UNIVERSITETET I SØRØST NORGE

DATASYSTEMDESIGN

EN_DAT2000-1 18V

TPU og maskinlæring

Forfatter
Magnus HAUKEBØE

Forfatter
Sigurd HOLMEN

Innholdsfortegnelse

1 Innledning **2**

2 Nevralt nettverk **3**

3 TPU **7**

3.1 Unike trekk 7

3.2 Nyere Versjoner 11

4 Annen Hardware **12**

4.1 CPU 12

4.2 GPU 12

5 Benchmarks **13**

5.1 Treningstid 13

5.2 Treneingskostnad 13

6 Konklusjon **14**

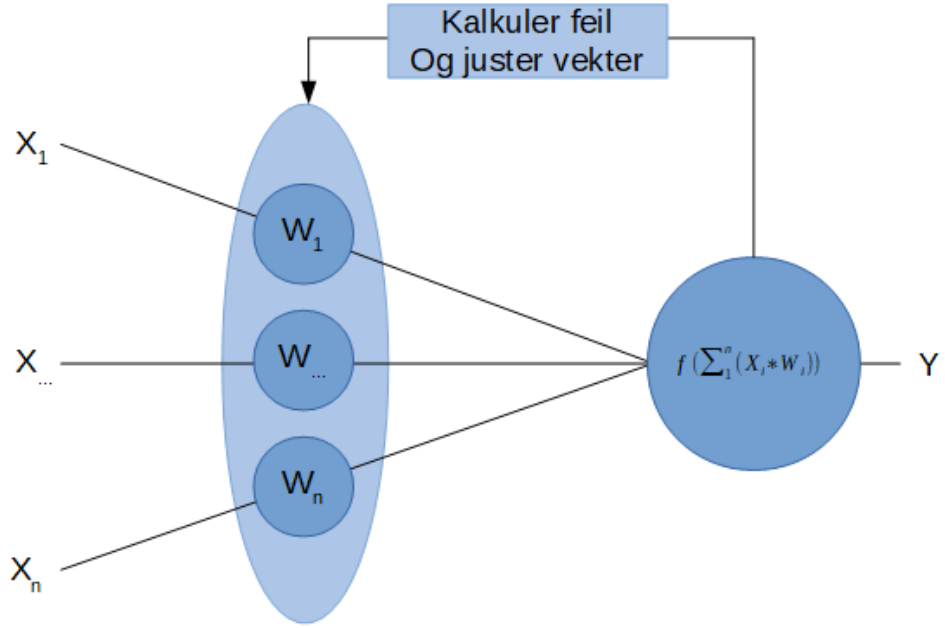
1 Innledning

I mange år har ytelsesforbedringene til CPU-er vært tilstrekkelig til å dekke behovet hos brukere, men med en økt interesse innenfor maskinlæring trengs det mye prosesseringskraft. Ved bruk av domenespesifikk hardware kan en oppnå raskere prosessering til trening og inferens av maskinlærings modeller. I dette prosjektet satte vi oss målet med å lære om Google sin TPU, og bruke denne til å trene opp en objekt-detekter (Object Detection Classifier) og sammenligne treningsprosessen med trening på egen hardware. Vi vil begynne med å forklare nevrale nettverk og deretter studere Google sin TPU, som er bygget spesifikt for denne oppgaven.

2 Nevralt nettverk

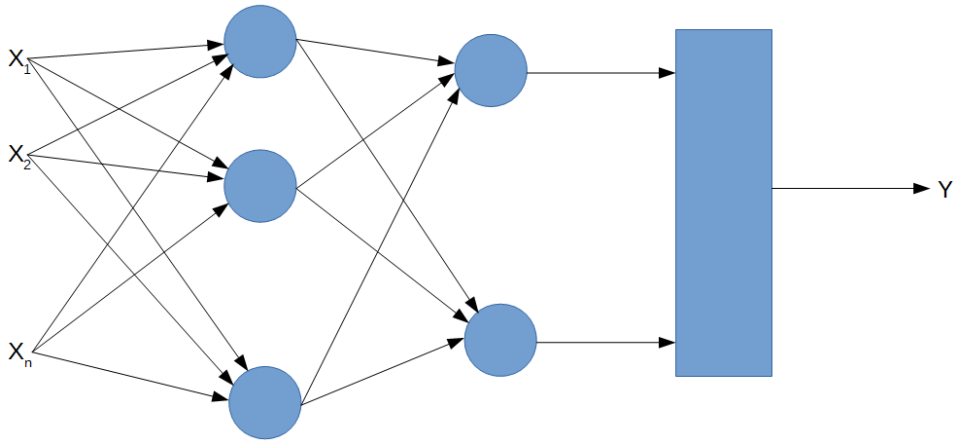
Dette er en teknikk innen maskinlæring som baserer seg på hvordan nevroner i hjernen funger for å lære seg ting. Systemet lærer ved å analysere et sett med eksempler på det den skal lære seg. I dette prosjektet så er det spillekort som skal gjenkjennes på bilder. Hele læreprosessen begynner med at kort på bilder blir markert, men ellers så har ikke algoritmen noen andre tegn å gå etter. Det finner algoritmen på selv.

Grovt sett så er et nevron i dette tilfellet et knutepunkt som lager en vektlagt sum. Denne verdien blir da kjørt 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 på [**Bonaccorso2017**].



Figur 1: Enkel illustrasjon av et neuron

Flere nevroner kan kobles sammen i et nettverk, der input blir kjørt inn i flere nevroner slik som bildet 2 viser. Hver nevron har da sine egne vekter og resultatet blir da kjørt inn i nye nevroner. I et slikt nettverk kan det være flere skjulte lag som gjør det samme som den første raden på bildet 2. Til slutt blir det tatt en beslutning basert på den siste rekken av nevroner [**Bonaccorso2017**]. Denne beslutningen blir tatt basert på verdien til det siste laget. En godt trent modell skal kunne gi kun et nevron som skiller seg ut med den høyeste verdien, og alle de andre har en verdi nærmest mulig null.



Figur 2: Flere nevroner danner et multi nettverk

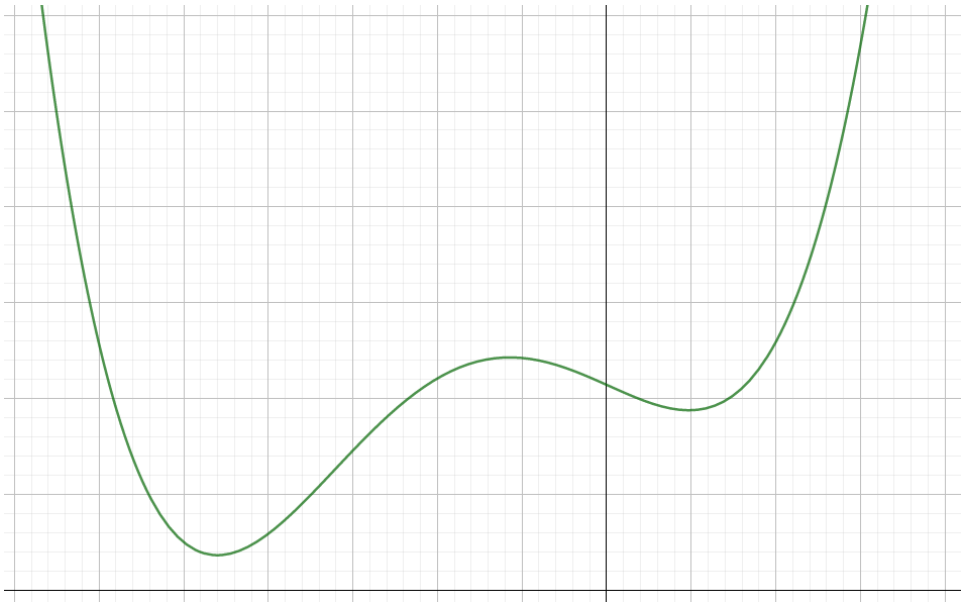
Når modellen blir trent, blir justeringen av vektene basert på en kostnad som blir beregnet ut fra den siste raden av nevroner [**neural_net**]. Hvis bildet 2 blir tatt som et utgangspunkt med et ukjent antall innganger, som kan være piksler fra et bilde, og to mulige utfall, for eksempel 'en' og 'to'. I starten vil modellen være utrent, det vil si at alle vektene er tilfeldige, og resultatet (Y) kan være hva som helst. En kostnad blir beregnet av summen av resultatet minus forventet resultat opphøyd i andre slik som formelen under viser. Her så er f forventet verdi, og r er resultatet.

$$\sum (r - f)^2 \tag{1}$$

Når modellen er utrent, vil kostnaden være høye, og vektene i alle nodene må bli beregnet for så å bli 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

[**neural_net**]. Ved å si at det bare en inngang, så vil man ende opp med en vanlig x,y graf som på bilde 3. Ved to innganger så hadde man endt opp med en 3-dimensjonal graf og ved flere innganger blir det vanskelig og visualisere. Men matten er lik, og 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 bildet 3, for da er målet å finne bunnpunktet med lavest verdi, en såkalt global minima, istedenfor de andre lokale bunnpunktene [**neural_net**]. Ved å ha ti-tusenvis 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.



Figur 3: Eksempel på en lokal og en global minima

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. Altså, hvis man legger alle vektene i en kolonne-matrise som under, og trekkker fra gradienten, så ender man opp med et sett med nye vekter [**neural_net**].

$$\begin{bmatrix} 3.67 \\ 3.95 \\ -1.27 \end{bmatrix} - \begin{bmatrix} 0.54 \\ -0.97 \\ 0.12 \end{bmatrix} = \begin{bmatrix} 3.13 \\ -4.92 \\ -1.39 \end{bmatrix} \tag{2}$$

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 trengte de ikke noe mer. I følge publikasjonen, var det i 2013 mer bruk av talesøk som førte til at Google måtte doble datasentrene for å imøtekomme 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.

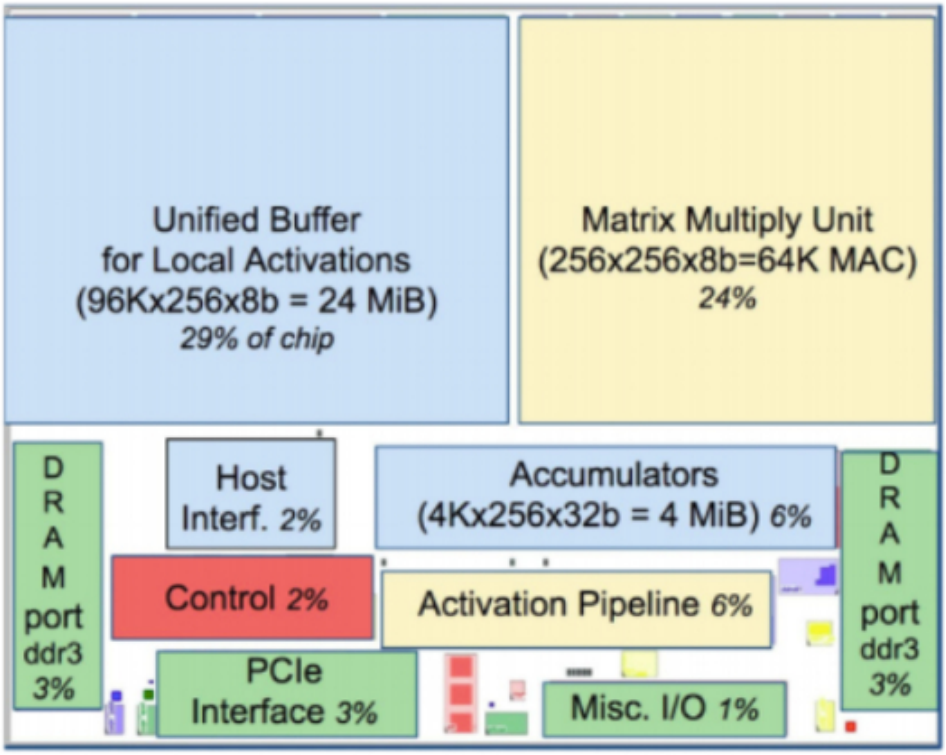
I en publikasjon av Google [**tpu_main**] tar de for seg de ulike elementene hos den første versjonen av TPU-en. Deres største behov i 2013 var å kjøre hele inferens-modeller raskt og energieffektivt. Ved å bruke matrise regning kan en effektivisere prosessen, som kan bli utført på tidligere teknologi, 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 TPUen 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 4, 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.

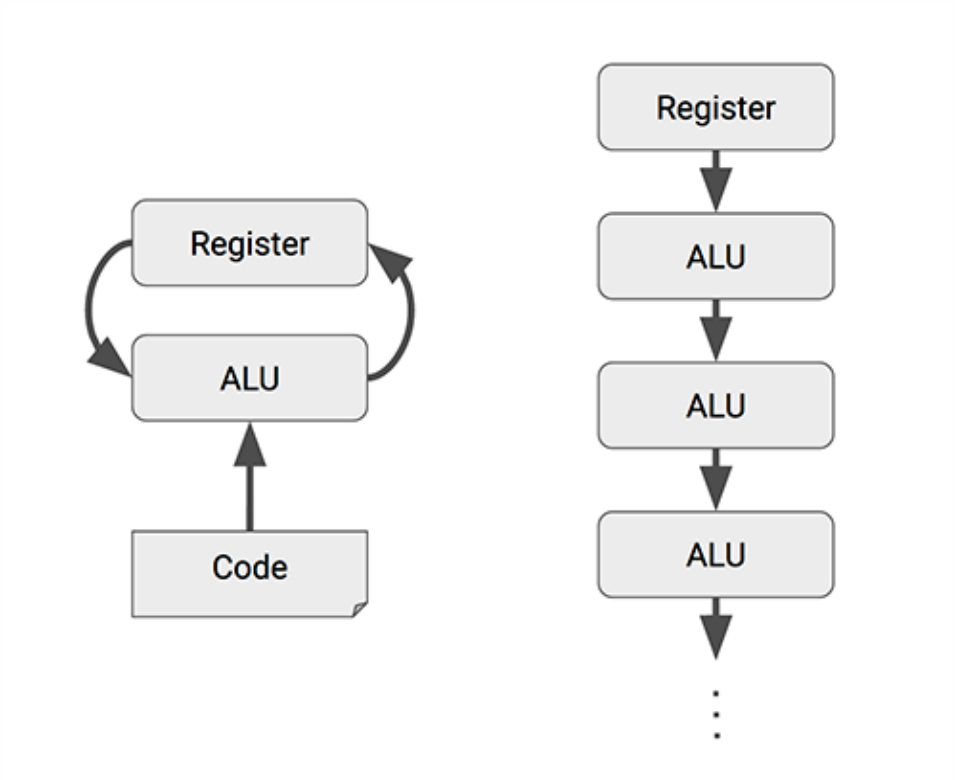


Figur 4: Transistorfordeling

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 instrukser). 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 5 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

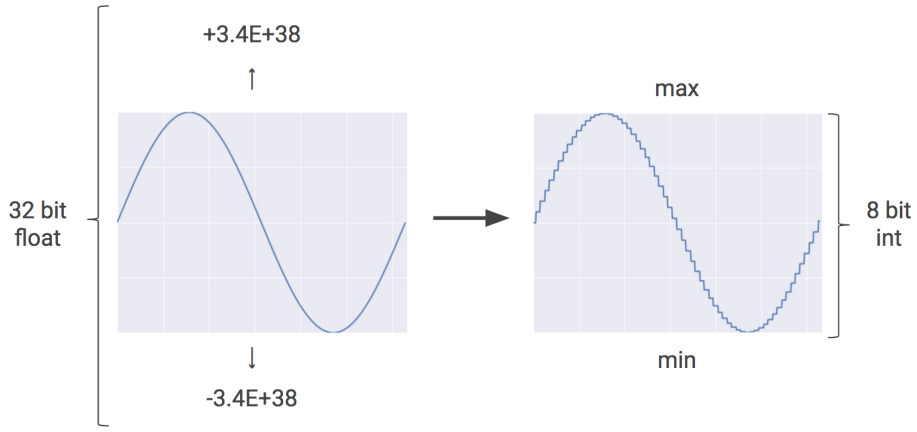
65,536 ALU-er og når klokkefrekvensen er på 700 MHz blir det totalt $4,6 * 10^{13}$ multiplikasjon/add operasjoner per sekund.



Figur 5: 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 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 6 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**].

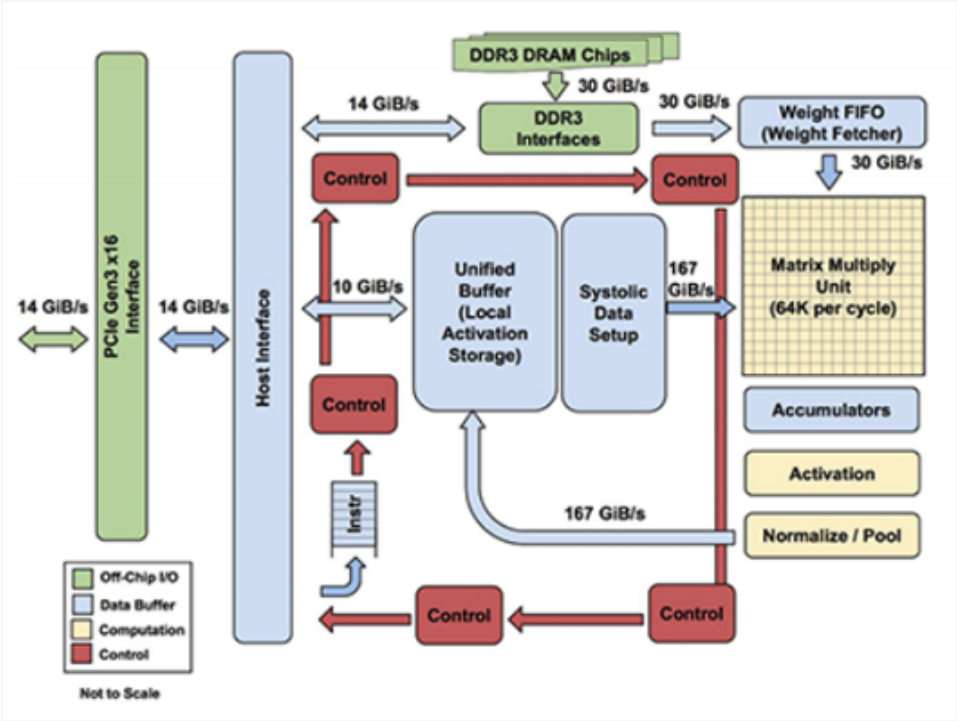
Brikken er også blitt laget så enkelt som mulig, og ungår mange nødvendigheter som finnes i dagens CPU-er som branch prediction, out-of-order execution, cacher, osv. Dette reduserer hvor mange transistorer som må brukes, og i figur 4 ser en at kun 2% av hele chipen blir brukt til kontrollflyten [**tpu_main**]. Vekter blir for



Figur 6: 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 koprosessor 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 fra 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-stegs 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 7 ser vi flyten inne i chipen. MMU-en får data fra Weight FIFO-en og et unified buffer. Resultatet av utregningene blir derretter sendt og lagret i accumulatorerer.



Figur 7: Blokkdiakgram for TPU-en

3.2 Nyere Versjoner

Siden den første versjonen av TPU-en kom ut i 2015 har det kommet flere 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 kan brukes til trening i tillegg. Den har også støtte for flyttall. Det er i tillegg mulig å koble flere Cloud TPUer 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.

4 Annen Hardware

Som sammenlikningmateriale i prosjektet 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.

4.1 CPU

En standard brikke av i7-6700k er klokket til 4.0GHz, men brikken som ble brukt er blitt overklokket 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 kan utføre mye forskjellig, men den er ikke optimalisert til noe spesiell oppgave.

4.2 GPU

GPUen er fabrikkoverklokket 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 mange piksler som skal vises på en skjerm, 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. Det at brikken kan gjøre slike beregninger parallelt og er eksepsjonelt god til pikselhånderinger, så står den i ganske stor kontrast til CPUen.

5 Benchmarks

Kommentar: Vi møtte på problemer underveis i prosjektet, og fant ut at test versjonen av Google Cloud ikke inkluderte bruk av TPU. Vi hadde også problemer med å kjøre koden lokalt. Derrimot kan vi se på benchmarks utført av andre.

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) og hvor mye prosessering som skjer gitt et tidsinterval. Det er også viktig at det er samme datasett som blir testet. En benchmark hostet hos Standford[**benchmark**] omhandler trening av en Image Classification der deltagere skal benytte seg av open-source bilder fra image-net [**image-net**].

5.1 Treningstid

I ulike kategorier blir ulike hardware-oppsett, 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, bestående av en blanding CPU-er og GPU-er.

5.2 Treneingskostnad

Denne kategorien går ut på å oppnå den minste prisen for å trene opp til 93%. På topp 3 fra forrige avsnitt er det ikke oppgitt kostnaden til treningen. Den raskeste med pris var Amazone sin tjeneste, som kostet 72 USD. Det første prisgitte innslaget av TPU koster 49 USD, men tok 7:28:30, en differanse på rundt 4,5 timer. Her må en vurdere hva som er viktigst av pris og ytelse om en skulle valgt oppsett til trening av andre modeller/treningssett.

6 Konklusjon

Siden vi støtte problemer med programmet underveis og at vi fant ut at vi ikke fikk brukt TPU-en slik vi trodde til å begynne med, ble ikke den praktiske delen gjennomført slik som vi hadde planlagt. Selv om prosjektet ikke gikk helt som vi hadde tenkt, har vi funnet ut hvordan forskjellige typer prosessorer fungerer, og da særlig om TPUen som har vært hovedfokuset ved siden av maskinlæring. Ved å studere de forskjellige prosessorene, så har vi funnet forskjeller som gir store utslag i effektiviseringen av maskinlæring som også har blitt vist i forskjellige benchmark som vi har sett nærmere på istedenfor å teste det selv.

Denne forståelsen av hva som gjør prosessoren effektiv har vi fått av å studere nærmere hvilke beregninger som skal til i maskinlæring, og hvordan disse beregningene henger sammen. Dette ble litt større enn planlagt. Derfor har vi bare skrapet litt på overflaten og prøvd å holde det relevant til det vi skulle lage.