

UNIVERSITETET I SØRØST NORGE

DATASYSTEMDESIGN

EN\_DAT2000-1 18V

---

# TPU og maskinlæring

---

*Forfatter*

Magnus HAUKEBØE

*Forfatter*

Sigurd HOLMEN

## **Innholdsfortegnelse**

<b>1</b>	<b>Innledning</b>	<b>2</b>
<b>2</b>	<b>Nevralt nettverk</b>	<b>3</b>
<b>3</b>	<b>TPU</b>	<b>6</b>
3.1	Unike trekk . . . . .	6
<b>4</b>	<b>Nyere Versjoner</b>	<b>10</b>
<b>5</b>	<b>Annen Hardware</b>	<b>11</b>
5.1	CPU . . . . .	11
5.2	GPU . . . . .	11
<b>6</b>	<b>Benchmarks</b>	<b>12</b>
6.1	Treningstid . . . . .	12
6.2	Treningstkostnad . . . . .	12
<b>7</b>	<b>Konklusjon</b>	<b>13</b>

## **1 Innledning**

CPU-er har med årene blitt raskere og økt antall beregninger den kan gjøre i løpet av en tidsperiode. Dette er på grunn av prosessor utviklere har vært i stand til å minske transistoren.

## 2 Nevralt nettverk

Dette er en teknikk innen maskinl ring som baserer seg p  hvordan nevroner i hjernen fungerer for   lære seg ting. Systemet lærer ved   analysere et sett med eksempler p  det den skal lære seg. I dette prosjektet s  er det kort som skal gjenkjennes p  bilder. Hele l reprosessen begynner med at kort p  bilder blir markert, men ellers s  har ikke algoritmen noen andre tegn   g  etter. Det finner algoritmen p  selv. Grovt sett s  er et nevron i dette tilfellet et knutepunkt som lager en vektlagt sum. Denne verdien blir da kj rt inn som verdien i en funksjon slik som bilde 1 viser. Algoritmen sjekker om det som kommer ut er det som er forventet eller ikke og juster vektene etter hvor mye som ble bommet p .

Flere nevroner kan kobles sammen i et nettverk, der input blir kj rt inn i flere

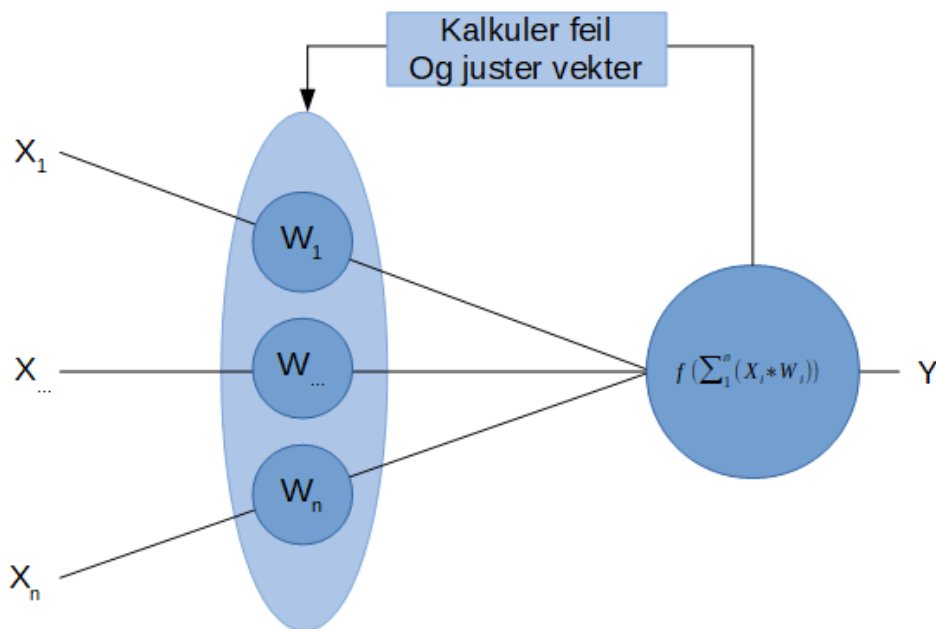


Figure 1: Enkel illustrasjon av et neuron

nevroner slik som bildet 2 viser. Hver nevron har da sine egne vektorer og resultatet blir da skj rt inn i nye nevroner. I et slikt nettverk kan det v re flere skjulte lag som gj r det samme som den f rste raden p  bildet. Til slutt blir det tatt en beslutning basert p  den siste rekken av nevroner. Denne beslutningen blir tatt basert p  verdien til det siste laget. En godt trent modell skal kunne gi kun et nevron som skiller seg ut med den h yeste verdien, og alle de andre har en verdi n rmest mulig null.

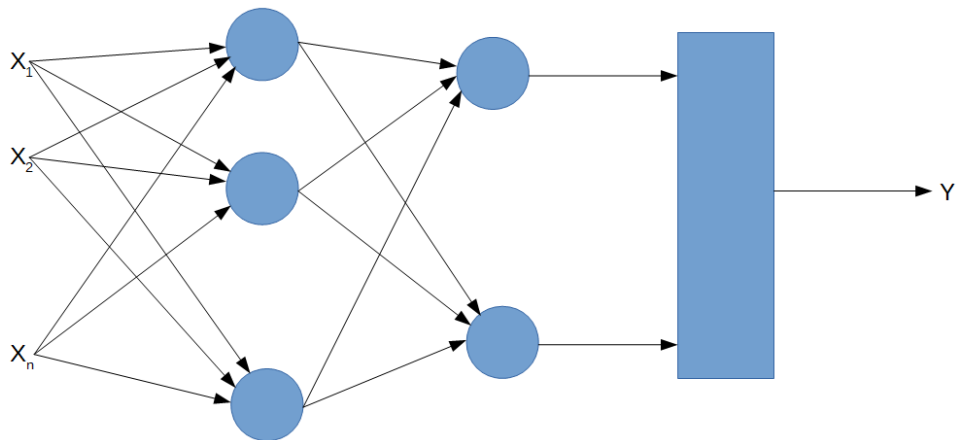


Figure 2: Flere nevroner danner et mult nettverk

Når modellen blir trent, så blir justeringen av vektene basert på en kostnad som blir beregnet ut fra den siste raden av nevroner. Hvis bildet2 blir tatt som et utgangspunkt med et ukjent antall innganger, som kan være piksler fra et bilde, og to mulige utfall, for eksempel 'en' og 'to'. Hvis modellen er utrent så vil utgangene være hva som helst og resultatet (Y) vil være helt tilfeldig. En kostnad blir beregnet av summen av resultatet minus forventet resultat opphøyd i andre. ( $sum((resultat - forventet)^2)$ )

Når modellen er utrent så vil kostnaden være veldig høy, og vektene i alle nodene må bli beregnet på nytt for så å testes på nytt. Målet er at kostnaden skal være så nærme null som mulig. Vektjusteringen blir ikke satt som en tilfeldig verdi. Denne beregningen kan man visualisere ved å lage en graf ut av kostnaden ved et gitt sett av input. Ved å si at det bare er en inngang, så vil man ende opp med en vanlig x,y graf som på bilde3. Ved to innganger så hadde man endt opp med en 3-dimensjonal graf og den ender da opp med å bli uoversiktlig. Målet ved vektjusteringen er da å finne bunnpunktet på grafen, for det er der funksjonen er optimalisert. X-verdien på grafen blir da vekten som er best egnet.

En stor utfordring er hvis man har flere bunnpunkter som på bildet3, for da er målet å finne bunnpunktet med lavest verdi, en såkalt global minima, istedenfor de andre lokale bunnpunktene. Ved å ha ti-tusenvise av input til modellen så sier det seg selv at det kommer til å være mange lokale minima og en graf-representasjon er helt ubrukelig.

For å finne de nye vektene, så må man først finne ut hvilken vei grafen går. Ved å legge til den negative gradienten på vektene så får man den nye vekten som skal

brukes. Hele denne prosessen kan gjøres ved å legge vektene og gradientene i en matrise, så kan man legge matrisene sammen. (kan trenge en visualisering av matrisene)

### 3 TPU

Nevrale nettverk og «deep learning» får stadig mer popularitet, og det oppdages stadig nye bruksområder. Til et nevralt nettverk kreves det en god del prosesseringskraft, og når prosessorutviklingen ikke er like rask som før, må en se på nye løsninger. En har begynt å se nærmere på hardware som er bygget for spesielle oppgaver, såkalt domenespesifikk hardware.

IT giganten Google har i mange av sine applikasjoner behov for maskin læring. Alt i fra tale- og bildegjenkjenning, til å slå eksperter i komplekse spill som Go [look\_at\_TPU]. I 2006 gjorde de en intern undersøkelse på behovet deres. Den gangen fant de ut at det var tilgjengelig databehandling i datasentrene, og derfor ikke trengte noe mer. I følge publikasjonen, var det i 2013 mer bruk av talesøk som førte til at Google måtte doble datasentrene for å møte behovet. I stedet for å bygge flere, valgte de å designe en ny hardware akselerator som kan gjøre denne jobben raskere og med et mindre energiforbruk enn tradisjonelle CPU-er og GPU-er.

I en publikasjon av Google [tpu\_main] tar de for seg de ulike elementene hos den første versjonen av TPU-en. Deres største behov i 2013 var å kjøre hele inferens-modeller raskt og energieffektivt. Ved å bruke matrise regning kan en effektivisere prosessen, som kan bli utført på tidligere teknologi, men som TPU-en har blitt spesifikt designet til å gjøre jobben bedre.

Denne versjonen av TPU-en er bygget med transistorer på 28nm i størrelse. Den kjører 700 MHz og har et effektforbruk på 40 W [tpu\_main]. Den er laget med en versjon 3 PCIe med \*16 datarate som tillot å sette ASIC-chipen rett inn i eksisterende servere som allerede bruker dette grensesnittet. I artikkelen sier de at de får en 12.5GB/s effektiv båndbredde mellom TPUn og prosessoren som styrer.

#### 3.1 Unike trekk

Men så er spørsmålet, hvis vi ikke skal bruke transistorene på å lage en generell prosesserings enhet, hva skal vi bruke dem på? Her har Google tenkt igjennom hva som behøves og ikke, og laget hardware som har en bedre ytelse på operasjoner som et nevralt nett trenger. Om en tar en titt på figur 3, kan en se at 24% av transistorene går til en spesiell hardware bit kalt ‘Matrix Multiply Unit’. Dette er en stor del av TPU-en og utfører kjerne funksjonen dens.

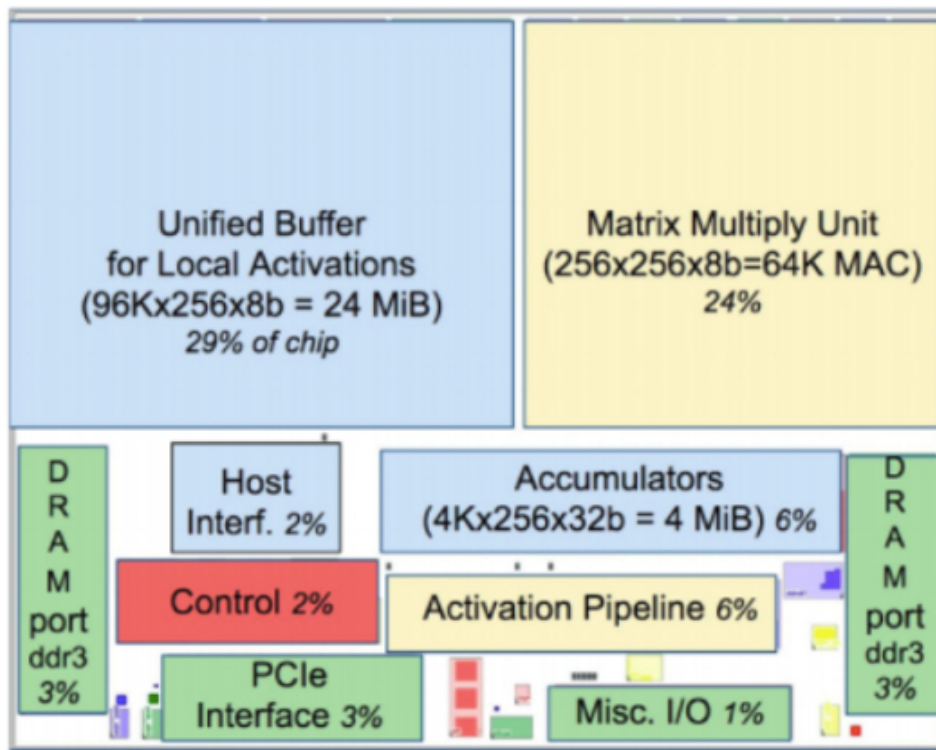


Figure 3: Transistorfordeling

RISC-prosessoren har enkle instruksjoner som i sammen kan gjøre det meste en ønsker. En instruksjon kan for eksempel være å legge sammen eller multiplisere, men det kreves flere klokkeperioder for å gjennomføre mer kompliserte funksjoner (satt sammen av flere instruksjoner). Noen CPU-er og GPU-er implementerer 'vector processing' [look\_at\_TPU] som innebærer at de kan utføre den samme instruksjonen for flere data (SIMD) på en periode.

Det TPU-en gjør annerledes er å innføre noe som kalles systolic array. Denne fungerer ved at verdier som kommer ut av en ALU blir sendt videre til neste ALU i rekken og utfører sammen en matrise beregning. Slik som i figur 4 er et slikt system mer effektivt enn en tradisjonell overføring mellom register og ALU som finnes i en CPU. Antall transistorer som trengs for å bygge en 8-bit ALU er også mindre enn 32- og 64-bit, som gjør at det er plass til flere på samme areal. Til sammen har TPU-en  $256 * 256$  ALU-er i sin MMU [look\_at\_TPU]. Dette tilsvarer 65,536 ALU-er og når klokkefrekvensen er på 700 MHz blir det totalt  $4,6 * 10^{13}$



multiplikasjon/add operasjoner per sekund.

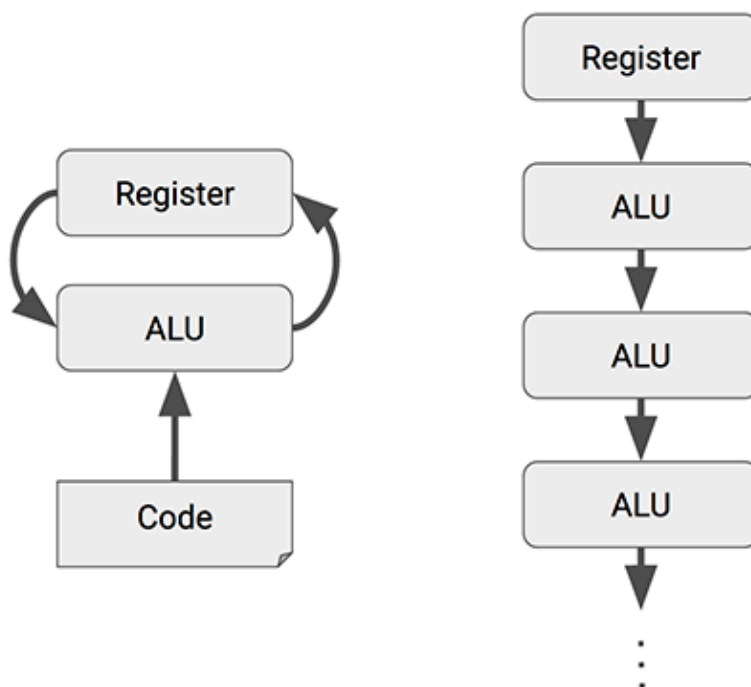


Figure 4: Konsept til Matrix multiply unit

Si at du skal regne ut hvor mange biler som kjører langs en motorvei, og du vil bare måle hvor mye kø det er når du skal lage en rute. Det er da ikke nødvendig med å vite akkurat hvor mange biler det er, bare et estimat om det er mange eller få. Samme kan sies i et nevralt nettverk. Om vi tenker oss nøyaktigheten til 32-bits flyttall trenger et typisk nevralt nettverk vanligvis ikke denne nøyaktigheten for hver node. I figur 5 ser vi at 8 bit (0 til 256) kan gi oss en tilnærmet kurve som dekker behovet. Ved å bruke quantization kan en finne 8-bits tallverdier mellom minimum og maksimum. [look\_at\_TPU].

Brikken er også blitt laget så enkelt som mulig, og unngår mange nødvendigheter som finnes i dagens CPU-er som branch prediction, out-of-order execution, cacher, osv. Dette reduserer hvor mange transistorer som må brukes, og i figur 3 ser en at kun 2% av hele chipen blir brukt til kontrollflyten [tpu\_main]. Vekter blir for

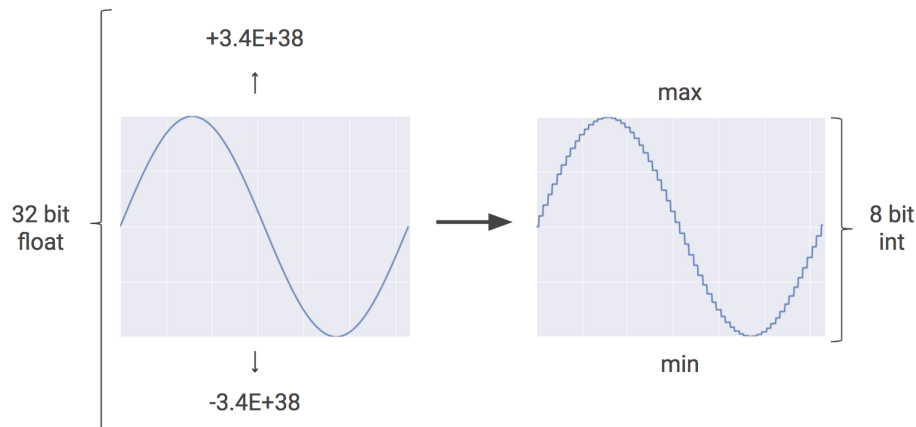


Figure 5: Kan oppnå en viss nøyaktighet selv med færre bits

eksempel lagret i en read-only DRAM, som fjerner problemet ved parallell programmering (siden ingen kan skrive over minnet, har alle samme informasjonen).

TPU-en ble designet til å være en koproessor til en CPU, og ved å bruke PCIe I/O busser, tillot det å putte den nye ASCI chipen rett inn i servere, slik som en kan gjøre med GPU-er. Den mottar TPU-instruksjoner ifra hosten sin, og bruker CISC (Complex Instruction Set Computer) prinsippet, for å kutte ned på antall instruksjoner som må bli overført. Instruksjonssettet inneholder rundt et dusin instruksjoner, der den gjennomsnittlige CPI-en er rundt 10-20. Selv med en 4-steps pipeline, så kan samme instruksjon bli kjørt over tusen ganger igjennom samme område, i motsetning til RISC-prosessoren som kjører igjennom en pipeline avdeling per klokkesyklus. I blokkdiagramet i figur 6 ser vi flyten inne i chipen. MMU-en får data ifra Weight FIFO-en og fra et unified buffer. Resultatet fra utregningene blir den sendt og lagret i accumulatorer.

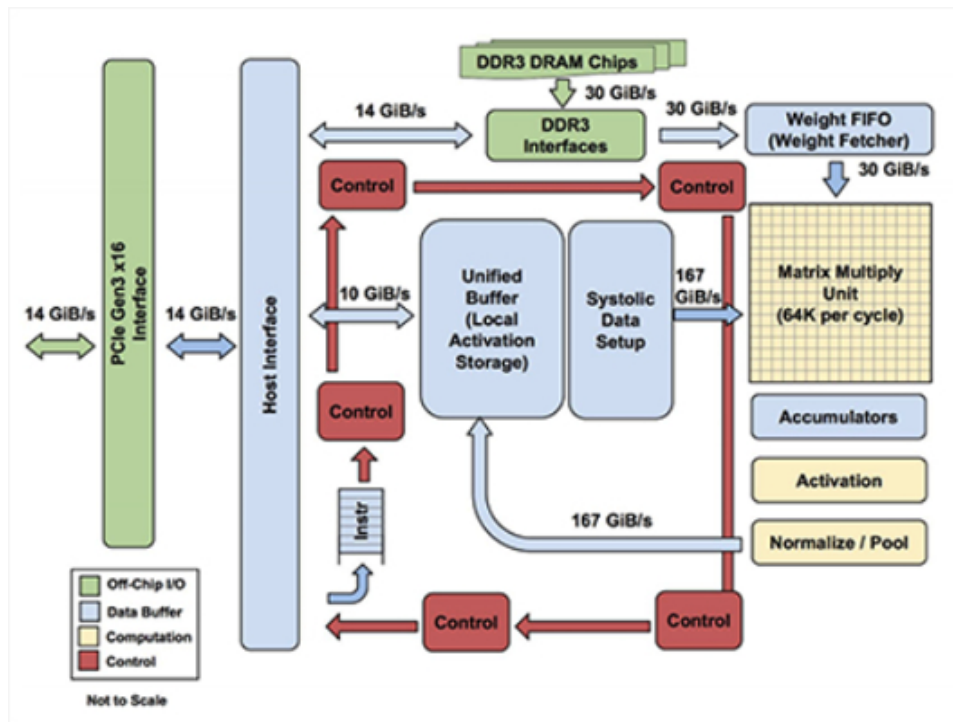


Figure 6: Blokkdiagram for TPU-en

## 4 Nyere Versjoner

Siden den første versjonen av TPU-en kom ut i 2015 har det kommet nyere utgaver [tpu\_video]. Der den første har 92 teraops og kun kunne brukes til inferens, har det kommet en cloud TPU (som vi hadde tenkt til å bruke i den praktiske delen av prosjektet) som har dobbelt så mange operasjoner per sekund og kunne brukes til trening i tillegg. Den har også støtte for flyttall. Det er i tillegg mulig å koble flere Cloud TPUs til et cluster(kalt TPU Pod) som har opp til 11.5 petaflops. I 2018 kommer versjon 3 av TPU Pod som klarer over 100 petaflops.

## 5 Annen Hardware

Som sammenlikningmateriale I prosjektet så blir algoritmen kjørt på en CPU og en GPU. CPUen som ble brukt er en intel i7-6700k prosessor som kjører skylake arkitekturen og GPUen er en EVGA Geforce GTX 1070 som kjører NVIDIA sin Pascal arkitektur.

### 5.1 CPU

En standard brikke av i7-6700k er klokke til 4.0GHz, men brikken som ble brukt er blitt overklokke til 4.4GHz og spenningen er satt til en fast spenning på 1.250 istedenfor en varierende spenning som kan føre til en ujevn test på maskinvaren. En annen grunn til å gjøre spenningen statisk er at man slipper at spenningen blir farlig høy over lengre perioder som leder til lengre levetid på prosessoren. Arkitekturen til chipen er laget for å være bakoverkompatibel, og har mange varierende instruksjoner. Dette fører til at brikken er veldig fleksibel, og gjør mye helt greit. Det betyr at den ikke er optimalisert til noe som helst.

### 5.2 GPU

GPUen er fabrikkoverklokke av EVGA og kjører derfor med noen differerende spesifikasjoner fra et standard kort fra NVIDIA. For eksempel så er klokkehastigheten økt fra 1506MHz(1683 boost) til 1594MHz(1784 boost). Kortet har blitt åpnet en gang for å legge på kjøleelementer på noen komponenter som stod i faresonen til å bli overopphetet. Dette kan medføre at kortet er bedre til å holde temperaturene nede, og forbedrer ytelsen i forhold til en brikke som kommer rett fra NVIDIA. Siden brikken er god på å generere masse piksler som skal vises på en skjerm, så er arkitekturen god på matriseregning. Dette er viktig fordi matriseregning er en effektiv måte å trene et nevralt nettverk som vist i kapittelet nevralt nettverk. Dert at brikken kan gjøre slike beregninger parallelt og er eksponert god til pikselhåndtering, så står den i ganske stor kontrast til CPUen.

## 6 Benchmarks

Når en skal utføre benchmark på nevrale nettverk er det flere ting en kan måle. Det kan være hvor lang tid det tar å nå en viss nøyaktighet, hvor mye det koster å kjøre algoritmen i skyen (server leie). Det er også viktig at det er samme datasettet som blir testet, og i en benchmark[**benchmark**] hostet hos Stanford omhandler Image Classification som benytter seg av open-source bilder fra image-net [**image-net**].

### 6.1 Treningstid

I ulike kategorier blir ulike hardwareoppsett, modeller og rammeverk målt opp mot hverandre, og blir her målt i tiden treningen bruker for å nå en 93% sikkerhet. På de tre første plassene på toppen ligger google sin TPU-pod oppdelt i 1/2(0:30:43), 1/4(1:06:32) og 1/16(1:58:24). Neste på listen ligger på 2:57:28 som kjører et oppsett ifra Amazone sin skytjeneste.

### 6.2 Treneingskostnad

Denne kategorien går ut på å oppnå den minste prisen for å trene opp til 93%. Dessverre er det ikke oppgitt hvor mye topp 3 koster å trene. Amazone sin tjeneste kostet 72 USD, og første prisgitte TPU-innslaget koster 49 USD, men tok 7:28:30. Her må en vurdere hva som er viktigst av pris og ytelse.

## **7 Konklusjon**

En fin konklusjon