arbeider med den mer komplekse modellen. Gjennom NSGA utvikler vi en metode for å generere komplekse bilder med høy «neural activation coverage» som sløser bort prosseseringskraften til bildegjenkjenningsprogrammet. Fase to vil bestå av å utføre den samme prosessen, men denne gangen med det mer relevante ToN IoT datasettet.

For å programmere implementasjonen vår valgte vi å bruke Python. Python er et kraftig programmeringsspråk som kan brukes til å definere modeller og visualisere analyse. En av hovedgrunnene Python er et så mektig programmeringsspråk er på grunn av dets breie bibliotek med nedlastbare

programpakker. Et bredt utvalg av pakker er tilgjengelig for å hjelpe med utviklingen av dyplæringsmodeller, inkludert TensorFlow, Pandas, NumPy, Sci-Kit Learn, SciPy, Keras, Pymoo, Tqdm og Matplotlib. Disse pakkene kan brukes til å lage og trene modeller, utføre statistiske analyser, lage datavisualiseringer og mer. Med disse pakkene er det mulig å analysere trender, mønstre og innsikt i data som ville være vanskelig å avdekke uten bruk av Python. Python inkluderer også en simpel implementasjon av NSGA-II algoritmen blant deres programpakker, som gjør det enkelt å arbeide sammen med modellene og NSGA-II algoritmen.

2.2 Forbehandling av data

2.2.1 MNIST

Forbehandlingen av MNIST datasettet er hentet fra Keras selv[12] og er relativt enkelt ettersom MNIST er ett mer simpelt datasett. Først blir datasettet direkte lastet ned fra Keras sine hjemmesider og splittet inn i ett trening og test datasett. Deretter normaliserer vi pikslene i bildene og utvider den innerste dimensjonen med 1 for å vise at bildene er gråskala. Tilslutt trenger vi bare å «one-hot encode» svaralternativene 0 til 9, for å konvertere dem fra tekst til tall. «One-hot encoding» vil si å konvertere kategorier til binære vektorer hvor ett element er «hot» (1), mens resten er «cold» (0).

Kode 2.1: Forbehandling av MNIST datasettet.

```
num_classes = 10

(x_train, y_train), (x_test, y_test) = ...
    keras.datasets.mnist.load_data()

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

2.2.2 ToN IoT

ToN IoT datasettet er mer komplekst og inneholder mye mer data enn MNIST datasettet. Dette gjør at datasettet krever mer forbehandling. I vår dyplæringsmodell valgte vi å bruke feltet «type» til å være variabelen som modellen skal forutsi. Feltet har verdier som sier om nettverkspakken er normal altså ikke malisiøs, eller om den er en type nettverksangrep for eksempel DDoS eller XSS.

Det første vi gjør med datasettet etter å lese det inn er å fjerne alle feltene som vi ikke skal bruke i vår dyplæringsmodell. I koden under 2.2 er deler av denne linjen utelatt. Deretter skifter vi ut verdiene for feltet «type» med tall fra 0 til 9 før vi tar «one-hot encoding» på listen. Grunnen til at vi bruker «one-hot encoding» her selv om teskten allerede er konvertert til tall er for å gjøre listen om til en matrise slik at den har rett form for dyplæringsmodellen. Tilslutt deler vi bare å dele datasettet inn i et trenings og testdatasettet slik som for MNIST datasettet.

Kode 2.2: Forbehandling av ToN IoT datasettet.

```
ton_iot = pd.read_csv("Data/ToN_IoT.csv")

ton_iot = ton_iot.drop(['ts','src_ip','src_port','dst_ip',

ton_iot['type'] = ton_iot['type'].replace(['normal', ...
    'scanning',

y = ton_iot.type.values

ton_iot.drop("type", axis=1, inplace=True)

x = ton_iot.values

y = to_categorical(y)

x_train, x_test, y_train, y_test = train_test_split(x, y, ...
    test_size=0.2, random_state=20)
```

2.3 Kalkulering av neural activation coverage

Som tidligere nevnt kalkulerer «neural activation coverage» hvor mange nevroner som ble aktivert av modellen for å prosessere dataen som blir satt inn. For å kalkulere «neural activaton coverage» bruker vi python-biblioteket keract[36] til å finne aktivasjonsverdiene til nodene. Så tar vi vekk inndata «layeret» ettersom vi bare er interessert i aktivasjonsverdiene til «hidden layers». Deretter summerer vi opp alle nevronene som har aktivasjonsverdi større enn 0, det vil si alle aktiverte nevroner, og deler på totalt antall nevroner. Hensikten med å måle «neural activation» coverage er for å sammenligne om de komplekse inndataene produsert av NSGA aktiverer en betydelig forskjell i aktiverte noder enn hva den vanlige inndataen gjør. En bemerkelig forskjell i antall noder som blir aktivert vil bety at angrepsmetoden vi studerer vil fungere og at modellen sløser bort betydelig mer prosesseringskraft.

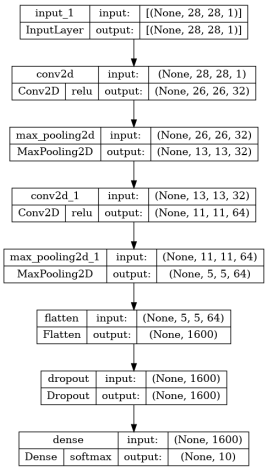
Kode 2.3: Funksjon for å kalkulere neural activation coverage

```
1
2       activations = get_activations(model, input_data, ...
            auto_compile=True)
3       del activations["input_1"]
4
5       total_neurons = 0
6       non_zeros = 0
7       for value in activations.values():
8           total_neurons += value.size
9           non_zeros += np.count_nonzero(value)
10
11      return non_zeros / total_neurons
12
13  nac = Neuron_Activation_Coverage(model, x_test[0])
14
15  print("NAC: ", nac)
```

2.4 AI modell

2.4.1 MNIST

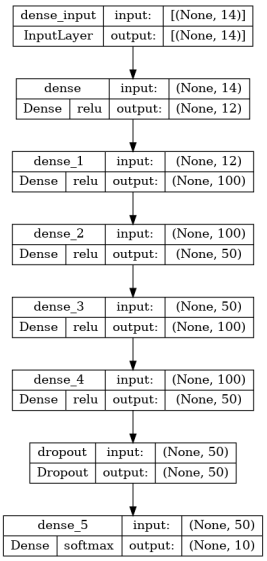
Modellen vi brukte for testing av NSGA algoritmen er en simpel Keras sekvensiell modell. Modellen bruker «convolutional» nevralt nettverk arkitektur Modellen er bygd på ett av Keras sine eksempel modeller fra deres hjemmeside[12]. Modellen bruker ett inndata «layer», med fem «hidden layers» som består av to «Pooling layers» med «pool size» (2, 2), to «Convolutional layers» hvor ett har 34 nevroner og ett har 64 nevroner, og ett «Flatten layer». Begge «Convolutional layersene» bruker ReLu aktiveringer og har kjerne størrelse på (3, 3). Modellen har også ett «dropout layer» med 0,5 i «dropout», og en «softmax-aktivering» for output «layeret».



Figur 2.1: Sammendrag av MNIST AI modellen.

2.4.2 ToN IoT

Modellen som blir implementert er en stokastisk sekvensiell Keras modell. Modellen er et «feedforward» nevralt nettverk med «multilayer perceptron» arkitektur. Modellen består av seks «dense layers» som blir både brukt til inndata, «hidden» og output «layer», og ett «dropout layer» blant «hidden layersene». Alle «dense layersene» bruker ReLu aktiveringer, untatt «output layeret» som bruker «softmax-aktivering». Inndata «layeret» har 12 nevroner og inndataen har formen til en vektor med størrelse 14. De fire «dense layersene» i «hidden layeret» består av to «dense layers» med 100 nevroner og to med 50 nevroner. «Dropout layeret» har 0,5 i «dropout» og output «layeret» består av 10 nevroner. For å trene modellen, brukes Adam «optimizer» sammen med «categorical cross-entropy» tapsfunksjonen. Modellen utføres og er trent med «early callbacks» aktivert.



Figur 2.2: Sammendrag av ToN IoT AI modellen.

2.5 NSGA

«Non-Donimated Sorting Genetic Algorithm» (NSGA) bruker vi for å legge støy til dataen for å gjøre den mer kompleks. Mer kompleks data skal produsere et høyere «neural activation coverage» og dermed krever mer prosesseringskraft enn den vanlige dataen. Det vil da være et tegn på at angrepsmetoden vi undersøker kan fungere.

For å implementere NSGA algoritmen trenger vi å lage problemklassen pymoo gir oss tilgang til. Problemklassen tar inn data, for MNIST er dette ett av MNIST bildene, mens for ToN IoT datasettet er det en nettverkspakke. Deretter vil NSGA automatisk generere en vektor, som for oss er støyvektoren. NSGA har en valgfri parameter for begrensninger som vi bruker for å begrense maksverdien i støyvektoren. Uten begrensningen vil NSGA legge til veldig mye støy som vil føre til at i et realistisk scenario så vil automatiske sikkerhetssystemer oppdage nettverkspakken og droppe den. Videre så legger vi støyvektoren til inndataen og kalkulerer «neural activation coverage» til den nye forstyrrede dataen. NSGA har bare en funksjon for å minimere resultatet, men vi ønsker å maksimere «neural activation coverage» til

dataen, det gjør at vi må snu om på problemet fra et maksimerings til et minimeringsproblem. Dette er grunnen til at vi tar $1.0/nc$ i kodelisting 2.4. Når vi kjørte selve algoritmen brukte vi samplingmetoden «FloatRandomSampling», «crossover SBX» og «mutation PolynomialMutation». Etter å ha iterert gjennom algoritmen flere ganger basert på populasjonsstørrelse og antall generasjoner, vil NSGA returnere den støyvektoren som økte «nerual activation coverage» mest for gitt inndata.

Kode 2.4: Implementasjon av problemklassen til NSGA for MNIST

```
1  class NCMax(ElementwiseProblem):
2      def __init__(self, input_data):
3          super().__init__(
4              n_var=784, n_obj=1, n_constr=1, xl=0.0, xu=1.0)
5          self.input_data = input_data
6
7      def _evaluate(self, x, out, *args, **kwargs):
8          x = np.reshape(x, (28, 28, 1))
9          perturbed_input = self.input_data + x
10         nc = Neuron_Activation_Coverage(model, ...
               perturbed_input)
11         ret_val = 1.0 / nc
12         out["F"] = ret_val
13         constr = x.max() - 0.8
14         out["G"] = constr
```

2.5.1 ToN IoT

Implementasjonen av NSGA for ToN IoT er veldig lik den for MNIST. Det er noen forskjeller slik at problemklassen passer for datasettet sin størrelse og form. Den viktige forskjellen er at vi bare vil legge støy til feltene for «source», ellers ville det igjen vært veldig enkelt for et sikkerhetssystem å merke at det er et angrep og dermed droppe nettverkspakken. Derfor legger vi bare støy til spesifikt feltene «src_bytes», «src_pkts», «src_ip_bytes» og «http_request_body_len».

Kode 2.5: Legger støy til spesifikke felter i nettverkspakken.

```
1           perturbed_input = np.copy(self.input)
2           perturbed_input[1] += x[0]   # src_bytes
3           perturbed_input[4] += x[1]   # src_pkts
4           perturbed_input[5] += x[2]   # src_ip_bytes
5           perturbed_input[11] += x[3]  # http_request_body_len
```

Kapittel 3

Resultater og diskusjon

I denne oppgaven har vi undersøkt en potensiell ny type DDoS angrep mot IoT baserte AI applikasjoner for å utmatte deres GPU/CPU ressurser. Dette ble gjort ved å lage dyplæringsmodeller for to datasett, MNIST og ToN IoT. For å ha en måling på hvor mye ressurser som blir brukt, brukte vi metrikken «neural activation coverage». Deretter brukte vi NSGA algoritmen for å lage nye komplekse data i håp om at de skaper en høyere «neural activation coverage» som vil bety at det kreves mer GPU/CPU ressurser.

3.1 Eksperimentelt oppsett

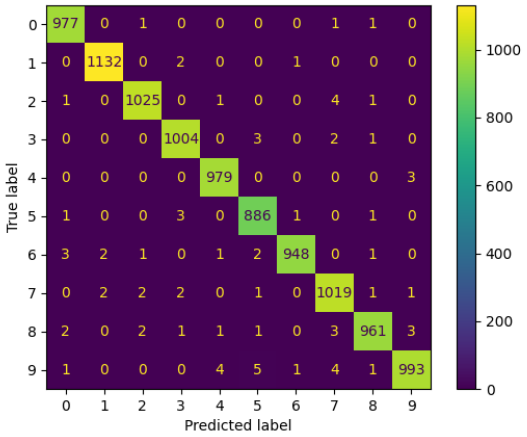
All testing ble gjort på en og samme PC. PC-en kjører Windows 11 og tabellen 3.1 viser spesifikasjonene til PC-en. Enheten har en Intel Core i7-10700KF prosessor med 8 kjerner og en klokkehastighet på 3.8GHz. Den har også totalt 32GB DDR4 ram med en hastighet på 3600MHz. GPU-en er en AMD Radeon RX 6700 XT med klokkehastighet på 2000MHz og 16GB minne.

Komponent	Spesifikasjon
OS	Microsoft Windows 11 Pro
CPU	i7-10700KF @ 3.80GHz
GPU	AMD Radeon RX 6700 XT
RAM	2x16GB DDR4 3600MHz

Tabell 3.1: Spesifikasjoner til PC-en brukt i testingen.

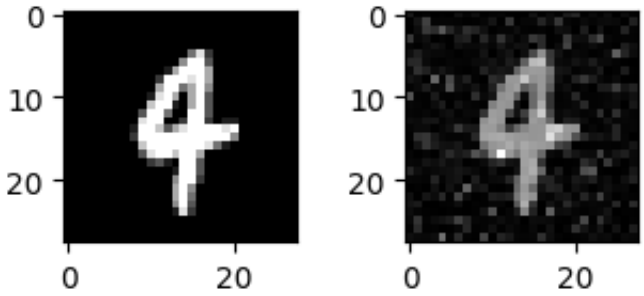
3.2 MNIST datasett analyse

MNIST dataopplæring og evaluering av tapsfunksjonen er en måte å måle hvor godt en modell er i stand til å lære fra data. Treningstapet av 0.032 indikerer at modellen er i stand til å lære datasettet nøyaktig. Tapstesten med målingen av 0.024 antyder at modellen er i stand til å generalisere treningsdataene og kan nøyaktig forutsi nye data. Treningsnøyaktigheten på 0,990 og testnøyaktigheten på 0,992 som tilsvarer respektivt 99.0% og 99.2% antyder at modellen er i stand til å nøyaktig forutsi effekten av datasettet. Dette indikerer at modellen gir gode resultater og er i stand til å lære og forutsi dataene nøyaktig. Figur 3.1 viser forvirringsmatrisen for vår AI modell på MINST datasettet. X-aksen er modellen sin antagelse og y-aksen er den korrekte verdien. Forvirringsmatrisen viser at modellen er i stand til å veldig ofte forutsi den riktige verdien. Modellens høye nøyaktighet gjør det til et dyktig program å teste NSGA-algoritmen på.



Figur 3.1: Forvirringsmatrise for Ai modellen på MNIST datasettet.

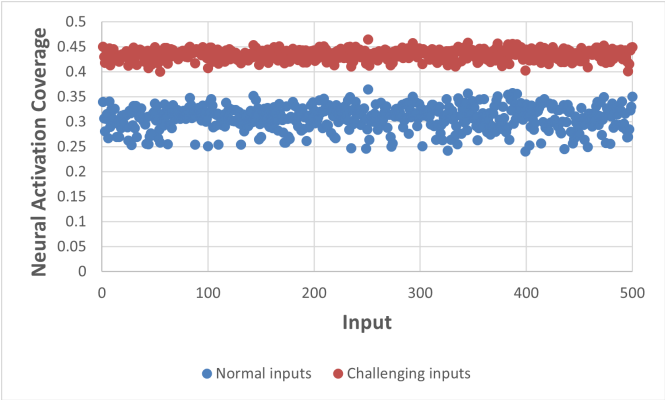
NSGA algoritmen tester et sett med bilder fra MNIST datasettet for å finne ut hvilke bilder som naturlig har høy «neural activation coverage». Dermed vil den runde ner til et smalt sett med bilder den lærer er komplekse, og begynne å manipulere disse bildene for å generere en høyere «neural activation coverage». I eksempelet nedenfor 3.2 fant NSGA ut at en av de mest optimale måtene å forvirre bildegjenkjenningsprogrammet på var gjennom å legge til gråtoning ovenfor tallet fire og legge til mye støy i bakgrunnen.



Figur 3.2: MNIST bilde før og etter NSGA manipulering.

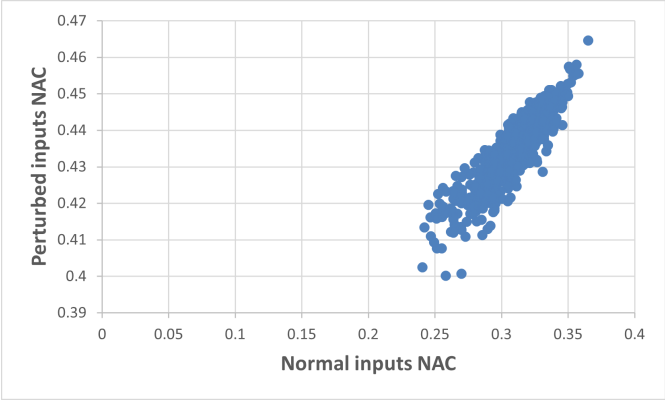
For å teste MNIST regnte vi ut «neural activation coverage» før og etter

NSGA legger til forstyrrelser på 500 tilfeldig valgte bilder. Målingen av MNIST datasettes «neural activation coverage» før NSGA bildemanipulering anviser en lav måling på gjennomsnittlig 0.30, noe som antyder at datasettet bruker lite prosesseringskraft. Noen av bildene fikk en «neural activation coverage» helt ned til 0.24, mens andre gikk opp til 0.36, som viser at det var litt variasjon fra bilde til bilde. Etter modellen ble injisert med komplekse bilder produsert av NSGA, ser vi en økning på 43% i modellens prosesseringsforbruk til en gjennomsnittlig «neural activaton coverage» måling på 0.43. Med en minimums måling på 0.40 og maksimum nå på 0.46 ser vi at variasjonen mellom minimum og maksimum etter bruk av NSGA har minket. I figur 3.3 kan man se et punktdiagram hvor y-aksen er «neural activation coverage» og x-aksen er inndataen. De blå punktene er «neural activation coverage» før bruk av NSGA, mens de rød er etter. Av diagrammet kan man se hvordan variansen har minket etter bruk av NSGA.



Figur 3.3: NAC før og etter bruk av NSGA plottet mot inndataen.

Det virker som at NSGA klarer å øke «neural activation coverage» mer for bilder som starter med en lavere verdi. Figuren 3.4 er enda et punktdiagram men denne gangen har vi plottet «neural activation coverage» etter bruk av NSGA på y-aksen og før bruk av NSGA på x-aksen. Y-aksen her starter på 0.39, det er siden den lavesete observerte verdien var rundt dette og det lar oss bedre se forskjellene i «neural activation coverage». Fra figuren kan vi se at x-aksen vokser mye raskere enn y-aksen, det indikerer igjen at NSGA virker å ha større effekt på bilder som starter med en lavere «neural activation coverage». I tillegg kan vi se av figuren at de høyeste målingene kommer fra de dataene som også har høyest måling før bruken av NSGA.

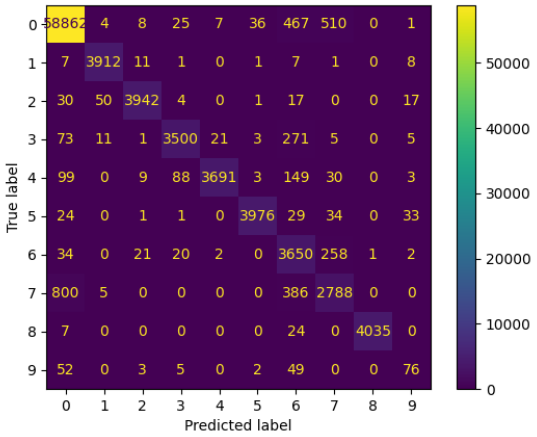


Figur 3.4: NAC etter bruk av NSGA plottet mot NAC før bruken av NSGA.

Generelt mener vi at det definitivt indikerer på å være et potensiale for sårbarhet innen angrep mot prosessoren selv i simple modeller. I noen tilfeller fikk vi en økning på rundt 70% i «neural activation coverage» som viser at de er muligheter for store økninger i nødvendig prosesseringskraft.

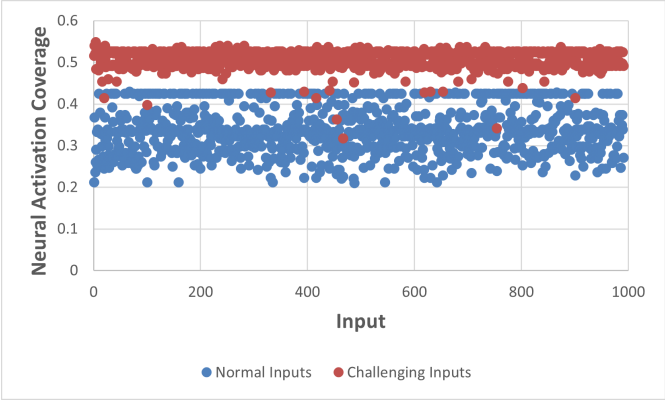
3.3 ToN IoT datasett analyse

Vår modell siktet på en nøyaktighet mellom 95-99% for ToN IoT datasettet. Dette er for å ha en så god som mulig modell slik at kalkulasjonen av «neural activation coverage» blir så realistisk som mulig. Etter lassevis med testing og justeringer oppnådde vi en tilfredstillende 96% nøyaktighet på vår modell sin egenskap til å gjenkjenne malisiøse angrep. Figur 3.5 under viser forvirringsmatrisen til dyplæringsmodellen. X-aksen viser modellen sin antagelse, mens y-aksen viser den faktiske verdien for gitt inndata, slik som for MNIST modellen. Fra dette kan man se at modellen generelt antar riktig men gjør noen feil som tilsvarer nøyaktigheten på rett rundt 96%. Mer interessant er at vi kan se hvor modellen gjør mest feil, for oss er det variablene 7 og 9 som tilsvarer respektivt «ransomware» og «mitm». I begge tilfellene gjør modellen en del ganger feilen ved å anta 0, altså at dataen er normal når den faktisk er malisiøs som i ett realistisk scenario er potensielt veldig farlig.



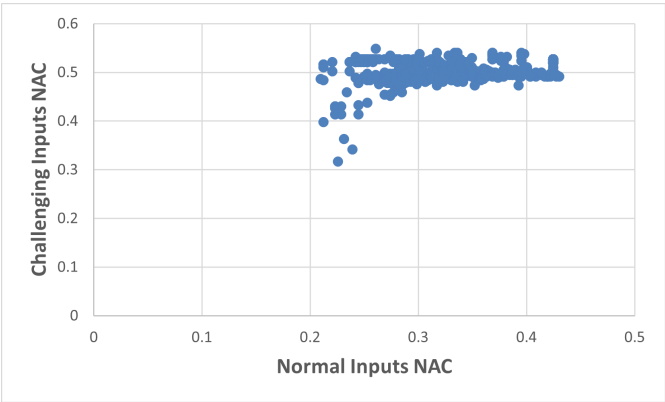
Figur 3.5: Forvirringsmatrise til AI modellen for ToN IoT datasettet.

Siden ToN IoT datasettet er veldig stort og inneholder mye data kan vi ikke realistisk kjøre NSGA på all dataen i datasettet. På grunn av det valgte vi å kjøre NSGA algoritmen på et tilfeldig utvalg av 1000 nettverkspakker. Fra disse målingene fant vi at datasettet sin «neural activation coverage» før manipulering fra NSGA er gjennomsnittlig 0.34 og den høyeste «neural activation coverage» var på 0.43, mens den minste målingen var helt nede på 0.21. Etter NSGA har blitt brukt for å manipulere dataen øker den gjennomsnittlige «neural activation coverage» med 54%, til en verdi på 0.50, en maks på 0.55 og en minimum på 0.32. Dermed kan vi se at den gjennomsnittlige økningen (0.16) var høyere enn den maksimale økningen (0.12). Dette tilsier at NSGA har større effekt på inndata som har lav «neural activation coverage» til å begynne med akkurat slik som det var med MNIST. I motsetning til MNIST så er minimumet etter NSGA veldig lav på noen spesifikke inndata som man kan se på figur 3.6. I tillegg kan man se at veldig mye av dataen før bruk av NSGA ligger på samme verdi rett over 0.40, som er unaturlig ettersom forskjellige data burde skape forskjellig «neural activation coverage». Dette for oss til å tro at resultatene for ToN IoT datasettet ikke er helt korrekt, og at vi mest sannsynlig har gjort noe feil underveis.



Figur 3.6: NAC før og etter bruk av NSGA plottet mot inndataen.

I figuren 3.7 under kan vi se sammenhengen mellom «neural activation coverage» før og etter bruk av NSGA. Her ser vi at NSGA er i stand til å øke «neural activation coverage» til rundt 0.50 uansett hva den er før bruk av NSGA, men at den gjør det mye oftere jo høyere «neural activation coverage» er før NSGA. Vi kan også se av figuren at veldig mye av dataen ender opp med samme «neural activation coverage» etter at NSGA har manipulert dem. Noe som igjen styrker tankene våre om at noe er galt.



Figur 3.7: NAC etter bruk av NSGA plottet mot NAC før bruken av NSGA.

Selv om det er noen merkelige resultater i dataen, så er det fortsatt noen av nettverkspakkene som produserer gode resultater og en sterk økning i

«neural activation coverage». I noen tilfeller økte den med rundt 140%, som allikevel får oss til å tro at det er muligheter for å øke prosesseringsmengden kraftig for en IoT basert AI applikasjon.

3.4 Diskusjon

Ut i fra våre resultater observerer vi at angrepsmetodens effektivitet varierer bemerkelig på hva type datasett og modell NSGA algoritmen blir utsatt for. En av våre antagelser før vi begynte vår modelltesting var at det ikke fantes noen grense på hvor mange nevroner NSGA kunne aktivere i en modell gjennom datamanipulering, men både vår MNIST testmodell og vår hovedmodell ville eventuelt nå en grense hvor den ikke sløste bort flere nevroner eller prosesseringskraft uansett hvor mye NSGA fortsatte å finjustere inndataen. Derimot hadde NSGA en bemerkelig effekt på «neural activation coverage» for alle modellene den ble utsatt for. På MNIST modellen kunne prosesseringskraften øke i gjennomsnittet 43% gjennom datamanipulering, mens i den mer komplekse ToN IoT datamodellen fløt prosseseringsøkningen rundt 54% etter datamanipulering. Hvor stor innflytelse en slik økning i prosesseringskraft vill ha på en modell sin tjenesteytelse vil variere på hva datasystem spesifikasjoner modellen kjører på, men kan ha en destruktiv effekt dersom et tjenestesystem bare er spesifisert til å takle rutinelig data og er dårlig justert til å håndtere noe som helst økning i prosesseringskraft.

Basert på resultatene fra ToN IoT datasettet hvor veldig mye av dataen hadde lik «neural activation coverage» før bruk av NSGA og fortsatt lik etter, mener vi at det er noe som ikke stemmer. Gjennom mye testing av dataforbehandling, AI modell og NSGA klarte vi ikke å finne utav helt hva som var galt og dermed heller ikke hvordan å fikse det. Men vi mener at feilen ligger i sammenhengen mellom forbehandlingen av dataen og AI modellen. Likevel er NSGA manipuleringen bevist til å være effektiv i å øke modellens prosesseringskraft, selv med feildata i datasettet eller modellen. Over 98%ãv all inndata produsert av NSGA algoritmen produserte høyere «neural activation coverage» enn hva selv datapakkene med feildata inni kunne gjøre, som også kan ha en effekt dersom systemet modellen kjører på ikke er justert til å takle en økning i prosesseringskraft.

Over alt er vi skuffet at dataen vi brukte endte opp med en del feildata, men

ut i fra sammenligning med de fungerende datapakkene observeres en åpenbar forskjell mellom de naturlige og manipulerte datapakkene. Dette viser at angrepsmetoden har en synlig effekt på modellens prosesseringskraft.

Kapittel 4

Konklusjon

Ved et DDoS angrep mot en IoT basert AI applikasjon gis inndata til en maskinlæringsmodell (ML) med hensikt til å få modellen til å sløse bort så mye prosesseringskraft som mulig. Disse angrepene er malisiøse forsøk på å villede og forvirre maskinlæringsmodeller. En angriper kan f.eks. endre et bilde på en måte som gjør at et datasynssystem ikke greier å gjette hva det er meningen bildet skal representere, og dermed tvinger datasynsystemet til å sløse så mye prosesseringskraft som mulig for å prøve å gjette seg fram til riktig svar. ML-modeller sine hovedfunksjoner kan bli satt ut av bruk gjennom DDoS AI angrep, og kan dermed bli videremanipulert til å få tilgang til sensitive data eller forstyrre systemets operasjon.

Vårt papir utforsker hvor effektivt denne nyoppdagede angrepsmetoden kan bli utført gjennom offentlig tilgjengelige dyplæringsprogrammer og evolusjonære algoritmer, spesifikt NSGA-II. Vi begynte med å utvikle en bildelæringsmodell for å utvikle NSGA algoritmen til en dataforstyrrende algoritme, og dermed brukte vi denne dataforstyrrende algoritmen på en kompleks nettverksangrep detektor modell.

Ett av våre hovedmål var å utforske om NSGA algoritmen ville ha noe som helst betydelig effekt på en modells prosesseringskraft. Gjennom testing på våre to modeller, ett for bildegjenkjenning og ett for nettverksangrep detektering, oppdaget vi at algoritmen vår kunne utgjøre en betydelig økning i hvor mye prosesseringskraft modellene brukte. Hvor mye prosesseringskraft

modellen brukte kom an på kompleksiteten av datasettet og hvor veltrent modellen var.

Et annet mål var å se om prosesseringskraft kunne maksimeres gjennom NSGA-II algoritmen. Dyplæringsmodeller hermer hjernens nevrale nodesystem, hvor jo flere noder blir aktivert jo mer prosesseringskraft blir brukt. Hensikten var å forsøke å få en modell til å aktivere hver eneste node den bruker til å analysere og prøve å bestemme riktig output med for å få modellen til å sløse bort så mye prosesseringskraft som mulig. Etter flere forsøk med justering og testing av vår algoritme kom vi fram til at hver modell har en grense hvor flere noder ikke kan bli aktivert uansett hva manipulering algoritmen vår gjør.

Ut i fra våre resultater kom vi fram til at angrepsmetoden kan ha en bemerkelig effekt på dyplæringsmodellenes prosesserngskraft, og at det bør implementeres metoder for å begrense hvor mange noder som kan bli aktivert etter hvor bra systemapparatet modellen kjører på er. Motstridelsesøving bør også bli tatt inn til konsiderasjon, hvor modellene øver på manipulert data for å kunne oppdage dem og ikke sløse bort prosesseringskraft på dem. Videreutforskning innen angrepsmetoden bør bli utført for å ikke utelate noen trusler.

4.1 Fremtidig arbeid

Etterforskningen vår var begrenset av at vi ikke hadde virkelige IoT enheter å teste på. Heller hadde vi ikke et bredt redskap av datamaskiner vi kunne bruke til å teste angrepsmetoden på. I optimale miljøer ville dyplæringsmodellen blitt angrepet av flere datamaskiner på samme tid. Dyplæringsmodellene var også simulert på samme enheter som manipulerngsalgoritmen ble kjørt på. Videreutforskning utenfor simulerte miljøer er kritisk for å kunne utvikle en bred forståelse innenfor denne angrepstypen og hva mulige konsekvenser det kan ha på IoT systemer, og hvordan vi forsikrer dem.

4.1.1 Etterforskningsspørsmål fra testing

Gjennom vår utforskning av dyplæringsmodellenes styrke mot DDoS AI angrep oppdaget vi at det virker som at modellene har en grense på hvor mange nevroner kan bli aktivert uansett hva form for videremanipulering NSGA-II algoritmen gjør på dataen. Et viktig spørsmål er hvorfor dyplæringsmodellene har disse grensene, og om det er mulig å manipulere dem til å gå forbi disse grensene. Et annet spørsmål vi ikke fant svar på var sammenhengen mellom modellens kompleksitet og dens «neural activation coverage» hadde noen sammenheng, som gjerne hadde vært et bra utforskningssubjekt for å finne ut hvilke typer modeller som er mest svake for denne type angrep.

4.1.2 Anbefalt DDoS AI angrepsforskning

Siden angrepsmetoden er relativt ny anbefales det videreutforskning innenfor angrepsmetoden. I vår egen utforskning brukte vi NSGA-II algoritmen for å teste om en offentlig tilgjengelig algoritme effektivt kunne øke en modells «neural activation coverage», videre bør det bli utforsket med andre evolusjonære eller spesiallaget algoritmer om hvilke som er best justert til å utføre angrep mot dyplæringsmodeller. Det bør også bli utført testing med et større sett av datamaskiner for å teste hvordan IoT enheter blir utsatt når de blir angrepet fra flere kanter.

4.1.3 Anbefalt sikkerhetsforskning

Ut i fra vår testing av hvordan dyplæringsmodeller reagerer på manipulert data anbefaler vi at det utvikles en metode eller funksjon for dyplæringsmodeller hvor brukere kan spesifisere en grense på hvor mange nevroner som kan bli aktivert før modellen velger å abortere prosessering av data for å unngå manipulering av prosesseringskraft. Dette kan også oppnås ved å utvikle en metode som gjør dette automatisk gjennom å adaptere seg til et systems spesifikasjoner. I tillegg anbefaler vi videre utvikling av motstridelsestrening («Adversarial training») metoden til å tilrettelegge for disse typer angrep. I motstridelsestrening blir modellen trent på både ren data og manipulert data. Under trening blir modellen eksponert for disse mani-

pulerte eksemplene og tvunget til å lære av dem i tillegg til ren data. Ved å gjøre dette blir modellen mer robust mot motstridelses angrep fordi den har lært å gjenkjenne og håndtere dem. Denne dyplæringsmetoden har stort sett blitt utilisert til å unngå manipulering av modellen til å få den til å utføre visse aksjoner, som for eksempel manipulering av en robot til å få den til å gjøre eller si noe den ikke skal. Vi anbefaler at motstridelseslæring blir videreutviklet til å oppdage data som er ment til å få modellen til å sløse prosesseringskraft.

Bibliografi

[1] Mnist dataset. `https://www.engati.com/glossary/mnist-dataset`.

[2] Mnist digits classification dataset. `https://keras.io/api/datasets/mnist/`.

[3] Michelle Adams, Christiaan Luminis, and Phil Butler. The global cybersecurity skills gap. *ISACA Journal*, 6:1–5, 2017.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. A survey on internet of things architectures. *IEEE Wireless Communications*, 18(1):26–36, 2011.

[5] Battista Biggio and Fabio Roli. Security and privacy challenges in machine learning. *Journal of Machine Learning Research*, 17(1):3937–3980, 2016.

[6] C. M. Bishop. Neural networks for pattern recognition. *Neural Networks for Pattern Recognition*, pages 1–482, 1995.

[7] Christopher M Bishop. Pattern recognition and machine learning. *Springer*, 2006.

[8] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[9] Tom B Brown, Dandelion Man'e, Aurko Roy, Mart'ın Abadi, and Justin Gilmer. Adversarial patch. In *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2842–2851. IEEE, 2018.

[10] Lei Chen, Jian Wang, Weifeng Zhou, and Shanchieh Jay Yang. Challenges and future research directions for cybersecurity analytics. *IEEE Transactions on Big Data*, 6(1):4–21, 2020.

[11] François Chollet et al. Keras. `https://keras.io`, 2015.

[12] François Chollet. Simple mnist convnet. `https://keras.io/examples/vision/mnist_convnet/`, 2019.

[13] Hung Q. Dang, Chen Zhang, Nan Zhang, Dejun Guo, and Jian Yin. A review of deep learning for cybersecurity. *IEEE Access*, 6:51829–51837, 2018.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2018.

[16] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[17] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Adversarial vulnerability of neural networks increases with input dimension. *arXiv preprint arXiv:1802.01421*, 2018.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 15:315–323, 2011.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. *MIT Press*, 1(2):1–800, 2016.

[20] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[21] Larry Greenemeier. Global business losses due to cybercrime top $400 billion annually. https://www.inc.com/will-yakowicz/cyberattacks-cost-companies-400-billion-each-year.html, 2018.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, 2015 International:1026–1034, 2015.

[23] Mohammad Shahadat Hossain and Ghulam Muhammad. A comprehensive survey on convolutional neural network architectures and learning paradigms. *Computers & Electrical Engineering*, 88:106872, 2020.

[24] KPMG International. The economic impact of cybercrime—no slowing down. *KPMG International Cooperative*, 2018.

[25] Sadiq Khan, Sandeep Kumar, and Sanjeev Kumar. Detection and mitigation of ddos attacks using machine learning techniques: A review. *IEEE Access*, 7:157181–157202, 2019.

[26] Kia Kokalitcheva. Why ai powered attacks represent a new threat to cybersecurity. *Fortune*, 2018.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[29] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2018.

[30] Jelena Mirkovic, Peter Reiher, and Lixia Wang. Attacking ddos at the source. *IEEE Network*, 18(1):22–29, 2004.

[31] Nour Moustafa. The ton_iot dataset. `https://research.unsw.edu.au/projects/toniot-datasets`.

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, 27(1):807–814, 2010.

[33] Yair Oren, Tanu Malik, and Sriram Subramanian. Hacking and cyber threats to critical infrastructure. *IEEE Security & Privacy*, 13(3):64–67, 2015.

[34] Honeynet Project. Know your enemy: Tracking botnets. `https://www.honeynet.org/papers/bots/`, 2004.

[35] Yifan Qian, Zhangyang Chen, Chao Liu, Zhanpeng Yang, Ting Liu, Li Xie, and Dawn Song Li. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 183–197. ACM, 2021.

[36] Philippe Remy. Keract: A library for visualizing activations and gradients. `https://github.com/philipperemy/keract`, 2020.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2015.

[39] Symantec. The cybersecurity talent gap. *Symantec*, 2019.

[40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[41] Jaikumar Vijayan. The cost of cybersecurity. *Communications of the ACM*, 61(6):16–18, 2018.

[42] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, Kevin J Lang, et al. Artificial neural networks applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(11):1629–1637, 1989.

[43] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[44] Xiaojun Yuan, Xiaojun Li, Pan He, Xiaolin Li, and Guojun Li. A survey of deep learning security: Threats and defense. *ACM Computing Surveys (CSUR)*, 52(5):1–37, 2019.

[45] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things (iot) applications and challenges in

smart cities: A review. *IEEE Internet of Things Journal*, 1(1):22–32, 2014.

Vedlegg A

Programlisting

Kildekode - mnist.7z
Kildekode - toniot.7z