FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Het optimaliseren van Human Activity Recognition machine learning modellen voor uitvoeren op Edge toestellen.

Arthur Leclercq
Studentennummer: 01908232

Promotoren: prof. dr. Femke Ongenae, prof. dr. ir. Sofie Van Hoecke
Begeleiders: ir. Marija Stojchevska, dr. ir. Bram Steenwinckel

Masterproef voorgelegd voor het behalen van de graad master in de Industriële Wetenschappen Elektronica ICT: ICT

Academiejaar: 2022 – 2023

UNIVERSITEIT
GENT

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Het optimaliseren van Human Activity Recognition machine learning modellen voor uitvoeren op Edge toestellen.

Arthur Leclercq
Studentennummer: 01908232

Promotoren: prof. dr. Femke Ongenae, prof. dr. ir. Sofie Van Hoecke
Begeleiders: ir. Marija Stojchevska, dr. ir. Bram Steenwinckel

Masterproef voorgelegd voor het behalen van de graad master in de Industriële Wetenschappen Elektronica ICT: ICT

Academiejaar: 2022 – 2023

UNIVERSITEIT
GENT

DANKWOORD

Als student voor de masteropleiding Industriële Wetenschappen Elektronica/ICT: ICT vormt deze scriptie het hoogtepunt en tevens de afsluiting van mijn academische carrière. Ik ben enorm dankbaar dat ik de kans heb gekregen om mijn interesse in de medische wereld te combineren met mijn passie voor technologie om dit hoofdstuk van mijn loopbaan af te sluiten. Dit zou niet mogelijk zijn geweest zonder de hulp en steun van verschillende mensen.

Allereerst wil ik mijn promotors en begeleiders, Prof. Dr. Femke Ongenae en Prof. Dr. Ir. Sofie Van Hoecke, Ir. Bram Steenwinckel en ir. Marija Stojchevska, van harte bedanken. Zij hebben mij met oprechte en opbouwende feedback door deze nieuwe uitdaging geleid, waardoor ik de diepgang en kwaliteit van deze scriptie continu kon verbeteren. Telkens wanneer ik vragen of problemen had met mijn onderzoek, kon ik rekenen op hun deskundige en nuttige adviezen. Ik wil hen bedanken voor de tijd, energie en inspanningen die zij in mij hebben geïnvesteerd.

Graag wil ik mijn dankbaarheid uiten aan mijn ouders, die mij de kans hebben gegeven om een opleiding te volgen aan de Universiteit Gent en mijn studie te financieren. Hun constante steun en aanmoediging hebben mij geholpen tijdens mijn studie en het schrijven van deze scriptie. Ik wil mijn vriendin bedanken voor het altijd klaarstaan en mij te steunen op momenten waarop ik het moeilijk had binnen deze scriptie. Verder heeft zij mij met grammatica sterk geholpen. Tot slot wil ik mijn stiefbroer bedanken voor de steun die hij doorheen deze scriptie heeft gebracht. Door zijn ervaring die hij heeft opgedaan toen hij zelf zijn scriptie schreef, wist hij mij gerust te stellen op momenten waar nodig. Zonder hen zou het niet mogelijk zijn geweest om te staan waar ik nu sta.

Dank jullie wel.

Arthur Leclercq, juni 2023

Het optimaliseren van Human Activity Recognition machine learning modellen voor uitvoeren op egde toestellen.

Door Arthur Leclercq

Masterproef ingediend met als doel het behalen van de academische graad van
Master of Science in Electronica/ICT: ICT
Academiejaar 2022-2023

Begeleiders: Prof. dr. Femke Ongenae, Prof. dr. ir. Sofie Van Hoecke
Promotors: Ir. Bram Steenwinckel, ir. Marija Stojchevska

Faculteit Ingenieurswetenschappen en Architectuur, Universiteit Gent

ABSTRACT

Dit proefschrift presenteert een studie naar de integratie van een *edge*-architectuur voor het *Human Activity Recognition machine learning* model binnen de mBrain-studie. De focus ligt op het verkennen en toepassen van optimalisatietechnieken voor de overgang naar een *edge*-architectuur. Het *machine learning* model dat in deze studie wordt behandeld is van het type *Convolutional Neural Network*. De toegepaste technieken omvatten *depthwise separable convolutions, pruning* en kwantisatie. De impact van deze technieken op de grootte, de inferentietijd en de nauwkeurigheid van het model worden onderzocht. Verder bevat de scriptie een vergelijkend onderzoek tussen een *cloud*-architectuur en een *edge*-architectuur. Deze twee architecturen worden opgezet om inzicht te krijgen in de verschillen tussen beide. Het proefschrift bevat een uitgebreid onderzoek naar beide architecturen en hun impact op de inferentietijd en de batterijconsumptie van een smartphone. Hierbij worden drie scenario's uitgewerkt voor de frequentie van dataverwerking. In het eerste scenario wordt de data elke zes seconden verwerkt, overeenkomend met één sample, in het tweede scenario elke vijf minuten, en tenslotte, in het derde scenario elk uur. Deze studie biedt inzicht in de verschillen tussen de *cloud*- en *edge*-architectuur, zodat een weloverwogen beslissing kan worden genomen over de eventuele overstap naar de *edge*-architectuur. Daarnaast biedt het een overzicht van de technieken die gebruikt kunnen worden om deze overstap te vereenvoudigen. Uit de resultaten van deze scriptie blijkt dat technieken zoals *depthwise separable convolutions* en een bepaalde vorm van kwantisatie voordelige effecten hebben op basis van inferentietijd en opslagcapaciteit. Andere technieken zoals pruning en andere vormen van kwantisatie tonen in mindere mate positieve effecten. Verder blijkt de *edge*-architectuur een interessant alternatief te zijn in vergelijking met de bestaande *cloud*-architectuur op basis van batterijverbruik en inferentietijd.

Trefwoorden: human activity recognition, edge, cloud, kwantisatie, pruning, convolutional neural network, depthwise separable convolutions.

Het optimaliseren van Human Activity Recognition machine learning modellen voor uitvoeren op edge toestellen.

Arthur Leclercq

Begeleiders: Prof. Dr. Ir. Sofie Van Hoecke, Prof. Dr. Femke Ongenae

Promotors: Ir. Bram Steenwinckel, Ir. Marija Stojchevska

Faculteit Ingenieurswetenschappen en Architectuur, Universiteit Gent

***Abstract*- Deze scriptie doet onderzoek naar de overgang van een *cloud*-architectuur naar een *edge*-architectuur voor het *Human Activity Recognition* model van de mBrain-studie. Het eerste deel onderzoekt de invloed van bestaande optimalisatietechnieken op het *machine learning* model van de mBrain-studie. Deze optimalisatietechnieken houden rekening met de beperkte rekenkracht, batterij en opslagcapaciteit van een smartphone waar het model op geplaatst zou worden. Hierbij worden vier verschillende technieken toegepast en hun invloed op de modelomvang, inferentietijd en nauwkeurigheid onderzocht. Het tweede deel bevat een uitgebreid onderzoek naar welke architectuur, *cloud* of *edge*, het voordeligst is om te implementeren binnen deze studie. Hierbij wordt onderzocht wat de invloed is van beide architecturen op het batterijverbruik van een smartphone en de inferentietijd. Uit de resultaten van deze scriptie blijkt dat technieken zoals *depthwise separable convolutions* en een bepaalde vorm van kwantisatie voordelige effecten hebben op basis van inferentietijd en opslagcapaciteit. Verder blijkt de *edge*-architectuur een interessant alternatief te zijn in vergelijking met de bestaande *cloud*-architectuur op basis van batterijverbruik en inferentietijd.**

***Trefwoorden*- Human Activity Recognition, edge, cloud, kwantisatie, pruning, depthwise separable convolutions, Convolutional Neural Network**

I. INTRODUCTIE

Het doel van de mBrain-studie is om meer inzicht te verkrijgen in migraineaanvallen waarbij gekeken wordt naar de levensstijl, gedrag en mogelijk eventuele triggers. Het richt zich onder andere op de ontwikkeling van een *machine learning* model voor het voorspellen en detecteren van migraineaanvallen. Daarbij hoort een model dat verantwoordelijk is voor het detecteren van de activiteiten van de persoon doorheen de dag. Het model maakt gebruik van accelerometergegevens van een smartphone. Deze gegevens bevatten de versnelling van de smartphone in drie verschillende richtingen, namelijk een x, y en z-richting.

Het huidige model van de mBrain-studie dat de activiteiten van een patiënt voorspelt, draait in de *cloud*. Deze opzet vereist een constante internetconnectie, zowel voor het verzenden van de data naar de virtuele node als voor het ontvangen van de predicties. Het verzenden van data over het internet is echter vatbaar voor hackpogingen, wat privacyrisico's inhoudt, zeker gezien het hier om patiëntendata gaat. In de toekomst kunnen er regelgevingen worden ingevoerd die restricties opleggen aan het verzenden van dergelijke gezondheidsdata over het internet. Bijkomend wil de mBrain-studie voorkomen dat predicties enkel mogelijk zijn bij beschikbaarheid van een internetconnectie. Mede om deze redenen wenst de mBrain-studie niet afhankelijk te zijn van wifi voor langdurige patiëntmonitoring. Een andere optie is de gegevens verwerken dichtbij de bron waar de gegevens worden verzameld, namelijk de *edge*-omgeving. Indien de overstap gemaakt wordt naar deze omgeving dan moeten de accelerometerdata niet meer over het internet verstuurd worden.

Deze scriptie verkent en beschrijft deze omgeving. Het brengt namelijk een aantal uitdagingen met zich mee in vergelijking met de *cloud*-omgeving. Deze uitdagingen zijn onder andere:

- Beperkte rekenkracht: De computationele bronnen in de *edge*-omgeving zijn minder krachtig, zoals op een smartphone, wat kan leiden tot tragere predicties.
- Opslagcapaciteit: De smartphone heeft een beperkte hoeveelheid opslag.
- Batterijduur: Een smartphone heeft een beperkte batterijduur voordat deze opnieuw moet worden opgeladen.

Daarentegen in een *cloud*-architectuur maakt de modelomvang, batterijconsumptie en complexiteit in mindere mate uit. Deze scriptie doet onderzoek naar de overgang naar een *edge*-omgeving waarbij het rekening houdt met de beperkingen van deze omgeving. Hiervoor worden drie technieken onderzocht, namelijk *pruning*, kwantisatie en *depthwise separable convolutions*, alsook een combinatie van *pruning* en kwantisatie. Hierbij wordt onderzocht of deze technieken de complexiteit, opslagcapaciteit en batterijconsumptie kunnen verminderen, zonder dat dit leidt tot een aanzienlijke daling van de nauwkeurigheid.

Het tweede deel doet onderzoek naar welke van deze twee omgevingen het voordeligst is op basis van batterijconsumptie en inferentietijd, specifiek binnen de mBrain-studie voor de *Human Activity Recognition (HAR)*. Er zijn twee architecturen gecreëerd, één voor beide omgevingen. Er worden metingen gedaan voor beide architecturen om de impact op de batterijduur en de inferentietijd te meten. Er worden drie verschillende scenario's onderscheiden. Hierbij wordt de data elke zes seconden (elke sample), om de vijf minuten of elk uur verwerkt. Op deze manier kan worden bepaald welk scenario

en welke architectuur het meest geschikt zijn voor deze toepassing.
Om dit doel te bereiken, streeft het proefschrift na de volgende onderzoeksvragen te beantwoorden:

1) Welke technieken bestaan er om een *machine learning* model lichter en sneller te maken en wat is de invloed hiervan op de inferentietijd, modelgrootte en de nauwkeurigheid?
2) Kan een *machine learning* model gebruikt worden op een smartphone zonder dat daarmee een aanzienlijk verlies aan accuraatheid gepaard gaat?
3) Heeft het gebruik van de *edge*-architectuur voor *machine learning* modellen een positieve impact op de batterijconsumptie en inferentietijd in vergelijking met een *cloud*-architectuur?

II. GERELATEERD WERK

De laatste jaren is de vraag naar geoptimaliseerde modellen voor een *edge*-omgeving sterk toegenomen. In de literatuur is er daarom ook al uitgebreid onderzoek gedaan naar technieken voor lichtere en efficiëntere modellen.

Een veelgebruikte toepassing is beeldverwerking op een smartphone zelf. Hierbij wordt een *machine learning* model op een smartphone geplaatst waarna het predicties kan uitvoeren op beelden. Voor deze toepassing wordt vaak gebruik gemaakt van een *Convolutional Neural Network* (CNN), dit soort netwerk bevat convoluties wat rekenintensief is. Xception [1] introduceerde als eerste het concept van *Depthwise Separable Convolutions* (DSC's), wat later ook werd geïntegreerd in MobileNet [2]. Het heeft als doel de reguliere rekenintensieve convoluties op te splitsen in twee minder rekenintensieve delen. De reguliere convolutie wordt opgesplitst in een *depthwise* convolutie en *pointwise* convolutie. Bij *depthwise* convoluties worden convoluties per kanaal uitgevoerd. Voor beelden is dit bijvoorbeeld het rode-, groene- en blauwe kleurkanaal apart. Voor HAR is dit de x-,y- en z-as apart. Na de *depthwise* convolutie volgt een *pointwise* convolutie die puntsgewijs over de drie kanalen heen convoluties uitvoert. Het aantal parameters vermindert hierdoor met $\frac{1}{N} + \frac{1}{D_K^2}$, waarbij N staat voor het aantal kernels en D_K voor de grootte van de filters.

T. Zebin et al. [3] passen kwantisatie toe om de complexiteit van een *machine learning* model te reduceren. Kwantisatie is een techniek waarbij getallen worden voorgesteld met een lagere precisie. T. Zebin et al. schakelen over van *floating point* getallen naar 8-bit *fixed point* getallen. De modelomvang van het CNN-model daalde met een factor 7.62, terwijl de accuraatheid met slechts 2,8% afnam. Deze studie heeft geen onderzoek gedaan naar de impact op de inferentietijd.

P. Dhiman et al. [4] gaan nog een stap verder. Naast post-training kwantisatie passen ze ook magnitude-gebaseerde *pruning* toe op een CNN. Met als doelstelling een efficiënter en lichter model te creëren zodat de detectie van ziektes op citrusvruchten direct op de landbouwvelden op de sensoren kan gebeuren. Door *pruning* toe te passen wordt een op voorhand gedefinieerd aantal gewichten van het model op nul gezet. Hierbij wordt specifiek gekozen voor magnitude-gebaseerde *pruning* waarbij de laagste gewichten gedurende de training op nul worden gezet. Dit heeft als reden dat deze lagere gewichten doorgaans een lichtere impact hebben op de uitgang van het model. Hierdoor zullen de gewichtsmatrices veel nulwaarden bevatten waardoor het model mogelijk op een efficiëntere manier zou kunnen worden opgeslagen. Nadat *pruning* werd toegepast daalde de grootte van het model met factor 1.88, en de nauwkeurigheid nam af met 2.74%. Wanneer ook post-training kwantisatie werd toegepast, daalde de grootte van het model met factor 2.56, de nauwkeurigheid daalde met 4.52%. Beide technieken kunnen dus gecombineerd worden om de modelgrootte verder te reduceren. Daarmee ging echter een groter verlies aan nauwkeurigheid gepaard. Ook deze studie heeft geen onderzoek gedaan naar de impact op de inferentietijd.

III. BASISMODELLEN

Om de invloed van verschillende optimalisatietechnieken op de inferentietijd, opslagomvang en nauwkeurigheid te onderzoeken, worden er twee basismodellen getraind. Beide modellen zijn CNNs en dienen voor *Human Activity Recognition*. Het eerste model, het WISDM-model, is getraind op de WISDM-dataset voor HAR [5], die publiek beschikbaar is en tri-axiale accelerometerdata bevat. Het tweede model, het mBrain-model, is getraind op de dataset van de mBrain-studie, welke eveneens tri-axiale accelerometerdata bevat. Het onderscheid tussen beide modellen ligt in het feit dat het WISDM-model over minder lagen beschikt, maar het bevat wel meer en grotere filters. Hierdoor is het aantal parameters en de modelomvang een stuk groter. Door beide modellen te onderwerpen aan dezelfde optimalisatietechnieken kan de variërende impact van elke techniek op twee verschillende modellen onderzocht worden.

Tabel. 1: *Eigenschapen van de basismodellen.*

	WISDM-model	mBrain-model
Aantal parameters	66 678	13 653
Modelomvang	272 772 bytes	71 960 bytes

IV. TESTOMGEVING

Om betrouwbare metingen uit te voeren, wordt er een specifieke testomgeving gecreëerd. Allereerst wordt er gedurende het volledige onderzoek gebruik gemaakt van dezelfde smartphone waarop de metingen worden uitgevoerd, namelijk een Samsung Galaxy S10e. Deze smartphone bevat een Samsung Exynos 9820 Octa-core CPU met 6 gigabyte werkgeheugen en een Lithium-Ion batterij met een capaciteit van 3000 mAh.

Om de *cloud*-architectuur te realiseren wordt er ook gebruik gemaakt van een virtuele node. Op deze virtuele node is een Flask-applicatie geïmplementeerd waarnaar de accelerometerdata gestuurd kan worden voor verwerking van de data. Na de verwerking worden de predicties teruggestuurd.

Tot slot zijn er twee Android-applicaties gemaakt om de metingen te kunnen uitvoeren. Deze applicaties worden geplaatst op de eerder vermelde smartphone waarop de metingen plaatsvinden. Deze applicaties zijn verantwoordelijk voor het opvragen van de smartphone accelerometerdata. Deze waarden worden dan vervolgens getoond in de applicatie aan de gebruiker en lokaal opgeslagen in een CSV-bestand. Bij de *cloud*-applicatie wordt de data, om de op vooraf bepaalde tijdsinterval, doorgestuurd naar de virtuele node voor de verwerking van de data. Bij de *edge*-applicatie wordt het te testen *machine learning model* in de applicatie geplaatst. De verzamelde accelerometerdata wordt vervolgens, om de op vooraf bepaalde tijdsinterval, gevoed aan het *machine learning* model op de smartphone. Bij beide applicaties worden de predicties lokaal opgeslagen.

V. OPTIMALISATIETECHNIEKEN

De drie technieken die in de literatuurstudie naar voren zijn gekomen, worden toegepast op beide modellen. Deze technieken zijn: *depthwise separable convolutions*, *pruning*, en kwantisatie. Voor de metingen wordt gebruik gemaakt van een licht andere *edge*-applicatie dan die beschreven werd in **Sectie IV**. Hierbij wordt telkens gebruik gemaakt van een bestand dat op voorhand werd gecollecteerd. Het bestand bestaat uit accelerometerdata van vijf minuten aan 32Hz en bevat bijgevolg 9600 datapunten voor elk van de drie assen.

A. DEPTHWISE SEPARABLE CONVOLUTIONS

Deze techniek slaagt er in het aantal parameters respectievelijk met een factor 6.63 en 2.18 te verminderen voor het WISDM-model en het mBrain-model. In **Sectie II** werd vermeld dat het aantal parameters vermindert met $\frac{1}{N} + \frac{1}{D_K^2}$, waarbij N staat voor het aantal kernels en D_K voor de grootte van de filters. Doordat het WISDM-model gebruik maakt van meer en grotere filters is de factoriale daling groter dan bij het mBrain-model.

***Tabel. 2**: Resultaten Depthwise Separable Convolutions op het WISDM-model en het mBrain-model.*

	WISDM-model		mBrain-model	
	Conv1D	SeparableConv1D	Conv1D	SeparableConv1D
#parameters	66 678	10 052	13 653	6 260
Omvang	272 772 bytes	47 532 bytes	71 960 bytes	47 664 bytes
Inferentietijd	172 ms	149 ms	136 ms	127 ms
Accuraatheid	87.83%	88.02%	77.75%	76.92%

Door de vermindering in parameters daalt de modelgrootte voor beide modellen ook aanzienlijk. Doordat er minder multiplicaties en optellingen dienen te gebeuren daalt de inferentietijd voor beide modellen. De daling bedraagt 1.15 en 1.07 respectievelijk voor het WISDM-model en het mBrain-model. Deze daling in inferentietijd is mogelijk voordelig voor het batterijverbruik.

B. PRUNING

Bij deze techniek krijgen, gedurende het trainingsproces, steeds meer gewichten van het CNN-model de nulwaarde toegewezen. Vervolgens wordt gebruik gemaakt van de *Tensor Algebra Compiler* om de uitgedunde gewichtsmatrices op een efficiëntere manier op te slaan [6].

Fig. 1 visualiseert de invloed van *pruning* op de modelomvang en de nauwkeurigheid voor het WISDM-model. Het model werd uitgedund tot 25%, 50%, 75%, 80%, 85% en 90%. De inferentietijd is niet veranderd na het toepassen van *pruning*. Uit de resultaten is het duidelijk dat deze techniek de modelomvang van het WISDM-model sterk kan doen dalen. Deze reductiefactor is afhankelijk van de mate waarin *pruning* wordt toegepast. De nauwkeurigheid neemt voor dit geval aanzienlijk af na 50%-uitdunning.

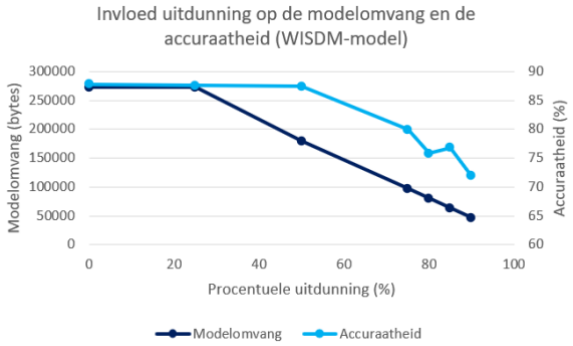


***Fig. 1:** Invloed uitdunning/pruning op de modelomvang en de accuraatheid bij het WISDM-model.*

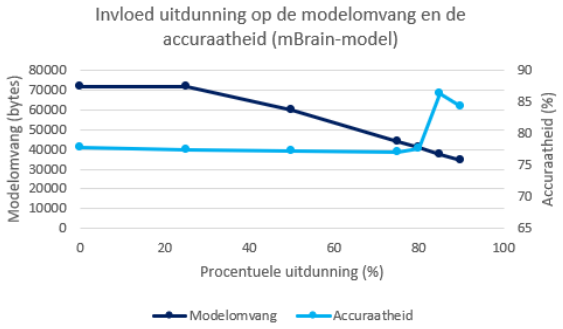


***Fig. 2**: Invloed uitdunning/pruning op de modelomvang en de accuraatheid bij het mBrain-model.*

Fig. 2 visualiseert de invloed van deze techniek op de modelomvang en de nauwkeurigheid voor het mBrain-model. Het model werd uitgedund tot 25%, 50%, 75%, 80%, 85%, 90%. Ook voor dit model blijft de inferentietijd hetzelfde na uitdunning. De resultaten liggen in dezelfde lijn als bij het WISDM-model. Hoe meer uitdunning, hoe compacter het model kan worden opgeslagen. Hierbij ligt het breekpunt voor de nauwkeurigheid weliswaar op 80%-uitdunning. Op de figuur is te zien dat de nauwkeurigheid stijgt vanaf deze mate van uitdunning. Dit geeft echter een vertekend beeld. Het model voorspelt vanaf dan nooit meer bepaalde minderheidsklassen. Dit is dus een ongewenst fenomeen. De

techniek is daarom enkel interessant tot 75%-uitdunning voor het mBrain-model.

C. POST-TRAINING KWANTISATIE

In **Tabel 3** worden de resultaten van deze techniek op de WISDM-model samengevat. De rij 'Geen' vertegenwoordigt de resultaten na de conversie van het Keras-formaat naar het TensorFlow Lite-formaat. Float16-kwantisatie slaagt erin de modelgrootte te verkleinen met een factor van 1.94, zonder aanzienlijk verlies in nauwkeurigheid. De inferentietijd blijft zo goed als onaangetast. Kotlin, de programmeertaal waarin de applicaties werden ontwikkeld, ondersteunt het datatype float16 niet, waardoor er nog steeds float32-operaties worden uitgevoerd. Hierdoor blijft de inferentietijd ongewijzigd. Uint8-kwantisatie zorgt voor een reductie in modelomvang met factor 3.43 en versnelt het inferentieproces met factor 1.16. De standaard optimalisator van TensorFlow (dynamisch), slaagt erin om de modelgrootte met factor 3.58 te reduceren, terwijl de nauwkeurigheid niet aanzienlijk daalt. Daarnaast daalt de inferentietijd met 1.18.

Tabel. 3: *Resultaten van post-training kwantisatie op het WISDM-model.*

Type	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	1 446 560 bytes	/	88.53%
Geen (float32)	272,772 bytes	172 ms	87.83%
Float16	140 832 bytes	169 ms	86.32%
Uint8	79 544 bytes	148 ms	86.23%
Dynamisch	76 288 bytes	146 ms	86.37%

In **Tabel 4** worden de resultaten post-training kwantisatie voor het mBrain-model samengevat. De algemene trend is gelijk aan die van het WISDM-model. De reden dat de reductiefactor voor modelgrootte en inferentietijd lager is, is omdat dit model minder trainbare parameters heeft (**Tabel 2**) waardoor de invloed minder is. Bij de uint8-kwantisatie stijgt de nauwkeurigheid van het model. Dit geeft echter een vertekend beeld, omdat het model vanaf dat punt geen minderheidsklassen meer voorspelt. Dit is een ongewenst resultaat.

Tabel. 4: *Resultaten van post-training kwantisatie op het mBrain-model.*

Type	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	152 248 bytes	/	77.88%
Geen (float32)	71 960 bytes	136 ms	77.75%
Float16	52.252 bytes	133 ms	77.76%
Uint8	40 864 bytes	131 ms	82.71%
Dynamisch	43 188 bytes	128 ms	77.83%

D. PRUNING EN POST-TRAINING KWANTISATIE

Als laatste wordt een combinatie van kwantisatie en *pruning* onderzocht. Eerst wordt het model gepruned en vervolgens wordt de standaard optimalisator van TensorFlow (dynamisch) gebruikt als kwantisatie.

Er is geen verdere daling zichtbaar voor de accuraatheid nadat ook kwantisatie wordt toegepast. Het model wordt nog kleiner in omvang na het ook toepassen van kwantisatie. De reductie in inferentietijd is dezelfde als wanneer enkel kwantisatie werd gebruikt.

Wanneer de combinatietechniek vergeleken wordt met de technieken afzonderlijk heeft deze enkel voordelen t.o.v. pruning. De combinatietechniek heeft voordelen t.o.v. kwantisatie tot een bepaalde uitdunning. Wanneer de uitdunning bij de combinatietechniek meer dan 50% is, dan is het voordeliger om enkel kwantisatie te gebruiken.

VI. CLOUD VS EDGE

Het tweede deel van deze scriptie onderzoekt de *cloud*- en *edge*-omgevingen. Er wordt onderzoek gedaan naar de invloed van beide architecturen op de inferentietijd en het batterijverbruik van de smartphone. De metingen worden uitgevoerd op de smartphone, virtuele node en de Android-applicaties, zoals beschreven in **Sectie II**. Er worden drie verschillende scenario's onderzocht. In het eerste scenario wordt de data voortdurend verwerkt (elke zes seconden), in het tweede scenario elke vijf minuten en in het derde scenario elk uur. Daarnaast werd nog bijkomend onderzoek gedaan naar de invloed van het internetwerk op de batterijverbruik en inferentietijd voor de *cloud*-architectuur. De metingen werden uitgevoerd op de TFLite-geconverteerde versie van het mBrain-model zonder optimalisatietechnieken. Elke meting duurde in totaal één uur.

A. BATTERIJCONSUMPTIE

Allereerst wordt gemeten hoeveel batterijcapaciteit de basisapplicatie verbruikt zonder dat er data wordt verwerkt (357.84mAh). Het is belangrijk om op te merken dat tijdens deze metingen het scherm voortdurend aanstond op maximale helderheid. Dit verklaart namelijk waarom het verbruik zo hoog is.

Vervolgens worden dezelfde metingen uitgevoerd voor de eerder genoemde drie scenario's. Om een goed overzicht te bieden wordt het verbruik van de basisapplicatie afgetrokken voor de drie onderzochte scenario's. Op deze manier wordt gezien hoeveel extra batterij het kost om de data te verwerken in de *cloud* of op de smartphone zelf (*edge*).

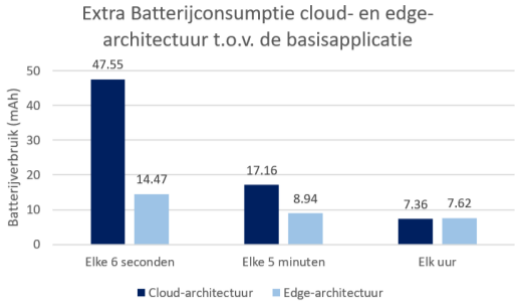


***Fig. 3**: Extra batterijconsumptie voor de cloud-architectuur voor de drie scenario's t.o.v. de basisapplicatie.*

Uit de metingen blijkt dat het scenario waarbij de verwerking elke zes seconden gebeurt het meest batterij verbruikt, dit geldt voor beide omgevingen. Voor de *cloud-omgeving* wilt dit zeggen dat de datapakketten die over het internet verstuurd worden kleiner zijn dan bij de andere twee scenario's. Ondanks dit, is dit scenario het meest energie-intensief. De frequentie van het versturen van datapakketten over het internet weegt zwaarder door dan de grootte van deze datapakketten. In totaal wordt de accelerometerdata 600 keer verstuurd en wordt de voorspelling 600 keer teruggestuurd naar de applicatie tijdens deze meting. Voor het tweede scenario is dit 12 keer en bij het derde scenario slechts één keer.

***Tabel. 5**: Overzicht aantal datapunten per vrijheidsgraad en bestandgroottes bij de drie verschillende scenario's.*

Scenario	Duur	datapunten per axis	Grootte (kB)
1	6 sec.	192	54
2	5 min.	9600	408
3	1 uur	115 200	4 710

Bij het *edge* scenario is hetzelfde patroon zichtbaar: hoe frequenter de data verwerkt wordt, hoe meer batterij dit kost. De toename in batterijverbruik is voornamelijk te wijten aan het frequenter activeren van de bronnen (processors, geheugenmodules of andere hardwarecomponenten), wat een inherente opstartkost met zich meebrengt. Precies deze opstartkost is een doorslaggevende factor in het hogere energieverbruik. Dit verklaart waarom er aanzienlijk minder batterijcapaciteit wordt verbruikt wanneer het tijdsinterval voor het verwerken van de data toeneemt.

Wanneer beide architecturen naast elkaar worden gelegd is de *edge*-architectuur de minst energie-intensieve. Het grootste verschil zit in het scenario waarin de verwerking elke zes seconden plaatsvindt. Het gebruik van de *cloud*-architectuur resulteert in 3.29 keer meer milliampère-uur bovenop de basisapplicatie. Ook het verschil in verbruik is nog duidelijk te merken in het scenario waarbij de data elke vijf minuten wordt verwerkt, maar al een stuk minder dan bij het eerste scenario. Bij het derde scenario is er geen verschil meer zichtbaar.

Wanneer batterijconsumptie enorm belangrijk is, is het dus voordelig om zo weinig frequent mogelijk de data te verwerken, ongeacht de architectuur die gebruikt wordt. Echter, een nadeel hiervan is dat de voorspellingen dan pas elk uur zichtbaar zullen zijn. Daarnaast blijkt de *edge*-architectuur voordeliger te zijn dan de *cloud*-architectuur. Hoe minder frequent de data wordt verwerkt, hoe gelijkaardiger het verbruik wordt tussen beide architecturen.

B. INFERENTIETIJD

Hierbij verwijst de inferentietijd naar de tijd die nodig is om de voorspellingen zichtbaar te maken voor de gebruikers van de applicatie. Voor de *cloud*-architectuur wordt dit onderverdeeld in drie verschillende delen. Het eerste deel is de uplink-en downlinktijd. De uplinktijd is de tijd die nodig is om de accelerometerdata vanop de smartphone te versturen naar de virtuele node in de *cloud*. De downlinktijd is de tijd die nodig is om de predicties terug te sturen van de virtuele node naar de smartphone. Het tweede deel is de tijd die wordt besteed aan het effectief maken van predicties door het model, de effectieve inferentietijd. Het derde deel is de voor-en naverwerking, dit zijn alle andere tijden binnen virtuele node die niet worden besteed aan het effectief uitvoeren van voorspellingen. Op deze manier wordt de inferentietijd binnen de *cloud*-architectuur dus verdeeld in drie verschillende processen.

***Tabel. 6**: Overzicht van de inferenteitijd voor de verschillende delen in het cloud-proces voor de drie verschillende scenario's.*

Scenario	Uplink +Downlink (ms)	Inferentietijd (ms)	Andere (ms)	Totale Inferentietijd (ms)	Gemiddelde tijd per predictie (ms)
6 sec.	697.31	0.10	23.75	721.16	721.16
5 min.	973.42	4.83	71.42	1049.67	20.99
1 uur	4696.0	61.0	638.0	5393.0	8.99

Tabel 6 toont de inferentietijden voor de verschillende fasen in het *cloud*-proces voor de drie scenario's. Het wordt duidelijk dat grotere bestanden resulteren in langere uplink- en downlinktijden. Hetzelfde geldt voor de totale inferentietijd van het model, aangezien er meer voorspellingen worden gemaakt. Deze waarden weerspiegelen de tijd voor het verzenden en verwerken van een bestand van zes seconden, vijf minuten en één uur. Het is duidelijk dat de tijd die effectief wordt gespendeerd aan het maken van inferenties door het *machine learning* model extreem laag is. Het overgrote deel van de tijd gaat naar het versturen en ontvangen van datapakketten over het internet. De tijd die verloren gaat aan de uplink- en downlinktijd bedraagt respectievelijk 96.7%, 92.7% en 87.0% van de totale tijd. De procentuele tijd die gespendeerd wordt aan het effectief uitvoeren van inferenties is respectievelijk 0.01%, 0.46% en 1.13%. Het is duidelijk dat er in de *cloud*-architectuur sprake is van veel *overhead*, veroorzaakt door de uplink- en downlinktijd.

De gemiddelde tijd per voorspelling neemt af naarmate het tijdsinterval voor verwerking toeneemt. Dit is te verklaren als er gekeken wordt naar het aantal bestanden dat in elk scenario wordt uitgelezen en weggeschreven:

1. In het scenario waarbij de data elke zes seconden wordt verwerkt (over een periode van één uur), wordt er 600 keer een bestand uitgelezen en evenzo 600 keer een voorspelling weggeschreven.
2. Wanneer de data echter elke vijf minuten wordt verwerkt, gebeurt het uitlezen en wegschrijven slechts 12 keer.
3. Wanneer de data echter elk uur wordt verwerkt, gebeurt het uitlezen en wegschrijven slechts één keer.

Naast het aantal keren dat bestanden worden gelezen en geschreven, is ook de gemiddelde uplink/downlinktijd van invloed. Wanneer data elke zes seconden wordt verwerkt, zijn er meer frequente verbindingen nodig, wat de kans op variabiliteit in de verbindingstijd kan vergroten. In tegenstelling, bij minder frequente verwerking, zoals bij de scenario's van elke 5 minuten of elk uur, wordt deze verbindingstijd over het algemeen verminderd.
Ten slotte moet ook het model minder frequent worden aangeroepen bij langere tijdsintervallen, wat verder bijdraagt aan een kortere gemiddelde tijd per voorspelling.

Tabel. 7: *Overzicht van de inferentietijd voor de verschillende delen in het edge-proces voor de drie verschillende scenario's.*

Scenario	Voorverwerking (ms)	Inferentie (ms)	Naverwerking (ms)	Totale Inferentie tijd (ms)	Gemiddelde tijd per predictie (ms)
6 sec.	16.73	1.37	1.72	19.82	19.82
5 min.	89.75	17.33	11.83	118.91	2.38
1 uur	664.0	97	76	837.0	1.40

Dezelfde aanpak wordt toegepast voor de metingen voor de *edge*-architectuur. Hierbij werden ook drie delen onderscheiden binnen de totale inferentietijd. De eerste is de voorverwerking, dit is zoals bij de cloud-architectuur de tijd die nodig is om de gegevens te lezen uit de databestanden en samples te maken van 192 datapunten. Als tweede wordt de effectieve inferentietijd gemeten. Als laatste proces wordt de naverwerking gemeten, dit is de tijd die nodig is om de inferenties lokaal op te slaan. Het is het duidelijk dat hoe groter het bestand, hoe meer tijd nodig is voor inferentie. Dit is logisch: hoe meer gegevenssamples voorspeld moeten worden, hoe langer het zal duren. Ook hier is de tijd voor het effectief uitvoeren van de inferenties niet de grootste. Het grootste deel wordt besteed aan de voorverwerking. De procentuele tijd dat gespendeerd wordt aan het effectief uitvoeren van inferenties is respectievelijk 6.91%, 14.57% en 11.59%.

Wanneer dit vergeleken wordt met de effectieve inferentietijd bij de *cloud*-architectuur is dit procentueel gezien veel groter. Hierdoor is er minder *overhead* van andere delen, voornamelijk veroorzaakt door de uplink- en downlinktijd bij de *cloud*-architectuur. De tijd voor het effectief uitvoeren van de inferenties is bij de *cloud*-architectuur wel telkens aanzienlijk minder dan bij de *edge*-architectuur. Dit komt omdat de *cloud*-architectuur krachtigere computationele bronnen heeft dan de smartphone. Dit is terug te vinden op **Fig. 4**.

Ook bij deze architectuur neemt de gemiddelde tijd per predictie af naarmate het tijdsinterval voor verwerking toeneemt. Bij deze architectuur leidt het minder frequent uitlezen van bestanden, opslaan van voorspellingen en het aanroepen van het machine learning model tot dit verschijnsel.

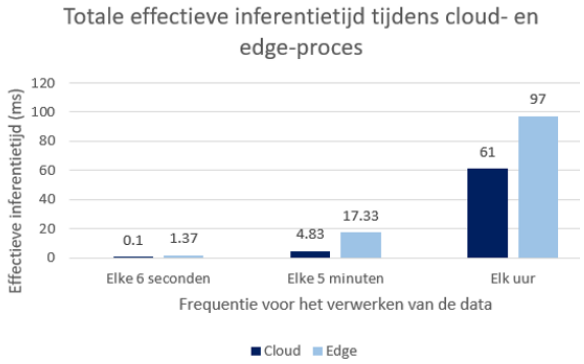


Fig. 4*: De effectieve inferentietijd bij de cloud- en edge-architectuur.*

Wanneer gekeken wordt naar de totale inferentietijd voor beide architecturen (**Fig. 5**), kan gesteld worden dat de *edge*-architectuur de snelste is. Zoals eerder werd vermeld weegt de *overhead* van de uplink- en downlinktijd zwaar door. In alle drie de scenario's is de *edge*-architectuur het voordeligst.

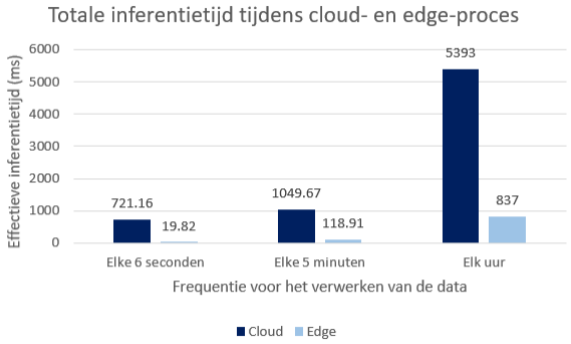


Fig. 5*: De totale inferentietijd bij de cloud- en edge-architectuur.*

C. INVLOED INTERNETNETWERK OP CLOUD-ARCHITECTUUR

Tot slot doet dit proefschrift onderzoek naar de invloed van het internetnetwerk op de batterijconsumptie en inferentietijd bij de *cloud*-architectuur. In deze architectuur wordt data over het internet verstuurd naar de virtuele node voor verwerking en worden de voorspelling over het internet verstuurt naar de smartphone. Het internet, waarover deze data verstuurd wordt, kan mogelijk een invloed hebben op zowel de batterijconsumptie als de inferentietijd.

Uit de resultaten blijkt dat de internetconnectie nauwelijks invloed heeft op de batterijconsumptie bij de *cloud*-architectuur. De resultaten voor de batterijconsumptie worden gevisualiseerd op **Fig. 6**.

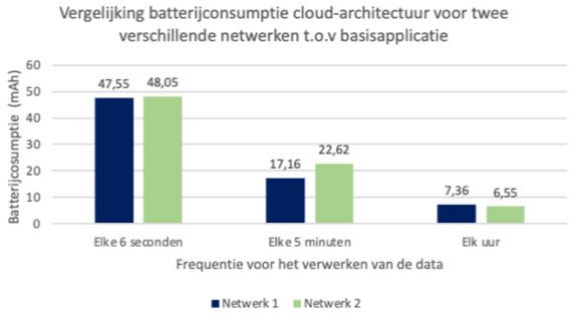


Fig. 6: *Vergelijking batterijconsumptie cloud-architectuur voor twee verschillende netwerken t.o.v. de basisapplicatie.*

Ook de tijden van de deelprocessen zijn gelijkaardig aan elkaar, dit is terug te vinden in **Tabel 8**. Enkel bij het scenario waarbij de data om het uur wordt verstuurd is een verschil merkbaar voor de uplink- en downlinktijd. Dit zou kunnen omdat het verkeer op het eerste netwerk op dat moment druk was. Aangezien de metingen zijn uitgevoerd op slechts twee netwerken volstaat het niet om een algemene conclusie te kunnen trekken over de invloed van het internetwerk op de *cloud*-architectuur. Daarnaast zijn er nog andere factoren zoals het aantal gebruikers op het netwerk en het verkeer over dat netwerk die invloed kunnen hebben op de metingen.

Tabel. 8: *Vergelijking inferentietijd cloud-architectuur voor twee verschillende internetwerken. (wit: zelfde netwerk als metingen voorgaande subsecties, grijs: nieuw netwerk).*

Scenario	Uplink/downlink -tijd (ms)	Effectieve inferentietijd (ms)	voor-en naverwerkingtijd (ms)
6 seconden	697.31	0.10	23.75
6 seconden	553.07	0.13	22.75
5 minuten	973.42	4.83	71.42
5 minuten	1014.25	4.67	72.16
1 uur	4694	61	638
1 uur	1967	63	642

VII. CONCLUSIE EN TOEKOMSTIG WERK

Dit proefschrift heeft als doel een overzicht te geven van de invloed van de *cloud*- en *edge*-architectuur op de batterijconsumptie en inferentietijd. Op basis hiervan kan besloten worden om mogelijk over te stappen van de huidige *cloud*-architectuur naar een *edge*-architectuur. Hierbij werd onderzoek gedaan naar de mogelijke optimalisatietechnieken om het model geschikt te maken voor een smartphone om de overgang te vereenvoudigen. De twee technieken die het meest voordelig zijn om te implementeren zijn de *Depthwise Separable Convolutions* en de *TensorFlow's Default Optimizer*. Deze technieken slagen er in om de grootte van het mBrain-model te reduceren respectievelijk met factor 1.51 en 1.67. Daarnaast slagen deze technieken er ook in de inferentietijd met 8-9 ms te verminderen. Er werd gemeten dat de effectieve inferentietijd 17.33 ms bedraagt. De technieken slagen er bijgevolg in de inferentietijd ongeveer te halveren. Dit allemaal zonder dat de nauwkeurigheid van het model aanzienlijk vermindert.

In het tweede deel wordt besloten dat de batterijconsumptie voor de *edge*-architectuur voordeliger was dan bij de *cloud*-architectuur. Dit verschil vermindert naarmate de data frequenter wordt verwerkt. Uit de metingen blijkt dat bij de *cloud*-architectuur het effectief uitvoeren van inferenties sneller is dan bij de *edge*-architectuur. Dit komt doordat het krachtere computationele bronnen bezit. Anderzijds is de *overhead* veroorzaakt door de uplink-en downlinktijd in de *cloud*-architectuur enorm. Hierdoor is de totale inferentietijd bij de *edge*-architectuur het snelst.

Er wordt besloten dat de overgang naar een *edge*-architectuur binnen de mBrain-studie voornamelijk voordelen met zich meebrengt. Zo is er geen nood aan een internetconnectie, is er minder nood aan bandbreedte, is de algemene batterijconsumptie minder en de inferentietijd is sneller. Daarnaast is een scenario waarbij de data elke vijf minuten wordt verwerkt een goede consensus voor batterijverbruik en inferentietijd t.o.v. de opslagvereiste die nodig is om de tijdelijk lokaal op te slaan. Het model kan geoptimaliseerd worden door het toepassen van *Depthwise Separable Convolutions* of de *TensorFlow's Default Optimizer* voor verdere vermindering van modelomvang en voor snellere inferenties, zonder aanzienlijk verlies aan nauwkeurigheid.

In deze scriptie is geen onderzoek gedaan naar een scenario waarbij meerdere *machine learning* modellen op een smartphone staan. Dit is een mogelijke uitbreiding op het onderzoek dat deze thesis heeft verwezenlijkt.

REFERENTIES

[1] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv [cs.CV]*, 2017.

[3] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan, "Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition," *IEEE Access*, vol. 7, pp. 133509–133520, 2019.

[4] P. Dhiman, V. Kukreja, and A. Kaur, "Citrus fruits classification and evaluation using deep convolution neural networks: An input layer resizing approach," in *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2021.

[5] "WISDM lab: Dataset," *Fordham.edu*. [Online]. Available: https://www.cis.fordham.edu/wisdm/dataset.php. [Accessed: 10-Jul-2023].

[6] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 1–29, 2017.

Optimizing Human Activity Recognition machine learning models for execution on edge devices.

Arthur Leclercq

Supervisors: Prof. Dr. Ir. Sofie Van Hoecke, Prof. Dr. Femke Ongenae

Counsellors: Ir. Bram Steenwinckel, Ir. Marija Stojchevska

Abstract- This thesis investigates the transition from a cloud architecture to an edge architecture for the Human Activity Recognition model of the mBrain study. The first part examines the impact of existing optimization techniques on the machine learning model of the mBrain study. These optimization techniques take into account the limited computational power, battery, and storage capacity of a smartphone where the model would be deployed. Four distinct techniques are applied and their effects on the model size, inference time, and accuracy are studied. The second part consists of an in-depth investigation into which architecture, cloud or edge, is most advantageous to implement within this study. This section explores the influence of both architectures on a smartphone's battery consumption and inference time. Results from this thesis indicate that techniques such as depthwise separable convolutions and a specific form of quantization have beneficial effects in terms of inference time and storage capacity. Additionally, the edge architecture emerges as an interesting alternative compared to the existing cloud architecture based on battery consumption and inference time.

Keywords- Human Activity Recognition, edge, cloud, quantization, pruning, depthwise separable convolutions, Convolutional Neural Network.

I. INTRODUCTION

The purpose of the mBrain study is to gain deeper insights into migraine attacks, examining lifestyle, behavior, and possible triggers. One focus is the development of a machine learning model for predicting and detecting migraine episodes. This includes a model responsible for tracking a person's daily activities. The model utilizes accelerometer data from a smartphone, capturing the device's acceleration in three directions: x, y, and z.

The mBrain study's current model, which predicts a patient's activities, operates in the cloud. This setup mandates a consistent internet connection, both for sending data to the virtual node and receiving predictions. However, transmitting data over the internet can be susceptible to hacking attempts, posing privacy risks, especially given that we're dealing with patient data. Future regulations might enforce restrictions on transmitting health-related data over the internet. Additionally, the mBrain study wants to ensure that predictions aren't solely dependent on internet connectivity. For these reasons, among others, the mBrain study aims not to rely on WIFI for prolonged patient monitoring. Another option is to process the data closer to the source, in an edge environment, negating the need to transmit accelerometer data over the internet.

This thesis delves into and describes this environment, highlighting its challenges compared to a cloud environment. These challenges include:

- Limited computational resources: Computational capabilities in the edge environment, such as on a smartphone, are less potent, which may result in slower predictions.
- Storage capacity: A smartphone offers limited storage.
- Battery life: A smartphone's battery has limited longevity before needing recharging.

Conversely, in a cloud architecture, factors like model size, battery consumption, and complexity are less concerning. This thesis investigates the transition to an edge environment, taking its limitations into account. Three techniques are studied in this context: pruning, quantization, and depthwise separable convolutions, as well as a combination of pruning and quantization. The study explores whether these techniques can reduce complexity, storage demands, and battery consumption without causing a significant drop in accuracy.

The second part investigates which of these two environments is most advantageous based on battery consumption and inference time, specifically within the mBrain study for Human Activity Recognition (HAR). Two architectures were designed, one for each environment. Measurements were taken for both to assess battery life and inference time impact. Three different scenarios were differentiated, processing the data every six seconds (each sample), every five minutes, or hourly. This way, the most suitable scenario and architecture for this application can be determined.

To achieve this goal, the thesis seeks to answer the following research questions:

1) What techniques exist to make a machine learning model lighter and faster, and what is their effect on inference time, model size, and accuracy?

2) Can a machine learning model be used on a smartphone without significant loss of accuracy?

3) Does the use of the edge-architecture for machine learning models have a positive impact on battery consumption and inference time compared to a cloud architecture?

II. RELATED WORK

A popular application is image processing directly on a smartphone. A machine learning model is placed on the smartphone, which can then make predictions on images. This application often uses a Convolutional Neural Network (CNN), a type of network that contains convolutions, which are computationally intensive. Xception [1] was the first to introduce the concept of Depthwise Separable Convolutions (DSC's), later integrated into MobileNet [2]. The purpose is to split the regular computationally intensive convolutions into two less computationally intensive parts: a depthwise convolution and a pointwise convolution. In depthwise convolutions, each channel, such as the red, green, and blue color channels for images, or the x, y, and z-axis for HAR, is processed separately. After the depthwise convolution, a pointwise convolution is applied over all three channels. This reduces the number of parameters by $\frac{1}{N} + \frac{1}{D_K^2}$, where N represents the number of kernels and D_K represents the filter size.

T. Zebin et al. [3] applied quantization to reduce a machine learning model's complexity. Quantization is a technique where numbers are represented with lower precision. They switched from floating-point numbers to 8-bit fixed-point numbers, decreasing the CNN model's size by a factor of 7.62, while accuracy only dropped by 2.8%. This study did not investigate the impact on inference time.

P. Dhiman et al. [4] took it a step further. In addition to post-training quantization, they also applied magnitude-based pruning to a CNN. Their objective was to create an efficient and lighter model so that disease detection in citrus fruits could be performed directly in the fields using sensors. Through pruning, a predefined number of the model's weights are set to zero. Specifically, magnitude-based pruning sets the smallest weights to zero during training. This is because these smaller weights typically have a lesser impact on the model's output. As a result, the weight matrices contain many zero values, potentially allowing for more efficient storage. After applying pruning, the model's size decreased by a factor of 1.88, and accuracy decreased by 2.74%. When post-training quantization was also applied, the model's size dropped by a factor of 2.56, and accuracy declined by 4.52%. Combining both techniques further reduced the model size but at the cost of a greater accuracy loss. This study also did not investigate the impact on inference time.

III. BASE MODELS

To investigate the influence of different optimization techniques on inference time, storage size, and accuracy, two base models are trained. Both models are CNNs and are used for Human Activity Recognition. The first model, the WISDM model, is trained on the WISDM dataset for HAR [5], which is publicly available and contains tri-axial accelerometer data. The second model, the mBrain model, is trained on the dataset from the mBrain study, which also contains tri-axial accelerometer data. The difference between the two models is that the WISDM model has fewer layers but contains more and larger filters. As a result, the number of parameters and the model size is significantly larger. By subjecting both models to the same optimization techniques, the varying impact of each technique on two different models can be investigated.

Table 1: *Properties of the base models.*

	WISDM-model	mBrain-model
#parameters	66 678	13 653
Model size	272 772 bytes	71 960 bytes

IV. TESTING ENVIRONMENT

To perform reliable measurements, a specific testing environment is created. Firstly, the same smartphone, a Samsung Galaxy S10e, is used throughout the entire study for the measurements. This smartphone has a Samsung Exynos 9820 Octa-core CPU with 6 gigabytes of RAM and a Lithium-Ion battery with a capacity of 3000 mAh.

To realize the cloud architecture, a virtual node is used. On this virtual node, a Flask application is implemented to which the accelerometer data can be sent for processing. After processing, the predictions are sent back.

Lastly, two Android applications have been developed to perform the measurements. These applications are installed on the smartphone where the measurements take place. These apps are responsible for retrieving the smartphone's accelerometer data. These values are then displayed in the app to the user and stored locally in a CSV file. For the cloud application, the data is sent to the virtual node for processing at predetermined time intervals. For the edge application, the machine learning model to be tested is placed within the app. The collected accelerometer data is then fed to the machine learning model on the smartphone at predetermined time intervals. Predictions are stored locally for both applications.

V. OPTIMIZATION TECHNIQUES

The three techniques that emerged from the literature study are applied to both models. These techniques are: depthwise separable convolutions, pruning, and quantization. For the measurements, a slightly different edge application is used than the one described in **Section IV**. In each case, use is made of a file that was collected in advance. The file consists of accelerometer data from five minutes at 32Hz and therefore contains 9,600 data points for each of the three axes.

A. DEPTHWISE SEPARABLE CONVOLUTIONS

This technique succeeds in reducing the number of parameters by a factor of 6.63 and 2.18 for the WISDM model and the mBrain model, respectively. In **Section II**, it was mentioned that the number of parameters is reduced by $\frac{1}{N} + \frac{1}{D_K^2}$, where N represents for the number of kernels and D_K represents the size of the filters. Because the WISDM model uses more and larger filters, the factorial decrease is larger than with the mBrain model.

***Table 2**: Results of Depthwise Separable Convolutions on the WISDM-model and the mBrain-model.*

	WISDM-model		mBrain-model	
	Conv1D	SeparableConv1D	Conv1D	SeparableConv1D
# parameters	66 678	10 052	13 653	6 260
Model size (TFLite-format)	272 772 bytes	47 532 bytes	71 960 bytes	47 664 bytes
Inference time	172 ms	149 ms	136 ms	127 ms
Accuracy	87.83%	88.02%	77.75%	76.92%

Due to the reduction in parameters, the model size also decreases significantly for both models. Since there are fewer multiplications and additions to be made, the inference time for both models decreases. The decrease is 1.15 and 1.07 respectively for the WISDM model and the mBrain model. This decrease in inference time may be beneficial for battery consumption.

B. PRUNING

In this technique, during the training process, more and more weights of the CNN model are assigned the zero value. The Tensor Algebra Compiler is then used to store the pruned weight matrices in a more efficient manner [6].

Figure 1 illustrates the impact of pruning on the model size and accuracy for the WISDM model. The model was pruned to 25%, 50%, 75%, 80%, 85%, and 90%. The inference time has not changed after applying pruning. From the results, it is clear that this technique is able to reduce the model size of the WISDM model. This reduction factor depends on the degree to which pruning is applied. The accuracy decreases considerably in this case after 50% pruning.

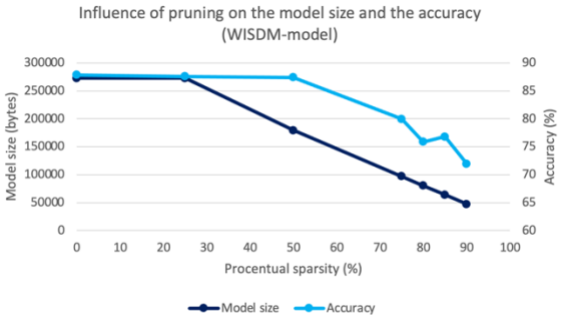


***Figure 1:** Influence of pruning on the model size and the accuracy on the WISDM-model.*

Figure 2 illustrates the impact of this technique on the model size and accuracy for the mBrain model. The model was pruned to 25%, 50%, 75%, 80%, 85%, and 90%. The inference time remains the same for this model after pruning. The results are in line with the WISDM model. The more pruning, the more compact the model can be stored. However, the tipping point for accuracy is at 80% pruning. The figure shows that the accuracy increases from this degree of pruning. However, this gives a distorted picture. From then on, the model never predicts certain minority classes. This is therefore an undesirable phenomenon. The technique is therefore only interesting up to 75% pruning for the mBrain model.

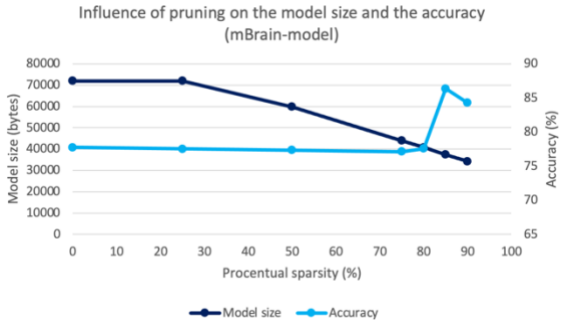


***Figure 2:** Influence pruning on the model size and the accuracy on the mBrain-model.*

C. POST-TRAINING QUANTIZATION

Table 3 summarizes the results of this technique on the WISDM model. The 'None' row represents the results after converting from the Keras format to the TensorFlow Lite format. Float16 quantization manages to reduce the model size by a factor of 1.94, without significant loss in accuracy. The inference time remains virtually untouched. Kotlin, the programming language in which the apps were developed, does not support the float16 datatype, so float32 operations are still performed. This keeps the inference time unchanged. Uint8 quantization results in a model size reduction factor of 3.43 and accelerates the inference process by a factor of 1.16. The standard TensorFlow optimizer (dynamic) manages to reduce the model size by a factor of 3.58, while accuracy does not decrease significantly. Additionally, the inference time decreases by 1.18.

***Table 3**: Results of post-training quantization on the WISDM-model.*

Type	Size	Inference time (files of 5 minutes)	Accuracy
Keras model	1 446 560 bytes	/	88.53%
None (float32)	272,772 bytes	172 ms	87.83%
Float16	140 832 bytes	169 ms	86.32%
Uint8	79 544 bytes	148 ms	86.23%
Dynamic	76 288 bytes	146 ms	86.37%

In **Table 4**, the results of post-training quantization for the mBrain model are summarized. The general trend is similar to that of the WISDM model. The reason the reduction factor for model size and inference time is lower is because this model has fewer trainable parameters (**Table 2**), so the influence is less. With the uint8 quantization, the accuracy of the model increases. However, this gives a distorted picture, because from that point on, the model no longer predicts any minority classes. This is an undesirable outcome.

***Table 4**: Results of post-training quantization on the mBrain-model.*

Type	Size	Inference time (files of 5 minutes)	Accuracy
Keras model	152 248 bytes	/	77.88%
None (float32)	71 960 bytes	136 ms	77.75%
Float16	52.252 bytes	133 ms	77.76%
Uint8	40 864 bytes	131 ms	82.71%
Dynamic	43 188 bytes	128 ms	77.83%

D. PRUNING AND POST-TRAINING QUANTIZATION

Lastly, a combination of quantization and pruning is investigated. First, the model is pruned and then the standard TensorFlow optimizer (dynamic) is used for quantization.

No further decrease in accuracy is observed after quantization is applied. The model size decreases even further after also applying quantization. The reduction in inference time remains the same as when only quantization was used.

When comparing the combined technique with the individual techniques, it only has advantages over pruning. The combined technique has benefits compared to quantization up to a certain thinning. When the thinning in the combined technique exceeds 50%, it's more advantageous to use only quantization.

VI. CLOUD VS EDGE

The second part of this thesis investigates the cloud and edge environments. The influence of both architectures on inference time and smartphone battery consumption is examined. Measurements are taken on the smartphone, virtual node, and Android apps, as described in **Section II**. Three different scenarios are explored. In the first scenario, data is continuously processed (every six seconds), in the second scenario every five minutes, and in the third scenario every hour. Additional research was also conducted on the influence of networking on battery consumption and inference time for the cloud architecture. The measurements were performed on the TFLite-converted version of the mBrain model without optimization techniques. Each measurement lasted a total of one hour.

A. BATTERY CONSUMPTION

Firstly, the battery capacity consumed by the basic application without data processing is measured (357.84mAh). It's important to note that during these measurements the screen was continuously on at maximum brightness. This explains why the consumption is so high.

The same measurements are then taken for the previously mentioned three scenarios. To provide a clear overview, the consumption of the basic application is subtracted for the three examined scenarios. This way, it's determined how much additional battery is required to process the data in the cloud or on the smartphone itself (edge).

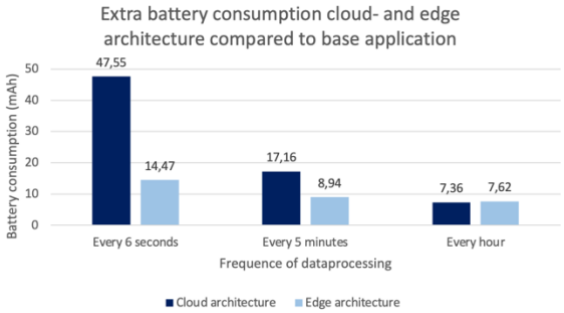


***Figure 3**: Extra battery consumption for both architectures compared to the base application.*

The measurements show that the scenario where processing occurs every six seconds consumes the most battery, applicable to both environments. For the cloud environment, this means that the data packets sent over the internet are smaller than in the other two scenarios. Despite this, this scenario is the most energy intensive. The frequency of sending data packets over the internet weighs more heavily than the size of these data packets. In total, accelerometer data is sent 600 times and the prediction is returned to the application 600 times during this measurement. For the second scenario, this happens 12 times and in the third scenario only once.

***Table 5**: Overview number of datapoints per axis and file size for the three different scenarios.*

Scenario	Duration	Data points per axis	Size (kB)
1	6 sec.	192	54
2	5 min.	9600	408
3	1 hour	115 200	4 710

In the edge scenario, the same pattern is evident: the more frequently the data is processed, the more battery it consumes. The increase in battery consumption is primarily due to the frequent activation of resources (processors, memory modules, or other hardware components), which entails an inherent startup cost. This very startup cost is a decisive factor in higher energy consumption. This explains why significantly less battery capacity is consumed when the time interval for processing data increases.

When both architectures are compared side by side, the edge architecture is the least energy intensive. The most significant difference is in the scenario where processing takes place every six seconds. Using the cloud architecture results in 3.29 times more milliamp-hours on top of the basic application. The difference in consumption is still clearly noticeable in the scenario where the data is processed every five minutes, but much less than in the first scenario. By the third scenario, no difference is discernible.

When battery consumption is extremely important, it is advantageous to process data as infrequently as possible, regardless of the architecture used. However, a drawback is that the predictions will only be visible every hour. Moreover, the edge architecture proves to be more advantageous than the

cloud architecture. The less frequently the data is processed, the more similar the consumption becomes between both architectures.

B. INFERENCE TIME

Here, inference time refers to the time needed to make the predictions visible to the application's users. For the cloud architecture, this is subdivided into three distinct parts. The first part is the uplink and downlink time. The uplink time is the duration needed to send accelerometer data from the smartphone to the virtual node in the cloud. The downlink time is the duration required to send the predictions back from the virtual node to the smartphone. The second part is the time spent on actually making predictions by the model, the effective inference time. The third part is the pre- and post-processing; these are all other times within the virtual node that are not dedicated to the actual execution of predictions. In this way, the inference time within the cloud architecture is thus divided into three separate processes.

Table 6: *Overview of the inference time for the different parts in the cloud-process for the three different scenarios.*

Scenario	preprocessing (ms)	Inference (ms)	postprocessing (ms)	Total (ms)	Average time per prediction (ms)
6 sec.	16.73	1.37	1.72	19.82	19.82
5 min.	89.75	17.33	11.83	118.91	2.38
1 uur	664.0	97	76	837.0	1.40

Table 6 shows the inference times for the different phases in the cloud process for the three scenarios. It becomes clear that larger files result in longer uplink and downlink times. The same applies to the model's total inference time since more predictions are being made. These values reflect the time for sending and processing a file of six seconds, five minutes, and one hour. It is evident that the time actually spent on making inferences by the machine learning model is extremely low. The vast majority of the time is dedicated to sending and receiving data packets over the internet. The time lost on uplink and downlink times accounts for 96.7%, 92.7%, and 87.0% of the total time, respectively. The percentage of time spent on actual inference execution is 0.01%, 0.46%, and 1.13%, respectively. It's clear that there's a lot of overhead in the cloud architecture, caused by the uplink and downlink times.

The average time per prediction decreases as the processing time interval increases. This can be explained when looking at the number of files that are read and written in each scenario:

1. In the scenario where the data is processed every six seconds (over a period of one hour), a file is read 600 times, and likewise, a prediction is written 600 times.
2. However, when the data is processed every five minutes, reading and writing occur only 12 times.
3. When the data is processed every hour, reading and writing happen only once.

Besides the number of times files are read and written, the average uplink/downlink time also influences. When data is processed every six seconds, more frequent connections are required, which can increase the chance of variability in connection time. In contrast, with less frequent processing, as in the scenarios of every 5 minutes or every hour, this connection time is generally reduced. Lastly, the model is invoked less frequently at longer time intervals, further contributing to a shorter average time per prediction.

Table 7*: Overview of the inference time for the different parts in the edge-process for the three different scenarios.*

Scenario	preprocessing (ms)	Inference (ms)	postprocessing (ms)	Total (ms)	Average time per prediction (ms)
6 sec.	697.31	0.10	23.75	721.16	721.16
5 min.	973.42	4.83	71.42	1049.67	20.99
1 uur	4696.0	61.0	638.0	5393.0	8.99

The same approach is applied to the measurements for the edge architecture. Within the total inference time, three parts were also distinguished. The first is the preprocessing, which, like in the cloud architecture, is the time required to read data from the data files and make samples of 192 data points. Second, the effective inference time is measured. The last process is the post-processing, which is the time needed to locally store the inferences. The larger the file, the more time is needed for inference. This makes sense: the more data samples that need to be predicted, the longer it will take. Here too, the time for effectively performing the inferences is not the longest. The majority is spent on preprocessing. The percentage of time spent on actual inference execution is 6.91%, 14.57%, and 11.59% respectively.

When this is compared to the effective inference time in the cloud architecture, it is proportionally much larger. As a result, there is less overhead from other parts, mainly caused by the uplink and downlink times in the cloud architecture. The time for actual inference execution in the cloud architecture is consistently considerably less than in the edge architecture. This is because the cloud architecture has more powerful computational resources than the smartphone. This can be seen in **Figure 4.**

Also, in this architecture, the average time per prediction decreases as the processing time interval increases. In this architecture, less frequent reading of files, storing predictions, and invoking the machine learning model lead to this phenomenon.

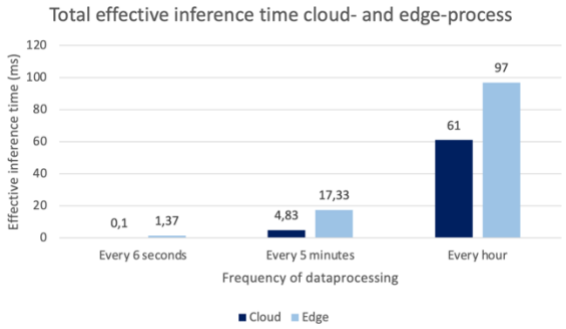


***Figure 4**: The effective inference time at the cloud and edge architecture.*

When looking at the total inference time for both architectures (**Figure 5**), it can be said that the edge architecture is the fastest. As mentioned earlier, the overhead of the uplink and downlink times weighs heavily. In all three scenarios, the edge architecture is the most advantageous.

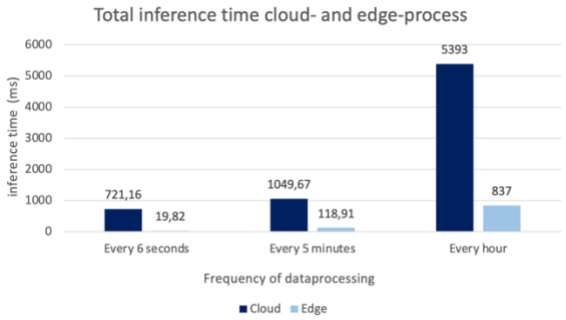


***Figure 5**: The total inference time at the cloud and edge architecture.*

C. INFLUENCE OF THE INTERNET NETWERK ON THE CLOUD-ARCHITECTUUR

Finally, this thesis investigates the influence of the internet network on battery consumption and inference time in the cloud architecture. In this architecture, data is sent over the internet to the virtual node for processing, and the predictions are sent over the internet to the smartphone. The internet, over which this data is sent, can potentially influence both battery consumption and inference time.

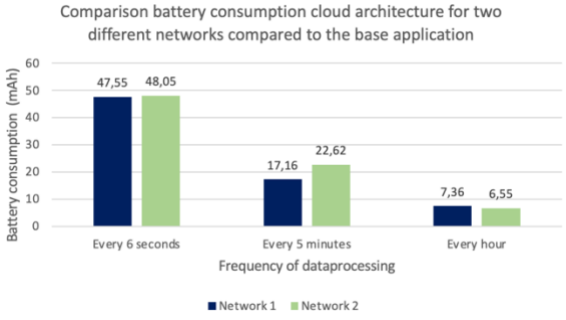


***Figure 6**: Comparison of the battery consumption cloud architecture for two different internet networks compared to the base application.*

The results show that the internet connection hardly affects the battery consumption in the cloud architecture. The results for battery consumption are visualized in **Figure 6.**

The times of the subprocesses are also similar to each other, which can be found in **Table 8**. Only in the scenario where the data is sent every hour is there a noticeable difference for the uplink and downlink time. This could be because the traffic on the first network was busy at that time. Since measurements were only taken on two networks, it is not enough to draw a general conclusion about the influence of the internet network on the cloud architecture. In addition, there are other factors such as the number of users on the network and the traffic over that network that can influence the measurements.

***Table 8**: Comparison of inference time for cloud architecture across two different internet networks. (white: same network as measurements from previous subsections, gray: new network).*

Scenario	Uplink/downlink time (ms)	Effective inference time (ms)	pre- and postprocessing time (ms)
6 seconds	697.31	0.10	23.75
6 seconds	553.07	0.13	22.75
5 minutes	973.42	4.83	71.42
5 minutes	1014.25	4.67	72.16
1 hour	4694	61	638
1 hour	1967	63	642

VII. CONCLUSION AND FUTURE WORK

The aim of this thesis is to provide an overview of the influence of cloud and edge architecture on battery consumption and inference time. Based on this, a decision can be made to possibly switch from the current cloud architecture to an edge architecture. Research was conducted into potential optimization techniques to adapt the model for a smartphone, thereby simplifying the transition. The two techniques that are most advantageous to implement are the Depthwise Separable Convolutions and TensorFlow's Default Optimizer. These techniques manage to reduce the size of the mBrain model by factors of 1.51 and 1.67, respectively. Additionally, these techniques also manage to reduce the inference time by 8-9 ms. Measurements showed that the effective inference time is 17.33 ms. As a result, these techniques almost halve the inference time. All this without significantly reducing the accuracy of the model.

In the second part, it is concluded that the battery consumption for the edge architecture was more favorable than for the cloud architecture. This difference decreases as the data is processed more frequently. Measurements show that in the cloud architecture, the actual execution of inferences is faster than in the edge architecture. This is because it has more powerful computational resources. On the other hand, the overhead caused by the uplink and downlink time in the cloud architecture is substantial. As a result, the total inference time in the edge architecture is the fastest.

It is decided that transitioning to an edge architecture within the mBrain study mainly brings advantages. For example, there is no need for an internet connection, less need for bandwidth, overall battery consumption is lower, and the inference time is faster. In addition, a scenario where data is

processed every five minutes provides a good balance between battery consumption and inference time relative to the storage requirement needed to temporarily save locally. The model can be optimized by applying Depthwise Separable Convolutions or TensorFlow's Default Optimizer for further reduction in model size and for faster inferences, without significant loss in accuracy.

In this thesis, no research was conducted into a scenario where multiple machine learning models are present on a smartphone. This is a potential extension of the research accomplished by this thesis.

REFERENCES

[1] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv [cs.CV]*, 2017.

[3] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan, "Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition," *IEEE Access*, vol. 7, pp. 133509–133520, 2019.

[4] P. Dhiman, V. Kukreja, and A. Kaur, "Citrus fruits classification and evaluation using deep convolution neural networks: An input layer resizing approach," in *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2021.

[5] "WISDM lab: Dataset," *Fordham.edu*. [Online]. Available: https://www.cis.fordham.edu/wisdm/dataset.php. [Accessed: 10-Jul-2023].

[6] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 1–29, 2017.

INHOUDSTABEL

LIJST VAN FIGUREN

LIJST VAN TABELLEN

LIJST MET AFKORTINGEN

Afkorting	Definitie	Pagina
ADB	Android Debug Bridge	89
COO	Coordinate List	56
CNN	Convolutional Neural Network	9
CSC	Compressed Sparse Column	56
CSR	Compressed Sparse Row	56
DSC	Depthwise Separable convolutions	17
DT	Decision Tree	6
FLS-SVM	Fuzzy Least Squares Support Vector Machine	8
FNN	Forwarded Neural Network	6
HAR	Human Activity Recognition	2
HMM	Hidden Markov Model	7
HTTP	Hypertext Transfer Protocol	50
IaaS	Infrastructure as a Service	84
kB	kilobyte	1
kNN	k-Nearest Neighbor	6
LMT	Logistic Model Tree	4
LS-SVM	Least Squared Support Vector Machine	8
LSTM	Long Short-Term Memory	11
mAh	milliampère-uur	89
MB	Megabyte	1
ML	Machine Learning	4
MLP	Multilayer Perceptron	23
NB	Naïve Bayes	6
NIST	National Institute of Standards and Technology	83
NTP	Network Transfer Protocol	92
PaaS	Platform as a Service	84

RBF	Radius Basis Function	7
ReLU	Rectified Linear Unit	18
RF	Random Forest	5
RNN	Recurrent Neural Network	11
RTT	Round Trip Time	92
SaaS	Software as a Service	84
SVM	Support Vector Machine	4
TACO	Tensor Algebra Compiler	56
TFLite	TensorFlow Lite	15
t.o.v.	Ten opzichte van	52
UI	User Interface	43

1. INTRODUCTIE

Migraine is een primaire hoofdpijnstoornis [1] die 12-14,4% van de wereldbevolking treft, waarbij hoofdpijn zelf het kernprobleem vormt en geen symptoom van een onderliggende ziekte of aandoening is [2,3]. De pijn van een migraineaanval is vaak invaliderend en weerhoudt patiënten van hun dagelijkse activiteiten, wat een enorme impact heeft op het sociaal en economische aspect van het leven van de patiënt. Eén van de belangrijkste triggers is stress. Wanneer een patiënt veel stress ervaart, wordt de kans op een migraineaanval vergroot. Het is bewezen dat meer beweging zorgt voor een verminderde stress, met als gevolg een kleinere kans op een migraineaanval [4]. Het monitoren van de verschillende activiteiten van een patiënt kan nuttig zijn bij het identificeren en beheren van mogelijke triggers om migraineaanvallen te voorkomen, te verminderen of te detecteren.

1.1. Probleemstelling

De mBrain-studie is bezig met het maken van een *machine learning* model voor het voorspellen en detecteren van migraineaanvallen. Daarbij hoort een model dat verantwoordelijk is voor het detecteren van de activiteiten van de patiënt doorheen de dag. Het huidige model van de mBrain-studie, dat de activiteiten van een patiënt voorspelt, draait in de *cloud*. Dit model doet op basis van tri-axiale accelerometerdata, gecollecteerd van de smartphone, predicties over de activiteiten doorheen de dag. Hierbij wordt de data in eerste instantie gedurende twee minuten opgeslagen op de smartphone. Vervolgens wordt deze data verstuurd naar een virtuele node waar een *machine learning* model op draait en elke vijf minuten predicties uitvoert. Wanneer de predicties gemaakt zijn, worden deze teruggestuurd naar de smartphone. Wanneer deze data gedurende vijf minuten wordt opgeslagen, is de omvang hiervan ongeveer 408 kilobyte (kB). Stel dat er gedurende één uur geen internetconnectie mogelijk is en deze data gedurende een uur niet verwerkt worden, is de omvang van dit databestand ongeveer 4710 kB of 4.7 megabyte (MB). Het versturen van deze grote hoeveelheid data kost bandbreedte, tijd en energie. Bovendien vereist deze opzet een constante internetconnectie, zowel voor het verzenden van de data naar de virtuele node als voor het ontvangen van de predicties. Het verzenden van data over het internet

is echter vatbaar voor hackpogingen, wat privacyrisico's inhoudt, zeker gezien het hier om patiëntendata gaat. In de toekomst kunnen er regelgevingen worden ingevoerd die restricties opleggen aan het verzenden van dergelijke gezondheidsdata over het internet. Bijkomend wil de mBrain-studie voorkomen dat predicties enkel mogelijk zijn bij beschikbaarheid van een internetconnectie. Mede om deze redenen wenst de mBrain-studie niet afhankelijk te zijn van wifi voor langdurige patiëntmonitoring.

1.2. Doelstelling

Een andere omgeving waar de verwerking van de data kan worden uitgevoerd is de *edge*-omgeving. Een *edge*-omgeving verwijst naar het uitvoeren van gegevensverwerking en analyse dicht bij de gegevensbron, zoals in dit geval op de smartphone. Dit staat in contrast met de *cloud*-omgeving waarbij gegevens worden verzonden naar centrale *servers* voor verwerking. De eigenschappen van deze omgeving zijn verschillend van de *cloud*-omgeving. Zo moet de data niet over het internet worden verstuurd, enkel de predictie van het model kan eventueel gestuurd worden naar de centrale server. Hierdoor wordt de nood aan bandbreedte gereduceerd en kan de monitoring verder gaan wanneer er geen ineternetconnectie aanwezig is. In deze scriptie wordt verder onderzoek gedaan naar de invloed van deze omgeving op de tijd die nodig is om een predictie uit te voeren en het energieverbruik dat hiermee gepaard gaat. Het doel van deze thesis is om een overzicht te geven over welke architectuur het interessantst is om te gebruiken voor het *Human Activity Recognition* model van de mBrain-studie. Hierbij wordt onderzocht wat de impact is van een *edge*-architectuur op de inferentietijd en batterijconsumptie in vergelijking met een *cloud*-architectuur.

OV1. Welke technieken bestaan er om een *machine learning* model lichter en sneller te maken en wat is de invloed hiervan op de inferentietijd, modelgrootte en de accuraatheid?

OV2. Kan een *machine learning* model uitgevoerd worden op een smartphone zonder dat daarmee een aanzienlijk verlies aan accuraatheid gepaard gaat?

OV3. Heeft het gebruik van de *edge*-architectuur voor *machine learning* modellen en positieve impact op de batterijconsumptie en inferentietijd in vergelijking met een *cloud*-architectuur?

1.3. Structuur van deze scriptie

De scriptie is als volgt georganiseerd. **Sectie 2** plaatst deze scriptie in de bestaande literatuur, met de nadruk op onderzoek naar verschillende *machine learning* types die gebruikt worden voor HAR. Daarnaast doet het ook onderzoek naar verschillende optimalisatietechnieken die gebruikt worden om een *machine learning* model lichter te maken. **Sectie 3** behandelt de twee basismodellen die gebruikt worden waarop de optimalisatietechnieken worden toegepast. **Sectie 4** beschrijft het evaluatiekader. **Sectie 5** behandelt de optimalisatietechnieken en de resultaten hiervan nadat ze werden toegepast op twee basismodellen. **Sectie 6** gaat dieper in op zowel de *cloud*-architectuur en de *edge*-architectuur. Daarnaast bevat het de bespreking van de resultaten op de inferentietijd en de batterijconsumptie van beide architecturen. Hierbij worden drie verschillende scenario's onderscheidden. Hierbij wordt de data gedurende zes seconden, vijf minuten of één uur opgeslagen vooraleer deze data wordt verwerkt. **Sectie 7** behandelt de duurzaamheidsreflectie van deze scriptie. Tot slot beschrijft **Sectie 8** de conclusies van deze scriptie en doet suggesties voor toekomstig werk op basis van de resultaten van deze scriptie.

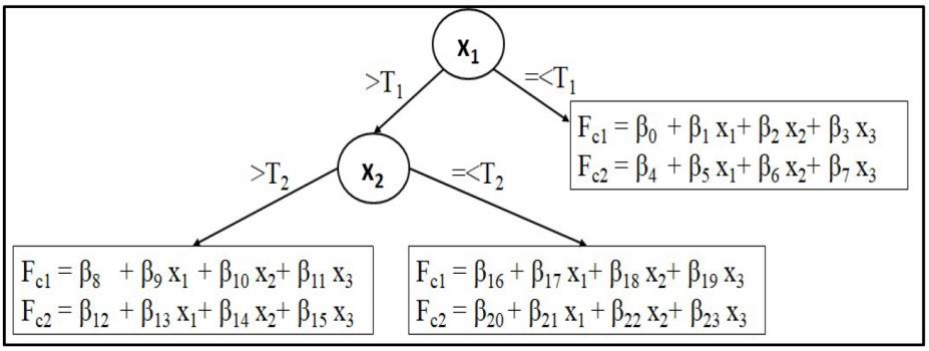
2. LITERATUURSTUDIE

Het doel van deze scriptie is om een *machine learning* (ML) model voor *Human Activity Recognition* (HAR) te modificeren en dit naar de *edge* te verplaatsen. Hierbij dient rekening te houden met de beperkte bronnen die beschikbaar zijn op randapparatuur zoals smartphones. In het eerste deel wordt onderzoek gedaan naar de verschillende types van modellen die in de literatuur naar voor komen voor HAR. Hier worden de drie meest frequent gebruikte modellen voor HAR besproken, namelijk boom-gebaseerde modellen, *Support Vector Machines* (SVM) en Neurale Netwerken (NN). In het tweede deel wordt de focus specifiek gelegd op onderzoeken voor HAR op de *edge*, evenals diverse methoden om modellen *edge*-vriendelijker te maken.

2.1. Human Activity Recognition modellen

2.1.1. Boom-gebaseerde modellen voor HAR

H. Nematallah et al. maken gebruik van een *Logistic Model Tree* (LMT) voor HAR. Ze combineren logistieke regressie met boom-gebaseerde modellen [5]. Het resultaat is een standaard boomstructuur met logistieke regressiefuncties in de bladeren. **Figuur 1** toont een eenvoudige LMT voor een binair classificatieprobleem.



Figuur 1: *voorbeeld van een LMT voor binair classificatieprobleem [5].*

Binnen dit onderzoek werd er gebruik gemaakt van publieke datasets, enerzijds de WISDM dataset [6] en anderzijds de UCI dataset [7]. De exacte eigenschappen van deze datasets staan vermeld op het einde van deze sectie in **Tabel 6**. De data werd gesegmenteerd in vensters met 50% overlap en de testen werden op verschillende venstergroottes uitgevoerd. Van elke sensor werden er 40 kenmerken geëxtraheerd, namelijk:

- Gemiddelde waarde en standaardafwijking per vrijheidsgraad (zes kenmerken)
- Het gemiddelde absolute verschil voor elke vrijheidsgraad (drie kenmerken)
- Grootte van elke sensormeting (één kenmerk)
- Histogram van elke sensormeting (30 kenmerken)

Om het model te evalueren is er gebruik gemaakt van *10-subject-cross validation*. Hierdoor wordt verzekerd dat data van gebruikers die aanwezig waren bij het trainingsgedeelte zeker niet aanwezig zijn bij het testgedeelte en zo te garanderen dat er geen sprake is van *data leakage*.

Activity	**Recognition Intervals of length 4 sec**			**Recognition Intervals of length 1 sec**		
	RF	***LR***	***LMT***	***RF***	***LR***	***LMT***
Jogging	0.98 / 0.98 / 0.98	0.98 / 0.98 / 0.98	0.99 / 0.99 / 0.99	0.97 / 0.98 / 0.98	0.96 / 0.98 / 0.97	0.98 / 0.98 / 0.98
Walking	0.92 / 0.94 / 0.93	0.88 / 0.93 / 0.90	0.93 / 0.96 / 0.95	0.90 / 0.92 / 0.91	0.80 / 0.94 / 0.86	0.88 / 0.96 / 0.92
Upstairs	0.75 / 0.74 / 0.75	0.64 / 0.57 / 0.60	0.83 / 0.79 / 0.81	0.69 / 0.67 / 0.68	0.59 / 0.39 / 0.47	0.77 / 0.66 / 0.71
Downstairs	0.71 / 0.65 / 0.68	0.58 / 0.47 / 0.52	0.78 / 0.72 / 0.74	0.65 / 0.61 / 0.63	0.57 / 0.33 / 0.42	0.74 / 0.60 / 0.66
Sitting	0.99 / 0.97 / 0.98	0.97 / 0.94 / 0.95	0.99 / 0.98 / 0.98	0.98 / 0.98 / 0.98	0.98 / 0.98 / 0.98	0.98 / 0.98 / 0.98
Standing	0.98 / 0.99 / 0.99	0.97 / 0.96 / 0.96	0.98 / 0.97 / 0.98	0.98 / 0.99 / 0.98	0.98 / 0.98 / 0.98	0.99 / 0.98 / 0.98
Weighted Avg	**0.91 / 0.91 / 0.91**	**0.87 / 0.88 / 0.87**	**0.93 / 0.93 /0.93**	**0.89 / 0.89 / 0.89**	**0.83 / 0.84 / 0.83**	**0.90 / 0.90 / 0.90**
Accuracy	**91.85 %**	**88.39 %**	**93.95%**	**89.57 %**	**84.76%**	**90.86%**

Figuur 2: *Resultaten voor venstergroottes van vier seconden en één second. Elke cel geeft respectievelijk de precisie, recall en f-score van het getrainde LMT model dat geïmplementeerd werd in vergelijking met Random Forest en Logistieke Regressie modellen van vergelijkbare grootte. [5]*

De studie concludeert dat de venstergrootte een invloed heeft op validatiescores. Een groter venster van vier seconden met 80 samples leidt tot een beter presterend model dan een venster met minder samples. Hoewel de nauwkeurigheid afneemt, blijft het LMT-model over het algemeen beter presteren dan andere modellen zoals Random Forest (RF) en Logistieke Regressie (LR). Logistieke Regressie presteert het slechtst voor beide vensterlengtes.

G. De Leonardis et al. vergelijken verschillende *classifiers* voor *real-time* toepassingen in HAR [8]. Dit artikel vergelijkt *k-Nearest Neighbor* (kNN), *Forwarded Neural Network* (FNN), *Support Vector Machine* (SVM), Naïve Bayes (NB) en *Decision Tree* (DT) voor HAR. In dit gedeelte wordt alleen het DT-model besproken, SVM en Neurale Netwerken (NN) worden later in deze sectie behandeld. Het onderzoek maakt gebruik van eigen proefpersonen. De details en kenmerken van deze dataset zijn beschreven in **Tabel 6**. De data is opgedeeld in vensters van vijf seconden zonder overlap. Er zijn 38 kenmerken geëxtraheerd:

- 21 kenmerken uit het tijd-domein (gemiddelde waarde, variantie, standaardafwijking, scheefheid, kurtosis, minimum- en maximumwaarde, 25e en 75e percentiel, interkwartielafstand en 11 centrale samples van de autocorrelatiesequentie)
- Drie kenmerken uit het frequentiedomein (gemiddelde waarde en mediaanfrequentie van het powerspectrum en de Shannon spectrale entropie)
- 14 kenmerken van het tijd-frequentiedomein (normen van de schatting en detailcoëfficiënten, met zeven niveaus van de decompositie van de Discrete Wavelet Transformatie)

Dit resulteert in de extractie van 342 (9 signalen × 38 kenmerken) elke vijf seconden. Vervolgens worden redundante kenmerken geïdentificeerd met behulp van de Pearson-correlatiecoëfficiënt, ρ. Kenmerken met een $\rho > 0{,}95$ worden beschouwd als sterk gecorreleerd. Bij elk sterk gecorreleerd paar wordt willekeurig één kenmerk gekozen en de andere wordt genegeerd in het trainingssubset.

Na de kenmerkselectie blijven er uiteindelijk 69 kenmerken over: 24 afkomstig van de accelerometer, 19 van de gyroscoop en 26 van de magnetometer. Bij het bekijken van de uiteindelijke DT die wordt gemaakt, vindt er nog een verdere reductie van kenmerken plaats. Deze DT vermindert het aantal kenmerken tot 13 en behaalt een nauwkeurigheid van 86% op de *5-fold-cross validation* en 91% op de uiteindelijke testdataset. In het artikel wordt de DT weergegeven, maar de uiteindelijke kenmerken die door de boom zijn geselecteerd, worden niet vermeld.

2.1.2. Support Vector Machine voor HAR

L. Cheng et al. vergelijken drie verschillende soorten modellen, namelijk *Support Vector Machine* (SVM), *Hidden Markov Method* (HMM) en Neural Networks (NN) [9]. In deze paragraaf ligt de focus voornamelijk op de resultaten van SVM. Er wordt gewerkt met een eigen gecreëerde dataset waarvan de details terug te vinden zijn op **Tabel 6**.

Er is gekozen voor een *Radius Basis Function* (RBF) kernel, omdat deze minder parameters vereist dan een polynomiaal kernel. Voor de C- en γ-parameters wordt gebruik gemaakt van een *gridsearch* om de optimale parameterwaarden te vinden. De C-parameter controleert de afweging tussen de toegestane misclassificaties van trainingsdata en de eenvoud van de *hyperplane*. Een lagere C-waarde resulteert in een zachtere beslissingsoppervlakte, terwijl een hogere C-waarde minder misclassificaties toestaat. Voor de C-waarden in de *gridsearch* zijn waarden gekozen tussen $[2^{-3}, 2^{-2}, \dots, 2^{3}]$. Voor de γ-waarden zijn in de *gridsearch* waarden gebruikt tussen $[2^{-6}, 2^{-2}, \dots, 2^{3}]$. De uiteindelijk gevonden parameterwaarden zijn te vinden in [9], sectie IV.

Tabel 1: *Resultaten SVM [9]*

Experiment	Accuraatheid
Persoon 1 vs. Persoon 1	92.5%
Persoon 2 vs. Persoon 2	99.5%
Persoon 3 vs. Persoon 3	93.5%
1 vs. iedereen	41.2%

De resultaten van de SVM in dit onderzoek worden weergegeven in **Tabel 1**. Er wordt gekeken naar *One-against-Own* en *One-against-All*. Voor de *One-against-Own* wordt 80% van de data van de desbetreffende persoon gebruikt om de SVM te trainen, terwijl 20% wordt gebruikt voor het testen. Voor de *One-against-All* wordt een lage nauwkeurigheid behaald. In het artikel wordt gesuggereerd dat dit mogelijk te wijten is aan de manier waarop personen zich voortbewegen, bijvoorbeeld door leeftijdsverschillen, wat niet als kenmerk in de dataset is opgenomen. Een andere mogelijkheid is de beperkte omvang van de dataset, die slechts vier proefpersonen bevat.

Er wordt aangegeven dat toekomstig onderzoek zich kan richten op het vinden van betere methoden voor het bepalen van optimale parameters. *Gridsearch* wordt beschreven als een intensieve procedure. Er wordt voorgesteld om eerst met een *cross-validation grid* te werken om het bereik van de optimale parameters te vinden, waarna een kleinere grid kan worden toegepast om het optimale resultaat te bereiken. Daarnaast wordt opgemerkt dat andere kernels mogelijk beter zouden kunnen presteren dan RBF.

Y. Cheng et al. introduceren een alternatieve vorm van een *Support Vector Machine* model voor HAR, namelijk de *Fuzzy Least Squares Support Vector Machine* (FLS-SVM) als classificatiemethode [10]. Er wordt gebruik gemaakt van een openbare dataset genaamd DLAs voor HAR [11]. De eigenschappen en details van deze dataset zijn terug te vinden in **Tabel 6**. De gegevens zijn gesegmenteerd in vensters van 2,5 seconden met een overlap van 50%.

Het gebruikte classificatiemodel is een alternatieve versie van SVM, geformuleerd voor tweeledige problemen. Om meerklasseproblemen op te lossen, wordt FLS-SVM voorgesteld en toegepast. In **Tabel 2** wordt een vergelijking gemaakt met vier andere modellen op dezelfde dataset, met dezelfde geëxtraheerde kenmerken en dezelfde kenmerkselectiemethode. De vier andere modellen zijn kNN, Bayesiaanse model, SVM en *Least Squared* SVM (LS-SVM).

***Tabel 2**: Vergelijking prestaties van de verschillende modellen in onderzoek [11]*

	kNN	Bayesiaanse model	SVM	LS-SVM	FLS-SVM
Gemiddelde accuraatheid	85.36	81.15	90.03	92.83	93.43

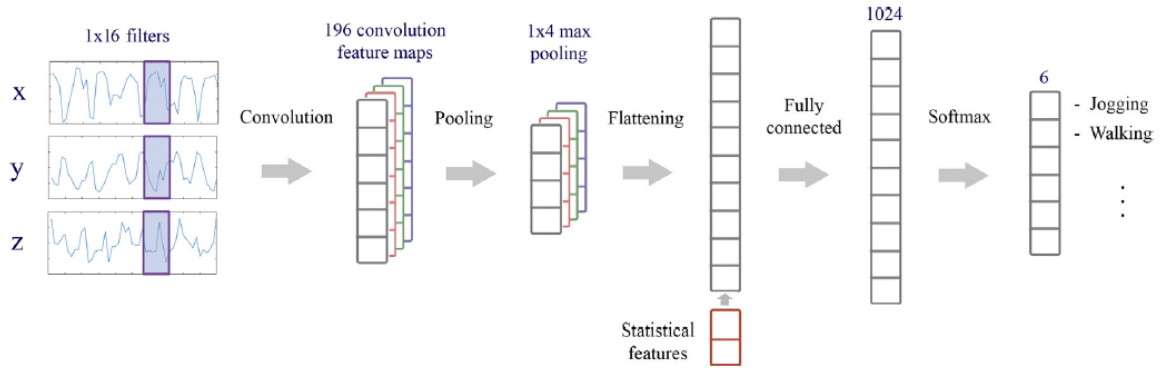
De validatiescores zijn berekend met behulp van *leave-one-subject-out cross validation*. De nauwkeurigheden per activiteit worden ook vermeld. Er wordt geconcludeerd dat FLS-SVM beter presteert voor meerklasseproblemen dan SVM en LS-SVM. Het belangrijkste verschil tussen SVM en LS-SVM ligt in het optimalisatieprobleem om de beslissingsgrens te vinden. Bij SVM is het optimalisatieprobleem kwadratisch, wat rekenkundig zwaar is. Bij LS-SVM is dit optimalisatieprobleem lineair, wat efficiënter kan worden opgelost en een voordeel kan zijn bij

toepassingen met grote datasets. Bovendien heeft LS-SVM een unieke oplossing, terwijl SVM mogelijk meerdere oplossingen heeft [12].

FSL-SVM breidt het standaard SVM-algoritme uit door *fuzzy sets* op te nemen in het classificatieproces. In plaats van dat er bijvoorbeeld gewerkt wordt met 0 en 1 om de klasse aan te duiden wordt er bij *fuzzy sets* een waarde tussen 0 en 1 toegekend. Hierdoor wordt rekening gehouden met de onzekerheid en vaagheid van gegevens. Dit zorgt voor meer flexibiliteit in de beslissingsgrens en kan de prestaties van het algoritme op bepaalde soorten datasets verbeteren [13].

2.1.3. Convolutional Neural Networks voor HAR

A. Ignatov maakt gebruik van een *Convolutional Neural Network* (CNN) voor het lokaal extraheren van kenmerken in combinatie met eenvoudige statistische kenmerken voor HAR [14]. Deze kenmerken geven informatie over de globale vorm van de tijdreeksgegevens. Een nadeel van CNN, dat in het werk wordt beschreven, is dat een CNN op zichzelf niet in staat is om globale eigenschappen van het tijdreeksignaal te vangen. Dit nadeel wordt opgevangen door de CNN te combineren met het toevoegen van deze kenmerken, bijvoorbeeld de gemiddelde waarde. Daarnaast wordt er ook onderzoek gedaan naar de invloed van tijdvensters op de nauwkeurigheid van het model. De gebruikte gegevens zijn afkomstig van de WISDM dataset [6] en de UCI dataset [7].



Figuur 3: *Voorgestelde CNN-architectuur van A. Ignatov [14].*

De voorgestelde architectuur wordt weergegeven op **Figuur 3**. Het verwerken van de accelerometerwaarden begint in de convolutielaag, waar 196 convolutiefilters worden gebruikt om zo de eigenschappen van de gegevens goed vast te leggen. De grootte van elke filter is 1x16. Daarna wordt *max pooling* van grootte 1x4 gebruikt om de kenmerk-voorstelling met een factor vier te verminderen. Na het *flattening*-proces worden vervolgens de globale statistische kenmerken toegevoegd (gemiddelde, variantie, som van de absolute waarden en histogram van elk gegevenskanaal). Dit wordt vervolgens verbonden met een volledig verbonden laag van 1024 neuronen. Er wordt gebruik gemaakt van stochastische gradiëntafdaling en *backpropagation*. Het model is publiek beschikbaar gemaakt [15].

Er wordt ook onderzoek gedaan naar de invloed van de grootte van het tijdvenster. Dit wordt getest op een aantal verschillende modellen. Er worden twee conclusies getrokken met betrekking tot de grootte van het tijdvenster:

1. Grotere segmenten leiden niet altijd tot betere prestaties.
2. De stijging in nauwkeurigheid is het meest significant bij een verandering van 20 *samples* naar 40-60 *samples*.

Daarnaast is er ook onderzoek gedaan naar verschillende voorverwerkingstechnieken. De resultaten zijn te vinden in **Tabel 3**.

Tabel 3: *classificatieresultaten met verschillende preprocessing technieken [14]*

Methode	Accuraatheid
CNN + Stat. kenmerken + *data centering*	97.63%
CNN + Stat. kenmerken	96.06%
CNN + Stat. kenmerken + datanormalisatie	95.48%
CNN	95.31%
CNN + d*ata centering*	92.35%
CNN + datanormalisatie	90.77%

Het CNN-model zonder voorverwerking behaalt een nauwkeurigheid van 95,31%. Wanneer de kenmerken worden verschaald (*data centering* of datanormalisatie) daalt de prestatie aanzienlijk. Dit wordt verklaard doordat de globale vorm van het ingangssignaal verloren gaat door de herschaling. Het verstoort de vorm van de tijdreeks en de magnitude-informatie, wat cruciaal is voor HAR. Dit wordt opgevangen door het toevoegen van globale statistische kenmerken. Door het centreren van de tijdreeks wordt de taak voor de CNN eenvoudiger.

Er wordt geconcludeerd dat het model efficiënt kan worden uitgevoerd op mobiele telefoons in *real-time*. Dit komt doordat het een lichte architectuur en een lage *runtime* heeft.

***Tabel 4**: Overzicht van resultaten van verschillende werken voor de WISDM [6] dataset. In deze tabel worden naast de besproken werken ook andere werken op de WISDM vermeld.*

Paper	Classificatiemethode	Testtechniek	Accuraatheid
[2]	Logistic Model Tree	10-subject-cross-validation	93.95%
[16]	Random Forests + handgemaakte kenmerken.	Leave-one-out	83.46%
[16]	Dropout + handgemaakte kenmerken	Leave-one-out	85.36%
[17]	LSTM	10-fold cross-validation	90.03%

***Tabel 5:** Overzicht van resultaten van verschillende werken voor de UCI [7] dataset. In deze tabel worden naast de besproken werken ook andere werken op de UCI vermeld.*

Paper	Classificatiemethode	Testtechniek	Accuraatheid
[2]	Logistic Model Tree	10-subject-cross-validation	94.02%
[14]	Convolutional Neural Network + Stat. kenmerken	21 trainings- en 9 testpersonen	97.63%
[18]	PCA + SVM	21 trainings- en 9 testpersonen	91.82%
[19]	Recurrent Neural Network (RNN)	75% training en 25% test	95.03%
[20]	CNN	21 trainings- en 9 testpersonen	95.18%
[21]	SVM	21 trainings- en 9 testpersonen	89%
[22]	CNN + FFT kenmerken	21 trainings- en 9 testpersonen	95.75%

***Tabel 6**: Overzicht van de besproken datasets voor HAR in deel 1 van de literatuurstudie.*

Dataset	Sensor(en)	Plaats sensor(en)	Aantal personen	Activiteiten	Referentie
WISDM	Accelerometer	pols	36	wandelen joggen trap omhoog zitten trap omhoog liggen	[5]
UCI	Accelerometer Gyroscoop	taille	30	wandelen joggen traplopen zitten staan liggen	[6]
G. De Leonardis	Accelerometer Gyroscoop Magnetometer	niet vermeld	15	zitten staan neerliggen wandelen trap oplopen trap aflopen omhoog wandelen omlaag wandelen	[7]
L. Cheng	Vier accelerometers	linker pols rechter pols linker enkel rechter enkel	4	zitten neerzitten staan rechtstaan wandelen	[9]
DLAs	Vier accelerometers	rechter pols heup borstkas linker enkel	19	zitten liggen staan afwassen stofzuigen dweilen wandelen trap oplopen trap afgaan hardlopen fietsen 50W fietsen 100W	[11]

2.1.4. Besluit

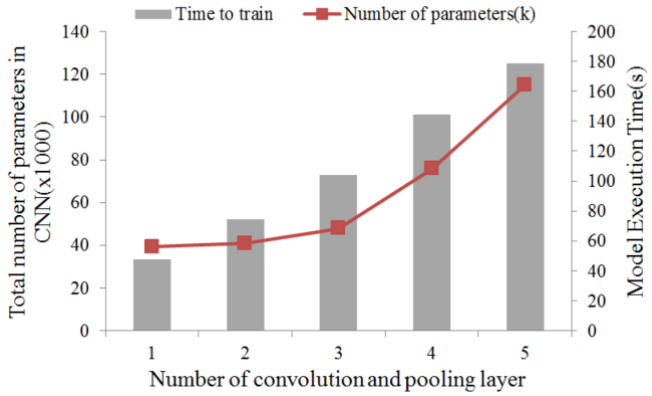
Het eerste deel van de literatuurstudie besprak de drie meest voorkomende modeltypes die gebruikt worden in de literatuur voor HAR. Op **Tabel 4** en **Tabel 5** worden de resultaten van verschillende modeltypes besproken die uitgevoerd zijn op de WISDM-dataset en UCI-dataset. Dit geeft een overzicht van verschillende modellen die worden getraind op dezelfde dataset waardoor een conclusie kan gevormd worden over de efficiëntie per type model voor HAR. Deze tabellen bevatten resultaten uit onderzoeken die werden besproken alsook van onderzoeken die niet werden besproken. In deze tabellen valt het op dat voornamelijk neurale netwerken en boom-gebaseerde modellen de beste scores behaalden. Dit geldt ook voor andere onderzoeken die werden besproken in het eerste deel die gebruik maakten van andere datasets. Dit in combinatie met het feit dat de huidig model van mBrain voor HAR in de *cloud* een CNN is, zal verder enkel nog aandacht worden gegeven aan dit type model.

2.2. Optimalisatietechnieken voor de edge

In het voorgaande deel van de literatuurstudie zijn diverse onderzoeken naar *Human Activity Recognition* (HAR) onderzocht en samengevat. Hierbij lag de focus op verschillende soorten modellen die in de literatuur voor HAR worden gebruikt. Dit deel van de literatuurstudie richt zich op technieken die worden toegepast om *machine learning* (ML)-modellen aan te passen, zodat ze geschikt zijn voor een *edge*-omgeving met beperkte rekenkracht en opslagbronnen. De aandacht gaat hierbij specifiek uit naar neurale netwerken omwille van de eerder benoemde redenen.

2.2.1. Convolutional Neural Networks

T. Zebin et al. implementeren een *Convolutional Neural Networks* (CNN) op een Samsung A5 2017 smartphone met beperkte middelen (1.2 GHz quad-core CPU met 3GB werkgeheugen) voor HAR [23]. De keuze voor een CNN-architectuur wordt verklaard doordat deze in staat is om ruwe tijdreeksen data te verwerken, kan omgaan met fouten via *backpropagation* en over het algemeen een hogere prestatiescore behaalt. Eerst wordt een CNN-model gecreëerd waarbij geen rekening wordt gehouden met beperkingen van de *edge*-omgeving. Vervolgens wordt dit model aangepast naar een *edge*-vriendelijkere versie. De data bestaat uit tri-axiale accelerometer- en gyroscoopgegevens. Er is gekozen voor een model met vier lagen, waarbij het aantal filters in elke laag wordt verdubbeld. Er is onderzoek gedaan naar de invloed van het aantal convolutielagen en *pooling*-lagen. Een toename van het aantal lagen resulteerde in betere prestaties van het model, maar het verhoogde ook de computationele kosten van het model. Daarnaast is er gekeken naar de invloed van het aantal lagen op de uitvoeringstijd van het model, waarbij deze tests zijn uitgevoerd op een computer met een Intel Core i7 CPU (2.4 GHz). De resultaten zijn weergegeven op **Figuur 4**.



***Figuur 4**: De invloed van het aantal convolutielagen en pooling-lagen op de executietijd uit onderzoek van T. Zebin et al [23].*

Er kan geconcludeerd worden dat een toename van het aantal lagen in het CNN-model een nagenoeg lineair verband heeft met de uitvoeringstijd van het model. Hoewel een toename van het aantal lagen een verbetering van de nauwkeurigheid met zich meebrengt, moet er rekening gehouden worden met het feit dat dit nadelige gevolgen heeft voor de uitvoeringstijd. Dit is een belangrijk aspect voor *real-time* toepassingen. Daarnaast is er onderzoek gedaan naar de grootte van de kernels, waaruit blijkt dat 1x12-kernels het best presteren en een verdere toename of afname in grootte leidt tot slechtere resultaten. In dit onderzoek zijn tijdvensters van 2,56 seconden genomen en werkten de sensoren op 50Hz, waardoor het tijdsvenster gelijkstaat aan 128 datapunten. De volgende stap is het aanpassen en implementeren van het model op een *edge*-device.

Om de inferentie stap mogelijk te maken op een smartphone, wordt gebruik gemaakt van de TensorFlow Lite (TFLite) bibliotheek. Er is onderzoek gedaan naar het aantal *floating-point* operaties dat wordt uitgevoerd door de *frozen graph* van het model. Met *frozen graph* wordt bedoeld: een opgeslagen versie van een getraind neuraal netwerkmodel, waarbij de parameters (gewichten en biases) van het model vastliggen en niet meer kunnen worden bijgewerkt. Er is ook gekeken naar de hoeveelheid geheugen en de uitvoeringstijd die nodig is voor het voorspellen van de activiteit per functie. De resultaten staan samengevat op **Figuur 5**. Uiteindelijk gebruikt het model 16 MB geheugen.

Node type	Count	Memory (KB)	Avg. execution time (ms)
Conv1D weights	72	5869.953	244.813
BiasAdd	73	5869.953	244.813
ReLU	72	5869.953	5.700
MaxPool	18	358.848	5.700
Concat	1	892.096	1.081
MatMul	1	4.032	0.660
Softmax	1	4.032	0.660
Reshape	1	0	1.081

Figuur 5: *Screenshot van de tabel met resultaten uit onderzoek van T. Zebin et al[23]. Onderzoek naar het aantal floating point operaties, vereiste geheugen en executie tijd per functie van het CNN-model.*

De volgende stap in dit onderzoek is het aanpassen van het model om het meer geschikt te maken voor *edge computing*. Dit wordt volledig en uitsluitend gedaan via kwantisatie. Het onderzoek richt zich dus op de effecten van modelkwantisatie. Kwantisatie in deze context houdt in dat het aantal bits dat wordt gebruikt om de gewichten en activeringen van het model weer te geven wordt verminderd.

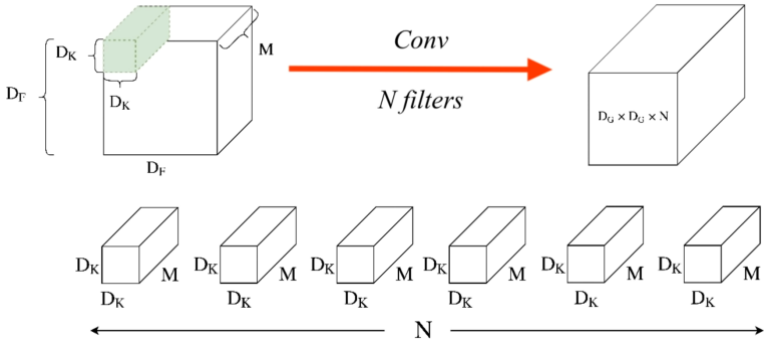
Network Type	32-bit floating point (float32) model				8-bit fixed point (unit8) quantized model		
	Memory footprint (MB)	Training (%)	Validation (%)	Test (%)	Test (%)	Memory footprint (MB)	Size ratio
Dense Neural Network (DNN)	8.2	97.99	89.04	86.55	87.6	1.37	5.98
Long-Short Term Memory Network (LSTM) [11]	17.4	98.38	96.69	92.2	93.51	4.1	4.24
Convolutional Neural Network (CNN)	16	99.23	95.92	96.4	93.6	2.1	7.62

Figuur 6: *Screenshot tabel uit onderzoek T. Zebin et al. [23] Verschil in accuraatheid en opslag door uint8-kwantisatie.*

Figuur 6 toont het verschil in accuraatheid en opslag als gevolg van kwantisatie in het onderzoek. Er vindt een overgang plaats van 32-bits *floating-point* kommagetallen naar 8-bits *fixed-point* getallen. Wanneer deze kwantisatie wordt toegepast op zowel de gewichtsfuncties als de activatiefuncties, leidt dit tot een reductie van de modelgrootte met een factor van 7,62. De nauwkeurigheid neemt hierbij af van 96,4% naar 93,6% op de testdata. Dit betreft post-training kwantisatie, waarbij het model eerst wordt getraind en opgeslagen in een *floating-point* formaat. Vervolgens wordt het omgezet naar een *fixed-point* formaat. Dit heeft als voordeel dat de nauwkeurigheid niet wordt aangepast tijdens het trainen van het model.

Depthwise Separable Convolutions

Xception [24] introduceerde als eerste het concept van *depthwise separable convolutions (DSC's)*, wat later ook werd geïntegreerd in MobileNet [25]. *Depthwise separable convolutions* zijn een type CNN-operatie die het aantal berekeningen aanzienlijk vermindert in vergelijking met standaard convoluties. DSC's bestaan uit twee afzonderlijke convoluties: *depthwise convolutions* en *pointwise convolutions*.

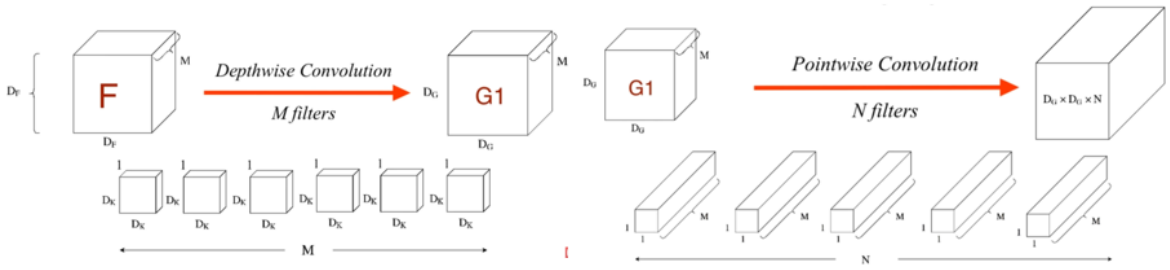


Figuur 7: *Voorstelling van een standaardconvolutie. Waar M staat voor het aantal ingangssignalen, D_K de grootte is van de filters en D_G het aantal samples. [33]*

Figuur 7 illustreert een standaardconvolutie waarbij M het aantal ingangssignalen, D_K de filtergrootte, D_G het aantal samples vertegenwoordigt en N het aantal toegepaste kernels is. Het aantal uit te voeren vermenigvuldigingen is gelijk aan $D_K^2 * D_G^2 * M * N$. DSC's worden gebruikt om het aantal bewerkingen te verminderen.

De eerste stap van DSC's zijn de *depthwise convolutions*, ook bekend als de filterfase. Deze fase past convoluties toe op één ingangssignaal per keer, in tegenstelling tot de standaardconvolutie die convoluties op alle ingangssignalen tegelijk toepast. Dit is te zien in **Figuur 8**, waar de filter de dimensie D_K X D_K X M heeft.

De volgende stap is de *pointwise convolution*, ook wel de combinatiefase genoemd. De invoer voor deze fase is de uitvoer van de vorige fase. De filtering gebeurt nu over alle ingangssignalen. De dimensie van de filters in deze stap is 1 x 1 x M en vertegenwoordigt een één-bij-één convolutie over alle lagen. Beide stappen zijn te zien in **Figuur 8.**



***Figuur 8**: Overzicht van fasen in DSC's. a) depthwise convolutions. b) pointwise convolutions. [33]*

In de eerste stap van depthwise separable convolutions is het aantal vermenigvuldigingen voor één convolutie gelijk aan D_K^2, wat resulteert in $D_K^2 * D_G^2$ vermenigvuldigingen per kanaal. Het totale aantal *depthwise* vermenigvuldigingen is dan $D_K^2 * D_G^2 * M$. In de tweede stap zijn er M vermenigvuldigingen voor één convolutie en $D_K * D_K *$ M vermenigvuldigingen per kernel. Het totale aantal *pointwise* vermenigvuldigingen is dus $N * D_K * D_K *$ M. Het totale aantal vermenigvuldigingen voor DSC's is de som van de *depthwise* en *pointwise* convoluties, wat neerkomt op $M * D_G^2 * (D_K^2 + N)$. De verhouding van deze methode ten opzichte van de standaardmethode is: $\frac{1}{N} + \frac{1}{D_K^2}$. Als er bijvoorbeeld 1024 filters worden toegepast en de filtergrootte is 3x3, heeft deze methode slechts 0.112 keer het aantal vermenigvuldigingen nodig in vergelijking met de standaard convolutiemethode, wat overeenkomt met 8.93 keer minder vermenigvuldigingen.

Naast het gebruik van *depthwise separable convolutions*, introduceert MobileNet twee globale parameters die de afweging maken tussen uitvoeringstijd en nauwkeurigheid. Gebruikers kunnen deze hyperparameters kiezen om de beste afweging voor hun *use case* te bepalen. Na elke convolutie, zowel *depthwise* als *pointwise*, worden batchnormalisatie en *Rectified Linear Unit* (ReLU) toegepast. ReLU is een veelgebruikte activatiefunctie in *machine learning*, die rekenkundig efficiënt is en zorgt voor snellere training van diepe neurale netwerken. Vervolgens wordt er gedownsampled met behulp van *strided* convolutie en wordt er een finale *average pooling* geïmplementeerd om de ruimtelijke resolutie te reduceren naar één, voor de volledig verbonden laag.

Het is belangrijk op te merken dat MobileNet oorspronkelijk is ontworpen voor toepassingen met afbeeldingen die drie kleurkanalen hebben (rood, groen en blauw). Afbeeldingen bestaan uit pixels met kleurinformatie, terwijl accelerometerwaarden bestaan uit tijdreeksen van bewegingsinformatie in de vorm van versnellingen langs de drie assen (x, y, en z). Hoewel er enige gelijkenis is tussen de twee in termen van het aantal dimensies, zijn de onderliggende structuren en de manier waarop deze gegevens moeten worden verwerkt om bruikbare informatie te verkrijgen, verschillend.

Het is mogelijk om een MobileNet-achtige architectuur aan te passen voor HAR met accelerometerwaarden. Echter, dit vereist enkele aanpassingen in de netwerkarchitectuur, zoals het gebruik van 1D-convoluties in plaats van 2D-convoluties. Dit komt doordat afbeeldingen tweedimensionale gegevens bevatten in de vorm van hoogte en breedte, terwijl tijdreeksen ééndimensionale gegevens zijn die variëren in de tijd.

***Tabel 7**: Resources die gebruikt worden in MobileNet per laagtype. [25]*

Type	Multiplicaties-Optellingen	Parameters
Pointwise convoluties (1X1)	94.86%	74.59%
Depthwise convoluties (3X3)	3.06%	1.06%
Convoluties 3X3	1.19%	0.02%
Fully Connected	0.18%	24.33%

De eerste geïntroduceerde hyperparameter is α, ook wel de *width multiplier* genoemd. Deze parameter is bedoeld voor toepassingen waar modellen nog kleiner en sneller moeten zijn. De rol van deze parameter is om het netwerk gelijkmatig uit te dunnen op elke laag. Voor een gegeven laag en *width multiplier* wordt het aantal ingangskanalen M, gereduceerd tot αM en het aantal uitgangskanalen N, wordt gereduceerd tot αN, waarbij $\alpha \in [0,1]$.

De tweede hyperparameter is ρ, ook wel de *resolution multiplier* genoemd. Deze parameter is ontworpen om de computationele kosten verder te verlagen en vermindert de kosten met ρ^2. Wanneer deze vermenigvuldiger wordt toegepast op het ingangssignaal en de interne representatie van elke laag, wordt elke laag verkleind met dezelfde vermenigvuldiger ρ. De

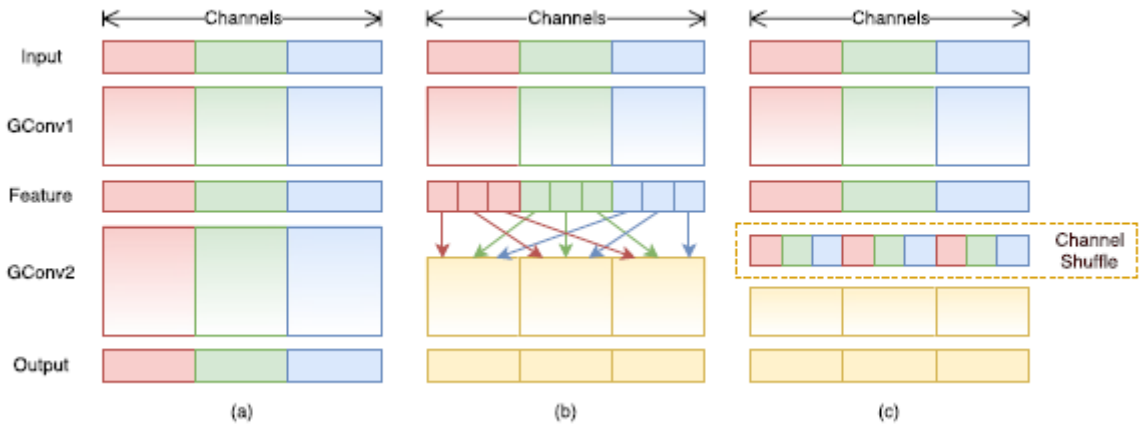
uiteindelijke computationele kosten voor CNNs met *depthwise separable convolutions*, een *width parameter* en *resolution parameter* zijn te vinden in **Formule 1**. De computationele kosten met gewone convoluties en zonder beide hyperparameters zijn te vinden in **Formule 2**.

(1) $$D_K * D_K * \alpha M * \rho D_G * \rho D_G + \alpha M * \alpha N * \rho D_G * \rho D_G$$
$$\Rightarrow \alpha * \rho^2 * M * D_G^2 * (\alpha * D_K^2 + N).$$

(2) $$M * N * D_G * D_G * D_K * D_K$$

Het aantal bronnen dat per laagtype wordt gebruikt, is weergegeven in **Tabel 7**. Uit de tabel kan worden geconcludeerd dat het grootste deel van de computationele kosten afkomstig is van de *pointwise convoluties*.

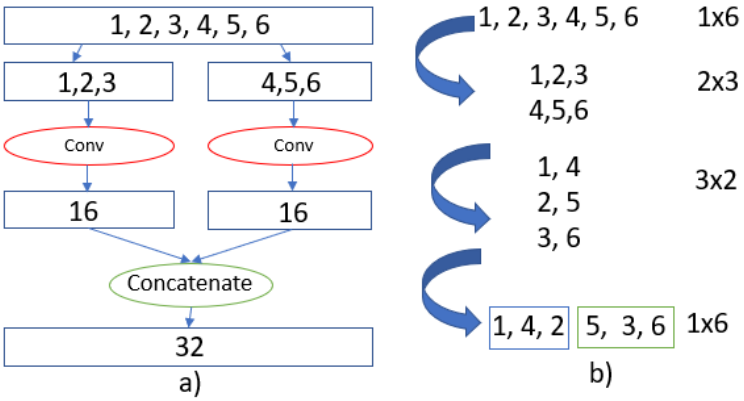
ShuffleNet wil de laatste stap van *depthwise separable convolutions*, *pointwise convoluties* nog verder minimaliseren, aangezien deze een hoge kost hebben voor kleinere netwerken [26]. Het introduceert een computationeel efficiënte CNN-architectuur die speciaal is ontworpen voor mobiele apparaten met beperkte rekenkracht. De architectuur maakt gebruik van twee nieuwe operaties: *pointwise group convolutions* en *channel shuffle*. Op deze manier wordt de rekenkost gereduceerd terwijl de nauwkeurigheid van het model behouden blijft.



Figuur 9: *Channel Shuffle met twee opeengestapelde group convoluties. GConv staat voor group convolution. a) Twee opeengestapelde convolutielagen met hetzelfde aantal groepen. Elk uitgangskanaal is alleen gerelateerd aan het ingangskanaal binnen dezelfde groep. b) Ingangs- en uitgangssignalen zijn volledig gerelateerd wanneer GConv2 data ontvangt van de verschillende groepen na GConv1. c) Equivalent aan implementatie b) met gebruik van channel shuffle. [26]*

ShuffleNet maakt gebruik van *group convolutions* voor zowel de 3x3-convoluties als de 1x1-convoluties die worden gebruikt over de verschillende ingangssignalen. In combinatie daarmee maakt het gebruik van *Channel Shuffle*. Een visualisatie hiervan wordt voorgesteld op **Figuur 9**. Het doel van de *shuffle*-operatie is om *cross-group* informatie uit te wisselen over verschillende groepen convolutielagen. X. Zhang et al. hebben ook onderzoek gedaan naar de invloed van het aantal groepen met de *shuffle*-operatie en *group convolutions* [26]. Er wordt geconcludeerd dat modellen met de *shuffle*-operatie beter presteren, met name voor modellen met een groter aantal groepen.

Op **Figuur 10** wordt het nut van ShuffleNet duidelijker gemaakt. Het voorbeeld heeft zes ingangssignalen of *feature maps*, waarbij gekozen is om het aantal groepen op twee vast te leggen. Op **Figuur 10.a** worden de zes ingangssignalen opgesplitst in twee groepen. Op deze twee groepen worden afzonderlijk convoluties uitgevoerd, wat resulteert in bijvoorbeeld 16 *feature maps*. Dit aantal kan meer of minder zijn. Vervolgens worden deze samengevoegd. In dit voorbeeld gaat men van zes *feature maps* naar 32 *feature maps*. In plaats van een 1x1x6x32-tensor wordt het gedaan via 2x1x1x3x16-tensor. Hierbij staat 2 voor het aantal groepen, wordt er per groep een 1x1-kernel toegepast, 3 staat hier voor het aantal *feature maps* per groep en 16 is het aantal uitgangen/*feature maps* na convolutie van één zo een groep. Het is een simplistische voorstelling van de *group convolutions* die worden geïmplementeerd door ShuffleNet.



Figuur 10: *Voorbeeld ShuffleNet met zes ingangen en twee groepen.*

Channel Shuffle is een operatie waarbij de kanalen worden geschud. Om dit te doen, worden een aantal stappen doorlopen. Op **Figuur 10.b** wordt een voorbeeld van deze operatie uitgewerkt. Zoals te zien, worden de kanalen geschud tussen de ingangstensor en de uitgangstensor. Dit is het basisidee achter de *Channel Shuffle*.

Technieken zoals die toegepast zijn in ShuffleNet, Xception en MobileNet worden tot nu toe voornamelijk gebruikt in computervisie toepassingen. CNN staat namelijk ook bekend om zijn sterke prestaties in classificatie van toepassingen waarbij afbeeldingen betrokken zijn.

B. Aricioğlu et al. zijn de eerste in de literatuur die gebruik van grafische afbeeldingen van tijdreeksen voor classificatie met *deep learning*-methoden [27]. Op deze manier worden deze technieken dus ook gebruikt voor tijdreeksen toepassingen. Verschillende CNN netwerkmodellen worden met elkaar vergeleken. Het gaat hier om netwerken die belangrijk zijn in termen van snellere verwerking, verbeterde prestaties en snellere leerprocessen. In **Tabel 8** worden de resultaten van het onderzoek weergegeven. Deze tabel zegt niet dat het ene netwerk beter is dan het andere netwerk. Het zijn verschillende soorten CNN-netwerken die elk op hun manier een standaard CNN lichter, sneller, kleiner of energiezuiniger maken. ShuffleNet haalt een score van gemiddeld 92,26%.

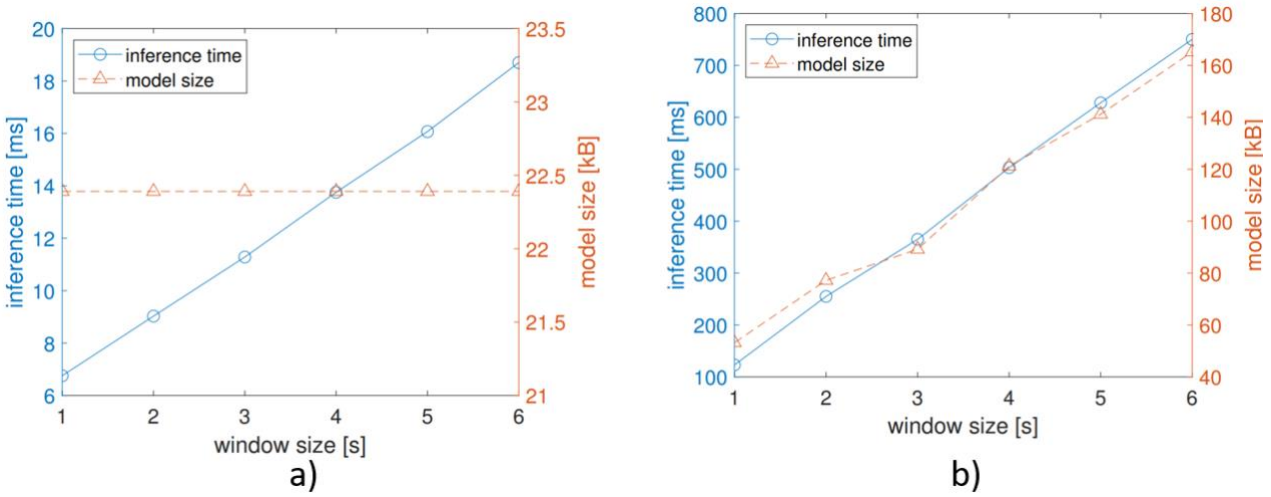
Tabel 8: *Resultaten van verschillende CNN netwerken op afbeeldingen op tijdreeks data [38].*

	Accuraatheid			
Netwerk	x-variabele	y-variabele	z-variabele	gemiddelde
ShuffleNet [26]	88.64%	89.90%	98.23%	92.26%
SqueezeNet [28]	96.97%	96.97%	97.22%	97.05%
AlexNet	91.92%	91.67%	98.48%	94.02%
ResNet50	95.20%	96.72%	99.75%	97.22%
ResNet101	93.94%	95.45%	99.75%	96.38%
DenseNet201	96.97%	96.97%	99.49%	97.81%
GoogleNet	94.44%	92.42%	98.23%	95.03%

E. Lattanzi et al. Hebben onderzoek gedaan naar de efficiëntie van Artificiële Neurale Netwerken (ANN) voor *Human Activity Recognition* (HAR) waarbij het besluitvormingsmodel wordt geïmplementeerd op de *wearable* zelf [29]. Er wordt onderzoek gedaan naar de efficiëntie van twee soorten ANN, namelijk het *Multilayer Perceptron* (MLP) en een CNN. Het doel is om de efficiëntie op het gebied van CPU-cycli, gebruikte geheugen en energieverbruik van de modellen te analyseren. Het is een van de weinige werken in de literatuur die effectief onderzoek doet naar het energieverbruik van modellen.

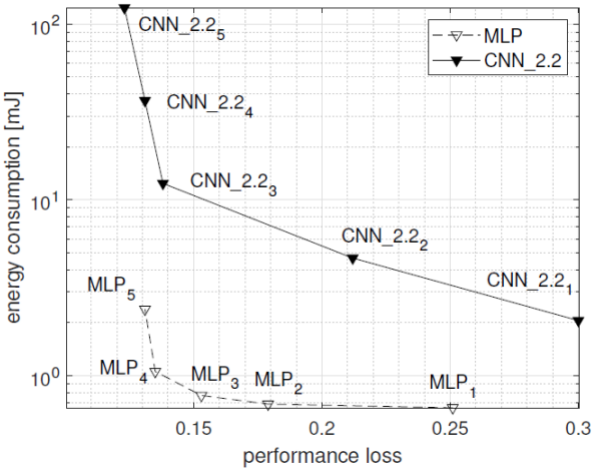
De keuze voor een CNN is gebaseerd op eerdere literatuur waarbij CNN zijn nut al heeft bewezen voor HAR-toepassingen. Daarnaast is er bij CNN geen noodzaak voor technieken voor kenmerkextractie en kenmerkselectie. Dit heeft soms wel als nadeel dat er een verhoogde computationele complexiteit en geheugengebruik mee gepaard gaat. Daarom wordt er ook gekeken naar MLP. Dit staat bekend als een solide *machine learning*-model dat potentieel minder belastend is voor bronbeperkte apparaten. Voor de MLP werden verschillende kenmerken geëxtraheerd. Na kenmerkselectie bleven er slechts drie kenmerken over per signaal, namelijk de gemiddelde waarde, de standaarddeviatie en de maximale waarde. Er worden twee CNN-modellen gemaakt: de eerste met slechts één convolutielaag en de tweede met twee convolutielagen. Voor beide soorten modellen wordt er gebruik gemaakt van *dropout* om *overfitting* te voorkomen. De *dropout-rate* wordt gezet op 0,5.

De CNN met één convolutielaag haalde een gemiddelde nauwkeurigheid van 84,2% en de CNN met twee convolutielagen een gemiddelde van 87,7%. MLP haalde met de uiteindelijke drie kenmerken een gemiddelde nauwkeurigheid van 86,5%. Op basis van nauwkeurigheid haalden de modellen dus een vergelijkbare score. Daarnaast werd er ook onderzoek gedaan naar de invloed van de venstergrootte op de uitvoeringstijd en de modelgrootte. De resultaten zijn te zien op **Figuur 11**, respectievelijk voor het MLP-model en CNN-model.



Figuur 11: *Invloed van venstergrootte op executietijd en modelgrootte [29]. A) Op het MLP-model. B) op het CNN_2.2-model.*

Voor zowel MLP als CNN is er sprake van een lineair verband tussen de venstergrootte en de executietijd. Voor de MLP verandert de modelgrootte niet wanneer de venstergrootte wordt aangepast. De invloed van de venstergrootte op de nauwkeurigheid van het model is bijna constant, maar behaalt de slechtste score bij één seconde en de beste score bij drie seconden. Uiteindelijk wordt er gekozen voor een grootte van drie seconden. Bij CNN heeft de venstergrootte wel een invloed op de modelgrootte, namelijk een lineair verband. Ook hier wordt tenslotte gekozen voor een grootte van 3 seconden. Het CNN-model heeft een executietijd per inferentie van ongeveer 350 ms en een modelomvang van 90 kB. Het MLP-model heeft een executietijd per voorspelling van ongeveer 11 ms en een modelomvang van 22.5 kB.



Figuur 12*: energieverbruik in mJ in functie van prestatieverlies voor beide modellen. De sub index geeft een beeld van het aantal hidden neurons die gebruikt werden in het model.*

Het energieverbruik per model en per venstergrootte wordt weergeven op **Figuur 12**. Het indexgetal geeft een indicatie voor het aantal verborgen neuronen dat gebruikt wordt voor het model. Hoe groter het getal, hoe meer verborgen neuronen er gebruikt worden. Het is duidelijk te zien dat om eenzelfde prestatie te verkrijgen, CNN aanzienlijk meer energie moet verbruiken. Het werk wil in de toekomst verder onderzoek doen naar het toepassen van de *early exit*-techniek om diepere CNNs te maken die dynamisch de computationele complexiteit zal aanpassen. Belangrijk hierbij is om te vermelden dat hier gebruik wordt gemaakt van één-dimensionale convoluties.

P. Dhiman et al. stellen een magnitude-gebaseerde *pruning*-methode voor in combinatie met post-training kwantisatie om de modellen aan te passen aan de beperkte bronnen van randapparaten [30]. De *use-case* in het onderzoek is het detecteren van ziektes op citrusvruchten. De doelstelling is om de verwerking van de foto's van deze citrusvruchten op het veld zelf te doen. In eerste instantie wordt een CNN- en een CNN-LSTM-architectuur voorgesteld. Vervolgens worden op deze modellen eerst magnitude-gebaseerde *pruning* toegepast en vervolgens post-training kwantisatie. *Pruning* is een techniek die de laagste gewichten van een neuraal netwerk op nul zet. Op deze manier worden sommige operaties niet meer uitgevoerd wat voordelig kan zijn voor de inferentietijd. Daarnaast zorgt het er ook voor dat door de grote aanwezigheid van nul-elementen de matrices voor de gewichten op een efficiëntere manier worden opgeslagen. Hierdoor kan de grootte van het model mogelijk dalen.

Evaluation Parameter	Without Pruning CNN Model	Without Pruning CNN-LSTM Model	With Pruning CNN-LSTM Model	Pruning + Post Quantization CNN-LSTM Model
Accuracy %	96.98	98.87	96.13	94.35
Loss	0.1713	0.0831	0.165	0.1867
Precision %	94.09	96.65	96.05	94.16
Recall %	93.4	97.74	94.13	93.44
F-Score %	94.88	97.72	92.71	93.71
Size MB	56.65	52.82	28.16	20.62

Figuur 13: *Screenshot van tabel met resultaten uit onderzoek P. Dhiman et al. [30]. De invloed van pruning + post training kwantisatie op de accuraatheid, verlies, precisie, recall, F-score en de grootte van het model.*

Voor het CNN-model werd alleen *pruning* toegepast, terwijl voor het CNN-LSTM-model beide technieken werden toegepast. De resultaten van dit onderzoek worden beschreven in **Figuur 2.13**. Het toont de invloed van de technieken op nauwkeurigheid, verlies, precisie, recall, F-score en modelgrootte weergegeven. Het CNN-model na *pruning* is met 50,29% gereduceerd, terwijl de nauwkeurigheid slechts daalt met 0,85%. Bij de CNN-LSTM-architectuur is de grootte van het model 6,67% kleiner dan het CNN-model en 1,89% nauwkeuriger. Op dit model werden beide technieken toegepast. Dit resulteert in een reductie van 60,96% en een verlies in nauwkeurigheid van 4,5%. Het verlies in nauwkeurigheid is te wijten aan het feit dat door *pruning* het aantal gedetecteerde kenmerken afneemt. Bovendien zorgt de post-training kwantisatie voor een verminderde nauwkeurigheid van de oorspronkelijk getrainde gewichten. De combinatie van deze factoren zorgt ervoor dat de modelgrootte afneemt, maar ook dat de nauwkeurigheid vermindert. P. Dhiman et al. concluderen dat de resultaten het gebruik aanmoedigt van deze technieken in omgevingen waar de bronnen beperkt zijn. Omdat door deze technieken het model lichter wordt terwijl het verlies aan nauwkeurigheid minimaal is.

2.2.3. Besluit

Het tweede deel van de literatuurstudie behandelde verschillende technieken om CNNs en boom-gebaseerde modellen naar de *edge* te brengen. Voor de CNNs werden *depthwise separable convolutions*, *pruning* en post training kwantisatie besproken als technieken om dit soort modellen *edge*-vriendelijker te maken. Deze technieken kunnen mogelijks de modelgrootte en de inferentietijd reduceren. Echter kan dit leiden tot een verlies van accuraatheid van het model. Het is een afweging die gemaakt moet worden.

2.3. Conclusie

Deze literatuurstudie behandelde twee andere onderwerpen die aan elkaar verbonden zijn. Het eerste deel, **Sectie 3.1**, focuste zich op het onderzoeken van de bestaande modellen in de literatuur die gebruikt worden voor *Human Activity Recognition* (HAR). Hierbij werd de focus gelegd op de drie meest voorkomende modeltypes voor HAR: *Support Vector Machines* (SVM), boom-gebaseerde modellen en neurale netwerken (NN). Uit het onderzoek bleek dat de neurale netwerken en de boom-gebaseerde modellen de sterkste accuraatheid behaalden. Dit in combinatie met het feit dat het huidig model van mBrain voor HAR in de *cloud* een CNN is, worden SVM en boom-gebaseerde architecturen verder niet meer besproken.

Sectie 3.2 onderzocht welke technieken er bestaan voor *Convolutional Neural Network* (CNN). Er zijn drie verschillende soorten technieken besproken die hun nut hebben bewezen in de literatuur. De eerste techniek is magnitude-gebaseerd *pruning*. Deze techniek houdt in dat gewichten met het laagste gewicht op nul worden gezet. Op deze manier kunnen de matrices efficiënter worden opgeslagen en zijn de operaties met de nul-gewichten niet meer nodig. Dit leidt mogelijks tot een kleinere modelgrootte en kortere inferentietijd. De tweede techniek die werd behandeld, is post-training kwantisatie. Deze techniek stelt de gewichten en activaties van het model voor met datatypes die minder bits in beslag nemen. Op deze manier wordt de modelgrootte kleiner en daalt de inferentietijd mogelijks. De derde en laatste techniek die werd besproken was *depthwise separable convolutions*. Hierbij wordt het convolutieproces opgesplitst in twee stappen: *depthwise convolution* gevolgd door een *pointwise convolution*. Deze techniek vermindert het aantal parameters en berekeningen. Dit resulteert in een kleiner model en snellere inferentietijd. Ondanks dat de accuraatheid van het model doorgaans afneemt door de reductie van de complexiteit, blijft de gehele prestatie over het algemeen goed. Deze drie technieken zijn aantrekkelijk voor toepassingen met beperkte rekenkracht en opslagruimte.

Deze scriptie gaat *depthwise separable convolutions*, *pruning* en kwantisatie toepassen op het huidige CNN-model dat in de *cloud* draait. Deze technieken hebben in de literatuur reeds bewezen dat ze een positieve impact kunnen hebben op de conversie naar de *edge*. Er zullen analyses gemaakt worden over de impact van deze technieken op het CNN-model van mBrain. Daarnaast zal ook nog een extra CNN-model getraind worden waarop dezelfde technieken worden toegepast. Op deze manier kan een analyse worden uitgevoerd over de modelafhankelijkheid van deze technieken. Dit extra model maakt gebruik van een publieke dataset die in het eerste deel van de literatuurstudie naar voor kwam, namelijk de openbare WISDM-dataset. Op deze manier zijn de modellen getraind op andere datasets.

Verder gaat deze scriptie onderzoek doen naar de invloed van een *edge*-architectuur t.o.v. een *cloud*-architectuur op de bronnen die aanwezig zijn op een smartphone. Hierbij gaat onderzoek worden gedaan naar het energieverbruik van de smartphone en de inferentietijd voor beide omgevingen. Dit zal geëvalueerd worden voor meerdere scenario's. Er zal gekeken worden naar het verschil wanneer er constant *real-time* inferentie wordt gedaan en wanneer er gewerkt wordt met een bepaald tijdsinterval voor het uitvoeren van inferenties.

***Tabel 9**: Overzicht van grote bijdragen voor machine learning op de edge.*

Doel	Paper	Korte beschrijving	Besproken
Lichtere en snellere architecturen	[26]	ShuffleNet: Een efficiënt CNN-model voor mobiele en ingebedde systemen	✓
	[25]	MobileNets: Kleine, laag-computationele en snelle CNN-modellen.	✓
	[33]	Efficiënte schaling van CNN architectuur terwijl de accuraatheid behouden blijft	✘
	[35]	Overparameteriseren tijdens training om kwaliteit van model te verbeteren	✘
	[36]	Onstabiele training met behouden van accuraatheid en kleine modelomvang	✘
Kwantisatie en modelcompressie	[31,32]	Bonsai: een tree-based algoritme voor ML-predicties resource-constrained toestellen	✘
	[36]	ProtoNN: kNN-model voor resource-constrained toestellen.	✘
	[37]	Kwantisatie voor CNN deployment op Edge-toestellen.	✘
	[39]	Verminderde geheugenverbruik gebruikmakend van 16-bit fixed-point getallen voor gewichten	✘
	[28]	SqueezeNet: een CNN architectuur met 50 keer minder parameters dan AlexNet	✓
	[39]	Gebruikte 8-bit om CNN gewichten voor te stellen wat resulteerde in een modelcompressie van factor vier. Net zoals [31] dat werd besproken in deze literatuurstudie.	✘
Gedistribueerde predicties	[39]	Ontwikkelde een gedistribueerd platform voor mobiele *deep learning*	✘
	[40]	Diepe neurale netwerken over gedistribueerde mobiele verwerkingseenheden voor snellere predictietijden	✘
	[41]	Snellere predicties door het partitioneren van modellen over meerdere toestellen	✘

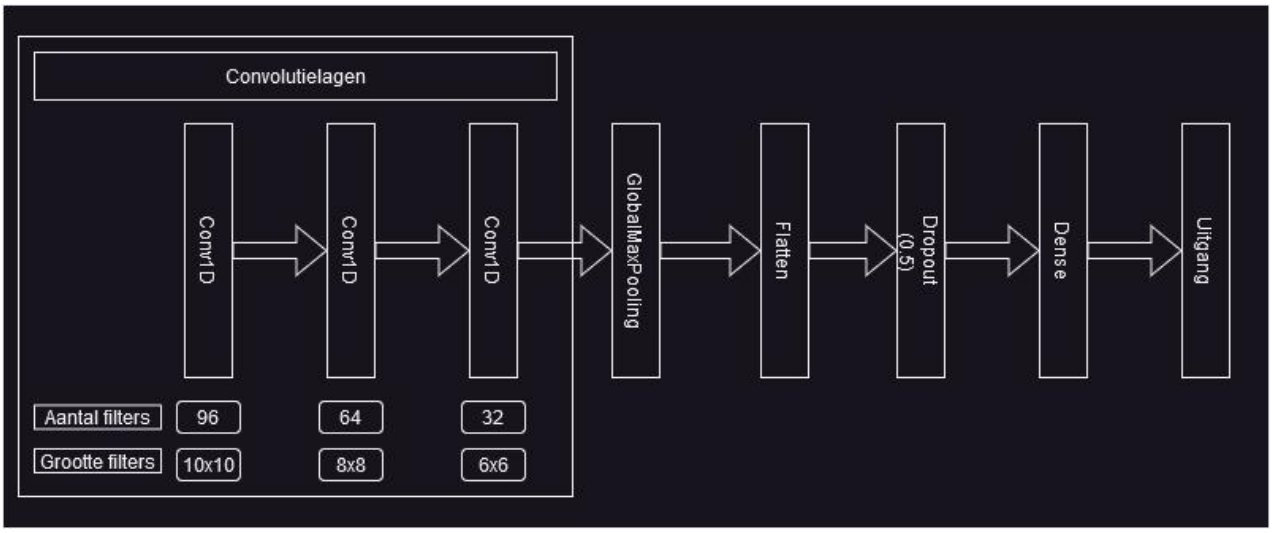
3. BASISMODELLEN

In deze scriptie worden metingen uitgevoerd op twee verschillende *machine learning* modellen. Op deze manier kan verder onderzoek worden gedaan naar de invloed van verschillende optimalisatietechnieken op verschillende architecturen. Deze sectie gaat dieper in op de eigenschappen en de architectuur van beide modellen.

1.1. Het WISDM-model

Het eerste model dat binnen deze scriptie wordt gebruikt, heeft de naam WISDM-model gekregen. De naam is afkomstig van de dataset waarop dit model is getraind, namelijk de publieke *'Wireless Sensor Data Mining*'-dataset voor *Human Activity Recognition* (HAR). Deze dataset bevat data van zowel de accelerometer als gyroscoop van een smartphone. Daarnaast is er ook nog een subset voorzien die enkel de accelerometerdata bevat. Het is deze subset die gebruikt wordt voor het trainen en testen van het WISDM-model. De WISDM-dataset voor HAR is een publieke dataset en is terug te vinden in [6]. De distributie van het aantal samples per activiteit wordt weergegeven op **Figuur 14** en andere eigenschappen van deze dataset worden vermeld in **Tabel 11**.

***Figuur 14**: Aantal samples per activiteit in de WISDM-dataset.*



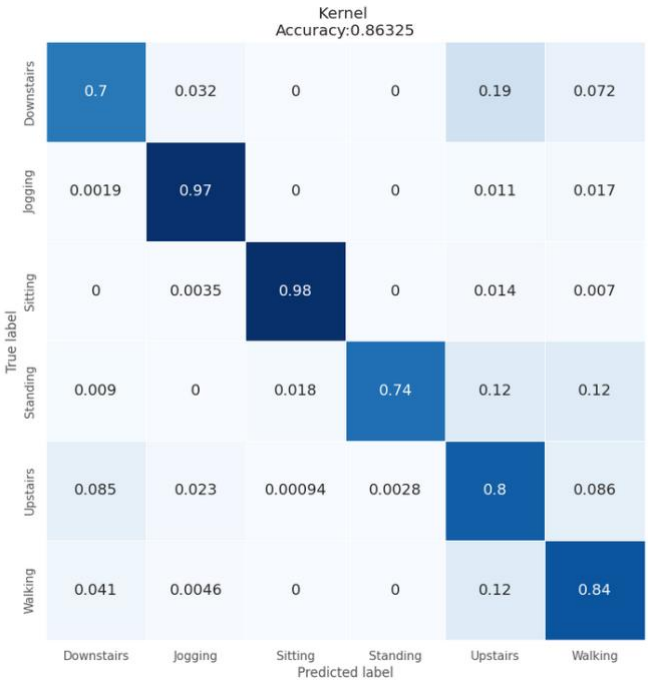
***Figuur 15**: Visualisatie van de architectuur van het WISDM-model.*

De architectuur van het WISDM-model wordt gevisualiseerd op **Figuur 15**. Er wordt gebruik gemaakt van drie opeenvolgende convolutielagen met *Rectified Linear Unit* (ReLU) als activatiefunctie. Via de eerste convolutionele laag leert het model de kenmerken van de acceleratiewaardes per activiteit aan. Dankzij de tweede laag worden complexere kenmerken geleerd door de combinatie van kenmerken die in de vorige laag zijn geleerd. Via de derde laag leert het model nog complexere kenmerken van de data op basis van uitgangen van de voorgaande lagen.

De volgende laag is een *GlobalMaxPooling*-laag. Deze laag neemt het maximum van alle tijdsstappen voor elk *featuremap*. Hierdoor wordt de dimensionaliteit van de data verminderd, waarbij alleen de meest relevante informatie (die met de hoogste waarde) per *featuremap* overblijft. Vervolgens zorgt de *Flatten*-laag voor de omzetting van de multidimensionale uitgang van de vorige laag om in een ééndimensionale vector. De zesde laag is een *dropout*-laag, bedoeld om overfitting te voorkomen. Het *dropout*-percentage is ingesteld op 50%. Tijdens het trainingsproces wordt willekeurig 50% van de neuronen uitgeschakeld, waardoor het model gedwongen wordt om robuustere kenmerken te leren zonder te sterk te leunen op individuele neuronen. De laatste laag is een *dense*-laag, oftewel een volledig verbonden laag. Deze laag combineert de in de voorgaande lagen geleerde kenmerken om de uiteindelijke uitgang te

genereren. Het aantal neuronen in deze laag is gelijk aan het aantal mogelijke klassen, in dit geval zes. Dit resulteert in een ééndimensionale vector van lengte zes, waarbij elke index de waarschijnlijkheid aangeeft dat de invoersample tot een bepaalde klasse behoort.

Aanvankelijk bestond de trainingsdataset uit 70% van de samples, terwijl 30% werd gebruikt als testdataset. Na het trainen bereikte het model een hoge nauwkeurigheid van ongeveer 90%. Echter, toen het model later op de smartphone werd geïmplementeerd en getest, presteerde het minder goed. Daarom werd de dataset op een andere manier opgesplitst: er werd gekozen voor een methode van groepsdeling. Hierbij wordt een bepaald percentage van de personen beschouwd in de trainingsgroep en de rest in de testgroep. Op deze manier wordt het model getraind op data van een specifieke groep en geëvalueerd op data van een andere groep. Hierdoor worden de tests uitgevoerd op data van personen die tijdens de trainingsfase niet werden meegenomen. Dit resulteerde in een lagere nauwkeurigheid van 86,33%, wat verklaard kan worden door het feit dat er getraind werd op andere personen dan waarop getest werd. Het model werkt met tijdvensters van vier seconden, dus elke sample komt overeen met 4*20 (bemonsteringsfrequentie) = 80 datawaarden.



***Figuur 16**: Verwarringsmatrix WISDM-model.*

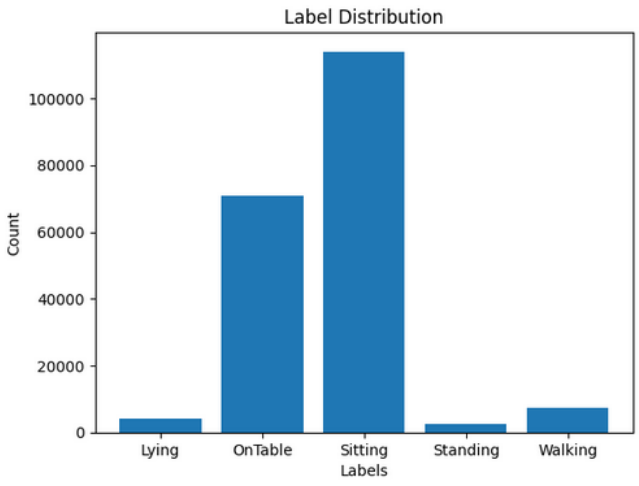
Voor het basismodel werd gewerkt met een totaal van 30 epochs. Een epoch verwijst naar het aantal keren dat het model door de volledige trainingsdataset gaat tijdens het trainingsproces. Bij dit aantal epochs convergeerde het model naar de uiteindelijke nauwkeurigheid van 86,33%. **Figuur 16** toont de verwarringsmatrix van het WISDM-model. Dit is een tabel die de prestaties van een classificatiemodel evalueert door de voorspelde klasse te vergelijken met de daadwerkelijke klasse. Het geeft meer inzicht in de soorten fouten die het model maakt en helpt bij het identificeren van klassen die het model vaak verkeerd voorspelt. De y-as toont de uitgevoerde activiteit en de x-as toont welke activiteit het model voorspelde. De verwarringsmatrix is overwegend diagonaal, met enkele uitschieters. Bijvoorbeeld, de activiteiten 'trap aflopen' en ‘staan' worden door het model relatief vaak verkeerd voorspeld.

Het WISDM-model werd in eerste instantie gebruikt om vertrouwd te raken met *Convolutional Neural Networks* (CNNs), die zoals besproken in **Sectie 2** vaak worden gebruikt voor *Human Activity Recognition* (HAR). Later werd dit model ook gebruikt om de modelafhankelijkheid van verschillende optimalisatietechnieken te evalueren.

1.2. Het mBrain-model

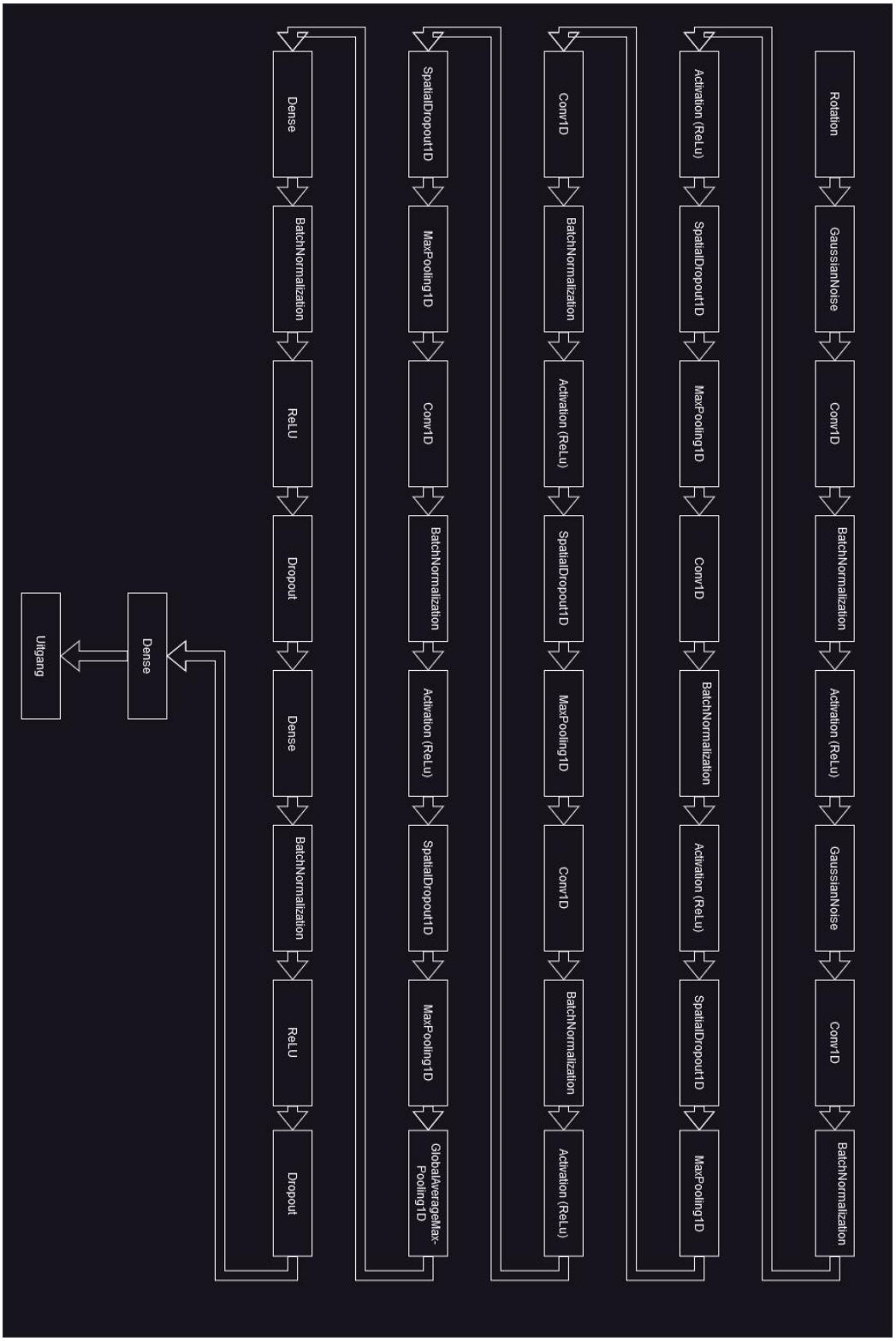
Het WISDM-model, het eerste model dat beschreven werd, is een model dat specifiek voor deze thesis is ontwikkeld. Het mBrain-model daarentegen, is een model dat al eerder werd gebruikt in een *cloud*-omgeving door de mBrain-studie. Dit model vormt de kern van de thesis, aangezien het de bedoeling is om dit model te implementeren op de *edge*. Voor dit model werd geen gebruik gemaakt van een publieke dataset, maar van een intern verzamelde dataset. Deze dataset onderscheidt vijf verschillende activiteiten van elkaar. Deze activiteiten verschillen van de activiteiten die werden opgenomen in de andere dataset, de WISDM-dataset. Een overzicht van de eigenschappen van deze dataset staan vermeld in **Tabel 11**.

Het is van belang om te melden dat de labels van de dataset niet handmatig zijn toegewezen. Deze zijn toegewezen via *machine learning*-voorspellingen op basis van een andere modaliteit, wat betekent dat de labels mogelijk niet optimaal zijn en enige onnauwkeurigheid kunnen bevatten. Tijdens de gegevensverzameling droegen de deelnemers een polsband. De labels voor het trainen van dit model zijn afgeleid van de voorspellingen van een ander model dat eerder werd getraind op de gegevens van deze polsband. Daarnaast zijn de monsters waarbij de GSM op tafel lag, verkregen met een algoritme dat gebaseerd is op drempelwaarden. Er werd een specifieke drempelwaarde gebruikt om te bepalen of de GSM op tafel lag, in plaats van een meer geavanceerde *machine learning*-benadering.



***Figuur 17**: Aantal samples per activiteit in de mBrain-trainingsdataset.*

De volledige gegevensverzameling omvat 31 unieke deelnemers, waarbij een groepsdeling is toegepast om de test- en trainingsgegevens te differentiëren. In totaal zijn er twintig deelnemers geïncludeerd in de trainingsdataset en tien in de testdataset. **Figuur 17** illustreert de verdeling van de activiteiten in de mBrain-trainingsdataset, waarbij opvalt dat de dataset significant ongebalanceerd is. Om deze onbalans te compenseren, wordt de 'compute_class_weight'-functie uit de sklearn-bibliotheek aangewend om gewichten toe te kennen aan elke klasse. Deze benadering garandeert dat de minderheidsklassen een groter gewicht krijgen dan de meerderheidsklasse, om te voorkomen dat alleen de meerderheidsklassen voorspeld worden.

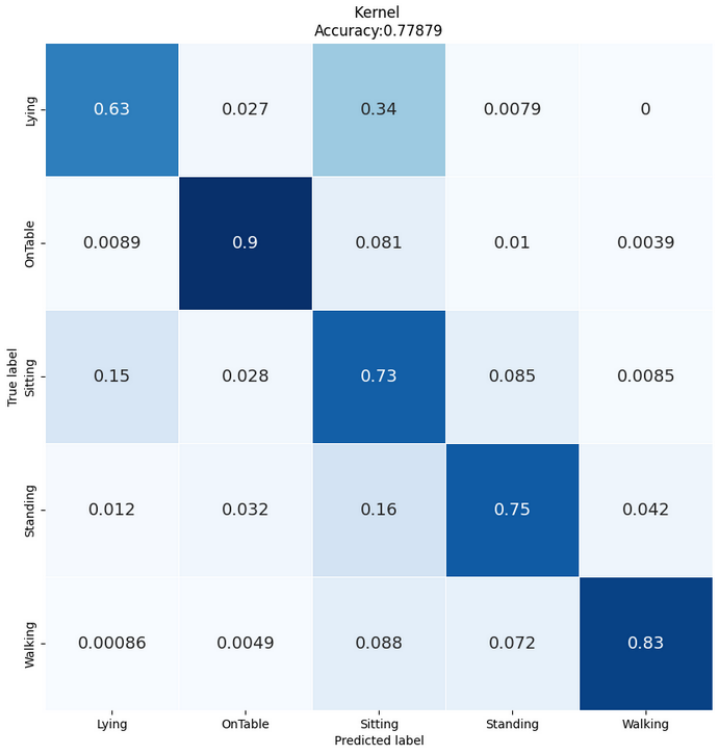


***Figuur 18**: Visualisatie van de architectuur van het mBrain-model.*

De architectuur van het mBrain-model is aanzienlijk complexer dan die van het WISDM-model, hoewel beide modellen van het *Convolutional Neural Network* (CNN) type zijn. De architectuur bevat meer lagen en meerdere soorten lagen. Net als het WISDM-model, bevat het ééndimensionale convolutielagen die complexe kenmerken van de invoergegevens detecteren. Daarnaast zijn er *dropout*-lagen om *overfitting* te voorkomen en dense-lagen. Maar dit model voegt ook andere soorten lagen toe:

- ***RotationLayer* en *GaussianNoise***: Deze lagen passen data-augmentatie toe en voegen ruis toe aan de ingangsdata om op deze manier de generalisatie van het model te verbeteren.
- ***BatchNormalization***: Deze lagen normaliseren de uitgangen van vorige lagen om de trainingsstabiliteit te verbeteren en voor convergentie.
- ***SpatialDropout1D***: Dit is een speciale vorm van een *dropout*-laag. Het doet hetzelfde als een gewone *dropout*-laag, maar dan in het ruimtelijke domein om *overfitting* verder te voorkomen.
- ***MaxPooling1D***: Deze lagen reduceren het aantal ruimtelijke dimensies om op deze manier meer globale kenmerken te gebruiken.
- ***GlobalAveragePooling1D*:** Deze laag berekent het gemiddelde van alle waarden langs de ruimtelijke dimensie. Hierdoor wordt de data gereduceerd naar een vaste grootte.

De combinatie van deze lagen zorgt ervoor dat zowel lokale als globale kenmerken worden gedetecteerd in de data. Hierdoor is het model in staat om te gaan met complexe patronen in de ingangsdata en generaliseert het goed tegen ongeziene data. De bemonsteringsfrequentie bedraagt 32 Hz en er wordt gewerkt met tijdvensters met 50% overlap van zes seconden. De ingangssamples bestaan dus uit een batch van 192 datasamples.



***Figuur 19**: Verwarringsmatrix van het mBrain-model.*

Figuur 19 illustreert de verwarringsmatrix van het mBrain-model, waarbij opvalt dat ook de minderheidsklassen in algemene zin accuraat worden geclassificeerd. Dit resultaat kan toegeschreven worden aan de eerder vermelde implementatie van gewichten per klasse, waardoor de minderheidsklassen een groter gewicht krijgen tijdens de trainingsfase. De uiteindelijke nauwkeurigheid van het model op de testdataset is 77.88%. Een vergelijkbare score werd gerealiseerd op de trainingsdataset.

Op **Tabel 10** staat een overzicht van de eigenschappen van de twee modellen. De kolom 'aantal samples' geeft weer hoeveel batchsamples er waren in de dataset waar de data vandaan kwam. Hierbij is het belangrijk om op te merken dat één zo'n *sample* bij het WISDM-model overeenkomt met 80 (4s * 20Hz) datapunten en 192 (6s * 32Hz) bij het mBrain-model. Bij het mBrain-model wordt verder ook gewerkt met tijdvensters met 50% overlap. Daarnaast worden ook de trainbare gewichten per model vermeld. Deze gewichten worden tijdens het trainingsproces aangepast om de nauwkeurigheid van het model te maximaliseren. Het aantal trainbare gewichten is afhankelijk

van de architectuur van het model, inclusief het aantal lagen, het aantal neuronen in elke laag en eventuele specifieke structuren zoals convolutielagen. Dit is belangrijke informatie voor **Sectie 5**, waarin de optimalisatietechnieken worden besproken voor elk model.

***Tabel 10**: Overzicht eigenschappen van de gebruikte machine learning modellen.*

Model	Accuraathei d	Aantal epochs	Nieuw model	Train/test	Trainbare parameters
WISDM-model	86.33%	30	Ja	70% van de personen als traingingsdataset	66 678
mBrain-model	77.88%	250	Nee	21 personen voor trainingsdataset en 10 personen voor testdataset	13 254

***Tabel 11**: Overzicht van de eigenschappen van de gebruikte datasets.*

	WISDM-dataset	mBrain-dataset
Aantal samples	12 410	287 551
Wandelen	38.6%	3.7%
Liggen	5.5%	57.4%
Joggen	31.2%	2.0%
Zitten	4.4%	/
Trap oplopen	11.2%	/
Trap aflopen	9.1%	/
Gsm op tafel	/	35.7%
Staan	/	1.2%
Plaats accelerometer	Smartphone in de broekzak	Smartphone vastgebonden rond de nek
Bemonsteringfrequentie	20 Hz	32 Hz

1.3. Integratie

Deze sectie behandelde twee verschillende *machine learning* modellen die worden gebruikt voor *Human Activity Recognition* (HAR): het WISDM-model en het mBrain-model. Beide modellen maken gebruik van *Convolutional Neural Networks* (CNNs), maar ze verschillen aanzienlijk in architectuur, de gebruikte datasets en de plaats van de accelerometer.

Het WISDM-model is een nieuw ontwikkeld model dat is gebaseerd op een publieke dataset. Bij de verzameling van de data werd een smartphone gebruikt in de broekzak van de gebruiker. Aan de andere kant, het mBrain-model, gebruikt een andere dataset dat gebruik maakt van een smartphone die met een koord aan de nek van de gebruiker was bevestigd voor de datacollectie.

Beide modellen worden gebruikt voor het evalueren van de verschillende optimalisatietechnieken die behandeld worden in **Sectie 5**. Aangezien de exacte architectuur van beide modellen verschillend is, kan de architectuurafhankelijkheid van de optimalisatietechnieken worden besproken. Het evaluatiekader dat gebruikt wordt voor het uitvoeren van deze testen wordt beschreven in de volgende sectie, **Sectie 4**.

4. EVALUATIEKADER

Deze scriptie onderzoekt eerst de impact van diverse optimalisatietechnieken op de inferentietijd, nauwkeurigheid en opslagvereisten van een CNN-model. Het tweede deel van het onderzoek is gericht op het gebruik van een *cloud*- en *edge*-architecturen voor het HAR-model. Hierbij wordt specifiek de focus gelegd op de verschillen in batterijverbruik en inferentietijd tussen deze twee architecturen. Beide onderzoeksgebieden maken gebruik van hetzelfde evaluatiekader. Per segment zijn er kleine verschillen in de testmethode, deze verschillen worden later in de desbetreffende secties besproken. Deze sectie presenteert het overkoepelende evaluatiekader dat in beide onderzoeken wordt toegepast. Eerst worden de kenmerken van de gebruikte smartphone besproken. Nadien wordt de virtuele node die gebruikt werd beschreven. Tot slot worden de zelfontwikkelde Android-applicaties beschreven die gebruikt worden in de evaluaties.

4.1. Toestel

Voor alle metingen wordt gebruik gemaakt van hetzelfde toestel. Alle resultaten zijn verkregen en geëvalueerd op de Samsung Galaxy S10e. Deze smartphone beschikt over de volgende resources:

- **Chipset:** Samsung Exynos 9820 Octa-core CPU met volgende configuratie:
    - 2x Mongoose M4 kernen met kloksnelheden tot 2.7Ghz (prestatiekernen)
    - 2x Cortex-A75 kernen met kloksnelheden tot 2,3 GHz (prestatiekernen)
    - 4x Cortex-A55 kernen met kloksnelheden tot 1,9 GHz (efficiëntiekernen)
- **Werkgeheugen:** 6 GB
- **Batterij:** Lithium-Ion met een capaciteit van 3000 mAh

Voorafgaand aan de metingen is het apparaat volledig gereset om eventuele invloeden van andere applicaties te minimaliseren. Dit resulteerde in een apparaat dat alleen de standaard Android-applicaties bevat.

4.2. Virtuele Node

Het gebruik van virtuele nodes biedt de mogelijkheid om processen te beheren en te coördineren zonder zich zorgen te hoeven maken over de onderliggende hardware- en systeemcomplexiteiten. Dit concept is fundamenteel in cloud *computing*, waarin gebruikers rekenkracht, opslag en andere bronnen huren als virtuele nodes.

Op deze virtuele node is een Flask-applicatie geïmplementeerd. Deze applicatie heeft twee *endpoints*. De eerste endpoint, de basisroute "/", dient als een *'Hello World'*-endpoint. De tweede *endpoint* is toegankelijk via *'/upload'*. Wanneer een bestand met accelerometerdata via een *POST*-verzoek wordt verzonden naar *http://193.190.127.213:8080/upload*, begint de dataverwerking. Er worden vier verschillende functies gebruikt.

De eerste geïmplementeerde functie is *remove_nul_chars*. Deze functie vervangt eventuele NUL-karakters in het binnenkomende bestand door lege strings. Deze functie wordt gebruikt als buffer indien er een fout gebeurde bij het opslaan van de data of bij de datatransmissie. De tweede functie, *process_csv*, is verantwoordelijk voor het maken van samples van 192 datapunten. De derde functie, *make_inference*, laadt een *batch* in het model en geeft de inferentie terug.

Ten slotte is er een laatste functie, *handle_csv_upload*, die wordt aangeroepen wanneer er een POST-verzoek binnenkomt op *http://193.190.127.213:8080/upload*. Deze laatste functie verbindt de eerste drie functies en stuurt uiteindelijk een antwoordbericht terug naar de smartphone-applicatie. Dit responsbericht is een CSV-bestand dat de inferenties van het model bevat.

4.3. Android-applicaties

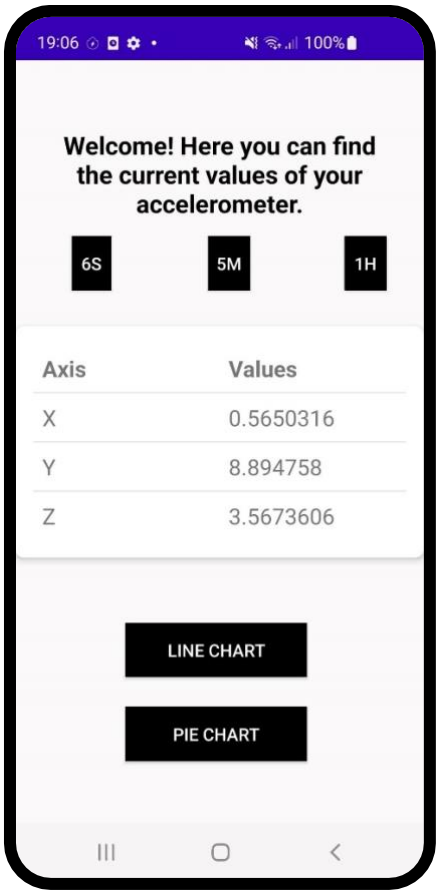
Voor de metingen zijn twee specifieke Android-applicaties ontwikkeld, bestaande uit drie verschillende *Activities* en een achtergrondservice. In de context van Android-ontwikkeling is een *Activity* een component die een specifiek scherm met een gebruikersinterface vertegenwoordigt. Het is in wezen een enkele, gefocuste interactie die een gebruiker met de app kan hebben. De achtergrondservice is een component die langdurige operaties kan uitvoeren en ook op de achtergrond kan werken, zelfs als de gebruiker niet interactief is met de applicatie. In deze sectie worden de architecturen van beide applicaties beschreven.

4.3.1. Edge-applicatie

MainActivity

De eerste *Activity* is de *MainActivity*. Deze *Activity* dient als een centrale hub voor de gebruikersinterface of de *User Interface* (UI). Daarnaast haalt het de sensorgegevens op en presenteert deze aan de gebruiker. Om dit te bereiken, implementeert het een achtergrondservice, namelijk de *AccelerometerService*. Op deze service wordt later nog dieper ingegaan. **Figuur 20** toont deze gebruikersinterface.

Er is gekozen voor een simplistische gebruikersinterface die voornamelijk wordt gebruikt om te tonen hoe de waarden binnenkomen. Er worden drie knoppen voorzien om het gewenste scenario te kiezen, namelijk verwerking elke zes seconden, elke vijf minuten of elk uur. Verder zijn er nog twee knoppen voorzien. Beide knoppen leiden naar een andere *Activity*. De 'LINE CHART'-knop leidt naar de *LineChartActivity*, en de 'PIE CHART'-knop verwijst naar de *PieChartActivity*. Voordat deze gebruikt kunnen worden, moet er een service worden voorzien die deze data verwerkt, zodat de activiteiten kunnen worden gevisualiseerd.



***Figuur 20**: Gebruikersinterface van de applicaties.*

AccelerometerService

Zoals al eerder werd vermeld wordt deze service opgeroepen door de MainActivity bij het openen van de applicatie. Deze achtergrondservice zorgt voor het verwerken van de accelerometerdata. Dit is een *service* die doorgaat met het uitvoeren van operaties, zelfs als de gebruiker niet interactief is met de applicatie, of als de applicatie zelf niet op de voorgrond draait. Dit is nuttig voor taken die onafhankelijk van de huidige gebruikersinterface moeten doorgaan. Op deze manier wordt er continu naar de accelerometer geluisterd en blijft het proces lopen wanneer de gebruiker een andere applicatie opent. Dit is van belang voor de verzameling van data voor *Human Activity Recognition* (HAR).

Deze *service* is verantwoordelijk voor:

- Het activeren van de sensorluisteraar.
- Het lokaal opslaan van de accelerometerdata.
- Het verwerken van de accelerometerdata aan de hand van een *machine learning* model.
- Het lokaal opslaan van de predicties van het model voor HAR.
- Het updaten van de user interface.

Er wordt een luisteraar gedefinieerd waarbij bepaald wordt welke sensor moet worden beluisterd en met welke bemonsteringsperiode in microseconden dit moet gebeuren. Ter illustratie van hoe dit werd geïmplementeerd, wordt hieronder de betreffende functie weergegeven. De bemonsteringsperiode is ingesteld op 20.000 microseconden (50Hz).

```
private fun setUpSensorListener(){
        val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        val accelerometerSensor = SensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
        sensorManager.registerListener(this, accelerometerSensor, 20000)
}
```

Door gebruik te maken van deze interface moeten er twee functies gedefinieerd worden:

```
override fun onSensorChanged(event: SensorEvent?) {}
override fun onAccuracyChanged(p0: Sensor?, p1: Int) {}
```

De functie *onSensorChanged* wordt opgeroepen telkens wanneer er een nieuwe waarde beschikbaar is. De frequentie hiervan werd bepaald bij het definiëren van de sensorluisteraar, namelijk 50 keer per seconde. De *onAccuracyChanged*-functie kan gebruikt worden om de nauwkeurigheid van de veranderingen mee te geven. Deze functie werd leeggelaten.

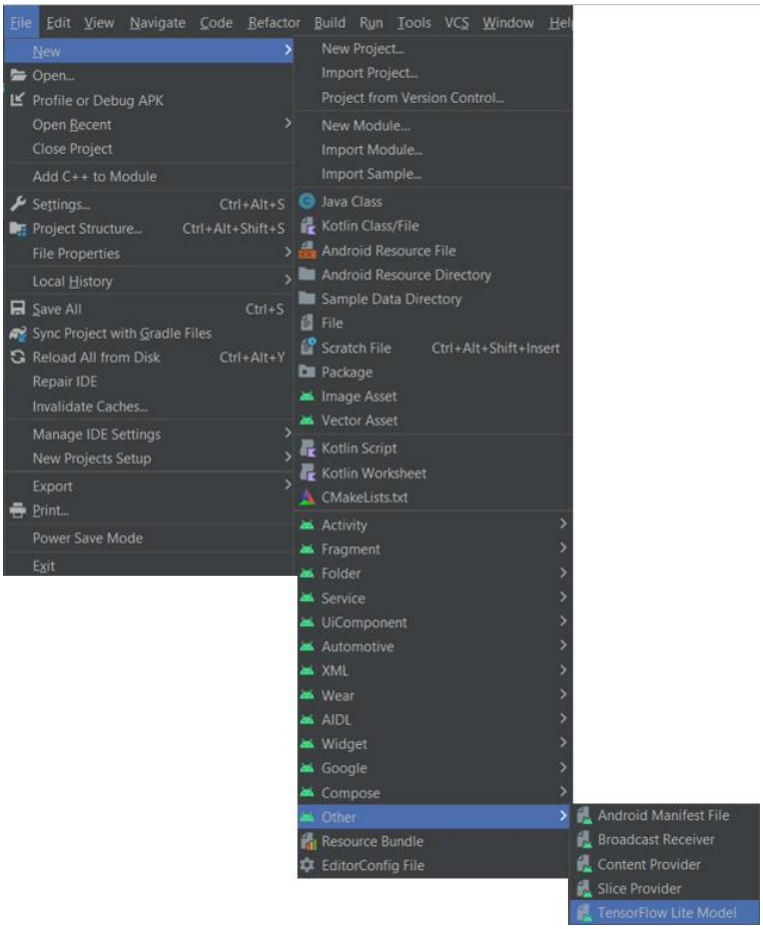
Het is belangrijk om aan te geven dat het toestel waarop de metingen werden uitgevoerd enkel in staat is om te samplen aan 50Hz of 100Hz. Aangezien zowel de WISDM-dataset als de mBrain-dataset werd gecollecteerd op andere frequenties, respectievelijk 20Hz en 32Hz, wordt er gebruik gemaakt van lineaire interpolatie.

Hieronder wordt een voorbeeld beschreven van hoe er tewerk wordt gegaan bij metingen met het mBrain model:

- Er wordt een Array van 50 elementen en een Array van 32 elementen aangemaakt.
- De datapunten worden gedurende 1 seconde opgeslagen aan 50Hz.
- De ratio wordt bepaald, namelijk 50Hz/32Hz = 1.5625.
- Voor elke index i in de Array van 32 elementen wordt de overeenkomstige index van de originele samples berekent, namelijk door i * 1.5625. (Bijvoorbeeld, voor i=0 is index = 0, voor i=1 is index = 1.5625 voor i=2 is index = 3.125, etc.)
- In het eerste geval is het dus eenduidig en wordt de eerste sample genomen. Bij het tweede geval is de overeenkomstige index=1.5625 tussen index = 1 en index = 2, vervolgens wordt een gewogen gemiddelde genomen (0.4375 voor index = 1 en 0.5625 voor index = 2).
- Dit wordt voor elke 'i' gedaan, namelijk 32 keer.
- Vervolgens is er een geïnterpoleerde Array beschikbaar.

Voor het lokaal opslaan van deze data worden twee verschillende bestanden gebruikt. Op deze manier wordt het conflict vermeden waarbij er tegelijkertijd naar een bestand wordt geschreven en uit het bestand wordt gelezen. Vooraleer de data wordt verwerkt, wordt de nieuwe data weggeschreven naar het andere bestand. Het bestand, waar tot dan toe naar geschreven werd, gaat naar de volgende fase, namelijk de verwerkingsfase. Zodra alle data uit een bestand is ingelezen en de verwerking ervan klaar is, wordt het bestand leeggemaakt, zodat het later opnieuw kan worden gebruikt.

De verwerking van de accelerometerdata wordt uitgevoerd door een *machine learning* model dat draait in de applicatie op de smartphone. Android Studio biedt een eenvoudige methode om zo een *machine learning* model aan de applicatie toe te voegen, zolang het in een TFLite-formaat is. **Figuur 21** illustreert hoe zo een model geïmplementeerd kan worden. Vervolgens wordt dit model opgeslagen in de ml-*directory* en kan het worden geïmporteerd.



Figuur 21: *Toevoegen TFLitemodel aan applicatie.*

Voordat het *machine learning* model wordt gebruikt, wordt de data voorbereid. Het bestand met de data wordt regel voor regel uitgelezen en er worden *batches* gemaakt met het benodigde aantal datapunten. Voor het WISDM-model worden batches van 80 datapunten gemaakt, en voor het mBrain-model worden batches van 192 datapunten gemaakt. De *batches* worden opgeslagen onder het *Array<Array<FloatArray>>*-datatype. Voordat deze batches in het *machine learning* model geladen kunnen worden, moeten ze worden omgezet naar een *ByteBuffer*-datatype. Om dit te realiseren, werd de volgende functie ontwikkeld:

```
    private fun convertInputArrayToByteBuffer(inputArray: Array<Array<FloatArray>>): ByteBuffer {
        val numElements = inputArray.size * inputArray[0].size * inputArray[0][0].size
        val byteBuffer = ByteBuffer.allocate(numElements * java.lang.Float.BYTES)
        byteBuffer.order(ByteOrder.nativeOrder())
        for (i in inputArray.indices) {
            for (j in inputArray[i].indices) {
                for (k in inputArray[i][j].indices) {
                    byteBuffer.putFloat(inputArray[i][j][k])
                }
            }
        }
        byteBuffer.rewind()
        return byteBuffer
    }
```

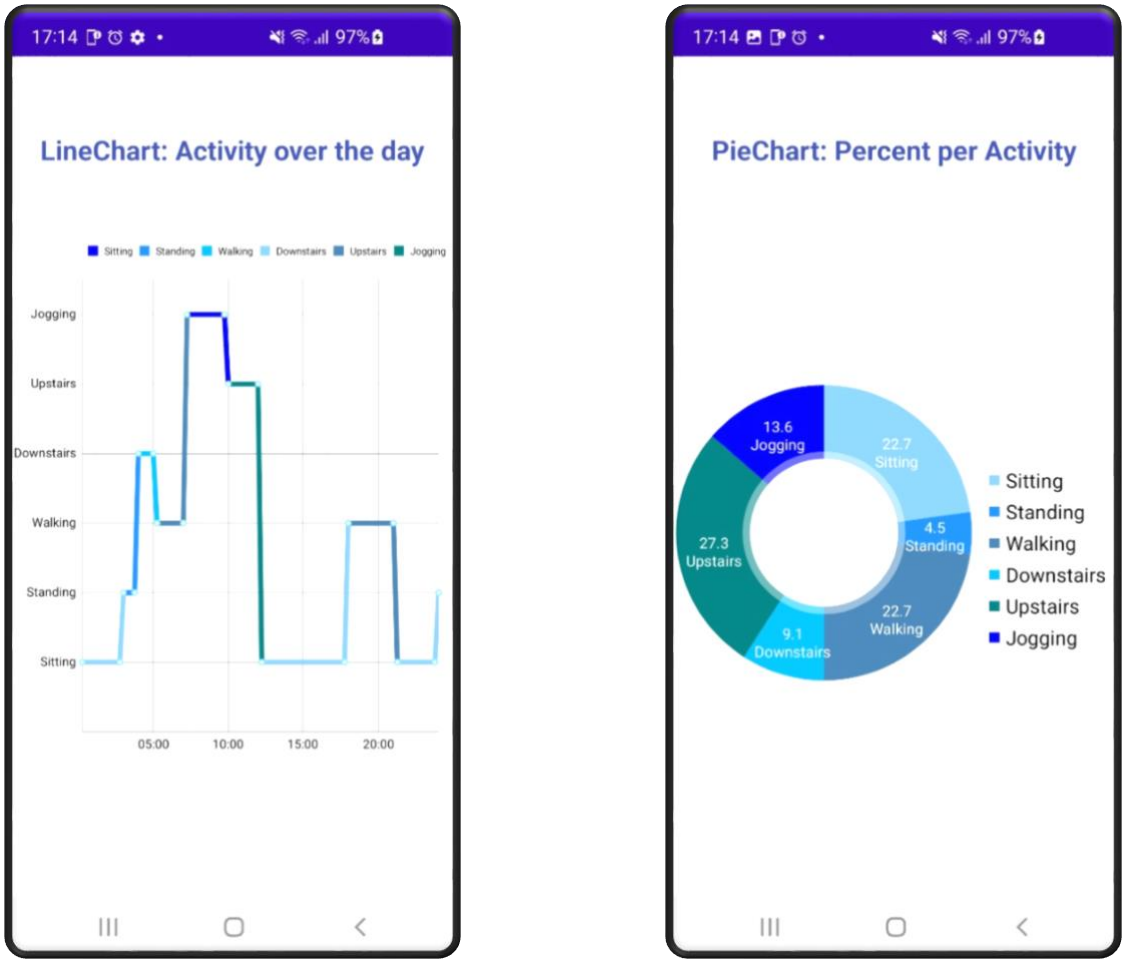
Na deze conversie kan het model worden geladen met zo'n ByteBuffer dat een array van bytes voorstelt. Vervolgens doet het model voorspellingen. Deze voorspellingen worden vervolgens lokaal opgeslagen. Deze voorspellingen kunnen later worden gebruikt om te visualiseren voor de eindgebruiker. Op deze manier kan een eindgebruiker bijvoorbeeld zijn activiteiten gedurende de dag inzien. Deze voorspellingen kunnen eventueel ook later naar een cloud worden gestuurd, waar deze data dan wordt opgeslagen. Dit is echter niet geïmplementeerd.

Daarnaast wordt in de achtergrondservice elke 100 milliseconden de *updateUI*-methode opgeroepen. Deze methode maakt gebruik van de *LocalBroadcastManager* om de huidige waarde te weergeven op de UI van de MainActivity.

LineChartActivity en PieChartActivity

De LineChartActivity en PieChartActivity zijn twee *Activities* die gebruikt kunnen worden om de activiteiten van de eindgebruiker gedurende de dag te visualiseren. Dit is een uitbreiding en geeft een voorbeeld van hoe de activiteiten gedurende de dag kunnen worden gevisualiseerd voor de eindgebruiker. Beide grafieken zijn afgebeeld op **Figuur 22**. Het is echter belangrijk om te

vermelden dat deze grafieken niet gebaseerd zijn op een echte meting. Ze maken gebruik van een zelfgeschreven CSV-bestand. Dit bestand bevat de activiteiten gedurende de dag met het bijbehorende tijdstempel.



***Figuur 22**: LineChart (links) en PieChart (rechts)*

Bij gebruik van echte data zou de *LineChart* mogelijk iets chaotischer ogen. Indien de echte data gebruikt zou worden, zou een bepaalde vorm van nabewerking noodzakelijk kunnen zijn. Hierbij zou men bijvoorbeeld het gemiddelde van elke 15 minuten kunnen berekenen om een datapunt te vertegenwoordigen. Dit zou eventueel ook kunnen helpen om mogelijk onjuiste voorspellingen te voorkomen. Daarnaast zou het bijvoorbeeld mogelijk zijn om korte activiteiten, zoals opstaan om koffie te halen en vervolgens weer gaan zitten, te filteren.

4.3.2. Cloud-applicatie

De applicatie voor de *cloud*-architectuur verschilt enigszins van die voor de *edge*-architectuur. Zowel de *MainActivity*, *PieChartActivity* en de *LineChartActivity* zijn hetzelfde. Het enige verschil tussen de applicaties is de *AccelerometerService*.

AccelerometerService

Bij deze architectuur is er geen noodzaak om een *machine learning* model in de applicatie te verwerken. De *AccelerometerService* is ook voor deze architectuur verantwoordelijk voor het opslaan van de accelerometerdata. De implementatie voor het opslaan van de data is hetzelfde als voor de *edge*-architectuur. Voor de verwerking van de data wordt gebruik gemaakt van een virtuele node, die eerder in deze sectie al werd besproken.

Om de data te verzenden, wordt de *sendToCloud*-methode geïnitialiseerd. Als er geen Wifi-connectie is, wordt de functie vroegtijdig afgesloten. Vervolgens wordt het juiste CSV-bestand gekozen om door te sturen. Daarna wordt een *aparte* thread gestart om de netwerkcommunicatie met de Flask-applicatie te behandelen. De methode maakt gebruik van de OkHttp-bibliotheek om de Hypertext Transfer Protocol (HTTP) *POST*-verzoeken te versturen.

Binnen de *thread* wordt een OkHttpClient geïnitialiseerd met behulp van de *getUnsafeOkHttpClient* methode. De *getUnsafeOkHttpClient* is een zelfgeschreven functie om communicatie met HTTP-berichten mogelijk te maken, en niet enkel met HTTPS-berichten. De *client* maakt vervolgens een *POST*-verzoek aan de Flask-applicatie met het CSV-bestand als bijlage in het verzoek. Als het verzoek succesvol is, wordt de serverrespons verwerkt. De serverrespons bevat een CSV-bestand met de inferenties die door de Flask-applicatie zijn gegenereerd. Dit bestand wordt opgeslagen op de lokale opslag van het apparaat. Als het verzoek niet succesvol is, wordt een foutbericht getoond in de console.

4.4. Overzicht

Deze sectie beschreef de verschillende onderdelen van het evaluatiekader die gebruikt worden in **Sectie 5** en/of **Sectie 6**. Indien nodig, wordt er in elk afzonderlijk onderzoek nog dieper worden ingegaan op de testomgeving. Hieronder wordt een samenvatting van de belangrijkste elementen van het evaluatiekader gegeven:

- **Toestel**: Alle metingen worden uitgevoerd op hetzelfde toestel.
- **Virtuele node**: Er wordt gebruik gemaakt van een virtuele node die *POST*-verzoeken kan ontvangen. Deze verzoeken bevatten CSV-bestanden met accelerometerdata. De virtuele node verwerkt de data met behulp van een *machine learning* model en stuurt vervolgens een CSV-bestand terug met de voorspellingen.
- **Edge-applicatie**: Dit is een Android-applicatie die accelerometerdata van de smartphone kan opvragen, deze data kan verwerken en de voorspellingen lokaal kan opslaan.
- **Cloud-applicatie**: Dit is een Android-applicatie die accelerometerdata van de smartphone kan opvragen, deze data kan versturen naar de virtuele node en de teruggestuurde voorspellingen lokaal kan opslaan.

***Tabel 12**: Overzicht van de verschillende delen van de evaluatiekader die gebruikt worden in Sectie 5 of Sectie 6.*

Onderdeel	Sectie 5	Sectie 6
Toestel	✓	✓
Virtuele Node	×	✓
Edge-applicatie	✓	✓
Cloud-applicatie	×	✓

5. OPTIMALISATIETECHNIEKEN

De laatste tijd is de vraag naar snellere en efficiëntere modellen binnen de wereld van kunstmatige intelligentie *en machine learning* een cruciale uitdaging geworden. In dit hoofdstuk van de scriptie worden verschillende optimalisatietechnieken voor *machine learning* modellen besproken, met name voor gebruik in *edge*-omgevingen zoals smartphones. Deze omgeving presenteert unieke uitdagingen, zoals beperkte rekenkracht, opslagcapaciteit en batterijduur. Het optimaliseren van modellen voor deze omgevingen betekent dat er rekening gehouden wordt met deze beperkingen, terwijl tegelijkertijd het nauwkeurigheidsverlies tot een minimum wordt beperkt.

Er wordt onderzoek gedaan naar drie technieken voor het optimaliseren van *machine learning* modellen voor *edge*-omgevingen: *depthwise separable convolutions*, p*runing*, kwantisatie en een combinatie van *pruning* en kwantisatie. Deze technieken worden toegepast op beide modellen die besproken zijn in **Sectie 3**. Deze technieken komen voort uit **Sectie 2**, de literatuurstudie.

5.1. Testomgeving

Er is een Android-applicatie ontwikkeld die enigszins afwijkt van de *edge*-applicatie die eerder besproken werd in **Sectie 4**. Deze aangepaste applicatie is ingezet voor metingen met betrekking tot de inferentietijd. De app verzamelt gegevens van de smartphone's accelerometer en slaat deze lokaal op in bestanden. Daarnaast is er een TFLitemodel in deze applicatie verwerkt om inferenties uit te voeren. De inferentietijd wordt gemeten door te loggen binnen de applicatie. Allereerst wordt de tijd in milliseconden genoteerd voordat de data naar het model wordt gestuurd. Vervolgens wordt de tijd na de inferentie ook vastgelegd. De inferentietijd is het verschil tussen deze twee tijdstippen. Deze tijd wordt vervolgens zichtbaar gemaakt in de applicatie of op de console. Het verschil bij deze versie (t.o.v. de versie besproken in **Sectie 4**) is de mate waarin de accelerometerwaarden worden weergegeven aan de gebruiker. Daarnaast worden de metingen gedaan op bestanden die vijf minuten lang de accelerometerdata capteerden. Dit wordt in totaal tien keer gedaan, waarna het gemiddelde genomen wordt van de gemeten inferentietijden. De data werd dus op voorhand al verzameld.

In eerste instantie werd ook de TensorFlow Benchmark Tool [42] gebruikt om de inferentietijden vast te leggen. Echter, deze tool leverde instabiele resultaten op. De gemeten tijden varieerden bij elke uitvoering. Vanwege deze inconsistentie zijn de resultaten niet opgenomen in deze scriptie.

5.2. Depthwise Separable Convolutions

De techniek *Depthwise Separable Convolutions* (DSC's) behoort tot de technieken die naar voren zijn gekomen in het literatuuronderzoek. Deze methode wordt toegepast in modellen zoals Xception, MobileNet en ShuffleNet [24,25,26] en is voornamelijk geëvalueerd op beeldgegevens. In deze scriptie wordt echter gebruik gemaakt van tijdreeksdata.

De tijdreeksdata die in deze scriptie wordt gebruikt bevat drie verschillende soorten kanalen: de x-richting, de y-richting en de z-richting, terwijl afbeeldingen doorgaans drie kleurkanalen bevatten: rood, groen en blauw. Hoewel er enige gelijkenis is in het gebruik van drie kanalen, is er een belangrijk onderscheid in de dimensie van de data. Afbeeldingen zijn tweedimensionaal, met een gedefinieerde hoogte en breedte, terwijl de tijdreeksdata die gebruikt worden voor HAR ééndimensionaal is, zonder concept van hoogte en breedte. Hierdoor worden in deze scriptie ééndimensionale convoluties gebruikt, terwijl bij beeldgegevens tweedimensionale convoluties worden toegepast. Dit verschil leidt tot een unieke implementatie van DSC's, die afwijkt van die in de eerdergenoemde studies.

De details van de werking van DSC's werd al besproken in **Sectie 2**. DSC's vormen een variant op standaard convoluties in neurale netwerken, die tot doel hebben het aantal benodigde berekeningen te verminderen. De convoluties worden in twee stappen uitgevoerd: eerst wordt de convolutie op elk kanaal afzonderlijk toegepast (*depthwise*), vervolgens over de drie kanalen heen (*pointwise*). Dit proces zorgt voor een efficiënter/minder gebruik van operaties. Hoewel DSC's de efficiëntie van het model verbeteren door het aantal berekeningen te verminderen, kunnen ze potentieel ook het vermogen van het model om complexe kenmerken te leren verminderen. Dit is omdat ze minder modelparameters bevatten in vergelijking met standaardconvoluties.

Deze techniek wordt toegepast op zowel het WISDM-model als het mBrain-model, waarvan de structuren besproken zijn in **Sectie 3**. Om deze techniek te implementeren, is elke *Conv1D*-laag vervangen door een *SeparableConv1D*-laag. De *SeparableConv1D*-laag, die beschikbaar is in de TensorFlow-bibliotheek, maakt gebruik van DSC's in plaats van standaard convoluties.

***Tabel 13**: Resultaten Depthwise Separable Convolutions op het WISDM-model en het mBrain-model.*

	WISDM-model		mBrain-model	
	Conv1D	SeparableConv1D	Conv1D	SeparableConv1D
Aantal parameters	66 678	10 052	13 653	6 260
Modelgrootte (in TFLite-formaat)	272 772 bytes	47 532 bytes	71 960 bytes	47 664 bytes
Inferentietijd	172 ms	149 ms	136 ms	127 ms
Accuraatheid	87.83%	88.02%	77.75%	76.92%

Uit de resultaten blijkt dat het aantal parameters aanzienlijk vermindert bij de toepassing van *Depthwise Separable Convolutions* (DSC's) in vergelijking met het gebruik van reguliere convoluties. Voor het WISDM-model en het mBrain-model vermindert het aantal parameters respectievelijk met een factor 6.63 en 2.18. Het is belangrijk om op te merken dat het WISDM-model hoofdzakelijk uit convolutielagen bestaat, terwijl het mBrain-model meer verschillende typen lagen gebruikt, waaronder normalisatielagen, die ook parameters vereisen. Deze variëteit aan lagen in het mBrain-model vormt een van de redenen waarom het WISDM-model een grotere factoriale reductie van parameters vertoont dan het mBrain-model. Dit komt doordat deze techniek alleen de vereiste parameters van de Conv1D-lagen vermindert.

Bovendien gebruikt het WISDM-model grotere filters. In **Sectie 2**, het literatuuronderzoek, werd de formule voor het aantal operaties door DSC's vergeleken met het aantal operaties voor reguliere convoluties. Het is aangetoond dat DSC's minder operaties vereisen dan normale

convoluties, met als factor: $\frac{1}{N} + \frac{1}{D_K^2}$. Hierbij staat N voor het aantal filters D_K voor de dimensie van de filters. Zowel het aantal filters als de dimensies van de filters zijn groter in het WISDM-model dan in het mBrain-model. Dit verklaart verder waarom de factoriale daling van het aantal parameters tussen de twee modellen verschilt. Het verklaart ook waarom de modelgrootte van het WISDM-model aanzienlijk meer afneemt dan bij het mBrain-model. Daarnaast resulteert het in beide modellen in een vermindering van de inferentietijd, gemeten op de Android-applicatie, terwijl de nauwkeurigheid bij de modellen niet aanzienlijk verandert.

Uit deze analyse blijkt dat *Depthwise Separable Convolutions* een gunstige invloed hebben op de complexiteit van beide modellen die werden onderzocht. Het aantal parameters neemt aanzienlijk af, wat leidt tot een verkleining van zowel de modelgrootte als de inferentietijd. Dit alles terwijl de nauwkeurigheid vrijwel hetzelfde blijft.

5.3. Pruning

P*runing*, of uitdunning in het Nederlands, is een techniek die wordt gebruikt om de complexiteit van neurale netwerken te verminderen door minder belangrijke neuronen of gewichten te verwijderen. Deze techniek heeft als doel een compacter en efficiënter model te creëren dat minder rekenkracht vereist door minder vermenigvuldigingen en optellingen uit te voeren, dankzij de uitgedunde/*sparse* matrices. Het is belangrijk om te benadrukken dat de effectiviteit van deze techniek mogelijks afhangt van het specifieke model en de toepassing. Daarom wordt de techniek geëvalueerd op beide modellen die werden behandeld in **Sectie 3**. Daarnaast worden ook verschillende hoeveelheden procentuele uitdunning toegepast. Zo worden het aantal nulwaarden van de gewichtsmatrices vastgelegd op 25%, 50%, 75%, 80%, 85% en 90%.

Een veel gebruikte *pruning*-techniek houdt in dat gewichten worden geëlimineerd op basis van de magnitude ervan, wat leidt tot een *sparse* (verdund) model. Bij deze benadering worden gewichten met de laagste magnitude geëlimineerd, op basis van de veronderstelling dat deze gewichten minder invloed hebben op de uitgang van het model en bijgevolg op de nauwkeurigheid. Deze techniek begint vanuit een getraind model, waarbij vervolgens de kleinste vooraf gedefinieerde gewichten worden gezet naar 0. Andere strategieën identificeren structurele redundantie, zoals het verwijderen van neuronen of hele lagen in plaats van individuele gewichten.

5.3.1. Dense en Sparse matrices

Dense matrices zijn matrices waarbij de meerderheid van de elementen niet-nul elementen zijn. *Sparse* matrices daarentegen, zijn matrices waarbij de meeste elementen nul zijn. Er zijn verschillende formaten beschikbaar voor het efficiënt opslaan van *sparse* matrices, zoals *Coordinate List* (COO), *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC) en *Tensor Algebra Compiler* (TACO)[43]. Door gebruik te maken van deze formaten kan de opslag- en computationele complexiteit worden gereduceerd in vergelijking met het formaat voor *dense* matrices.

Binnen deze thesis wordt er gewerkt met de TensorFlow en TensorFlow Lite (TFLite) bibliotheken. TFLite biedt een optimalisatiefunctie voor *sparse* matrices aan, genaamd 'EXPERIMENTAL_SPARSITY'. Na het toepassen van deze optimalisator worden de matrices opgeslagen in een *sparse* matrix formaat, in het bijzonder het TACO-formaat. Deze optimalisator is speciaal ontworpen voor modellen op mobiele en *edge*-toestellen [44].

TACO is een geavanceerde techniek die, afhankelijk van de tensor, een opslagformaat kiest zoals COO, CSR, CSC, etc. Daarnaast kan het ook technieken combineren. Het is moeilijk om te voorspellen hoe TACO reageert op een gegeven tensor. Het gaat op zoek naar de efficiënte wijze voor het opslaan van deze uitgedunde matrices.

5.3.2 Integratiemethode

De *pruning*-techniek die wordt besproken en toegepast op beide modellen is de eerder vermelde methode die gebaseerd is op de magnitude van de gewichten. Dit kan verder worden onderverdeeld in de niet-iteratieve methode en de iteratieve methode.

Voor het uitdunnen van het model is gebruik gemaakt van twee TensorFlow-bibliotheken, namelijk *tensorflow-model-optimization* en *tensorflow.keras*. Hieronder wordt stap voor stap uitgelegd hoe *pruning* kan worden gerealiseerd. Vervolgens wordt de werkwijze die in deze scriptie is toegepast besproken.

Iteratieve methode:

1. Train een basismodel.
2. Definieer en kies *pruning* parameters. Er zijn vier verschillende soorten die gedefinieerd kunnen worden, deze worden vermeld in **Tabel 14**.
3. Plaats de *prune_low_magnitude-wrapper* om het model heen, waarbij de *pruning_parameters* worden meegegeven. De *prune_low_magnitude-wrapper* werkt door

de gewichten in het model te rangschikken op basis van hun absolute waarden (magnitudes).

4. Compileer en train het model met de vooraf gedefinieerde *pruning_parameters*. Na deze stap zijn de gewichten van het model uitgedund volgens het vooraf gedefinieerde percentage.
5. Vorm de dense matrices met nulwaarden om naar een *sparse* matrix formaat.

***Tabel 14**: Verschillende parameters voor pruning.*

Pruning Parameter	**Opties**	**Beschrijving**
pruning_schedule	ConstantSparsity	Behoudt vanaf de eerste stap dezelfde procentuele uitdunning.
	PolynomialDecay	Past een 'polynomiaal afname'-functie toe voor de uitdunning tijdens het proces. Start met a% nul-gewichten en geleidelijk naar b% nul-gewichten.
	Zelfgeschreven	---
pruning_policy	PruneForLatencyOnXNNPack	Enkel 1x1 Conv2D-lagen *prunen.*
	Zelfgeschreven	---
block_size	*Tuple* van integers	Default: (1,1). Wanneer (3,3) wordt gekozen wordt het blok als 1 gewicht beschouwd.
block_pooling_function	'AVG', 'MAX' of 'MIN'	Functie die wordt toegepast op blok wanneer block_size groter is dan (1,1).

Niet-iteratieve methode:

1. Train een basismodel.
2. Zet de x%-laagste gewichten op nul.
3. Vorm de dense matrices met nulwaarden om naar een *sparse* matrix formaat.

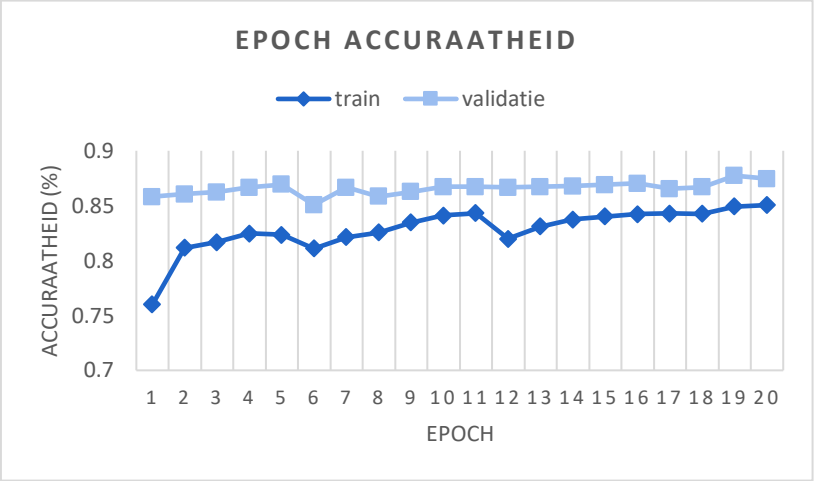
In deze scriptie is gekozen voor het gebruik van iteratieve *pruning*. Bij toepassing van niet-iteratieve *pruning* resulteerde dit in een veel lagere nauwkeurigheid van het model. Door de

training voort te zetten en geleidelijk nulwaarden in de matrices aan te brengen, werd een beter resultaat bereikt. Dit is dan ook de methode die verder wordt besproken en toegepast.
De parameters die voor de evaluatie worden gebruikt zijn:

- ***pruning_schedule***: Er is gekozen voor *PolynomialDecay*, met een initiële uitdunning van 10%. Daarnaast is de beginstap gezet op 0, zodat de *pruning* wordt toegepast vanaf de eerste stap tijdens de nieuwe trainingsfase. De eindstap is bepaald door het aantal epochs te vermenigvuldigen met het aantal stappen per epoch. Zo wordt tijdens deze fase de uitdunning bij elke stap uitgevoerd.
- ***pruning_policy***: Deze parameter heeft geen waarde gekregen en gebruikt dus de standaardwaarde. Elke laag met de *wrapper* om zich wordt bijgevolg uitgedund/gepruned met de op voorhand bepaalde hoeveelheid.
- ***block_size***: Deze parameter heeft geen waarde gekregen en gebruikt dus de standaardwaarde. Dit betekent dat er geen blok van gewichten wordt genomen, maar dat er per gewicht wordt gekeken.
- ***block_pooling_function***: Deze parameter heeft ook geen waarde gekregen. Aangezien de *block_size*-parameter op (1,1) wordt genomen, is het ook overbodig. De drie opties voor deze parameters zouden in dit geval telkens dezelfde resultaten opleveren.

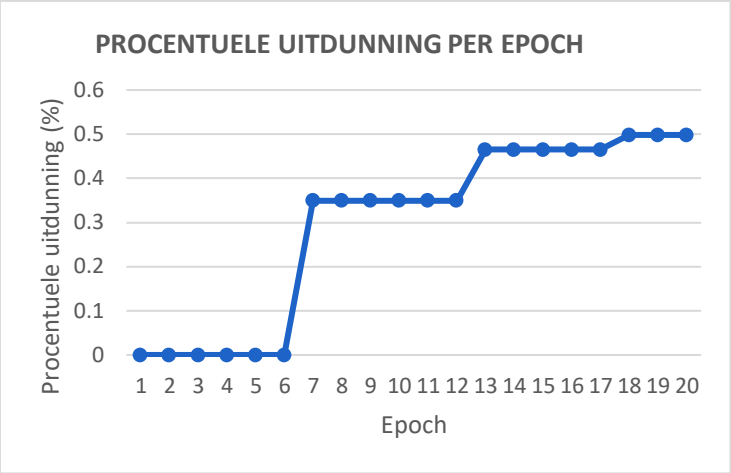
In deze scriptie wordt verder gebruik gemaakt van de TensorBoard visualisatietool die beschikbaar wordt gesteld door de TensorFlow-bibliotheek. TensorBoard is een *tool* die helpt om de interne aspecten van *machine learning* modellen te visualiseren, te begrijpen, te debuggen en te optimaliseren. Hierdoor kan gekeken worden naar de prestaties van het model en kunnen eventuele knelpunten of *bottlenecks* in het model geïdentificeerd worden. In deze scriptie is de visualisatietool voornamelijk gebruikt om *bottlenecks* op te sporen en het proces te visualiseren. Er werden geen opmerkelijke of ongewone knelpunten gedetecteerd. Voor de visualisatie van het proces worden een aantal grafieken voorzien.

Een van de grafieken die ter beschikking wordt gesteld door de TensorBoard is de *epoch_accuracy-grafiek*. Deze grafiek toont de nauwkeurigheid van het model op zowel de eerder meegegeven testdata als validatiedata. Een andere handige *tool* is dat grafieken worden voorzien van de procentuele uitdunning per epoch. Op deze manier kan er inzicht verkregen worden over wanneer en in welke mate tijdens de trainingsfase de *pruning* wordt toegepast. Deze grafiek is interessant om te gebruiken in combinatie met de eerder vermelde grafiek die de nauwkeurigheid van het model per epoch visualiseert. Op **Figuren 23 en 24** worden respectievelijk de nauwkeurigheid van het model per epoch voorgesteld en de procentuele uitdunning per epoch van een willekeurig gekozen laag. Er wordt gebruik gemaakt van *PolynomialDecay*, waarbij de procentuele uitdunning stapsgewijs wordt doorgevoerd. De uiteindelijke uitdunningsparameter werd gezet op 50% voor dit voorbeeld.



***Figuur 23**: Voorbeeld TensorBoard-grafiek epoch accuraatheid.*

In dit specifieke voorbeeld toont **Figuur 23** dat de accuraatheid in de zesde epoch aanzienlijk afneemt. Dit is ook terug te zien op **Figuur 24**, waar te zien is dat in epoch zes ongeveer 35% van de gewichten op nul worden gezet. Vervolgens traint het model weer verder tot epoch 12, waarna er nog 10% extra wordt gepruned. Dit is wederom te zien op de grafiek van de nauwkeurigheid. Hetzelfde geldt voor epoch 17, waar het uiteindelijk de gewenste 50% uitdunning bereikt. De combinatie van beide grafieken geeft meer inzicht in de relatie tussen de uitdunning en de nauwkeurigheid en helpt bij het begrijpen van hoe het *pruning*-proces werkt.



Figuur 24: *Voorbeeld TensorBoard-grafiek procentuele uitdunning per epoch.*

Tot slot wordt het model omgezet naar een TFLite-formaat. De *tf.lite.TFLiteConverter.from_keras_model()* converteert het Keras-model naar een TFLitemodel dat geoptimaliseerd is voor het gebruik op mobiele en IoT-toestellen. Daarnaast is het heel belangrijk dat de *tf.lite.Optimize.EXPERIMENTAL_SPARSITY* wordt toegepast. Dit zorgt ervoor dat de matrices, met veel nul-elementen, worden omgezet naar een formaat speciaal voor uitgedunde matrices. Dit gebeurt met behulp van de eerder vermelde *Tensor Algebra Compiler* (TACO).

```
converter = tf.lite.TFLiteConverter.from_keras_model(stripped_pruned_model)
converter.optimizations = [tf.lite.Optimize.EXPERIMENTAL_SPARSITY]
pruned_50_tflite = converter.convert()

with open('pruned_50.tflite', 'wb') as f:
  f.write(pruned_50_tflite)
```

Figuur 25*: Codefragment voor de omzetting Keras-model naar TFLitemodel gebruikmakend van TACO.*

5.3.3. Resultaten

De *pruning* techniek wordt op verschillende niveaus toegepast op beide modellen, met de volgende procentuele uitdunning: 25%, 50%, 75%, 80%, 85%, en 90%. Op deze manier wordt de impact van verschillende hoeveelheden uitdunning geëvalueerd.

***Tabel 15**: Resultaten iteratieve pruning op het WISDM-model.*

Procentuele uitdunning	Opslaggrootte	Inferentietijd (data van 5 minuten)	Accuraatheid
Keras model	1 446 560 bytes	/	88.53%
TFlite	272 772 bytes	172 ms	87.83%
TFlite 25%	272 772 bytes	173 ms	87.58%
TFlite 50%	179 544 bytes	176 ms	87.40%
TFlite 75%	96 852 bytes	174 ms	79.98%
TFlite 80%	80 312 bytes	172 ms	75.85%
TFlite 85%	63 772 bytes	174 ms	76.80%
TFlite 90%	47 172 bytes	178 ms	71.94%

Tabel 15 geeft de resultaten weer na het toepassen van het pruning-proces op verschillende niveaus op het WISDM-model. Het is opmerkelijk dat de conversie van Keras naar TFLite resulteert in een aanzienlijke vermindering van de modelomvang. Het TFLite-formaat is meer dan vijf keer zo klein, zelfs zonder het toepassen van *pruning*.

Er zijn verschillende redenen waarom een TFLite-model kleiner is dan het oorspronkelijke Keras-model dat geconverteerd is. Een eerste reden hiervoor is de vereenvoudiging van de modelstructuur/graaf tijdens de conversie. Dit houdt in dat het operaties verwijdert die enkel en alleen nodig zijn tijdens de training, een voorbeeld hiervan is een *dropout*-laag. Door het verwijderen van dit soort lagen zal het TensorFlow Lite-formaat kleiner zijn. Dit komt doordat dit formaat uitsluitend gebruikt wordt voor inferentie en niet voor het trainen van modellen. Daarnaast kunnen er tijdens het conversieproces nog andere optimalisaties plaatsvinden, zoals kwantisatie, het tot nul reduceren van gewichten die dichtbij nul liggen, en het samenvoegen van meerdere opeenvolgende operaties tot één enkele operatie.

De modelgrootte is een cruciaal kenmerk voor de aanpassing naar een *edge*-vriendelijke omgeving. *Pruning* heeft aangetoond dat het een aanzienlijke impact kan hebben op de modelgrootte. Dankzij de combinatie van nul-gewichten en de *Tensor Algebra Compiler* (TACO) worden de tensors/matrices op een efficiëntere manier opgeslagen, wat leidt tot een vermindering van de modelgrootte. Echter, bij een uitdunning van 25% lijkt er geen verandering in de grootte te zijn. Dit is te wijten aan het feit dat TACO heeft bepaald dat het huidige formaat, het dense-formaat, de meest efficiënte opslagmethode is. Het past dus geen COO, CSR, CSC of een hybride toe op het model aangezien dit in dit geval niet leidt tot een efficiëntere opslag. Omdat de aanwezigheid van nul-elementen beperkt is, blijft de omvang van het model ongewijzigd. Vanaf 50%-uitdunning en hogere percentages is er echter wel sprake van een reductie in grootte. Dit komt doordat er meer nulwaarden in de matrices aanwezig zijn waardoor TACO een efficiëntere manier vindt om de matrices op te slaan.

De mate van reductie in grootte hangt af van de hoeveelheid nulwaarden en de locatie ervan in het model wat het moeilijk maakt om de exacte reductie in grootte te voorspellen. In dit specifieke geval lijkt een uitdunning van 50% de nauwkeurigheid van het model nauwelijks te beïnvloeden. Als de nauwkeurigheid bekeken wordt, is de algemene trend dat meer *pruning* leidt tot een lagere nauwkeurigheid. Dit komt doordat er minder gewichten beschikbaar zijn in het model. Vanaf een uitdunning van 75% is er een merkbare afname in nauwkeurigheid. In dit geval lijkt 50%-uitdunning de optimale keuze te zijn aangezien de nauwkeurigheid vrijwel onveranderd blijft, terwijl de omvang van het model aanzienlijk vermindert.

Een andere cruciale factor van het model is de inferentietijd omdat deze gerelateerd is aan het batterijverbruik. Bij gebruik van het model op dezelfde smartphone zal een langere inferentietijd resulteren in meer batterijverbruik. Uit de resultaten blijkt dat de inferentietijd in dit geval niet verandert.

Het is van belang om een juiste balans te vinden tussen het behoud van nauwkeurigheid en het verkleinen van de modelgrootte en inferentietijd zodat het model geschikt is voor *edge*-apparaten met beperkte bronnen. Over het algemeen heeft *pruning* op dit model vooral een impact op de modelgrootte. De inferentietijd daarentegen neemt niet af.

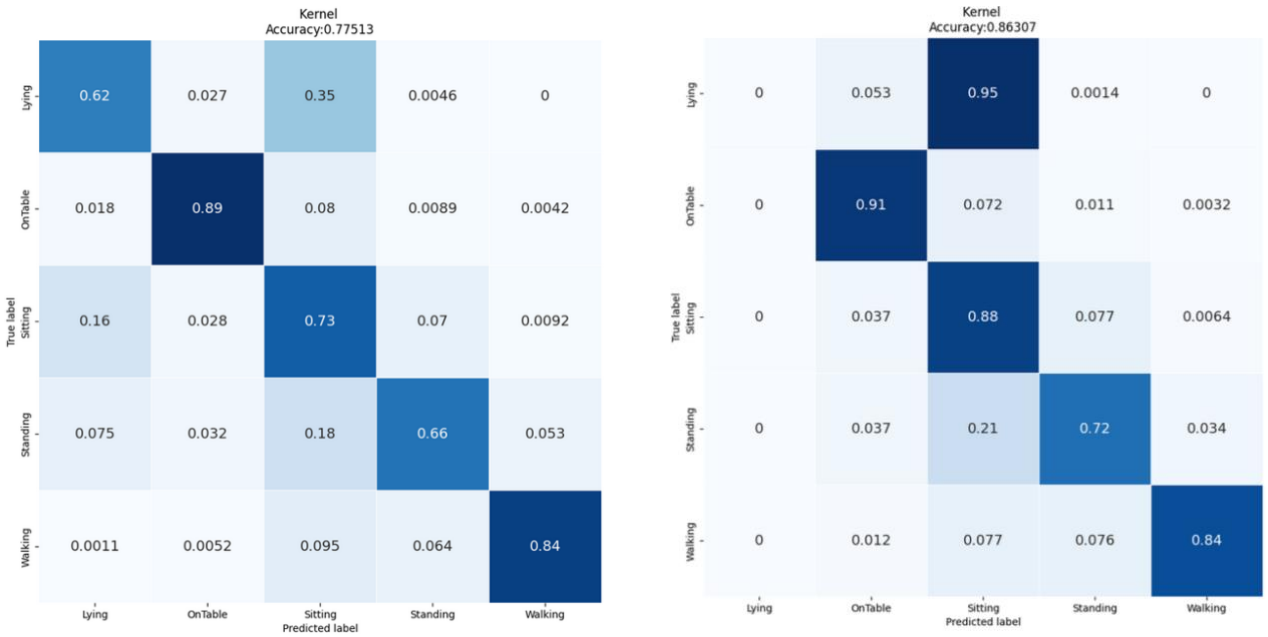
***Tabel 16**: Resultaten iteratieve pruning op mBrain-model*

Procentuele uitdunning	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	172 072 bytes	/	77.88%
TFlite	71 960 bytes	136 ms	77.75%
TFlite 25%	71 960 bytes	134 ms	77.51%
TFlite 50%	59 772 bytes	136 ms	77.35%
TFlite 75%	43 916 bytes	135 ms	77.13%
TFlite 80%	40 768 bytes	134 ms	77.55%
TFlite 85%	37 404 bytes	137 ms	86.34%
TFlite 90%	34 248 bytes	137 ms	86.29%

De *pruning*-techniek is ook toegepast op het tweede model, met dezelfde verschillende gradaties van uitdunning. De resultaten hiervan zijn samengevat in **Tabel 16**. Evenals bij het eerste model is de meest significante verbetering waarneembaar in de modelgrootte. Een gelijksoortige trend is zichtbaar waarbij het model verkleint naarmate er intensievere uitdunning wordt toegepast. De nauwkeurigheid van het mBrain-model blijft relatief stabiel na de uitdunning in vergelijking met het WISDM-model. Echter doet er zich een bijzonder fenomeen voor bij een uitdunning vanaf 85%, waarbij de nauwkeurigheid met bijna 10% stijgt. Dit wordt veroorzaakt doordat het model bij deze mate van uitdunning voornamelijk de klassen met de meeste leden voorspelt, ondanks de toegekende gewichtsverdeling per klasse (**Sectie 3.2**). Hoewel de totale nauwkeurigheid verbetert bij deze mate van uitdunning is dit in de praktijk een minder gewenst resultaat.

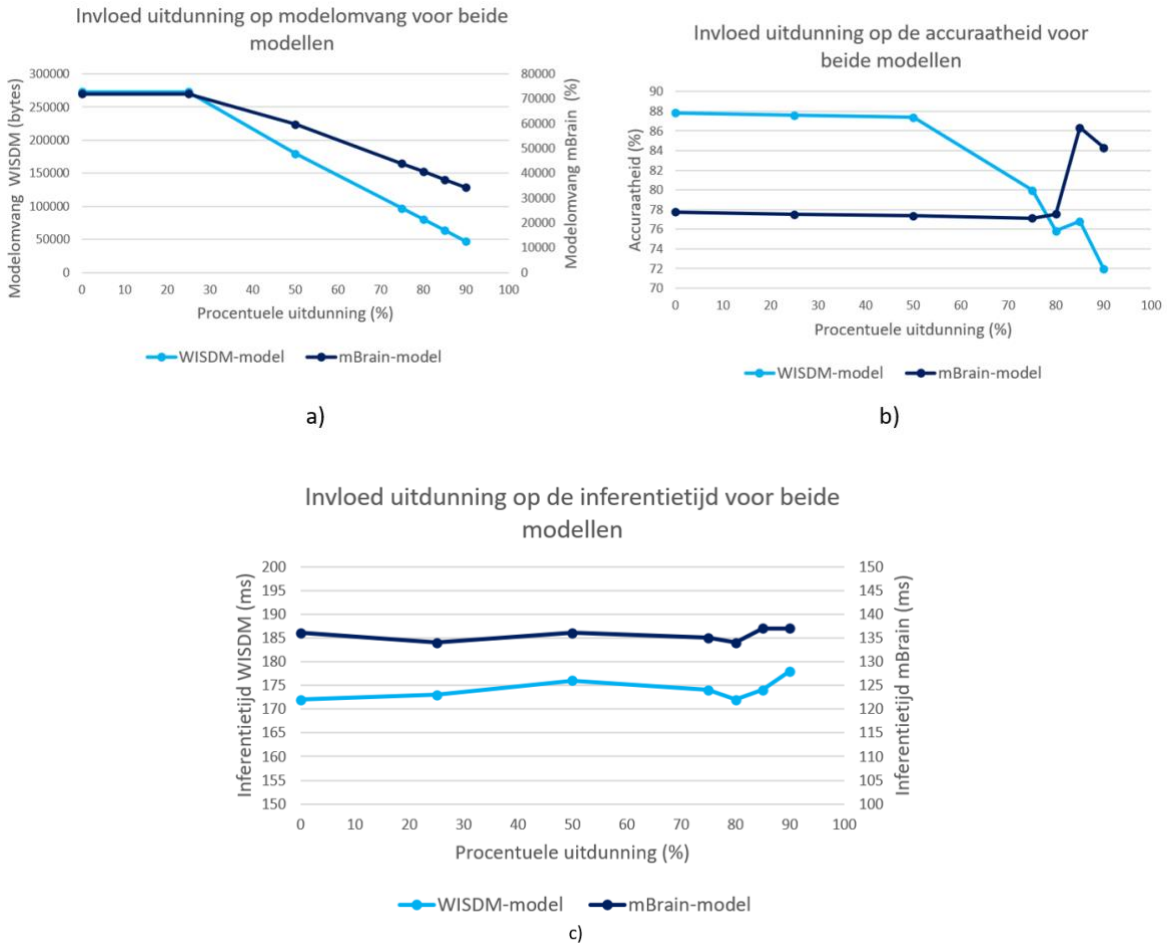
Figuur 26 illustreert de verwarringsmatrix voor 25%- en 90%-uitdunning om het verschil te benadrukken. Alle verwarringsmatrices worden voorzien in **Bijlage A**.

De resultaten van de Inferentietijd voor het tweede model zijn vergelijkbaar met die van het eerste model. Uit de resultaten blijkt dat ook in dit geval de inferentietijd niet verandert.



***Figuur 26**: Verwarringsmatrix 25%-uitdunning mBrain-model (links), verwarringsmatrix 90%-uitdunning mBrain-model (rechts)*

Over het algemeen wordt geconcludeerd dat *pruning* voornamelijk invloed heeft op de grootte van het model, en niet op de inferentietijd. Hoewel het prunen weldegelijk is toegepast, zoals blijkt uit de grote aanwezigheid van nullen in de gewichtsmatrices, leidt dit enkel tot een efficiëntere opslag. De techniek van '*sparse inference*', die in theorie zou leiden tot minder multiplicatieve berekeningen, wordt in dit geval niet gebruikt. TensorFlow geeft aan dat deze techniek specifiek wordt toegepast op Conv2D-lagen. Aangezien het model echter Conv1D-lagen gebruikt vanwege de eendimensionale aard van de accelerometerdata, verklaart dit waarom de inferentietijd niet afneemt.



Figuur 27: *Invloed pruning op beide modellen. a) invloed op de modelomvang. b) invloed op de accuraatheid. c) invloed op de inferentietijd.*

Op **Figuur 27** wordt de vergelijking gemaakt tussen de twee modellen na *pruning*. Er kan geconcludeerd worden dat de trends in grote lijnen voor beide modellen vergelijkbaar zijn:

- Afname van modelgrootte met toenemende gewichtsuitdunning.
- Geen verschil in inferentietijd.
- Afname van nauwkeurigheid met toenemende gewichtsuitdunning, na 50% voor het WISDM-model en vanaf 80% voor het mBrain-model.

Zoals besproken in **Sectie 2**, hebben P. Dhiman et al. een gelijkaardig onderzoek uitgevoerd naar de invloed van *pruning* op een CNN-LSTM-model [30]. De specifieke mate van uitdunning in dit onderzoek is niet vermeld. Daarnaast werden zowel de activaties als de gewichten van het model uitgedund. Dit resulteerde in een aanzienlijke verkleining van de modelgrootte, met een reductiefactor van iets meer dan twee. Hierbij bleef de nauwkeurigheid nagenoeg dezelfde. Het onderzoek heeft echter geen specifieke aandacht besteed aan de invloed van *pruning* op de inferentietijd.

Desalniettemin wordt geconcludeerd dat zowel het huidige onderzoek als dat van P. Dhiman et al. aantonen dat uitdunning*/pruning* een significant effect heeft op de grootte van het model, terwijl de nauwkeurigheid van het model slechts in beperkte mate wordt beïnvloed. Dit is weliswaar afhankelijk van de mate waarin de uitdunning wordt toegepast. Het is echter belangrijk op te merken dat P. Dhiman et al. geen onderzoek hebben gedaan naar de invloed op de inferentietijd. In deze scriptie werd hier wel onderzoek naar gedaan. Hierbij wordt besloten dat er geen verschil is in inferentietijd na uitdunning.

5.4. Post-training Kwantisatie

Een andere bekende optimalisatietechniek om *machine learning*-modellen geschikter te maken voor apparaten met beperkte rekenkracht, zoals smartphones, is kwantisatie. Het doel is vergelijkbaar met dat van *pruning*: de computationele complexiteit van het model verminderen zonder significant verlies van nauwkeurigheid. Hierdoor zouden batterijverbruik, inferentietijd en modelgrootte kunnen afnemen. In deze sectie wordt onderzocht wat de invloed is van verschillende vormen van post-training kwantisatie op deze eigenschappen.

Bij de overgang van het keras.h5-formaat naar een .TFLite-formaat vindt er al een eerste vorm van kwantisatie plaats. Deze eerste stap is moeilijk te voorspellen en varieert per model, dit werd al eerder besproken in **Sectie 5.3.3**. Daarnaast biedt TensorFlow nog drie verschillende mogelijkheden om de gewichten van een model te kwantiseren. In deze scriptie worden drie vormen van kwantisatie toegepast, waarbij alleen de gewichten worden gekwantiseerd, maar niet de biases van neuronen.

5.4.1. Float16

De eerste vorm van kwantisatie die kan worden toegepast, is het omzetten van float32-waarden naar float16-waarden voor de gewichten van het model. Beide datatypes worden gebruikt om decimale getallen te representeren. Het verschil tussen de twee ligt in de hoeveelheid geheugen die ze in beslag nemen en de precisie waarmee ze getallen kunnen weergeven. Float16 gebruikt 16 bits om een getal te representeren, terwijl Float32 gebruik maakt van 32 bits.

5.4.2. UINT8

De tweede vorm van kwantisatie betreft de omzetting van float32-getallen naar uint8-getallen. Hierbij wordt de overgang gemaakt van *floating-point* getallen naar *fixed-point* getallen. Door deze conversie wordt de precisie van de gewichten lager. De getallen die in het model worden geladen, moeten dan eerst geconverteerd worden naar uint8-waarden. De gewichten van het model nemen een bereik aan tussen -128 en 127. Hierbij is het belangrijk om op te merken dat, hoewel het bereik van de gewichten zal toenemen, de precisie zal afnemen. Een voorbeeld van hoe de matrix van een laag eruitziet, wordt getoond op **Figuur 28**. Dit is een voorbeeld van de vijfde laag van het WISDM-model na uint8-kwantisatie.

```
Layer  5  weights:
[[  29  -28  -78   51   86  -33  -57   48  -36  -21   16  -50   50   53
   -18    6]
 [ -51  -67   62   76  -57  -23  -26   51   -6   74   82   63  -82   61
   -70  -34]
 [ -41  -94   69  -51   78   27   83  -67   14  -15  -48   24    5  -16
    32   13]
 [ -81   42  -42  -34  -20   95  -97  -16  -67   -4  -40   40 -127  -78
    52    5]
 [  23   18   -2   68  -84  -35   24  -65  -51  -37  -41   51   34   47
   -46  -31]
 [  19   12  -52  -70  -16   96   23   -6   65  -53   87  -22   76   -6
    18  119]]
```

***Figuur 28**: Voorbeeld van gewichtswaarden van een machine learning laag na uint8-kwantisatie.*

Een ander belangrijk aspect is dat de waarden die in het model worden geladen ook van het datatype 'uint8' moeten zijn. Hierbij is het belangrijk om rekening te houden met het bereik van de mogelijke inputwaarden. Als er per inputbatch een standaardconversie naar het datatype 'uint8' wordt uitgevoerd, levert dit geen goede resultaten op. Als er twee verschillende inputbatches in het model worden ingevoerd en er eerst een standaardconversie naar het uint8-datatype wordt uitgevoerd, is dit het resultaat:

```
[20.36  56.23  20.]                [20  56  20]
[ 0.36   6.23  -7.]        ---->   [ 0   6 249]
[50.36 -30.23  -5.]                [50 226 251]

[ 0.36   0.23   0.]                [20  56  20]
[ 0.36   2.23  -2.]        ---->   [ 0   2 254]
[ 5.36  -3.23   0.]                [ 5 253   0]
```

Bij de eerste inputbatch wordt -7 omgezet naar 249 en -5 naar 251. Bij de tweede wordt -2 omgezet naar 254 en -3.23 naar 253. Het is dus afhankelijk van de batch hoe de conversie wordt toegepast. Het hangt namelijk af van de minimum- en maximumwaarde per batch. Om dit probleem te omzeilen, is een functie geschreven die op basis van voorkennis van de inputdata de conversie uitvoert. Hierbij werd op voorhand berekend wat de maximale en minimale waarden waren van de WISDM-dataset en de mBrain-dataset.

Als dit niet werd gedaan, resulteerde dit tot een aanzienlijk lagere accuraatheid bij beide modellen. Toen dit wel werd gedaan, waren de resultaten een stuk beter. Dit toont aan dat hier zeker aandacht aan besteed moet worden. Hieronder staat de functie voor de omzetting van *floating-point32* naar *uint8* getallen. Hierbij wordt er op voorhand berekend wat de maximale en minimale waarden zijn van de WISDM-dataset en de mBrain-dataset.

```
flattened_array = x_train.flatten()
min_float = flattened_array.min()
max_float = flattened_array.max()

def float32_to_uint8(float_array):

    float_array = np.clip(float_array, min_float, max_float)
    float_array = (float_array - min_float) / (max_float - min_float)
    uint8_array = np.uint8(float_array * 255)

    return uint8_array
```

Bij deze conversie naar het TFLite-formaat voor het *machine learning* model is het aanbevolen om een representatieve dataset mee te geven aan de converter. Op deze manier kan de converter beter inschatten wat het bereik van de inputwaarden is. Zonder een representatieve dataset kan het kwantisatieproces een gok zijn voor wat betreft het bereik van activatiewaarden, wat zou kunnen leiden tot een aanzienlijk verlies van modelnauwkeurigheid. De hele dataset werd telkens gebruikt als representatieve dataset. Dit zorgde voor een langer trainingsproces maar op deze manier werd verzekerd dat het zeker gaat om een dataset die de mogelijke inputwaarden representeert.

5.4.3. TensorFlow 's default optimalisator

De derde vorm van kwantisatie die onderzocht is betreft de dynamische kwantisatie optimalisator van TensorFlow Lite, die kan worden opgeroepen met tensorflow.lite.Optimize.DEFAULT. Het is een standaardoptimalisator en doet meer dan enkel kwantisatie. Omdat het zich voornamelijk richt op kwantisatie wordt het toch besproken in deze sectie. De verschillende optimalisaties die gebruikt worden zijn:

- **Gewichtskwantisatie**: Deze techniek converteert de 32-bits *floating-point* getallen die de gewichten van neuronen in het model beschrijven, naar een lagere voorstelling (uint8 of float16). De exacte vorm van kwantisatie hangt af van het model.
- **Activatiefusie**: Dit combineert opeenvolgende operaties in het model tot enkele, gecombineerde operaties. Dit kan leiden tot een snellere uitvoering en lager geheugengebruik.
- **Verwijderen van ongebruikte operaties en tensors**: Deze techniek verwijdert delen van het model die niet bijdragen aan de output, wat de modelgrootte verder kan verkleinen.

Het is niet mogelijk om exact te bepalen hoe deze optimalisator te werk gaat. De verschillende technieken die het toepast zouden echter wel voor een reductie van complexiteit kunnen zorgen.

5.4.4. Resultaten

***Tabel 17**: Resultaten van post-training kwantisatie op het WISDM-model*

Type	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	1 446 560 bytes	/	88.53%
Geen (float32)	272,772 bytes	172 ms	87.83%
Float16	140 832 bytes	169 ms	86.32%
Uint8	79 544 bytes	148 ms	86.23%
Dynamisch	76 288 bytes	146 ms	86.37%

***Tabel 18:** Resultaten van post-training kwantisatie op het mBrain-model.*

Type	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	152 248 bytes	/	77.88%
Geen (float32)	71 960 bytes	136 ms	77.75%
Float16	52.252 bytes	133 ms	77.76%
Uint8	40 864 bytes	131 ms	82.71%
Dynamisch	43 188 bytes	128 ms	77.83%

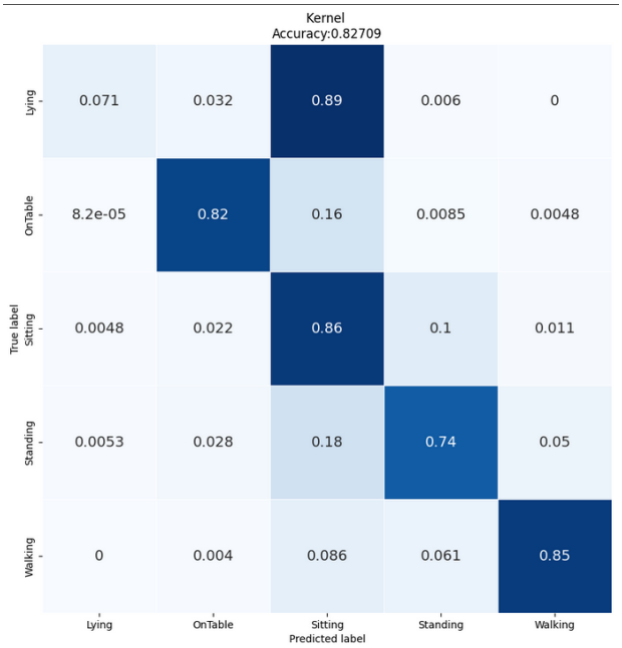
In Tabel 17 en 18 worden respectievelijk de resultaten van post-training kwantisatie voor het WISDM-model en het mBrain-model gepresenteerd. Bij toepassing van float16-kwantisatie is er een afname in modelgrootte, maar de inferentietijd blijft gelijk aan die van het float32-model. Dit kan verklaard worden door:

1. Kotlin, de gebruikte programmeertaal voor de applicaties, ondersteunt geen float16-waarden. Er wordt dus geen conversie gedaan van float32 naar float16 voor de accelerometerdata. Hierdoor zijn de ingeladen accelerometerwaarden nog steeds van het type float32, wat resulteert in float32-operaties.

2. CPU's zijn geoptimaliseerd voor float32-operaties. Zelfs als een conversie naar float16 mogelijk was, zouden er nog steeds float32-operaties plaatsvinden.

Zolang de conversie van accelerometerwaarden naar float16 niet kan plaatsvinden en het toestel geen GPU bezit die float16 operaties ondersteunt, zal de inferentietijd ongewijzigd blijven.

Het gebruik van *fixed-point* (uint8) getallen zorgt voor een aanzienlijke vermindering in modelgrootte voor beide modellen. De grootte van het WISDM-model neemt met ongeveer 3.5 keer af en de inferentietijd daalt met factor 1.16. Bij dit model blijft de nauwkeurigheid vrijwel ongewijzigd. Voor dit model blijkt dit dus de beste vorm van kwantisatie te zijn. Voor het mBrain-model geldt ongeveer hetzelfde. De verhouding van modelgrootte voor en na deze uint8-kwantisatie is ongeveer 1,8. Ook voor dit model daalt de inferentietijd met 1.04. Maar voor dit geval zijn de inferenties die worden uitgevoerd na deze techniek aanzienlijk slechter. De nauwkeurigheid stijgt maar de reden daarvoor is ongunstig. Het voorspelt namelijk nooit meer de minderheidsklasse 'liggen' (**Figuur 29**). De accuraatheid geeft in dit geval dus een vertekend beeld.



Figuur 29: *Verwarringsmatrix van het mBrain-model na uint8-kwantisatie.*

De dynamische standaardoptimalisator van TensorFlow is over de hele linie meest stabiele vorm van kwantisatie. Het zorgt bij beide modellen voor een vermindering van modelgrootte en inferentietijd zonder dat de nauwkeurigheid aanzienlijk wordt aangetast. De vermindering in nauwkeurigheid is voor het WISDM-model slechts 1,46%, de modelomvang is verminderd met factor 3,58 en de inferentietijd ongeveer 1,18 keer zo klein. Voor het mBrain-model worden er vergelijkbare resultaten behaald. De nauwkeurigheid blijft ook hier nagenoeg dezelfde, de modelomvang is gereduceerd met factor 1,69 en de inferentietijd is ongeveer 1,06 keer kleiner. Deze vorm van kwantisatie is dus het meest stabiel en behaalt gewenste resultaten voor *edge*-toestellen.

Zoals besproken in **Sectie 2**, hebben T. Zebin et al. eveneens de float32-conversie naar uint8 toegepast. Deze aanpassing was bedoeld om het model geschikt te maken voor het specifieke apparaat waarop het zou draaien namelijk een smartphone. Het onderzoek richtte zich echter uitsluitend op de nauwkeurigheid en opslaggrootte van het model na kwantisatie zonder specifieke aandacht voor de inferentietijd. Zowel de activaties als de gewichten van het CNN-model werden gekwantiseerd. Als resultaat werd een aanzienlijke reductie in modelgrootte bereikt, met een factor 7.62. Voor het WISDM-model en het mBrain-model bedroegen de reductiefactoren respectievelijk 3.43 en 1.79. Dit verschil kan worden toegeschreven aan het aantal trainbare parameters in beide modellen. Het onderzoek van T. Zebin et al. maakte gebruik van convolutionele lagen met grotere filters, wat resulteert in een hoger aantal gewichten en activaties. Hierdoor heeft de kwantisatie een grotere invloed op de totale omvang van het model. Het WISDM-model, dat werkt met aanzienlijk grotere filters, bevat veel meer trainbare gewichten dan het mBrain-model, dit werd reeds vermeld op het einde van **Sectie 3**. Dit verschil is dan ook duidelijk zichtbaar in de factoriale afname van beide modellen na de uint8-kwantisatie. In dit proefschrift wordt wel onderzoek gedaan naar de invloed van uint8-kwantisatie op de inferentietijd. Hierbij wordt besloten deze vorm van kwantisatie het predictieproces kan versnellen.

5.5. Post-training Kwantisatie in combinatie met Pruning

5.5.1. Integratiemethode

Kwantisatie en *pruning* zijn twee technieken die in deze scriptie worden gebruikt om de omvang en rekenintensiteit van neurale netwerken te verkleinen. Deze technieken hebben als doel dat modellen efficiënter kunnen werken op apparaten met beperkte rekenkracht of opslagcapaciteit, zoals mobiele apparaten of andere *edge*-apparaten. Eerst werden beide technieken afzonderlijk toegepast, in deze subsectie worden ze gecombineerd en getest. Eerst wordt *pruning* toegepast met dezelfde verschillende hoeveelheden uitdunning als in **Sectie 5.3.** Vervolgens wordt de standaardoptimalisator van *TensorFlow* gebruikt die kwantisatie, activatiefusie en verwijdering van ongebruikte operaties en tensors uitvoert. Deze vorm van post-training kwantisatie is gekozen omdat deze vorm voor beide modellen het meest stabiel was in het reduceren van inferentietijd en modelomvang waarbij de accuraatheid nauwelijks wordt aangetast.

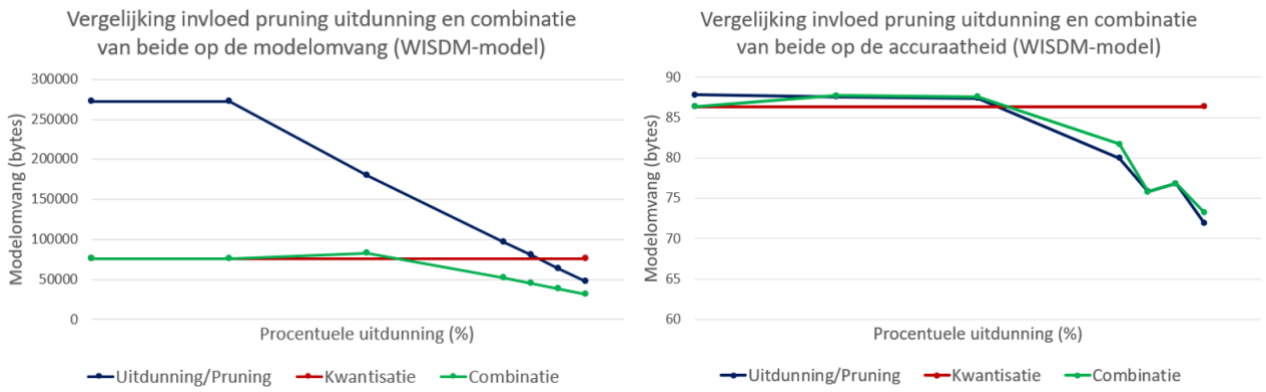
5.5.2. Resultaten

De resultaten van de combinatie van deze technieken op het WISDM-model zijn terug te vinden op **Tabel 19**.

Tabel 19: *Resultaten van pruning en post-training kwantisatie op het WISDM-model.*

Procentuele uitdunning	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	1 446 560 bytes	/	88.53%
TFlite	76 288 bytes	146 ms	86.37%
TFlite 25%	76 280 bytes	144 ms	87.73%
TFlite 50%	82 880 bytes	148 ms	87.57%
TFlite 75%	51 712 bytes	142 ms	81.69%
TFlite 80%	45 088 bytes	151 ms	75.80%
TFlite 85%	38 456 bytes	145 ms	76.83%
TFlite 90%	31 776 bytes	147 ms	73.24%

Bij het model dat voor 25% is uitgedund en gekwantiseerd, is de modelgrootte precies hetzelfde als wanneer alleen kwantisatie wordt toegepast. Dit resultaat is zoals verwacht omdat uit eerdere conclusies bleek dat de *Tensor Algebra Compiler* in eerste instantie geen overgang naar een *sparse* representatie van matrices maakte voor 25%-uitdunning. Het is echter opmerkelijk dat het model dat voor 50% is uitgedund en gekwantiseerd is, een iets grotere modelomvang heeft dan wanneer alleen post-training kwantisatie wordt toegepast. Een mogelijke verklaring is dat de matrices worden opgeslagen in een formaat, bepaald door de *Tensor Algebra Compiler* (TACO), waardoor mogelijks minder operaties worden gefuseerd. De combinatie van technieken leidt tot betere resultaten in termen van modelgrootte vergeleken met het afzonderlijk toepassen van de technieken na 50% uitdunning.



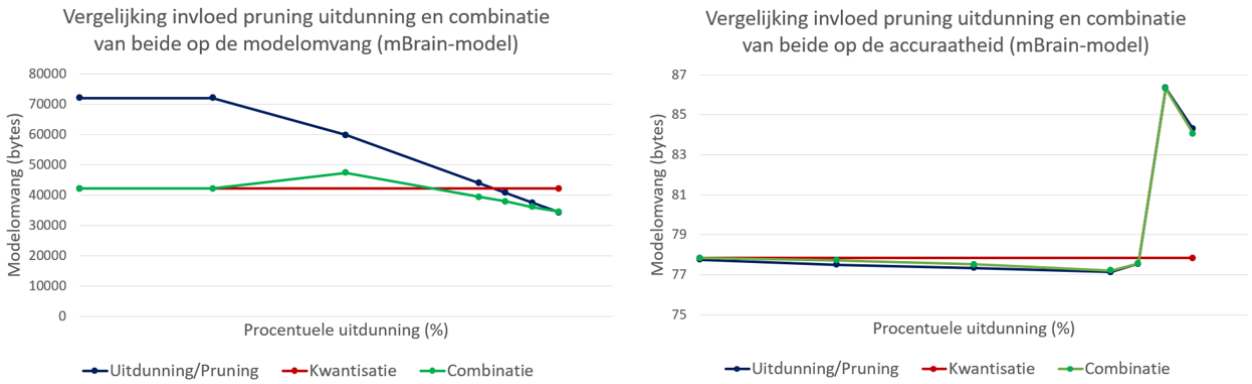
***Figuur 30**: Vergelijking invloed pruning, kwantisatie en de combinatie van beide technieken op de modelomvang (links) en op de accuraatheid (rechts) op het WISDM-model.*

De combinatie van beide technieken op het WISDM-model toont geen verlies aan nauwkeurigheid vergeleken met de versies waar enkel uitdunning wordt toegepast. Bovendien behaalt het betere scores voor modelomvang dan wanneer de technieken afzonderlijk worden toegepast wanneer er voldoende uitdunning wordt gehanteerd. De inferentietijd blijft echter dezelfde als wanneer enkel kwantisatie werd toegepast, daarom wordt hiervoor geen grafiek voorzien.

Tabel 20: *Resultaten van pruning en post-training kwantisatie op het mBrain-model.*

Procentuele uitdunning	Opslaggrootte	Inferentietijd (bestanden van 5 minuten)	Accuraatheid
Keras model	172 072 bytes	/	77.88%
TFLite	42 152 bytes	128 ms	77.83%
TFlite 25%	42 152 bytes	128 ms	77.73%
TFlite 50%	47 376 bytes	132 ms	77.52%
TFlite 75%	39 392 bytes	128 ms	77.22%
TFlite 80%	37 840 bytes	127ms	77.56%
TFlite 85%	36 040 bytes	129 ms	86.31%
TFlite 90%	34 472 bytes	129 ms	84.06%

Hetzelfde onderzoek werd uitgevoerd voor het mBrain-model. Ook bij dit model heeft de combinatietechniek bij 25%-uitdunning geen invloed ten opzichte van enkel kwantisatie om dezelfde reden als bij het WISDM-model. Daarnaast heeft de combinatie van 50%-uitdunning hier ook een nadelig effect op de modelgrootte in vergelijking met enkel kwantisatie. Vanaf de combinatie van 75%-uitdunning treedt er voor dit model ook een voordelig effect op. Ook hier volgt de nauwkeurigheid van de combinatiemethode de curve van de gewone *pruning*-methode.



Figuur 31*: Vergelijking invloed pruning, kwantisatie en de combinatie van beide technieken op de modelomvang (links) en op de accuraatheid (rechts) op het mBrain-model.*

Bij het analyseren van de invloed van de verschillende methoden op de inferentietijd zijn de resultaten vergelijkbaar met die van het WISDM-model. De inferentietijd blijft dezelfde als wanneer enkel kwantisatie werd toegepast, daarom wordt hiervoor geen grafiek voorzien.

Zoals besproken in **Sectie 2**, hebben P. Dhiman et al. ook gebruik gemaakt van magnitude-gebaseerde *pruning*, in combinatie met post-training kwantisatie. Specifieke details over de mate van uitdunning en de exacte vorm van post-training kwantisatie werden echter niet vermeld. Het is echter de moeite waard te vermelden dat ze geen gebruik hebben gemaakt van *floating-point* operaties, wat suggereert dat ze uint8-kwantisatie hebben toegepast. Vanwege het ontbreken van specifieke informatie over de mate van uitdunning en de verschillende benaderingen van post-training kwantisatie, is het moeilijk om directe vergelijkingen te maken. Desondanks concludeerden P. Dhiman et al. dat beide technieken elkaar versterken. Na het toepassen van *pruning* werd de nauwkeurigheid met 2.75% verlaagd, en na kwantisatie nam dit verder af met 1.78%. De grootte van het model werd na pruning ongeveer gehalveerd, van 56.65 Megabyte (MB) naar 28.16 MB, en na kwantisatie werd dit verder verminderd tot 20.62 MB. Deze resultaten komen overeen met de bevindingen die in deze scriptie zijn verkregen. Daarnaast werd in deze scriptie ook gekeken naar de invloed op de inferentietijd. Hierbij werd geconcludeerd dat deze niet verbeterde ten opzichte van wanneer enkel kwantisatie werd toegepast.

5.6. Conclusie

In deze sectie zijn drie verschillende technieken en één hybride vorm, die twee technieken combineert, toegepast om een *machine learning* model aan te passen aan de beperkte bronnen beschikbaar op een smartphone. De technieken zijn toegepast op twee verschillende architecturen en de tests zijn steeds uitgevoerd op hetzelfde toestel.

De eerste toegepaste techniek is het gebruik van *Depthwise Separable Convolutions* (DSC's) in plaats van reguliere convoluties. Deze techniek zorgt ervoor dat beide modellen aanzienlijk minder parameters bevatten dankzij de verminderde operaties. Afhankelijk van het initiële aantal parameters, het aantal filters en de grootte van de filters wordt een reductie in aantal parameters bereikt. Deze techniek zorgt voor een factoriale daling in parameters van 6.63 en 2.18, respectievelijk voor het WISDM-model en het mBrain-model. Hierdoor is de model omvang respectievelijk 5.74 en 1.51 keer kleiner. De daling in inferentietijd is 15% voor het WISDM-model en 7% voor het mBrain-model. Deze techniek verkleint dus zowel de modelomvang als de inferentietijd terwijl de nauwkeurigheid behouden blijft.

De tweede toegepaste techniek is magnitude-gebaseerde *pruning*. Deze techniek houdt in dat de gewichten met de laagste waarde gedurende het proces de waarde nul krijgen toegewezen. Alleen bij 25%-uitdunning leidt dit niet tot vermindering van de opslaggrootte. Bij de meest intense uitdunning, 90%, maakt deze techniek het WISDM-model meer dan vijf keer kleiner en het mBrain-model twee keer kleiner. De conclusie is dat hoe hoger de procentuele pruning/uitdunning, hoe kleiner de modelomvang. Daarnaast is er geen verandering in inferentietijd waargenomen. Echter, de nauwkeurigheid van het model neemt ook af. Hierbij geldt: hoe groter de procentuele uitdunning, hoe groter het verlies aan nauwkeurigheid. Bij het WISDM-model ligt het breekpunt voor stabiele nauwkeurigheid na 50%-uitdunning en het mBrain-model na 75% uitdunning. Uit analyse blijkt dat *pruning* bij voldoende toepassing de omvang van het model en de inferentietijd niet verbetert voor 1D-convoluties, maar wel ten koste gaat van de nauwkeurigheid.

De derde onderzochte en toegepaste techniek is post-training kwantisatie. Deze techniek representeert de gewichten van het model met datatypes die minder bits in beslag nemen. Er werd onderzoek gedaan naar de conversie naar twee datatypes die minder bits gebruiken: float16 en uint8. Daarnaast is ook een andere vorm onderzocht, namelijk TensorFlow's standaard optimalisator, die naast dynamische kwantisatie ook zorgt voor activatiefusie en het verwijderen van overbodige operaties. Float16-kwantisatie presteerde het minst goed vergeleken met de andere twee methoden. Ondanks dat de modelgrootte verkleind is met een factor van 1,5-2, waren er geen verbeteringen merkbaar in de inferentietijd. De reden hiervoor is dat het model op de smartphone niet geladen kon worden met float16-waarden, maar alleen met float32-waarden.

De uint8-kwantisatie toonde sterke resultaten voor de modelgrootte. Het WISDM-model daalde in grootte met een factor van 3,4 en het mBrain-model met een factor van 1,8. Deze voordelen gaan ten koste van de nauwkeurigheid van het model. De nauwkeurigheid daalt met 1,6% voor het WISDM-model. Bij het mBrain-model verbeterde de accuraatheid. Dit omdat het vaker de meerderheidsklassen voorspelden en nooit meer de minderheidsklasse ‘liggen’ voorspelde. De accuraatheid geeft in dit geval dus een vertekend beeld.

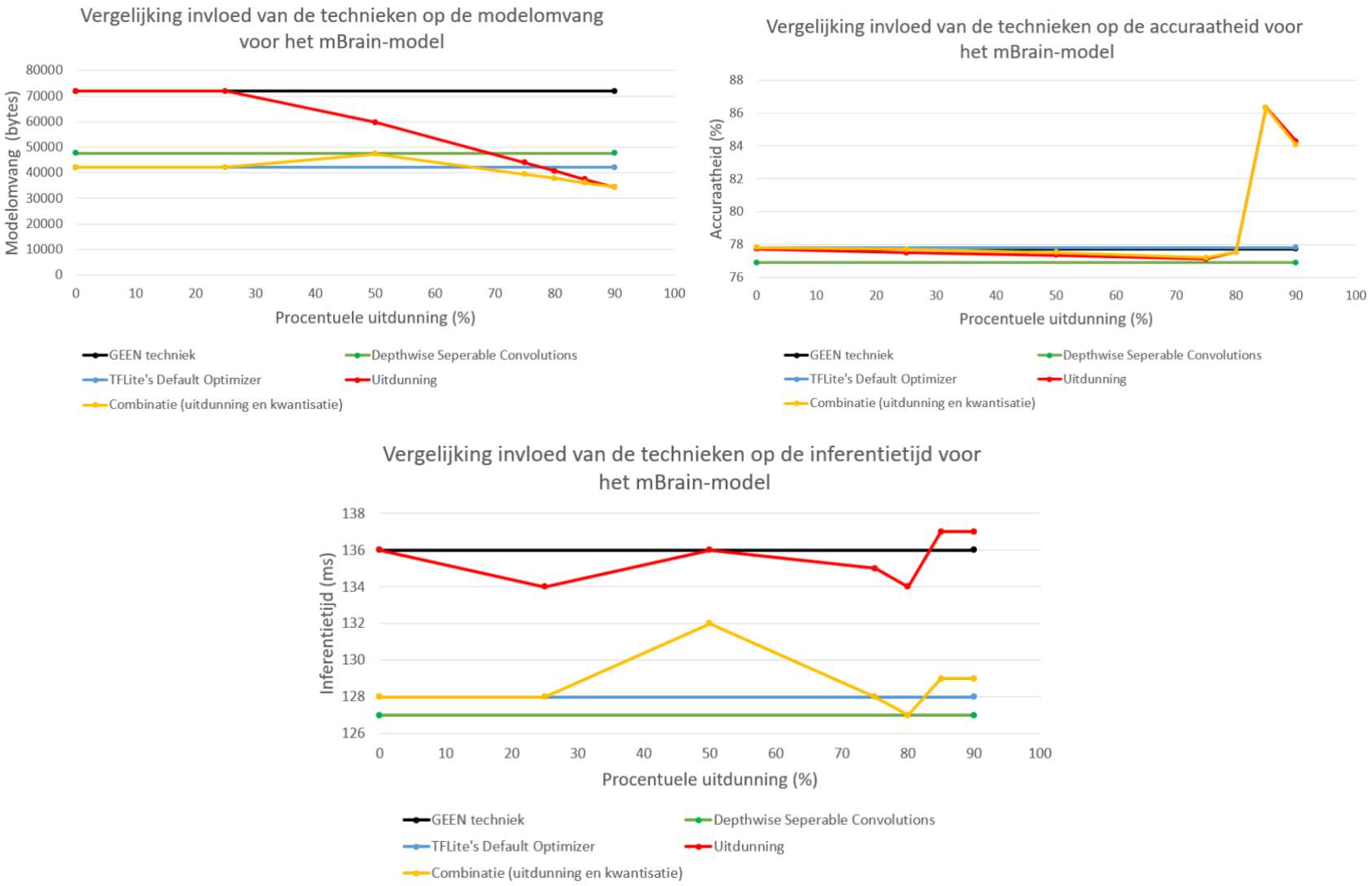
De laatste vorm van post-training kwantisatie die werd onderzocht, is *TensorFlow's default optimalisator*. Deze ingebouwde optimalisator voert naast dynamische kwantisatie ook activatiefusie uit en verwijdert onbelangrijke operaties. Door het toepassen van deze drie functies werden zowel de inferentietijd als de grootte van het model verkleind. Een reductie van factor 3,55 en 1,7 werd behaald, respectievelijk voor het WISDM-model en het mBrain-model. Voor het WISDM-model behaalde het dus een licht betere compressie en voor het mBrain-model maar een licht mindere score in vergelijking met uint8-kwantisatie. De inferentietijd verbeterde met factor 1,18 voor het WISDM-model. Voor het mBrain-model was de inferentietijd 1,07 keer sneller.

Tot slot werd een hybride van *pruning* en post-training kwantisatie toegepast. Hierbij werd gekozen voor *TensorFlow‘s default optimalisator* als kwantisatiemethode omdat deze het meest stabiel was voor beide modellen. Vanaf voldoende uitdunning/*pruning* versterken beide

technieken elkaar. Voor de compressie van het model versterken beide technieken elkaar vanaf 75%-uitdunning. Echter versterkt ook (ongewenst) het verlies aan nauwkeurigheid.

Daarnaast is er een vergelijking gemaakt met resultaten van soortgelijke onderzoeken waarin dezelfde optimalisatietechnieken zijn toegepast. Over het algemeen werden vergelijkbare scores behaald. Het is echter belangrijk om aandacht te besteden aan het aantal trainbare parameters in elk model. Het mBrain-model heeft het laagste aantal trainbare parameters, wat resulteert in een minder uitgesproken factoriale vermindering bij het toepassen van de optimalisatietechnieken, in vergelijking met het WISDM-model of het model dat gebruikt werd in de studie van T. Zebin et al. [23]. Bovendien werd in de andere onderzoeken geen specifieke aandacht besteed aan de inferentietijd van deze technieken. In deze scriptie is er echter wel uitgebreid aandacht besteed aan dit aspect.

Op de vraag welke optimalisatietechniek het best toegepast kan worden op het model voor de mBrain-studie, is er geen eenduidig antwoord. Het is telkens een afweging die gemaakt moet worden tussen de gebruikte bronnen en het verlies aan nauwkeurigheid van het model. Op **Figuur 32** wordt een vergelijking gemaakt van alle onderzochte technieken op de drie verschillende meetrieken. Wanneer de grootte van het model de belangrijkste factor is, zijn de DSC's of de TensorFlow's Default Optimizer de beste oplossing zonder aanzienlijke vermindering van accuraatheid. Hetzelfde geldt voor de inferentietijd. DSC's scoort iets beter voor de inferentietijd en de TensorFlow's Default Optimizer lichtelijk beter voor de opslagomvang. Pruning verbetert enkel de modelgrootte en vanaf te veel uitdunning daalt de accuraatheid aanzienlijk. Daarom hangt de beste methode af van de specifieke toepassing en het specifieke apparaat waarop het model wordt gebruikt. Over het algemeen zijn DSC's en TensorFlow's Default Optimizer de meest gunstige technieken voor het mBrain model.



Figuur 32: *Vergelijking van de invloed van de toegepaste optimalisatietechnieken op het mBrain-model.*

6. Cloud VS. Edge

Deze sectie presenteert een vergelijkend onderzoek tussen een *cloud*-architectuur en een *edge*-architectuur, met een specifieke focus op hun toepassing in de context van *Human Activity Recognition* (HAR) binnen de mBrain-studie. Eerst worden beide architecturen geïntroduceerd, gevolgd door uitleg over de testmethodiek ontworpen om de impact van deze architecturen op de batterijconsumptie en inferentietijd te meten. Daarna worden de resultaten van dit onderzoek gepresenteerd. Tot slot wordt er een conclusie getrokken over welke architectuur het meest geschikt is voor deze specifieke toepassing.

6.1. Architecturen

6.1.1 Cloud architectuur

Ondanks het feit dat er tot 2008 geen eenduidige definitie voor *cloud computing* bestond [45], streven onderzoekers nog altijd naar een uniforme formulering. Het *National Institute of Standards and Technology* (NIST) [46] definieert cloud *computing* als een architectuur die altijd en overal toegang biedt tot een gedeeld netwerk van configureerbare computerbronnen. Deze bronnen kunnen snel worden geleverd en vrijgegeven met minimale interactie of beheerinspanning van de dienstverlener. *Cloud computing* is dus een methode voor het aanbieden van IT-diensten, waarbij gebruikers via het internet toegang hebben tot beschikbare bronnen zoals *servers*, opslag en applicaties. Deze diensten worden onder andere aangeboden door providers zoals Amazon Web Services, Microsoft Azure en Google Cloud.

Bij *cloud computing* betalen gebruikers uitsluitend voor de benodigde bronnen, zoals opslag en rekenkracht, waardoor investeringen in kostbare hardware en software overbodig zijn. De flexibiliteit van deze dienst draagt bij aan haar aantrekkelijkheid: het is mogelijk om tegen betaling extra bronnen te verkrijgen, maar net zo eenvoudig om de bronnen te verminderen en zo kosten te besparen.

Cloud computing kan worden ingedeeld op basis van dienstmodellen en implementatietypen. Deze classificatie helpt gebruikers bij het kiezen van het meest geschikte implementatietype en dienstmodel voor hun specifieke behoeften. Het implementatiemodel bepaalt het niveau van beschikbaarheid, terwijl het dienstmodel verschillende servicetypes onderscheidt.

Ongeacht het gekozen implementatiemodel (openbaar, privé of hybride), kan cloud uiteenlopende dienstmodellen aanbieden aan klanten. In het algemeen worden er drie soorten diensten onderscheiden: *Platform as a Service* (PaaS), *Infrastructure as a Service* (IaaS) en *Software as a Service* (SaaS). Hoewel deze drie soorten het meest gebruikelijk zijn, winnen andere dienstmodellen, zoals *Memory as a Service*, *Security as a Service*, *Data as a Service* en *Test Environment as a Service*, de laatste jaren aan populariteit [47].

Software as a Service (SaaS) staat bekend als het meest abstracte dienstmodel. De gebruiker heeft toegang tot een applicatie die wordt beheerd en ontworpen door de *cloud provider*. De software wordt via internet beschikbaar gesteld, waardoor installatie op de computer van de gebruiker overbodig is.

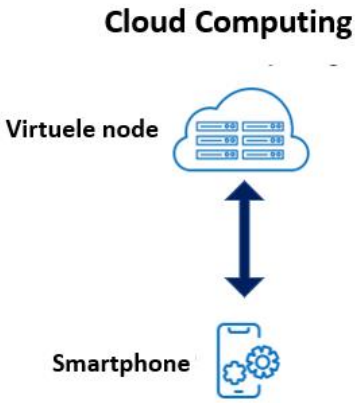
Platform as a Service (PaaS) biedt een platform voor het beheren, ontwikkelen en uitvoeren van applicaties met behulp van *cloud*-gebaseerde *tools* en diensten.

Infrastructure as a Service (IaaS) biedt besturingssystemen, infrastructuur en virtualisatie als dienst aan. Hierbij draagt de gebruiker de verantwoordelijkheid voor het beheer van alles, variërend van dataopslag tot aan de applicaties. IaaS kan worden gedefinieerd als het aanbieden van hardware, zoals servers en dataopslag, in combinatie met bijbehorende software, zoals besturingssystemen en virtualisatietechnologie.

Voor *Human Activity Recognition* (HAR) in de mBrain-studie wordt een virtuele node in de *cloud* gebruikt waar alle accelerometerdata naar worden verstuurd en verwerkt, waarna de inferentie terug naar de smartphone wordt gezonden.

Figuur 33 toont hoe de *cloud*-architectuur wordt toegepast in de context van *Human Activity Recognition* (HAR) in de mBrain-studie. Het proces omvat de volgende stappen:

1. De sensoren op de smartphone verzamelen diverse soorten data.
2. De Android-applicatie selecteert de gegevens van de accelerometer.
3. Deze onbewerkte gegevens worden gedurende een bepaalde periode op de smartphone opgeslagen, doorgaans twee minuten indien er een internetconnectie aanwezig is.
4. Mits een internetverbinding stuurt de smartphone deze data naar de virtuele node via zijn openbaar IP-adres.
5. In de cloud verwerkt het *machine learning* model de volledige onbewerkte data doorgaans om de vijf minuten.
6. Dit model voert vervolgens inferenties uit op de uitgevoerde activiteiten.
7. De voorspellingen worden teruggestuurd naar de smartphone en zijn zichtbaar voor de eindgebruiker in de applicatie.

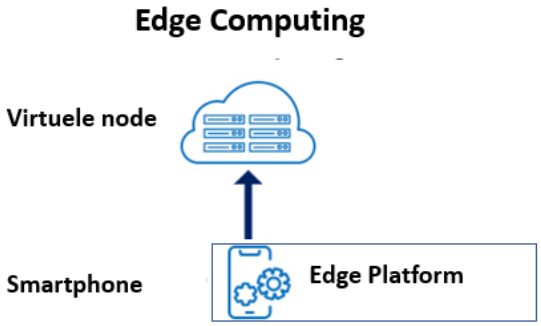


***Figuur 33**: Cloud architectuur voor HAR van de mBrain-studie.*

Voor deze architectuur is een internetverbinding vereist om voorspellingen te maken en ze vervolgens weer te geven in de applicatie. Hierbij worden grote hoeveelheden data periodiek door verschillende gebruikers naar de *cloud* gestuurd. In andere woorden, alle data wordt in de *cloud* gecentraliseerd. Het regelmatig verzenden van grote datavolumes vergt bandbreedte.

6.1.2 Edge architectuur

In een *edge*-architectuur vindt dataverwerking en analyse plaats aan de rand van het netwerk, dicht bij de bronnen van de gegevens, zoals sensoren of andere *Internet of Things* (IoT) apparaten. In plaats van gegevens naar een centrale server te sturen, voert *edge computing* de verwerking uit op lokale apparaten zoals smartphones, *wearables* of *gateways*. Doordat de data niet via het internet gestuurd moet worden voor verwerking, resulteert dit in een lagere behoefte aan bandbreedte.



***Figuur 34**: Voorgestelde edge architectuur voor HAR voor de mBrain-studie.*

In deze scriptie wordt een *edge*-architectuur gecreëerd. Hierbij wordt de data lokaal op de smartphone verwerkt, wat ook de bron van de accelerometerwaarden is. Voor de realisatie van deze architectuur is een Android-applicatie ontwikkeld die in staat is om de gegevens op te vragen, te verwerken en te visualiseren. De details van deze applicatie werden eerder behandeld in **Sectie 4,** omtrent het evaluatiekader. In deze architectuur draait het *machine learning* model op de smartphone zelf. **Figuur 34** illustreert de voorgestelde en toegepaste *edge*-architectuur. In vergelijking met de *cloud*-architectuur, zoals weergegeven in **Figuur 33**, is te zien dat de pijl tussen de smartphone en de virtuele node slechts in één richting wijst. In deze *edge*-architectuur hoeven enkel de voorspellingen van het *machine learning* model naar de virtuele node te worden teruggestuurd, niet de volledige accelerometerdata.

In deze *edge*-architectuur worden de volgende stappen doorlopen:

1. De sensoren op de smartphone verzamelen diverse soorten data.
2. De Android-applicatie vraagt alleen de waarden van de accelerometer op.
3. Deze onbewerkte gegevens worden gedurende een bepaalde periode opgeslagen op de smartphone.
4. Het *machine learning* model op de smartphone verwerkt de volledige ruwe data via de Android-applicatie.
5. Dit model voert vervolgens inferenties uit voor de uitgevoerde activiteiten.
6. Als laatste stap kunnen de inferenties na een bepaalde tijd naar een virtuele node worden gestuurd, waar deze voorspellingen kunnen worden opgeslagen.

Op deze manier kan op een later moment worden teruggekeken naar wanneer welke activiteiten zijn uitgevoerd. Omdat alleen deze voorspellingen periodiek naar een centrale node worden gestuurd, en niet de volledige gegevens van de accelerometer, wordt de behoefte aan bandbreedte verminderd.

6.2. Testmethode

In dit onderzoek wordt de invloed van beide architecturen, *cloud* en *edge*, op het batterijgebruik en de inferentietijd van de smartphone onderzocht. Het evaluatiekader is eerder beschreven in **Sectie 4**, waarin de Android-applicaties voor zowel de *cloud*- als de *edge*-architectuur in detail zijn besproken. Voor beide architecturen worden drie verschillende scenario's geanalyseerd.

Het eerste scenario houdt in dat inferenties continu worden uitgevoerd. In de *cloud*-architectuur wordt de data gedurende zes seconden opgeslagen op de smartphone, wat overeenkomt met één sample. Na deze periode wordt de data via een *POST*-bericht naar de virtuele node gestuurd die beschikbaar is op zijn openbare IP-adres en zijn eindpunt '/upload', dat luistert op poort 8080. In dit geval wordt het bericht verstuurd naar: *http://193.190.127.213:8080/upload*. Op deze virtuele node bevindt zich onder andere het *machine learning* model van de mBrain-studie. Zodra de inferentie is voltooid, wordt deze als reactiebericht naar de Android-applicatie gestuurd, waarna de voorspelling lokaal in een bestand wordt opgeslagen. Voor de *edge*-architectuur geldt hetzelfde, behalve dat de data niet naar de virtuele node wordt gestuurd en de verwerking lokaal plaatsvindt. Het model dat gebruikt wordt voor beide architecturen is het geconverteerde mBrain-model naar TFLite zonder optimalisatietechnieken. Zodra de verwerking is voltooid en de inferentie is uitgevoerd, wordt het bestand met de accelerometerdata verwijderd. Dit geldt voor beide architecturen.

Het tweede scenario is vergelijkbaar met het eerste scenario voor beide architecturen. In dit geval wordt de data echter eerst vijf minuten opgeslagen voordat deze wordt doorgestuurd naar de virtuele node of lokaal wordt verwerkt. Het derde scenario is een uitbreiding van de vorige twee, waarbij hetzelfde proces wordt gevolgd, maar dan over een tijdspanne van één uur.

Het energieverbruik wordt gemonitord met behulp van een Android-applicatie, BatteryGuru, die beschikbaar is in de Google Store. Deze applicatie geeft gedetailleerde informatie over het batterijverbruik van de smartphone waarop het is geïnstalleerd. Het toont een overzicht in de vorm van een grafiek van het batterijverbruik van de afgelopen minuut, tien minuten, een uur of

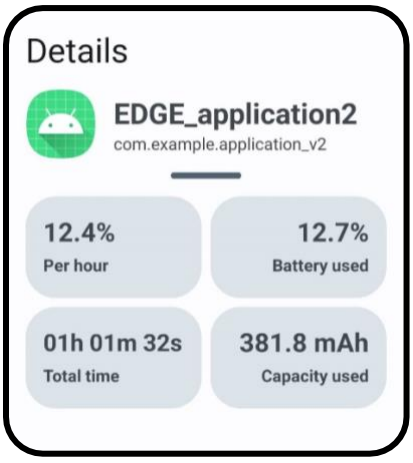
zes uur. In deze grafiek wordt het verbruik in milliampère-uur (mAh) over een bepaalde tijd weergegeven. De applicatie biedt ook advies over hoe de batterij het meest efficiënt kan worden gebruikt, hoe snel de smartphone oplaadt en andere kenmerken van de batterijstatus.

Voor het testen van het batterijverbruik wordt de *App Usage*-functie van BatteryGuru gebruikt. Deze functie geeft een gedetailleerde weergave van hoeveel procent van de batterij en hoeveel mAh er wordt verbruikt door de verschillende applicaties. De te monitoren applicatie moet hiervoor wel actief zijn. Op deze manier kan het batterijverbruik per architectuur worden bekeken zonder de taken die op de achtergrond draaien mee te tellen.

Om de *App Usage*-functie te gebruiken, moet er toestemming worden gegeven om deze op de smartphone te laten werken. De permissie kan gegeven worden via het volgende commando:

- *adb -d shell pm grant com.paget96.batteryguru android.permission.DUMP*

Op deze manier wordt de *android.permission.DUMP* gegeven aan de desbetreffende applicatie. Door deze permissie kan de app, BatteryGuru, informatie over de systeemdiagnostieken en statusinformatie van het toestel verkrijgen.



Figuur 35: *Voorbeeld 'App Usage'-functie van BatteryGuru.*

Nadat deze stappen zijn doorlopen en de benodigde toestemmingen zijn verleend, kunnen beide applicaties van de architecturen gemonitord worden. Op **Figuur 35** wordt een voorbeeld getoond van hoe metingen worden weergegeven. Deze afbeelding is afkomstig van een uitgevoerde meting. Het toont een overzicht van hoeveel mAh er wordt verbruikt door de applicatie, hoe lang de meting duurde en het percentage van het verbruik plus het percentage van het verbruik per uur. Elke meting die werd uitgevoerd, duurde ongeveer één uur, waarna de waarden werden omgezet naar het werkelijke verbruik per uur.

Deze metingen worden uitgevoerd op hetzelfde apparaat dat is beschreven in de sectie over het evaluatiekader, **Sectie 2**. De smartphone wordt telkens tot 100% opgeladen voor het begin van de meting. Bovendien wordt de helderheid voor elke test op maximaal gezet en werd de adaptieve helderheid uitgeschakeld. De smartphone was tijdens elke meting verbonden met hetzelfde netwerk. Verder worden alleen de *BatteryGuru*-applicatie en de testapplicatie op de smartphone geïnstalleerd, naast de standaard door Android geleverde applicaties. Op deze manier wordt de verstoring door achtergrondtaken tot een minimum beperkt. De metingen worden op dezelfde manier uitgevoerd voor de *edge*- en de *cloud*-architectuur. Voor beide architecturen wordt gebruik gemaakt van hetzelfde TFLite-geconverteerde mBrain-model. Op deze manier kan een nauwkeurige vergelijking worden gemaakt tussen beide architecturen.

Inferentietijd

Cloud omgeving

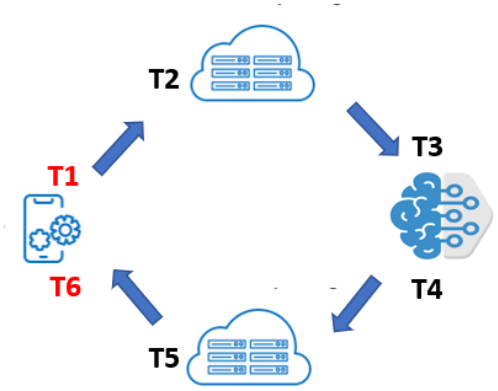
De methodologie voor het berekenen van de inferentietijden in de *cloud* gebruikt metingen die worden uitgevoerd door middel van *loggen* in de virtuele node en binnen de Android-applicatie. Deze procedure omvat zes verschillende momenten waarop de tijd wordt geregistreerd. Door deze geregistreerde tijden vervolgens te vergelijken, kunnen verschillende fasen in het proces worden geïdentificeerd en kan de duur van elk segment worden vastgesteld. Drie specifieke segmenten worden in dit proces onderscheiden.

1. **Uplink- en Downlinktijd**: De uplinktijd verwijst naar de benodigde tijd om het bestand met accelerometerdata te verzenden van de smartphone naar de virtuele node. De downlinktijd is daarentegen de benodigde tijd om het inferentiebestand van de virtuele te zenden naar de smartphone.
2. **Inferentietijd van het model**: Dit segment vertegenwoordigt de tijd die het model in de *cloud* nodig heeft om de voorspellingen uit te voeren.
3. **Overige:** Dit segment omvat alle overige tijdsfactoren, zoals de voorverwerking van gegevens, het opslaan van de inferentie in een CSV-bestand na de verwerking, het controleren van het correcte bestandstype, het verwijderen van NULL-elementen uit het accelerometer databestand, etc.

De keuze om de uplink- en downlinktijd samen te voegen komt voort uit de beperkingen van de meetmethodologie. Deze werkt met tijdsmarkeringen die gedurende het hele proces worden vastgelegd en vervolgens vergeleken om de duur van elk processegment te bepalen. Een uitdaging hierbij is dat de Android-applicatie op de smartphone is ontwikkeld in Kotlin en draait op het Android-toestel, terwijl de code voor de Flask-applicatie op de virtuele node is geschreven in Python. Hierdoor worden twee verschillende bibliotheken gebruikt om de huidige tijd in milliseconden op te vragen: *System.currentTimeMillis()* in de Android-applicatie en *int(time.time() * 1000)* in de virtuele node. Een synchronisatieprobleem ontstaat omdat deze klokken verschillende waarden hebben. Ondanks dit verschil slechts een aantal honderd milliseconden

bedraagt, kan het leiden tot situaties waarin de ontvangen tijd voor het accelerometer databestand eerder is dan de verzendtijd, wat onmogelijk is. Er zijn verschillende oplossingen onderzocht, zoals het gebruik van het *Network Time Protocol* (NTP), maar zelfs na implementatie was de synchronisatie niet perfect. Daarom worden de uplink- en downlinktijden samen genomen. De *round trip time* (RTT) kan gemeten worden in de applicatieomgeving, vervolgens kan ook de volledige tijd in de *cloud* gemeten worden in de cloud omgeving. Door deze twee tijden af te trekken kan de uplink-en downlinktijd bepaald worden.

Zoals eerder aangegeven, wordt er gedurende het gehele proces gebruik gemaakt van tijdstempels om de duur van elk processegment te berekenen. Er zijn zes verschillende tijdstempels in gebruik, waarvan twee (T1 en T6) zich bevinden binnen de Android-omgeving en de resterende vier (T2, T3, T4, en T5) zich bevinden binnen de virtuele node-omgeving. Deze verdeling wordt geïllustreerd in **Figuur 36**, waarbij de tijdstempels in de Android-omgeving zijn gemarkeerd in het rood en die in de *cloud*-omgeving in het zwart.



***Figuur 36**: Visuele voorstelling van de verschillende tijdstempels in het cloud proces.*

- T1: Gemeten net voor het verzenden van het POST-bericht naar de virtuele node.
- T2: Gemeten zodra een POST-bericht wordt ontvangen.
- T3: Gemeten net voordat de accelerometerdata in het model wordt geladen.
- T4: Gemeten zodra alle voorspellingen zijn voltooid.
- T5: Gemeten net voor het verzenden van het response-bericht.
- T6: Gemeten zodra het inferentiebestand door de applicatie is ontvangen.

Deze tijdstempels worden opgeslagen in een CSV-bestand, zowel op de virtuele node als op de Android-applicatie. Na het uitvoeren van de metingen worden deze bestanden overgebracht naar een lokale computer voor verdere analyse. Met behulp van deze tijdstempels kan de duur van elk processegment worden berekend. In **Tabel 21** wordt een overzicht gegeven van hoe de tijd van de verschillende processegmenten worden berekend op basis van de tijdstempels.

***Tabel 21**: Overzicht van hoe de verschillende onderdelen van de totale inferentietijd van hoe cloud proces wordt berekend.*

	Tekst	**Formule**
Uplink- en downlinktijd	Volledige *Round Trip Time* (RTT) – de tijd in de cloud.	(T6-T1) – (T5-T2)
Inferentietijd van het model	Tijd na inferentie – tijd voor inferentie	(T4 – T3)
Andere	Tijd in de *cloud* – inferentietijd van het model	(T5-T2) – (T4-T3)

Edge omgeving

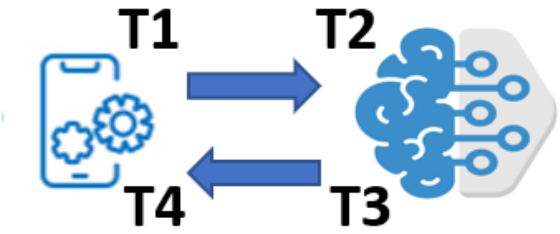
De methodologie voor het berekenen van de inferentietijden voor de *edge*-architectuur vereist ook metingen die worden uitgevoerd door middel van loggen binnen de Android-applicatie. Deze procedure omvat vier verschillende momenten waarop de tijd wordt geregistreerd. Door deze geregistreerde tijden vervolgens te vergelijken, kunnen verschillende fasen in het proces worden geïdentificeerd en kan de duur van elk segment worden vastgesteld. Drie specifieke segmenten worden in dit proces onderscheiden.

1. **Voorverwerking**: Deze fase omvat de tijd die nodig is om de data in te lezen. De accelerometerdata wordt lokaal opgeslagen in de vorm van een CSV-bestand. Dit bestand wordt onderverdeeld in batches van 192 datasamples voor het mBrain-model. Daarnaast moet deze data worden omgezet naar een *ByteBuffer*-datatype.
2. **Inferentietijd van het model**: Dit is de tijd die het model nodig heeft om inferentie te doen op de smartphone zelf.
3. **Naverwerking**: Dit is de tijd die nodig is om de inferenties lokaal op te slaan.

Voor deze architectuur wordt ook gebruik gemaakt van tijdstempels om de verschillende delen te meten. Hierbij worden vier verschillende tijdstempels gebruikt:

- T1: Gemeten net voor het uitlezen van de accelerometerdata.
- T2: Gemeten voor elke *sample* wordt ingeladen in het *machine learning* model.
- T3: Gemeten na elke inferentie.
- T4: Gemeten na opslaan van inferentie.

Figuur 37 visualiseert de plaatsing van deze tijdstempels doorheen het proces. **Tabel 22** geeft een overzicht van hoe elk deel vervolgens berekend wordt.



***Figuur 37**: Visuele voorstelling van de verschillende tijdstempels in het edge proces.*

***Tabel 22**: Overzicht van hoe de verschillende onderdelen van de totale inferentietijd van het edge-proces wordt berekend.*

	Tekst	**Formule**
Voorverwerking	Tijd voor inferentie – tijd voor voorwerking	(T2 - T1)
Inferentietijd van het model	Tijd na inferentie – tijd voor inferentie	(T3– T2)
Naverwerking	Tijd na opslaan inferenties – tijd na inferentie	(T4-T3)

6.3. Resultaten

In deze subsectie worden de resultaten van de metingen geanalyseerd die uitgevoerd zijn om de batterijconsumptie en inferentietijd van de *cloud*- en de *edge*-architectuur te vergelijken. Bijkomend onderzoek wordt gedaan voor de *cloud*-infrastructuur waarbij de metingen worden uitgevoerd op een ander netwerk. Op deze manier wordt de invloed van het internetnetwerk op de *cloud*-architectuur in kaart gebracht. Tot slot wordt een conclusie beschreven die de resultaten nogmaals samenvat.

6.3.1. Batterijconsumptie

Zoals eerder werd vermeld in **Sectie 4**, met betrekking tot het evaluatiekader, zijn er twee applicaties gemaakt. Op deze manier is er per architectuur een aparte applicatie voorzien. De eerste fase van beide applicaties is hetzelfde, namelijk het opslaan van de accelerometerdata. De andere fase is verschillend voor elke architectuur, namelijk het verwerken van de data. Aangezien de eerste fase dezelfde is, wordt een eerste meting gedaan op de applicatie waarbij enkel deze fase plaatsvindt. Op deze manier wordt een basis gelegd van hoeveel de applicatie verbruikt zonder de data lokaal te verwerken, of door te sturen naar een virtuele node.

Net zoals elke meting die zal volgen, werd deze meting gedurende een uur uitgevoerd. Wanneer de applicatie enkel en alleen data opslaat en niet aan inferentie doet, verbruikte de applicatie 357.84 mAh. Op deze basis wordt verder gewerkt en hiernaar wordt vaak terugverwezen.

6.3.1.1. Batterijconsumptie *cloud*-architectuur

Op **Tabel 23** worden de resultaten voor de *cloud*-architectuur voor alle drie scenario's samengevat, samen met het verbruik van de basisapplicatie. In deze tabel wordt het mAh-verbruik van de applicatie onderscheiden van het totale verbruik in mAh gedurende de meting. Hetzelfde geldt voor het batterijpercentage. Het is duidelijk dat het belangrijk is om de achtergrondtaken uit te filteren en niet zomaar te kijken naar de vermindering van het batterijpercentage van de smartphone. Anders zou er geen verschil opgemerkt zijn tussen het 5-minuten scenario en het 1-uur scenario. De gedetailleerde informatie die BatteryGuru, **Sectie 6.2.1**, voorziet per applicatie specifiek, is dus van cruciaal belang. Op deze manier kan een gedetailleerd onderscheid gemaakt worden tussen de drie verschillende scenario's.

Daarnaast is het ook belangrijk om te vermelden dat de smartphone gedurende de volledige meting aanstond op de hoogste helderheid. Dit is de reden waarom het basismodel en alle andere modellen aanzienlijk veel energie verbruiken. Wanneer de gsm niet voortdurend open zou zijn op maximale helderheid zou dit een stuk minder zijn. Daarom wordt in volgende delen ook telkens het verbruik van het basismodel vermeld.

***Tabel 23**: Batterijconsumptie cloud architectuur voor de drie verschillende scenario's en de basis applicatie.*

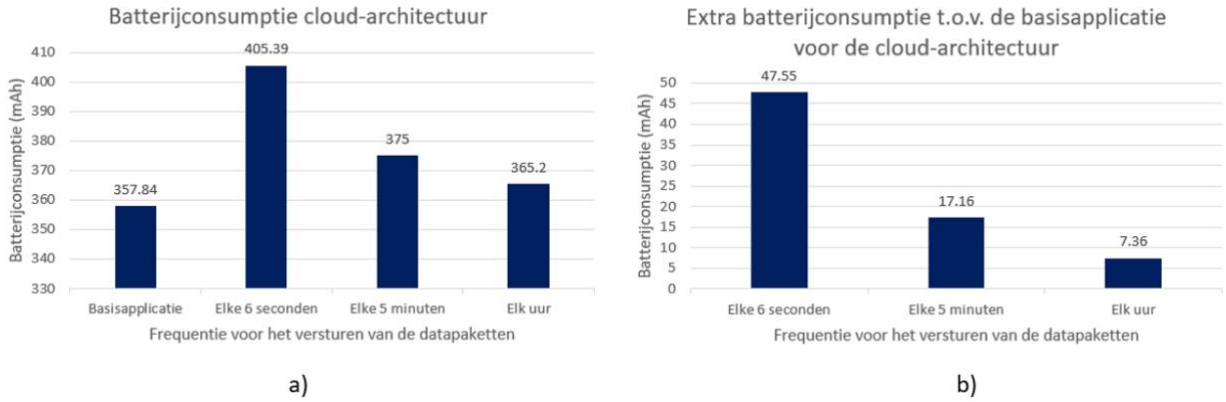
Scenario	mAh verbruikt door applicatie	Totale mAh verbruikt	Verbruik in batterijpercentage door applicatie	Totaal verbruik in batterijpercentage
Geen inferentie	357.84	450	11.91%	15%
6 seconden	405.39	510	13.51%	17%
5 minuten	375.0	480	12.50%	16%
1 uur	365.20	480	12.17%	16%

De meest frequente verbinding tussen de smartphone en de virtuele node verbruikt het meeste energie. Het uitwisselen van bestanden om de zes seconden resulteert in een verbruik van 405.39 mAh. Wanneer het verbruik van de basisapplicatie hiervan wordt afgetrokken, blijft er nog 47.55 mAh over. In dit scenario verzendt zowel de applicatie als de virtuele node elk 600 bestanden per

uur. Ondanks dat de datapakketten die over het internet verstuurd worden relatief klein zijn, zorgt dit toch voor een aanzienlijke stijging in verbruik.

Wanneer er gekeken wordt naar het tweede scenario, waarbij de databestanden elke vijf minuten worden uitgewisseld, resulteert dit in een aanzienlijke afname ten opzichte van het eerste scenario. Hierbij zijn de datapakketten die over het internet gestuurd worden weliswaar een stuk groter, maar worden er per uur slechts 12 bestanden verstuurd en ontvangen door de smartphone. In dit geval wordt er 375.0 mAh verbruikt door de applicatie, wat 1.08 keer minder is dan het eerste scenario. Wanneer het verbruik van het basismodel hiervan wordt afgetrokken, blijft er slechts 17.16 mAh over. Dit betekent dat de dataverwerking voor dit scenario 17.16 mAh kost. Dit is aanzienlijk minder dan het eerste scenario dat 47.55 mAh kost. De keuze voor deze methode is daarom 2.77 keer voordeliger dan het eerste scenario.

Het derde scenario stuurt slechts één bestand door per uur. Dit bestand is wel veruit het grootste. Dit scenario is het meest voordelige scenario met betrekking tot het batterijverbruik van de smartphone. Hoe minder frequent de datapakketten worden verstuurd, ondanks dat ze groter zijn, hoe minder batterij hiervoor gebruikt moet worden. Het verbruikt slechts 7.36 mAh meer dan de basisapplicatie en is op deze manier bijna 2.33 keer zo energiezuinig als het scenario waarbij de data elke 5 minuten verstuurd werd. Wanneer vergeleken wordt met het scenario waarbij elke 6 seconden de data werd verstuurd, is dit maar liefst 6.46 keer energiezuiniger.



Figuur 38: *a) Batterijconsumptie cloud architectuur van de basisapplicatie en de drie verschillende scenario's. b) Extra batterijconsumptie van de drie scenario's t.o.v. de basisapplicatie.*

6.3.1.2. Batterijconsumptie edge-architectuur

Tabel 24 vat de resultaten voor de *edge*-architectuur voor dezelfde drie scenario's samen, inclusief het verbruik van de basisapplicatie. In dit geval is er geen sprake van bestanden die gestuurd dienen te worden over het internet.

***Tabel 24**: Batterijconsumptie edge architectuur voor de drie verschillende scenario's en de basisapplicatie.*

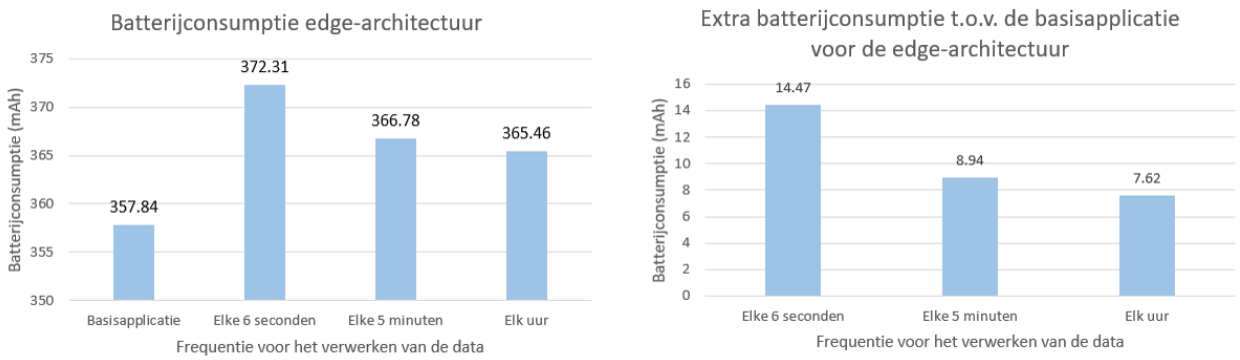
Scenario	mAh verbruikt door applicatie	Totale mAh verbruikt	Verbruik in batterijpercentage door applicatie	Totaal verbruik in batterijpercentage
Geen inferentie	357.84	450	11.91%	15%
6 seconden	372.31	480	12.41%	16%
5 minuten	366.78	480	12.23%	16%
1 uur	365.46	450	12.18%	15%

Het *edge*-scenario dat de meeste energie verbruikt, is het scenario waarbij het model elke zes seconden voorspellingen uitvoert. In dit geval wordt elke zes seconden het model geladen met 192 datasamples, waarna het een voorspelling doet en deze vervolgens opslaat in een bestand. Het verbruikt in totaal 372.31 mAh, wat 14.47 mAh meer is dan de basisapplicatie. In totaal worden hierbij de bestanden met accelerometerdata in totaal 600 keer gelezen en worden er 600 keer opnieuw voorspellingen toegevoegd aan het bestand met de predicties.

Het scenario waarbij het model elke vijf minuten wordt geladen met de opgeslagen accelerometerdata, verbruikt minder energie dan het eerste scenario. Het kost 8.94 mAh extra ten opzichte van het basismodel. Op deze manier is dit scenario 1.62 keer goedkoper in energieverbruik dan het eerste scenario. In dit geval wordt er slechts 12 keer gelezen uit databestanden en slechts 12 keer worden de voorspellingen toegevoegd aan het bestand met de predicties. Deze bestanden bevatten weliswaar meer datapunten dan het eerste scenario.

Bij het derde scenario wordt er slechts één keer per uur, en dus maar één keer tijdens deze meting, gegevens gelezen uit het databestand. Hetzelfde geldt voor het opslaan van de voorspellingen. Dit leidt dan ook tot het meest energie-efficiënte *edge*-scenario, waarbij slechts 7.62 mAh extra wordt gebruikt ten opzichte van de basisapplicatie. Het is op deze manier 1.17 keer goedkoper dan het tweede scenario en maar liefst 1.90 keer efficiënter dan het eerste scenario.

Over het algemeen kan gesteld worden dat hoe frequenter de inferenties lokaal worden uitgevoerd, hoe meer energie dit kost. Een mogelijke reden hiervoor is dat het minder verschillende bestanden moet uitlezen, hoewel het totale aantal regels dat moet worden gelezen hetzelfde blijft. In dit geval worden de bronnen die nodig zijn pas elk uur gebruikt en gewekt door de applicatie. Bij de andere twee scenario's gebeurt dit frequenter, waardoor meer energie wordt verbruikt.



***Figuur 39**: a) Batterijconsumptie cloud architectuur van de basisapplicatie en de drie verschillende scenario's. b) Extra batterijconsumptie van de drie scenario's t.o.v. de basisapplicatie.*

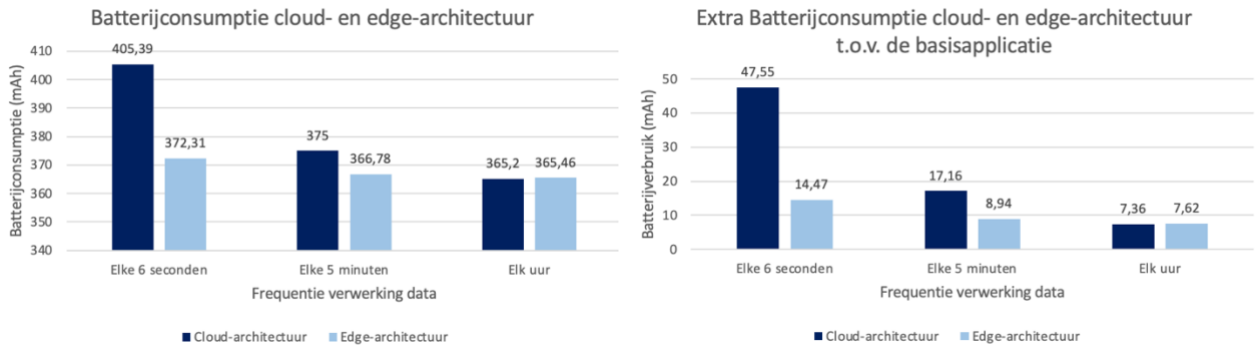
6.3.1.3. Vergelijking energiegebruik architecturen

Tabel 25 presenteert een vergelijking van het energieverbruik voor de *cloud*- (wit) en *edge*-architectuur (grijs) in de drie verschillende scenario's en de basisapplicatie.

***Tabel 25**: Batterijconsumptie cloud (wit) en edge-architectuur (grijs) voor de drie verschillende scenario's en de basisapplicatie*

Scenario	mAh verbruikt door applicatie	Totale mAh verbruikt gedurende meting	Verbruik in batterijpercentage door applicatie	Totaal verbruik in batterijpercentage gedurende meting
Geen inferentie	357.84	450	11.91%	15%
6 seconden	405.39	510	13.51%	17%
6 seconden	372.31	480	12.41%	16%
5 minuten	375.00	480	12.50%	16%
5 minuten	366.78	480	12.23%	16%
1 uur	365.20	480	12.17%	16%
1 uur	365.46	450	12.18%	15%

Voor beide architecturen geldt hetzelfde principe: hoe frequenter de data wordt verwerkt, hoe meer batterij dit kost. Dit gaat echter hand in hand met het feit dat het databestand met de accelerometerdata groter is. Op deze manier wordt er minder efficiënt omgegaan met de opslagcapaciteit van de smartphone.



***Figuur 40**: a) Batterijconsumptie cloud en edge-architectuur van de basisapplicatie en de drie verschillende scenario's. b) Extra batterijconsumptie van de drie scenario's t.o.v. de basisapplicatie.*

Uit de resultaten blijkt dat de *edge*-architectuur voor de eerste twee scenario's het meest efficiënt is voor de batterij van het toestel. Voor het derde scenario is er geen aanzienlijk verschil te merken. Voornamelijk voor het eerste scenario bereikt de *edge*-architectuur een aanzienlijke verbetering ten opzichte van de *cloud*-architectuur. In dat geval zijn de kosten voor het voortdurend verzenden en ontvangen van data over het internet aanzienlijk groot. Het is dus energie-efficiënter om in dit geval de voorspellingen voortdurend uit te voeren op de smartphone. De energie die wordt verbruikt door de bronnen voor het uitvoeren van vermenigvuldigingen en optellingen tijdens inferentie is dus minder dan de energie die wordt gebruikt voor het voortdurend verzenden van de bestanden. De cloud-architectuur gebruikt in dit geval 47.55 mAh extra in vergelijking met de basisapplicatie en de *edge*-architectuur slechts 14.47 mAh, dit is terug te vinden op **Figuur 40b**. De *edge*-architectuur is in dit geval dus ongeveer 3.29 keer efficiënter met betrekking tot het energieverbruik.

Voor het tweede scenario is het verschil tussen de *cloud*- en *edge*-architectuur kleiner, namelijk slechts 8.22 mAh. Bij de *edge*-architectuur wordt het model minder vaak geopend en gebruikt en wordt de data ook minder frequent voorbewerkt in vergelijking met het scenario van zes seconden. Hierdoor worden de bronnen die hiervoor nodig zijn minder vaak opgeroepen, wat leidt tot een vermindering in verbruik. Hetzelfde geldt voor de *cloud*-architectuur, waarbij er 50 keer minder frequent data verstuurd wordt over het internet. Voor dit scenario presteert de *edge*-architectuur iets beter, maar het verschil is kleiner in vergelijking met het eerste scenario.

Bij het derde scenario is het verschil tussen beide architecturen bijna onzichtbaar. Beide scenario's verbruiken slechts een paar mAh meer dan de basisapplicatie. Dit komt omdat de frequentie waarmee de bronnen worden opgeroepen voor het inferentieproces in beide scenario's lager is. Verdere reductie in verbruik door bijvoorbeeld een langere tijdspanne te nemen zal waarschijnlijk geen aanzienlijk verschil maken. Beide architecturen voor dit scenario verbruiken slechts iets meer mAh dan de basisapplicatie en het is onmogelijk dat deze scenario's ooit minder zullen verbruiken dan de basisapplicatie. Daarom is ervoor gekozen om geen langere tijdspannes te onderzoeken.

Over het algemeen wordt geconcludeerd dat de *edge*-architectuur zorgt voor een verbetering in energieverbruik ten opzichte van de *cloud*-architectuur voor de kortere tijdspannes. Hoe langer de tijdspanne, hoe vergelijkbaarder het verbruik van beide architecturen wordt. Van de onderzochte scenario's is er geen waarbij de *edge*-architectuur aanzienlijk meer verbruikte dan de *cloud*-architectuur, maar het omgekeerde is wel het geval.

6.3.2. Inferentietijd

Een andere cruciale metriek die mogelijk varieert tussen de twee architecturen is de inferentietijd. Hierbij verwijst de inferentietijd naar de tijd die nodig is om de voorspellingen zichtbaar te maken voor de gebruikers van de applicatie. In dit gedeelte worden eerst de resultaten van de *cloud*-architectuur voor de drie eerdergenoemde scenario's behandeld. Vervolgens wordt de *edge*-architectuur op dezelfde wijze geanalyseerd. Tenslotte vindt een vergelijking plaats tussen beide architecturen.

6.3.2.1. Inferentietijd cloud architectuur

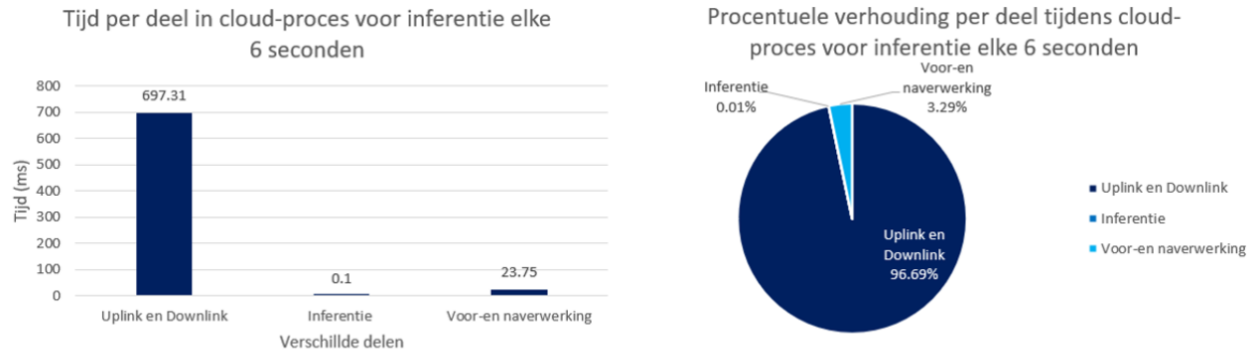
De resultaten die in deze subsectie worden besproken, betreffen de resultaten van de *cloud*-omgeving voor de drie verschillende scenario's. De methode van berekening en de gebruikte tijdstempels zijn reeds beschreven in **Sectie 6.1**.

Tabel 26: *Overzicht van de inferentietijd voor de verschillende delen in het cloud proces voor de drie verschillende scenario's.*

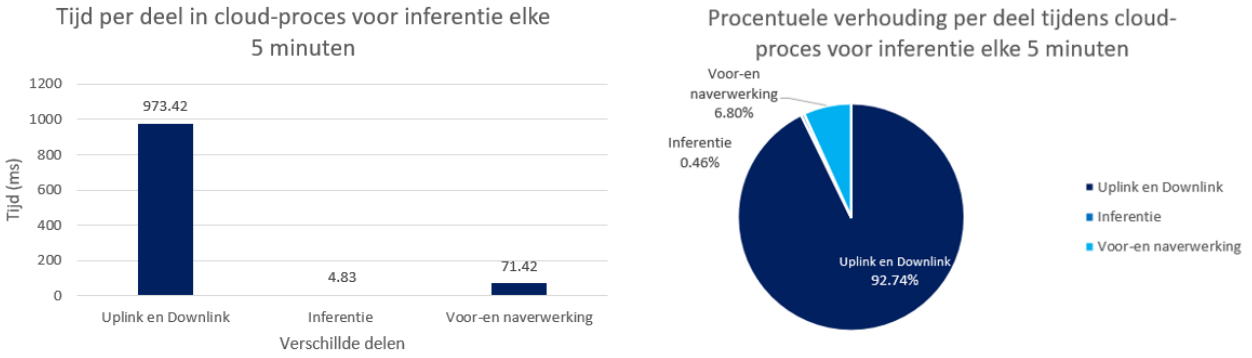
Scenario	Uplink en Downlink (ms)	Inferentietijd van het model (ms)	Andere (ms)	Totale Inferentietijd (ms)
6 seconden	697.31	0.10	23.75	721.16
5 minuten	973.42	4.83	71.42	1049.67
1 uur	4696.0	61.0	638.0	5393.0

Tabel 26 toont de inferentietijden voor de verschillende fasen in het *cloud*-proces voor de drie scenario's. Het is duidelijk dat grotere bestanden resulteren in langere uplink- en downlinktijden. Hetzelfde geldt voor de totale inferentietijd van het model, aangezien er meer voorspellingen worden gemaakt. Deze waarden weerspiegelen de kosten voor het verzenden en verwerken van een bestand van zes seconden, vijf minuten en één uur. Ze bieden echter geen inzicht in de tijd die nodig is per uitgevoerde inferentie.

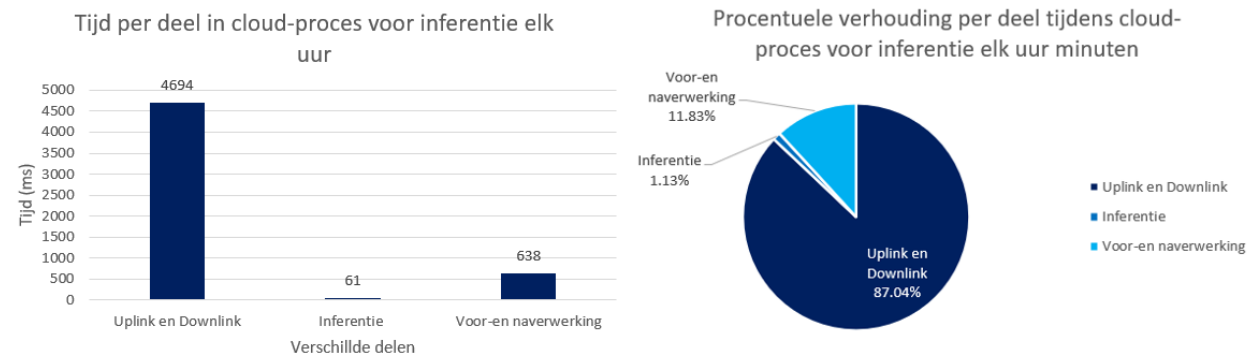
In het scenario van zes seconden moet de eindgebruiker ongeveer 721.16 milliseconden na zijn activiteit wachten om de uitgevoerde activiteit te zien. Voor het scenario van vijf minuten is dit 1.46 keer zoveel, namelijk 1049.67 milliseconden. Het scenario van één uur voorspelt de activiteiten die in het afgelopen uur zijn uitgevoerd, wat in totaal 5393.0 milliseconden duurde. Dit is 7.48 keer zoveel als het scenario van zes seconden en 5.14 keer zoveel als het scenario van vijf minuten. **Figuur 41** visualiseert deze metingen. Het staafdiagram toont voor elk deelproces hoeveel tijd het in beslag neemt. Het taartdiagram toont de procentuele verdeling van de verschillende processen ten opzichte van de totale tijd.



a)



b)



c)

***Figuur 41**: Staafdiagram en taartdiagram voor de visualisatie van de tijd van de verschillende delen voor inferentie van het cloud proces. a) scenario waarbij de data elke 6 seconden worden verstuurd en verwerkt. b) scenario waarbij de data elke 5 minuten worden verstuur en verwerkt. c) scenario waarbij de data elk uur worden verstuurd en verwerkt.*

Het is duidelijk dat de *overhead* in het eerste scenario het grootst is, waarbij slechts 0.01% van de totale tijd besteed wordt aan het daadwerkelijk uitvoeren van inferenties. Naarmate de data minder frequent worden verzonden, neemt deze *overhead* af en wordt een groter percentage van de totale tijd besteed aan de effectieve inferentie. Desondanks blijft de tijd voor de uplink en downlink met voorsprong de langste. Dit wordt duidelijk geïllustreerd in de taartdiagrammen van **Figuur 41**.

De voorgaande resultaten behandelden de tijd die nodig was om inferenties te maken op gegevens verzameld over periodes van zes seconden, vijf minuten en één uur. **Tabel 27** beidt een overzicht van de tijd per inferentie voor elk van de drie scenario's. Voor het eerste scenario blijft dit ongewijzigd. In het tweede scenario worden er in totaal 50 voorspellingen gemaakt, waardoor de waarden van **Tabel 26** worden gedeeld door een factor 50. Het derde scenario maakt in totaal 600 voorspellingen, bijgevolg worden alle waarden voor dit scenario van **Tabel 26** gedeeld door 600.

***Tabel 27**: Tijd nodig per inferentie voor de drie scenario's gebruikmakend van een cloud-architectuur.*

Scenario	Uplink en Downlink (ms)	Inferentietijd van het model (ms)	Andere (ms)	Totale Inferentietijd (ms)
6 seconden	697.31	0.10	23.75	721.16
5 minuten	19.47	0.10	1.43	20.99
1 uur	7.83	0.10	0.61	8.99

De uitgevoerde analyse onthult dat het derde scenario aanzienlijk minder tijd vereist per uitgevoerde inferentie dan de andere scenario's. Het eerste scenario, daarentegen, heeft beduidend meer tijd nodig per inferentie. Deze resultaten zijn in lijn met de metingen van het energieverbruik voor elk scenario (zoals beschreven in **Sectie 6.3.1**), waarbij het eerste scenario de meeste energie verbruikt, gevolgd door het tweede en derde scenario.

De correlatie tussen de inferentietijd en energieverbruik komt voort uit het feit dat bij het frequenter uitvoeren van inferenties, elke afzonderlijke inferentie meer tijd kost. In de context van de energieverbruiksmetingen werden er voor alle drie scenario's in totaal 600 inferenties

uitgevoerd. In het eerste scenario, waar de inferenties het meest frequent waren, was per inferentie aanzienlijk meer tijd nodig. Dit resulteert in een hoger energieverbruik vergeleken met de andere twee scenario's.

Het tweede scenario, dat minder frequent inferenties uitvoerde dan het eerste maar meer dan het derde, toonde ook een hoger energieverbruik dan het derde scenario. Dit ondersteunt verder de vastgestelde correlatie: hoe meer tijd er nodig is voor elke inferentie, hoe meer energie er wordt verbruikt. Dit betekent dat het optimaliseren van de tijd die elke inferentie kost, direct kan bijdragen aan het verminderen van het energieverbruik.

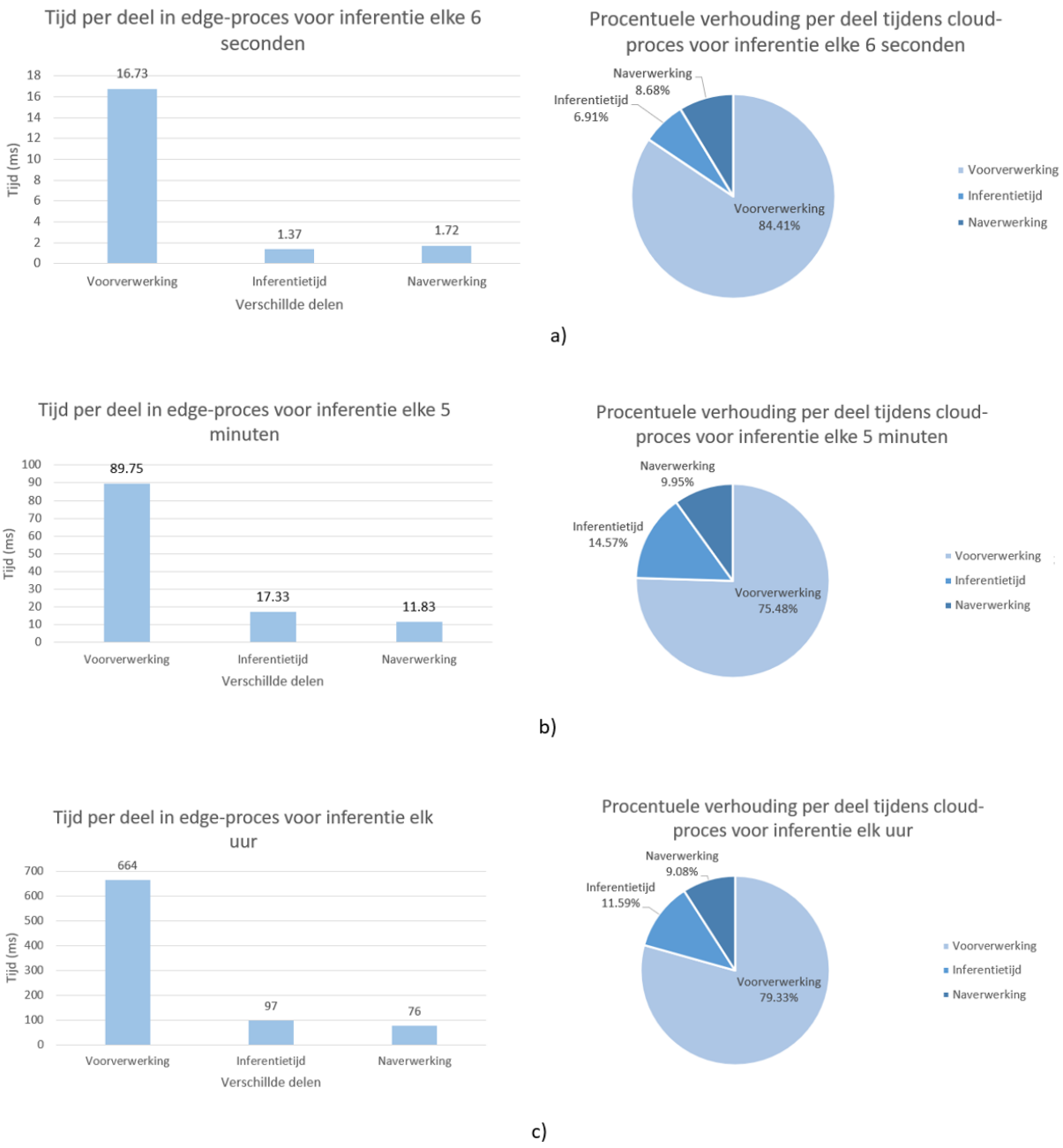
6.3.2.2. Inferentietijd *edge*-architectuur

De procedure voor het berekenen van de tijd en de gebruikte tijdstempels zijn reeds uiteengezet in **Sectie 6.1**. De metingen zijn opnieuw uitgevoerd voor de drie eerder genoemde scenario's, maar dit keer voor de *edge*-architectuur. **Tabel 28** biedt een overzicht voor de totale benodigde tijd voor het uitvoeren van inferentie(s) in elk scenario. Het biedt inzicht in hoe lang het duurt om inferenties uit te voeren op een bestand van zes seconden, vijf minuten en één uur aan accelerometerdata. Het proces wordt ook hier verdeeld in drie verschillende stadia: voorverwerking, inferentie door het model en naverwerking.

***Tabel 28**: Overzicht van de inferentietijd voor de verschillende delen in het edge proces voor de drie verschillende scenario's.*

Scenario	Voorverwerking (ms)	Inferentie door het model (ms)	Naverwerking (ms)	Totale Inferentietijd (ms)
6 seconden	16.73	1.37	1.72	19.82
5 minuten	89.75	17.33	11.83	118.91
1 uur	664.0	97	76	837.0

Het is duidelijk dat hoe groter het bestand, hoe meer tijd nodig is voor inferentie. Dit is logisch: hoe meer gegevenssamples voorspeld moeten worden, hoe langer het zal duren. Opmerkelijk is dat het overgrote deel van de tijd van het inferentieproces wordt besteed aan de voorverwerking. Hierbij gaat het over het uitlezen van de databestanden en het maken van samples van 192 datapunten.



***Figuur 42**: Staafdiagram en taartdiagram voor de visualisatie van de tijd van de verschillende delen voor inferentie van het edge-proces. a) scenario waarbij de data elke 6 seconden worden verstuurd en verwerkt. b) scenario waarbij de data elke 5 minuten worden verstuur en verwerkt. c) scenario waarbij de data elk uur worden verstuurd en verwerkt.*

Figuur 42 presenteert grafieken voor de drie verschillende scenario's. Het staafdiagram toont hoeveel tijd er per procesonderdeel wordt verbruikt. Het taartdiagram geeft de procentuele verhouding weer tussen deze verschillende onderdelen tijdens het inferentieproces. Wanneer dit wordt vergeleken met **Figuur 41**, dat dezelfde grafieken toont maar dan voor het *cloud*-proces, valt op dat er procentueel gezien in het *edge*-proces meer tijd wordt besteed aan het inferentieproces. Bij het *cloud*-proces ging meer tijd verloren aan andere zaken, voornamelijk aan de uplink- en downlinktijd.

In **Sectie 5** werd gemeten dat de totale inferentietijd bij de *edge*-architectuur voor het verwerken van data van vijf minuten 136 ms bedraagt. Zoals eerder werd vermeld, werden deze metingen op een licht andere applicatie-versie uitgevoerd. De accelerometerdata werd aan een hogere frequentie afgebeeld op de applicatie voor de gebruiker waardoor er meer bronnen werden gebruikt en de inferentietijd licht groter was dan in deze sectie. Aangezien het verschil klein is kan toch enige vergelijking gemaakt worden. In **Sectie 5** werd onder andere gemeten dat de *Depthwise Separable Convolutions* en de *TensorFlow's Default Optimizer* erin slagen de inferentietijd te verminderen met 8-9 ms. Uit de metingen van deze sectie blijkt dat de effectieve inferentietijd 17.33 ms bedraagt. Dit betekent dat beide technieken erin slagen om de effectieve inferentietijd ongeveer te halveren.

De bovenstaande cijfers presenteerden de totale inferentietijd voor de verwerking van bestanden met zes seconden, vijf minuten en één uur aan accelerometerdata. **Tabel 29** toont de tijd benodigd voor het uitvoeren van één enkele inferentie in elk scenario**.** Voor het eerste scenario blijft dit hetzelfde. Voor het tweede scenario worden er in totaal 50 voorspellingen gemaakt, waardoor de waarden van **Tabel 28** worden gedeeld door factor 50. Het derde scenario maakt in totaal 600 voorspellingen, dus alle waarden van **Tabel 28** worden gedeeld door 600.

***Tabel 29**: Tijd nodig per inferentie voor de drie scenario's gebruikmakend van een cloud-architectuur.*

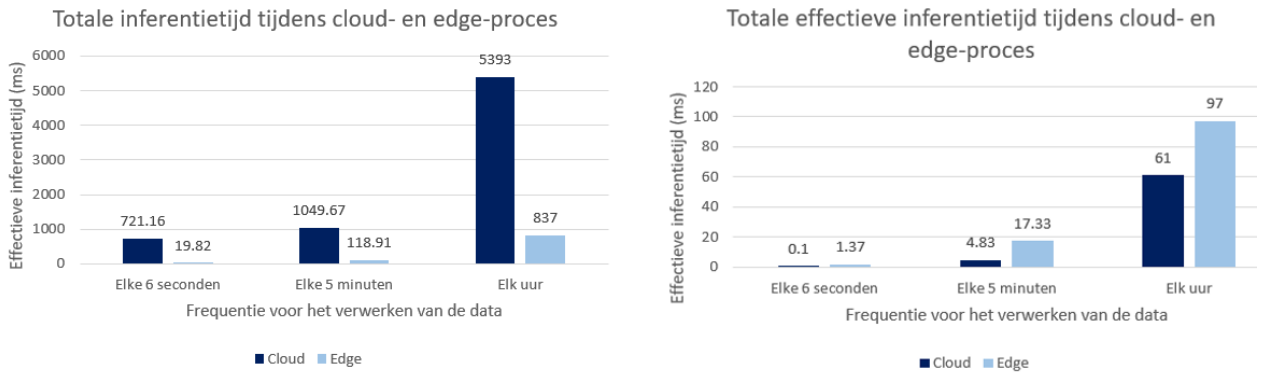
Scenario	Voorverwerking (ms)	Inferentie (ms)	Naverwerking (ms)	Totale Inferentietijd (ms)
6 seconden	16.73	1.37	1.72	19.82
5 minuten	1.80	0.34	0.24	2.38
1 uur	1.11	0.16	0.13	1.40

Uit analyse blijkt dat wanneer bestanden minder frequent worden uitgelezen voor verwerking, dit gemiddeld minder tijd kost. Het grootste verschil is te vinden in de tijd nodig voor inferenties. Hiervoor zijn verschillende mogelijke verklaringen. Een eerste mogelijkheid is dat er bepaalde opstartkosten verbonden zijn aan het laden of initialiseren van het model. Deze opstartkosten worden verdeeld over de uitgevoerde inferenties. Hoe minder inferenties er plaatsvinden, hoe hoger de relatieve kost per voorspelling zal zijn. Een andere mogelijkheid is *caching*. Als het model gegevens of berekeningen hergebruikt tussen inferenties, kunnen deze na de eerste inferentie in de cache worden opgeslagen, waardoor latere inferenties sneller worden uitgevoerd. Daarnaast is opvallend dat de voorverwerking- en naverwerkingtijd per inferentie afneemt. Dit komt doordat er minder frequent bestanden hoeven te worden geopend, gelezen en naar geschreven moet worden.

6.3.2.3. Vergelijking inferentietijd architecturen

De voorgaande twee subsecties behandelden de inferentietijden voor de *cloud*- en *edge*-architectuur voor drie verschillende scenario's. Hierbij werd gekeken naar de totale tijd die nodig was om per scenario inferenties te doen. Eerst werd besproken hoeveel tijd er nodig was om een bestand met zes seconden, vijf minuten en één uur aan accelerometerdata te verwerken (**Tabel 26 en 28**), voor beide architecturen. Daarna werd er per scenario ook gekeken naar de gemiddelde tijd die nodig was per inferentie (**Tabel 27 en 29**). Bij beide architecturen wordt vastgesteld dat, wanneer er over een grotere tijdsperiode voorspellingen worden uitgevoerd, dit relatief gezien (per inferentie) het voordeligst is. Bij de *cloud*-omgeving is dit voornamelijk te wijten aan het feit dat de relatieve tijd voor de *uplink* en *downlink* connectie aanzienlijk minder is. Ook de andere metrieken, zoals de effectieve tijd voor inferentie, dalen naarmate de tijdsperiode toenam. Bij de

edge-omgeving is dit ook het geval. Dit wordt verklaard door de initialisatiekosten van het model en mogelijke *caching*.



***Figuur 43**: De totale effectieve en de totale inferentietijd van de edge- en cloud-architectuur.*

In deze subsectie wordt een vergelijking gemaakt tussen beide architecturen. Het eerste dat opvalt, is dat de tijd die het model in de *cloud* nodig heeft voor het effectief uitvoeren van inferenties (**Figuur 43, rechts**) kleiner is dan het model op de smartphone (*edge*). Dit komt doordat de virtuele node beschikt over betere computationele bronnen dan de smartphone, waardoor de effectieve inferentie sneller verloopt. Echter, de totale tijd voor inferentie (**Figuur 43, links**) bij de *edge*-architectuur is aanzienlijk minder dan bij de *cloud*-architectuur. Dit komt doordat de *cloud*-architectuur veel tijd verliest aan het heen en weer sturen van de databestanden over het internet. Voor elk van de drie geteste scenario's is de *edge*-architectuur sneller dan de *cloud*-architectuur, ondanks de beperkte bronnen die beschikbaar zijn op de smartphone.

Over het algemeen kan worden gesteld dat:

- Hoe frequenter de databestanden worden verwerkt, hoe sneller de totale inferentietijd is voor beide architecturen. Dit is te wijten aan het feit dat er minder inferenties worden uitgevoerd.
- Bij minder frequente verwerking van databestanden is de individuele inferentietijd voor beide architecturen korter. Dit kan worden toegeschreven aan de initialisatiekosten van het model en het minder vaak schrijven naar en lezen van bestanden.

- De *edge*-architectuur zorgt voor een snellere inferentietijd dan de *cloud*-architectuur voor alle drie de onderzochte scenario's. Dit komt doordat de *uplink*- en *downlinktijden* bij de *cloud*-architectuur zwaar doorwegen.

6.3.3. Verschillende internetnetwerken

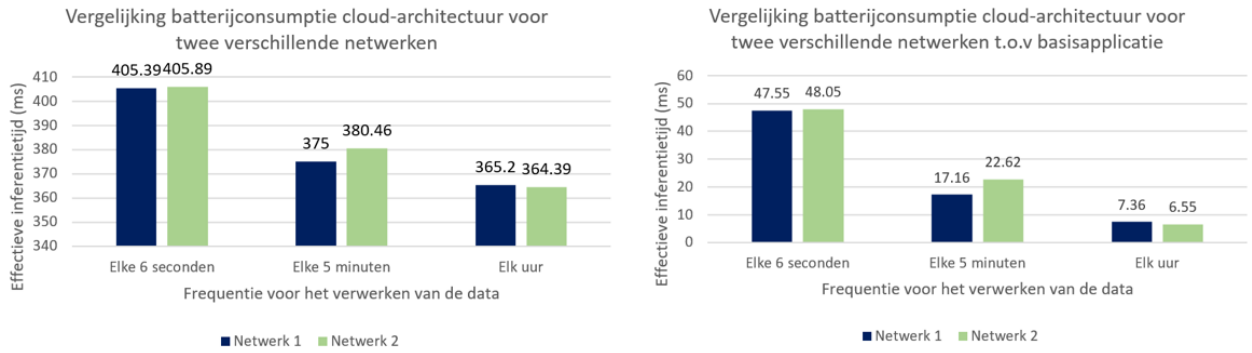
In voorgaande subsecties is een vergelijkend onderzoek uitgevoerd tussen een *cloud*- en *edge*-architectuur, waarbij de invloed op het batterijverbruik en de inferentietijd wordt onderzocht. Voor de *cloud*-architectuur is een internetverbinding nodig om de data naar een virtuele node te sturen waar inferentie plaatsvindt, waarna de voorspellingen terug naar de applicatie worden gestuurd. In deze subsectie worden dezelfde metingen uitgevoerd op een ander internetnetwerk om te bepalen hoe sterk de internetverbinding de resultaten omtrent het batterijverbruik en de inferentietijd beïnvloedt.

Tabel 30: *Vergelijking batterijconsumptie cloud-architectuur voor twee verschillende internetwerken. (wit: zelfde netwerk als metingen voorgaande subsecties, grijs: nieuw netwerk).*

Scenario	mAh verbruikt door applicatie	Totale mAh verbruikt gedurende meting	Verbruik in batterijpercentage door applicatie	Totaal verbruik in batterijpercentage gedurende meting
Geen inferentie	357.84	450	11.91%	15%
6 seconden	405.39	510	13.51%	17%
6 seconden	405.89	540	13.53%	18%
5 minuten	375.00	480	12.50%	16%
5 minuten	380.46	510	12.68%	17%
1 uur	365.20	480	12.17%	16%
1 uur	364.39	510	12.15%	17%

Tabel 30 vat de resultaten van het batterijverbruik samen voor beide netwerken. De witte rijen tonen de resultaten van **subsectie 6.3,** de grijze rijen presenteren de resultaten van metingen op een ander netwerk. Uit de metingen blijkt dat de invloed op het batterijverbruik nagenoeg gelijk is voor beide netwerken. Hoewel deze resultaten een eerste inzicht geven in de invloed van de internetverbinding op het batterijverbruik op twee verschillende netwerken, kunnen ze niet garanderen dat dit voor elk netwerk hetzelfde zal zijn, aangezien er slechts een vergelijking is

gemaakt tussen twee netwerken. Ook andere factoren, zoals netwerkverkeer, kunnen een invloed hebben. **Figuur 44** visualiseert de resultaten.

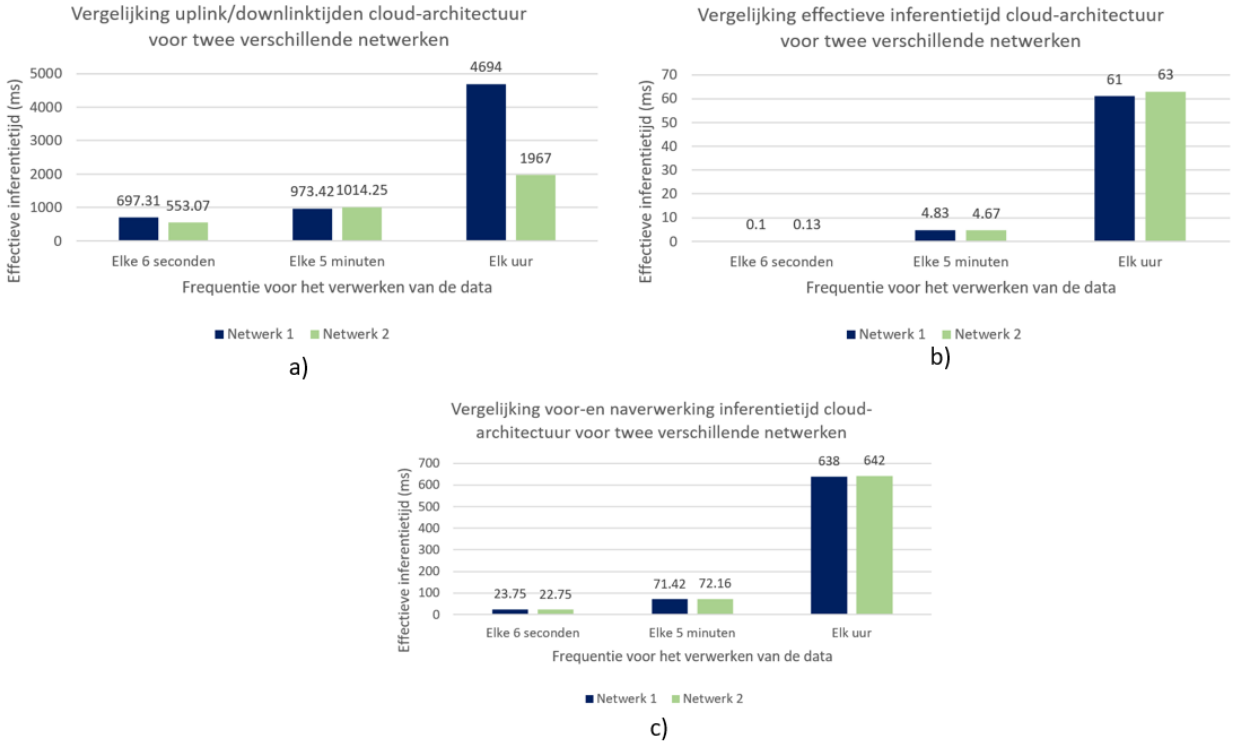


Figuur 44: *Vergelijking batterijconsumptie cloud-architectuur voor twee verschillende internetwerken. (Donkerblauw: zelfde netwerk als metingen voorgaande subsecties, lichtgroen: nieuw netwerk).*

De invloed op de inferentietijd voor beide netwerken is eveneens onderzocht, naast het effect op het batterijverbruik. De resultaten hiervan zijn terug te vinden in **Tabel 31**. Ook hier zijn de uitkomsten grotendeels gelijk. De belangrijkste kolom betreft de uplink- en downlinktijd, omdat deze kan variëren bij metingen op verschillende netwerken. De andere twee kolommen betreffen tijden in de *cloud* zelf en zijn onafhankelijk van het netwerk dat wordt gebruikt voor de datatransmissie. De resultaten voor de daadwerkelijke inferentietijd en de voor- en naverwerkingtijd zijn dan ook nagenoeg gelijk. Voor de uplink- en downlinktijd is er een groter verschil waarneembaar. In het scenario waarbij de verwerking per uur wordt uitgevoerd, is dit verschil het grootst. Dit kan mogelijk worden toegeschreven aan het netwerkverkeer dat op dat moment aanwezig was. Over het algemeen zijn de resultaten vrij onafhankelijk van het netwerk dat wordt gebruikt voor de datatransmissie. Op beide netwerken neemt de inferentietijd toe naarmate een groter bestand wordt verstuurd, en is de totale inferentietijd vergelijkbaar.

***Tabel 31**: Vergelijking inferentietijd cloud-architectuur voor twee verschillende internetwerken. (wit: zelfde netwerk als metingen voorgaande subsecties, grijs: nieuw netwerk).*

Scenario	Uplink/downlink -tijd (ms)	Effectieve inferentietijd (ms)	voor-en naverwerkingtijd (ms)
6 seconden	697.31	0.10	23.75
6 seconden	553.07	0.13	22.75
5 minuten	973.42	4.83	71.42
5 minuten	1014.25	4.67	72.16
1 uur	4694	61	638
1 uur	1967	63	642



***Figuur 45:** Vergelijking inferentietijd cloud-architectuur voor twee verschillende internetwerken. (Donkerblauw: zelfde netwerk als metingen voorgaande subsecties, lichtgroen: nieuw netwerk).*

6.4. Conclusie

Dit onderzoek heeft zich gericht op het analyseren van de invloed van twee verschillende architecturen, de cloud- en de *edge*-architectuur, op de batterijconsumptie en inferentietijd in het kader voor *Human Activity Recognition* voor de mBrain-studie. Er is specifiek gekeken naar de prestatieverschillen tussen deze architecturen binnen drie verschillende scenario's, waarbij data over zes seconden, vijf minuten en één uur werden verwerkt.

Wat betreft de batterijconsumptie, blijkt uit de resultaten dat de *cloud*-architectuur in de eerste twee scenario's aanzienlijk meer energie verbruikt dan de *edge*-architectuur. Dit verschil kan voornamelijk worden toegeschreven aan de energie die nodig is voor het uploaden en downloaden van data naar en van de virtuele node. Dit resulteert in een hoger energieverbruik. In het derde scenario, waarbij na elk uur verbinding wordt gemaakt met de *cloud*, is het energieverbruik echter minder en vergelijkbaar met dat van de *edge*-architectuur. De resultaten werden eerder geïllustreerd in **Sectie 6.3** op **Figuur 40**.

Wat de inferentietijd betreft, is de *edge*-architectuur in alle drie de scenario's sneller dan de *cloud*-architectuur. Hoewel de *cloud*-architectuur betere computationele bronnen heeft en daardoor sneller inferentie kan uitvoeren, wordt dit voordeel tenietgedaan door de tijd die nodig is voor het uploaden en downloaden van gegevens. Daarentegen wordt de inferentietijd van de *edge*-architectuur hoofdzakelijk bepaald door de tijd die nodig is voor de voorverwerking, wat in grote mate werd beïnvloed door de omvang van de te verwerken gegevens. De resultaten werden eerder geïllustreerd in **Sectie 6.3** op **Figuur 43**.

Een andere belangrijke metriek die niet specifiek werd besproken in deze subsectie is de vereiste opslaggrootte van de databestanden in de drie verschillende scenario's. In **Tabel 32** wordt een overzicht gegeven van het aantal datapunten per vrijheidsgraad en de bestandgroottes in de drie verschillende scenario's. In deze bestanden zaten naast de datapunten ook de tijdstempels van elk datapunt.

Tabel 32: *Overzicht aantal datapunten per vrijheidsgraad en bestandgroottes bij de drie verschillende scenario's.*

Scenario	Duur	Aantal datapunten per vrijheidsgraad	Grootte (kB)
1	zes seconden	192	54
2	vijf minuten	9600	408
3	één uur	115 200	4 710

Samengevat illustreert het onderzoek duidelijk dat zowel de frequentie van de dataverwerking als de keuze van de architectuur significante invloed uitoefenen op de batterijconsumptie en inferentietijd. Hoewel de *cloud*-architectuur profiteert van superieure computationele middelen, wordt dit voordeel grotendeels tenietgedaan door de *overhead* van datatransmissie, resulterend in zowel hogere batterijconsumptie als langere inferentietijden in vergelijking met de *edge*-architectuur. Aan de andere kant, ondanks dat de *edge*-architectuur beperkt is door de beschikbare bronnen op de smartphone, is er minder *overhead* en wordt de beschikbare tijd in grotere mate besteed aan het daadwerkelijk uitvoeren van inferenties. Dit leidt tot aanzienlijke verbeteringen voor zowel de batterijconsumptie als de snelheid van inferentie. Daarmee blijkt de *edge*-architectuur, gebaseerd op de onderzochte metrieken, de meest voordelige architectuur te zijn.

Bovendien is de keuze van de frequentie van gegevensverwerking van cruciaal belang. Dit onderzoek maakt duidelijk dat een hogere frequentie van gegevensverwerking leidt tot een hoger batterijverbruik en langere relatieve inferentietijd. Echter, ook de grootte van de lokaal opgeslagen databestanden moet worden meegewogen. Het eerste scenario scoort het slechtst qua batterijverbruik en relatieve inferentietijd, maar het best voor de opslagomvang van het databestand. Het derde scenario scoort het best voor batterijconsumptie en relatieve inferentietijd, maar vereist significant meer lokale opslagcapaciteit voor de data. Het tweede scenario vormt hier een evenwichtige tussenweg. Het presteert nauwelijks slechter op het gebied

van batterijverbruik en relatieve inferentietijd dan het derde scenario, en vereist aanzienlijk minder opslagcapaciteit.

Na analyse van de resultaten van dit onderzoek wordt geconcludeerd dat de *edge*-architectuur de meest efficiënte architectuur is volgens de onderzochte metrieken. Bovendien blijkt het vijf-minuten scenario een ideale balans te bieden tussen batterijconsumptie en vereiste opslagcapaciteit.

7. DUURZAAMHEIDSREFLECTIE

Duurzaamheid vormt een steeds belangrijkere pijler in het ontwerp en de implementatie van technologische oplossingen. In deze scriptie werd onderzoek gedaan naar de efficiëntie van de *cloud*- en *edge*-architectuur in het kader van de batterijconsumptie en inferentietijd voor *Human Activity Recognition* (HAR). Er werd ook onderzoek gedaan naar welke technieken er bestaan om efficiëntere en lichtere modellen te creëren. De focus van dit onderzoek op efficiëntie sluit aan bij verschillende *Sustainable Development Goals* (SDGs) van de Verenigde Naties. In het kader van duurzaamheidsreflectie worden vier verschillende SDGs besproken die relevant zijn en zich aansluiten voor de onderzoeken die werden gedaan.

1. SDG 3: Goede gezondheid en welzijn.
2. SDG 9: Industrie, innovatie en infrastructuur.
3. SDG 12: Verantwoorde consumptie en productie.
4. SDG 13: Klimaat

SDG 3 dat zich richt op goede gezondheid en welzijn, sluit aan bij dit onderzoek aangezien het deel is van een groter project in de mBrain-studie. Het grote project heeft als doel migraine te voorspellen en te detecteren. De *Human Activity Recognition* (HAR) draagt bij aan deze voorspellingen.

SDG 9 dat zich focust op industrie, innovatie en infrastructuur, benadrukt het belang van veerkrachtige infrastructuur, duurzame industrialisatie en innovatie. Dit onderzoek heeft een bijdrage geleverd aan deze doelstelling door te verkennen hoe de efficiëntie van data-inferentie kan worden geoptimaliseerd. De bevinding dat de *edge*-architectuur minder energie verbruikt en snellere inferenties biedt, heeft implicaties voor de veerkracht van infrastructuren.

SDG 12 richt zich op verantwoorde consumptie en productie. Dit is de SDG dat het meest toepasselijk is voor deze scriptie. Door de energie-efficiëntie van apparaten te verhogen, kan het energieverbruik verminderd worden en daarmee ook de productie van energiebronnen zoals batterijen. Dit heeft zowel economische als ecologische voordelen, aangezien het helpt om

grondstoffen te besparen en afval te verminderen. Door het onderzoek werd duidelijk welke scenario's per architectuur het meeste batterij van de smartphone verbruikten. Hierdoor kan gekozen worden voor een scenario en architectuur die minder batterij verbruikt waardoor het toestel minder frequent opgeladen moet worden. Op deze manier kan de energieconsumptie van de gebruiker gereduceerd worden.

Tot slot heeft dit onderzoek ook raakvlakken met SDG 13, klimaatactie. Technologie speelt een belangrijke rol in de strijd tegen klimaatverandering. Door het energieverbruik van apparaten te verminderen, kan de CO2-voetafdruk van technologische processen worden verkleind. Dit onderzoek toont aan dat door het optimaliseren van de architectuur en het zorgvuldig overwegen van de frequentie van dataverwerking, de impact van technologie op het klimaat kan verminderen.

8. CONCLUSIE EN TOEKOMSTIG WERK

Het afsluitende hoofdstuk van deze thesis omvat essentiële conclusies die zijn getrokken op basis van de verworven resultaten. Dit hoofdstuk biedt een grondige evaluatie van de bevindingen die zijn vastgesteld in de context van de *edge*-omgeving tijdens het verloop van het onderzoek. De eerder opgestelde onderzoekvragen worden behandeld in deze sectie. Er wordt niet alleen aandacht besteed aan de bereikte resultaten, maar ook aan de beperkingen en uitdagingen die naar voren zijn gekomen tijdens de uitvoering van deze studie. Het is essentieel om deze tekortkomingen te erkennen, aangezien zij de weg wijzen naar toekomstige verbeteringen en een verdieping van het onderzoek.

8.1. Antwoorden op de onderzoeksvragen

Dit proefschrift heeft als doel een overzicht te geven van de invloed van de *cloud*- en *edge*-architectuur op de batterijconsumptie en inferentietijd. Op basis hiervan kan besloten worden om mogelijk over te stappen van de huidige *cloud*-architectuur naar een *edge*-architectuur. Hierbij werd onderzoek gedaan naar mogelijke optimalisatietechnieken om het model geschikt te maken voor een smartphone, om de mogelijk overgang te vereenvoudigen. De volgende onderzoeksvragen worden beantwoord.

OV1. Kan een *machine learning* model uitgevoerd worden op een smartphone zonder dat daarmee een aanzienlijk verlies aan accuraatheid gepaard gaat?

ANT1. Door gebruik te maken van de Keras- en Tensorflow-bibliotheken kan een model worden omgezet naar een TFLite-formaat dat speciaal is ontworpen om te gebruiken op een *edge*-toestel zoals een smartphone. Hierbij is er nauwelijks tot geen verlies aan accuraatheid. Bovendien kan dit model eenvoudig geïmporteerd worden in Android Studio. Het verlies aan nauwkeurigheid bij deze omzetting is minimaal.

OV2. Welke technieken bestaan er om een *machine learning* model lichter en sneller te maken en wat is de invloed hiervan op de inferentietijd, modelgrootte en de accuraatheid?

ANT2. De eerste conversie die wordt gemaakt, is van het .Keras-formaat naar het TFLite-formaat. Dit leidde tot een eerste reductie in modelomvang met factor 2.12. Er werden drie technieken toegepast om het model verder te optimaliseren voor de edge. Deze technieken zijn: *depthwise separable convolutions*, *pruning*, post-training kwantisatie en een hybridevorm van de laatste twee. Het toepassen van *depthwise separable convolutions* zorgt voor een extra reductie in modelomvang met factor 1.51, de effectieve inferentietijd wordt gehalveerd en de nauwkeurigheid blijft nagenoeg dezelfde. *Pruning* bewijst een positieve impact te hebben op de grootte van het model. Hoe meer gewichten er worden verwijderd, hoe kleiner het model wordt, dit leidt echter ook tot een vermindering in nauwkeurig/accuraatheid. Voor de post-training kwantisatie werden drie verschillende vormen onderzocht. De conversie naar float16-gewichten leidt tot een extra reductie in modelomvang met factor 1.38. De uint8-dataype conversie resulteert in een extra afname van de omvang met factor 1.76, en de *TensorFlow's Default Optimizer* realiseert een gelijkaardige compressie als de uint8-kwantisatie. Daarnaast is de effectieve inferentietijd van het model ook dubbel zo snel na het toepassen van de *TensorFlow's Default Optimizer*. De hybridevorm, van pruning en kwantisatie, biedt geen extra voordelen t.o.v. enkel kwantisatie aangezien de nauwkeurigheid aanzienlijk daalt.

OV3. Heeft het gebruik van de *edge*-architectuur voor *machine learning* modellen en positieve impact op de batterijconsumptie en inferentietijd in vergelijking met een *cloud*-architectuur?

Per architectuur werden drie scenario's ontworpen. Het eerste scenario verwerkt de data elke zes seconden, wat overeenkomt met één sample. Het tweede scenario verwerkt de data elke vijf minuten. Het derde scenario verwerkt de data elk uur. Uit analyse blijkt dat de *edge*-architectuur de beste resultaten behaalt voor de inferentietijd voor alle drie de scenario's. Dit komt doordat de

uplink- en downlinktijd voor het verzenden en ontvangen van de data veel *overhead* veroorzaakt. Dit is verantwoordelijk voor 96.7%, 92.7% en 87.0% van de totale tijd, respectievelijk voor de drie scenario's. Bij de *edge*-architectuur is er geen sprake van uplink- en downlinktijden. De *overhead* is in deze architectuur bijgevolg veel kleiner. Wat betreft de batterijconsumptie presteert de *edge*-architectuur in de eerste twee scenario's het beste. Het grootste verschil is te merken in het eerste scenario, waarbij de *cloud*-architectuur 47.55 mAh/uur extra verbruikte t.o.v. de basisapplicatie, dat enkel de data opslaat en geen verwerking doet. Voor datzelfde scenario zorgt de *edge*-architectuur slechts voor een toename van 14.47 mAh/uur t.o.v. de basisapplicatie. Bij het tweede scenario is het verschil in verbruik tussen beide architecturen slechts 8.22 mAh. Voor het derde scenario was de consumptie nagenoeg dezelfde. Hoe minder frequent de data wordt verwerkt, hoe gelijkaardiger het energieverbruik tussen beide modellen is.

8.2. Algemene conclusies

De twee technieken die het meest voordelig zijn om te implementeren voor het *machine learning* model voor *Human Activity Recognition* van de mBrain-studie, zijn de *Depthwise Separable Convolutions* en de *TensorFlow's Default Optimizer*. Eerst werd het model omgezet naar een TFLite-formaat, wat leidde tot een reductie in modelomvang met factor 2.12. Deze twee technieken slaagden er vervolgens in om de grootte van het mBrain-model verder te reduceren respectievelijk met factor 1.51 en 1.67. De totale reductiefactor voor de modelomvang bedraagt respectievelijk 3.19 en 3.53. Daarnaast slagen deze technieken er ook in om de inferentietijd met 8-9 ms te verminderen. Er werd gemeten dat de effectieve inferentietijd 17.33 ms bedraagt. De technieken slagen er bijgevolg in de inferentietijd ongeveer te halveren. Dit alles zonder dat de nauwkeurigheid van het model aanzienlijk verminderd is.

In het tweede deel wordt besloten dat de batterijconsumptie voor de *edge*-architectuur voordeliger was dan bij de *cloud*-architectuur. Dit verschil vermindert naarmate de data frequenter wordt verwerkt. Uit de metingen blijkt dat bij de *cloud*-architectuur het effectief uitvoeren van inferenties sneller is dan bij de *edge*-architectuur. Dit komt doordat het krachtigere

computationele bronnen bezit. Anderzijds is de *overhead* veroorzaakt door de uplink-en downlinktijd in de *cloud*-architectuur enorm. Hierdoor is de totale inferentietijd bij de *edge*-architectuur het snelst.

Er wordt besloten dat de overgang naar een *edge*-architectuur binnen de mBrain-studie voornamelijk voordelen met zich meebrengt. Zo is er geen nood aan een internetconnectie, is er minder nood aan bandbreedte, is de algemene batterijconsumptie minder en de inferentietijd is sneller. Anderzijds dient een machine learning model op het toestel geplaatst te worden, wat een bepaalde hoeveelheid opslag inneemt. Dit nadeel kan worden gereduceerd door de eerder benoemde optimalisatietechnieken toe te passen. Daarnaast is een scenario waarbij de data elke vijf minuten wordt verwerkt een goede consensus voor batterijverbruik en inferentietijd t.o.v. de opslagvereiste die nodig is om het tijdelijk lokaal op te slaan. Het model kan geoptimaliseerd worden door het toepassen van *Depthwise Separable Convolutions* of de *TensorFlow's Default Optimizer* voor verdere vermindering van modelomvang en voor snellere inferenties, zonder aanzienlijk verlies aan nauwkeurigheid.

8.3. Tekortkomingen en toekomstig werk.

Deze scriptie heeft zich gericht op het gebruik van één *machine learning*-model voor beide architecturen op de smartphone. Toekomstig werk zou een gelijkaardig onderzoek kunnen uitvoeren waarbij meerdere modellen draaien in beide architecturen. Zo zouden er bijvoorbeeld drie verschillende modellen tegelijkertijd gebruikt kunnen worden in de *edge*-omgeving of de *cloud*-omgeving.

Daarnaast zijn alle metingen uitgevoerd op één enkel toestel om vergelijkbare resultaten te verkrijgen. Dit betekent ook dat de resultaten specifiek zijn voor dat toestel. Het zou in de toekomst interessant zijn om de invloed van het specifieke toestel, in dit geval een smartphone, op de onderzochte metrieken verder te onderzoeken. Hierbij zouden de testen dan op verschillende toestellen worden uitgevoerd en kan een vergelijking worden opgesteld.

Uit resultaten blijkt dat de voorverwerking en naverwerking van de data meer tijd in beslag nemen dan het daadwerkelijk uitvoeren van inferenties. Toekomstig onderzoek zou kunnen focussen op het versnellen van deze processen, wat zou leiden tot een verminderd energieverbruik.

Deze scriptie is er niet in geslaagd om een overlappend tijdsvenster voor het verwerken van de data voor beide architecturen te integreren. Het mBrain-model is echter wel getraind om te werken met een overlappend tijdsvenster. De resultaten zijn dus afkomstig van metingen waarbij tijdens de verwerking geen overlappend tijdsvenster werd gehanteerd. Tijdens de metingen werden bijgevolg minder predicties uitgevoerd. Uit de metingen blijkt dat voor de *cloud*-architectuur de *overhead* zich voornamelijk bevindt in de uplink-en downlinktijd, respectievelijk 96.7%, 92.7% en 87.0% van de totale tijd. Wanneer in deze architectuur dus meer predicties zouden worden uitgevoerd, zou het verschil in tijd en batterijconsumptie minimaal zijn, aangezien dit slechts een klein deel bevat. Voor de *edge*-architectuur is gebleken dat het verschil tussen de scenario's, het totaal te verwerken data, klein is. Het verschil in batterijconsumptie tussen het tweede scenario en het derde scenario is slechts 1.32 mAh. Wetende dat bij het derde scenario 12 keer meer predicties worden uitgevoerd, wordt geconcludeerd dat het aantal predicties dat wordt uitgevoerd weinig invloed heeft op de batterijconsumptie. Ook hierbij is de tijd voor het effectief maken van predicties klein t.o.v. het uitlezen van andere zaken. Het grootste deel van de tijd gaat naar het uitlezen van de bestanden. Of er tijdsvensters worden gebruikt of niet, de grootte van deze bestanden blijft hetzelfde. Er wordt aangenomen dat, indien er wel gebruik wordt gemaakt van overlappende tijdsvensters, dit nagenoeg dezelfde resultaten zal opleveren als deze scriptie heeft gepresenteerd. Toekomstig werk kan dit eventueel verder onderzoeken.

REFERENTIES

[1] R. B. Lipton, J. M. Pavlovic, S. R. Haut, B. M. Grosberg, and D. C. Buse, "Methodological issues in studying trigger factors and premonitory features of migraine," *Headache*, vol. 54, no. 10, pp. 1661–1669, 2014.

[2] "Headache disorders," *Who.int*. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/headache-disorders. [Accessed: 07-May-2023].

[3] GBD 2016 Neurology Collaborators, "Global, regional, and national burden of neurological disorders, 1990-2016: a systematic analysis for the Global Burden of Disease Study 2016," *Lancet Neurol.*, vol. 18, no. 5, pp. 459–480, 2019.

[4] P. Salmon, "Effects of physical exercise on anxiety, depression, and sensitivity to stress: a unifying theory," *Clin. Psychol. Rev.*, vol. 21, no. 1, pp. 33–61, 2001.

[5] H. Nematallah, S. Rajan, and A.-M. Cretu, "Logistic model tree for human activity recognition using smartphone-based inertial sensors," in *2019 IEEE SENSORS*, 2019.

[6] "WISDM lab: Dataset," *Fordham.edu*. [Online]. Available: https://www.cis.fordham.edu/wisdm/dataset.php. [Accessed: 07-May-2023].

[7] "UC Irvine machine learning repository," *Uci.edu*. [Online]. Available: https://archive-beta.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones. [Accessed: 07-May-2023].

[8] G. De Leonardis *et al.*, "Human Activity Recognition by Wearable Sensors : Comparison of different classifiers for real-time applications," in *2018 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, 2018.

[9] L. Cheng, Y. Guan, K. Zhu, and Y. Li, "Recognition of human activities using machine learning methods with wearable sensors," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 2017.

[10] Y. Chen, M. Guo, and Z. Wang, "An improved algorithm for human activity recognition using wearable sensors," in *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, 2016.

[11] "ActivityNet," *Fau.de*. [Online]. Available: https://www.mad.tf.fau.de/research/activitynet/. [Accessed: 07-May-2023].

[12] B. P. Soni, A. Saxena, and V. Gupta, "Support Vector Machine based approach for accurate contingency ranking in power system," in *2015 Annual IEEE India Conference (INDICON)*, 2015.

[13] D. Tsujinishi and S. Abe, "Fuzzy least squares support vector machines for multiclass problems," *Neural Netw.*, vol. 16, no. 5–6, pp. 785–792, 2003.

[14] A. Ignatov, "Real-time human activity recognition from accelerometer data using Convolutional Neural Networks," *Appl. Soft Comput.*, vol. 62, pp. 915–922, 2018.

[15] Andrey, *HAR*. .

[16] B. Kolosnjaji and C. Eckert, "Neural network-based user-independent physical activity recognition for mobile devices," in *Intelligent Data Engineering and Automated Learning – IDEAL 2015*, Cham: Springer International Publishing, 2015, pp. 378–386.

[17] V. Aswal, V. Sreeram, A. Kuchik, S. Ahuja, and H. Patel, “Real-time human activity generation using bidirectional long short term memory networks,” in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2020.

[18] Y. Li, D. Shi, B. Ding, and D. Liu, “Unsupervised feature learning for human activity recognition using smartphone sensors,” in *Mining Intelligence and Knowledge Exploration*, Cham: Springer International Publishing, 2014, pp. 99–107.

[19] M. Inoue, S. Inoue, and T. Nishida, “Deep recurrent neural network for mobile human activity recognition with high throughput,” *Artif. Life Robot.*, vol. 23, no. 2, pp. 173–185, 2018.

[20] W. Jiang and Z. Yin, “Human activity recognition using wearable sensors by deep convolutional neural networks,” in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015.

[21] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, “Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine,” in *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 216–223.

[22] C. A. Ronao and S.-B. Cho, “Human activity recognition with smartphone sensors using deep learning neural networks,” *Expert Syst. Appl.*, vol. 59, pp. 235–244, 2016.

[23] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan, “Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition,” *IEEE Access*, vol. 7, pp. 133509–133520, 2019.

[24] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[25] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *arXiv [cs.CV]*, 2017.

[26] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An extremely efficient convolutional neural network for mobile devices,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[27] B. Aricioğlu, S. Uzun, and S. Kaçar, “Deep learning based classification of time series of Chen and Rössler chaotic systems over their graphic images,” *Physica D*, vol. 435, no. 133306, p. 133306, 2022.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” *arXiv [cs.CV]*, 2016.

[29] E. Lattanzi, M. Donati, and V. Freschi, “Exploring artificial neural networks efficiency in tiny wearable devices for human activity recognition,” *Sensors (Basel)*, vol. 22, no. 7, p. 2637, 2022.

[30] P. Dhiman, V. Kukreja, and A. Kaur, “Citrus fruits classification and evaluation using deep convolution neural networks: An input layer resizing approach,” in *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2021.

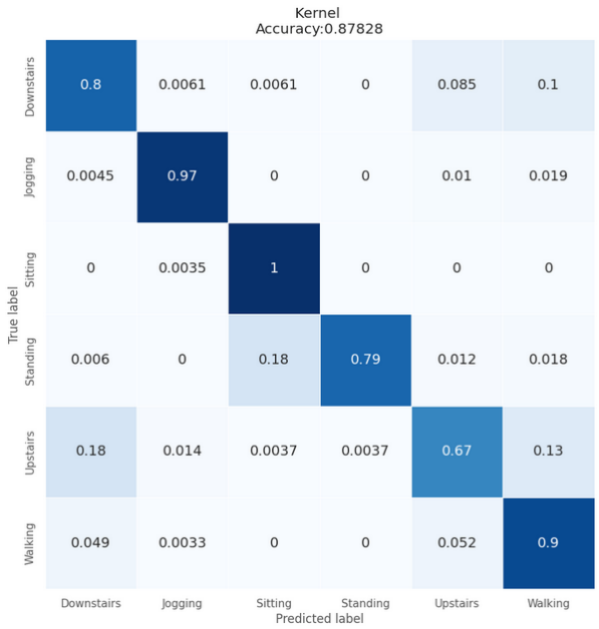
[31] A. Kumar, S. Goyal, and M. Varma, “Resource-efficient machine learning in 2 KB RAM for the internet of things,” *Manikvarma.org*. [Online]. Available: http://manikvarma.org/pubs/kumar17.pdf. [Accessed: 07-May-2023].

[32] “Cbonsai,” *GitLab*. [Online]. Available: https://gitlab.com/jallbrit/cbonsai. [Accessed: 07-May-2023].
[33] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional Neural Networks,” *arXiv [cs.LG]*, 2019.
[34] J. Tang *et al.*, “Understanding and improving knowledge distillation,” *arXiv [cs.LG]*, 2020.
[35] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, “FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network,” *arXiv [cs.LG]*, 2019.
[36] C. Gupta, “ProtoNN: Compressed and accurate kNN for resource-scarce devices,” in *Proceedings of the 34th International*, .
[37] K. Pradeep, K. Kamalavasan, R. Nathecsan, and A. Pasqual, “EdgeNet: SqueezeNet like convolution neural network on embedded FPGA,” in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018.
[38] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” *arXiv [cs.LG]*, 2015.
[39] S. S. Ogden and T. Guo, “MODI: Mobile deep inference made efficient by edge computing,” in *USENIX Workshop*, .
[40] S. Teerapittayanon, B. McDanel, and H. T. Kung, “Distributed deep neural networks over the cloud, the edge and end devices,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.
[41] Z. Huai, B. Ding, H. Wang, M. Geng, and L. Zhang, “Towards deep learning on resource-constrained robots: A crowdsourcing approach with model partition,” in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2019.
[42] “Performance measurement,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/lite/performance/measurement. [Accessed: 09-May-2023].
[43] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, “The tensor algebra compiler,” *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 1–29, 2017.
[44] “Build fast, sparse on-device models with the new TF MOT Pruning API,” *Tensorflow.org*. [Online]. Available: https://blog.tensorflow.org/2021/07/build-fast-sparse-on-device-models-with-tf-mot-pruning-api.html. [Accessed: 10-May-2023].
[45] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: Towards a cloud definition,” *Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2008.
[46] “The NIST definition of cloud computing,” in *Application Performance Management (APM) in the Digital Enterprise*, Elsevier, 2017, pp. 267–269.
[47] A. S. Al-Ahmad and H. Kahtan, “Cloud computing review: Features and issues,” in *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, 2018.

BIJLAGES

A. Verwarringsmatrices

A.1. WISDM-model

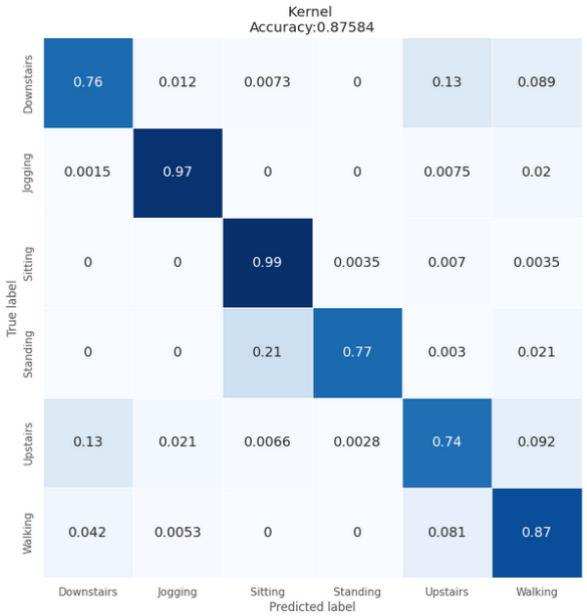


Figuur 46: *Verwarringsmatrix TFLite-versie WISDM-model.*

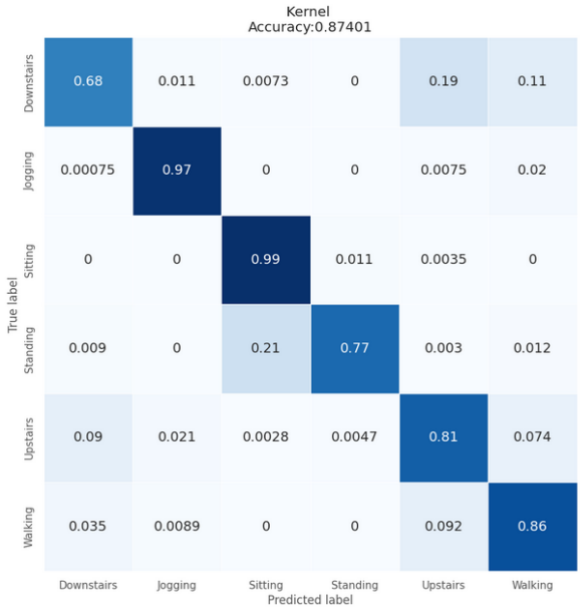
A.1.1. Depthwise Separable Convolutions

Figuur 47: *Verwarringsmatrix TFLite-versie WISDM-model met Depthwise Separable Convolutions.*

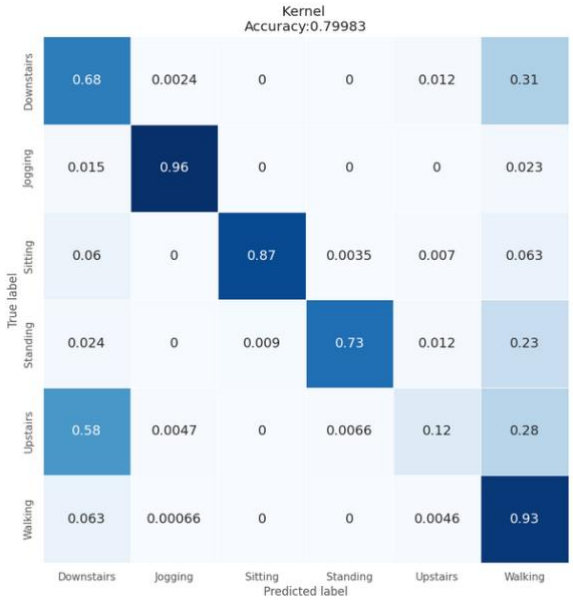
A.1.2. Pruning



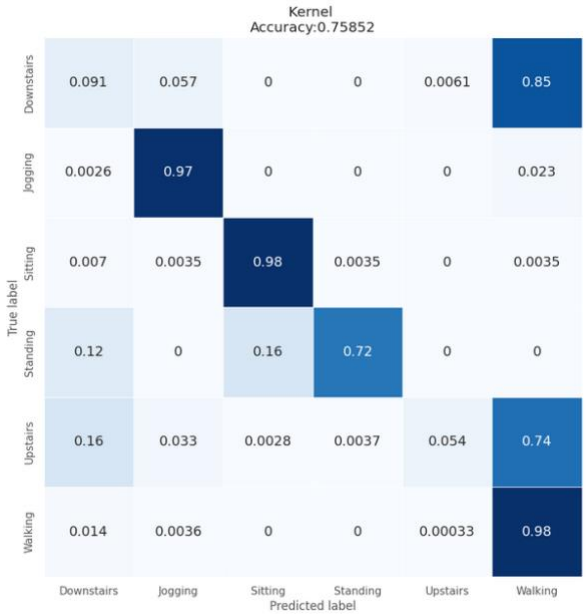
Figuur 48: *Verwarringmatrix TFLite-versie WISDM-model met met 25% uitdunning.*



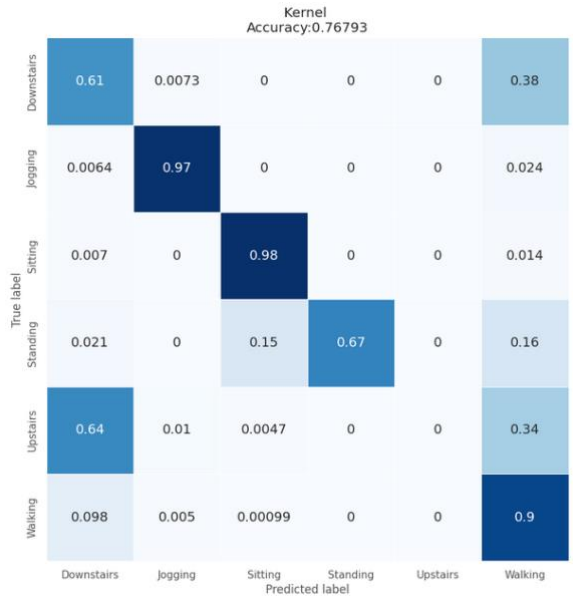
Figuur 49: *Verwarringmatrix TFLite-versie WISDM-model met 50% uitdunning.*



Figuur 50: *Verwarringmatrix TFLite-versie WISDM-model met 75% uitdunning.*

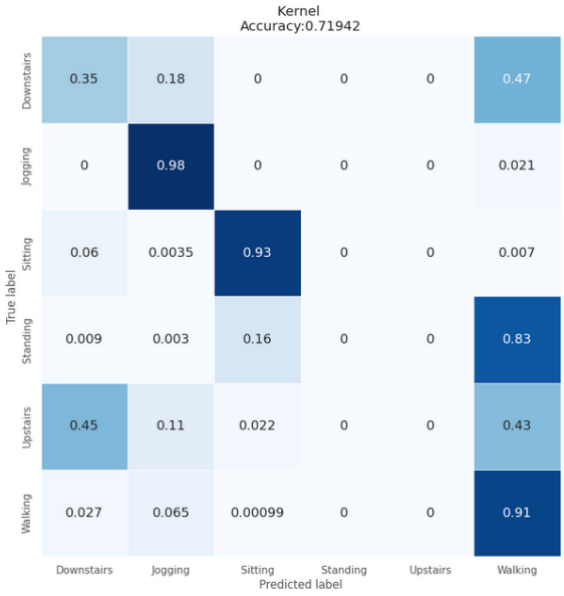


Figuur 51: *Verwarringmatrix TFLite-versie WISDM-model met met 80% uitdunning.*



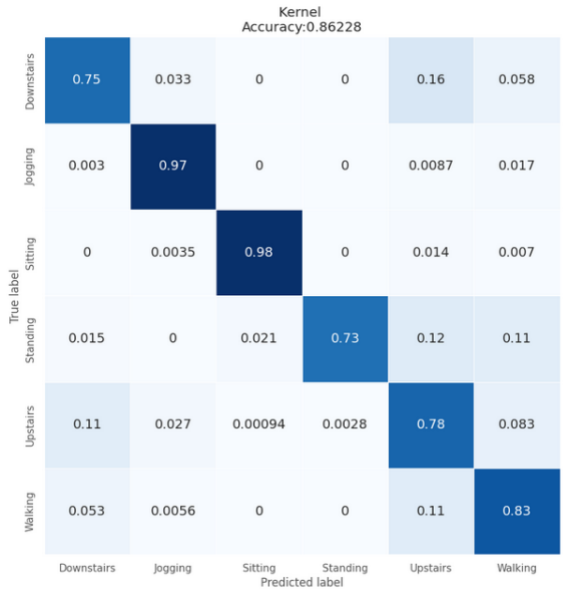
***Figuur 52**: Verwarringsmatrix TFLite-versie WISDM-model met 85% uitdunning.*

.



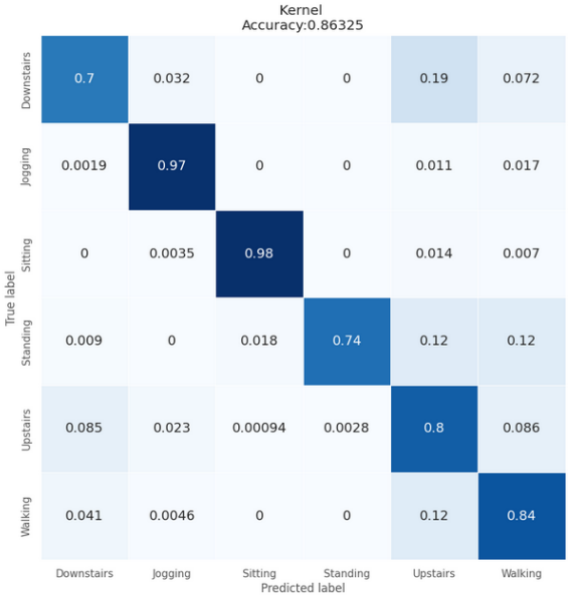
***Figuur 53**: Verwarringsmatrix TFLite-versie WISDM-model met 90% uitdunning.*

A.1.3. Kwantisatie

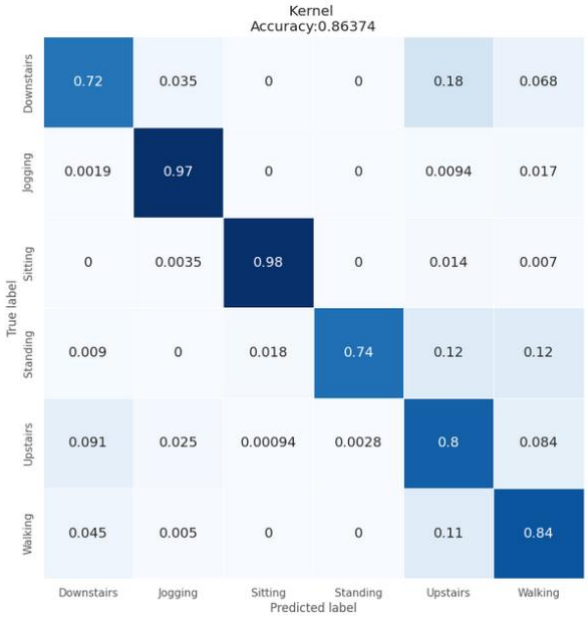


***Figuur 54**: Verwarringsmatrix TFLite-versie WISDM-model uint8-kwantisatie.*

.

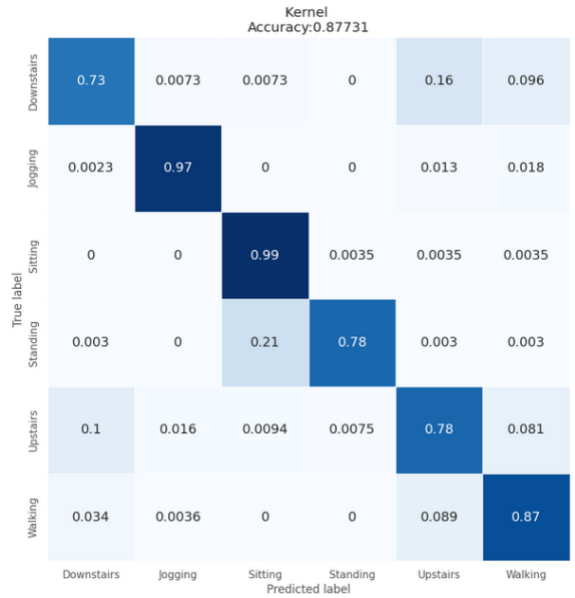


***Figuur 55:** Verwarringsmatrix TFLite-versie WISDM-model na float16-kwantisatie.*

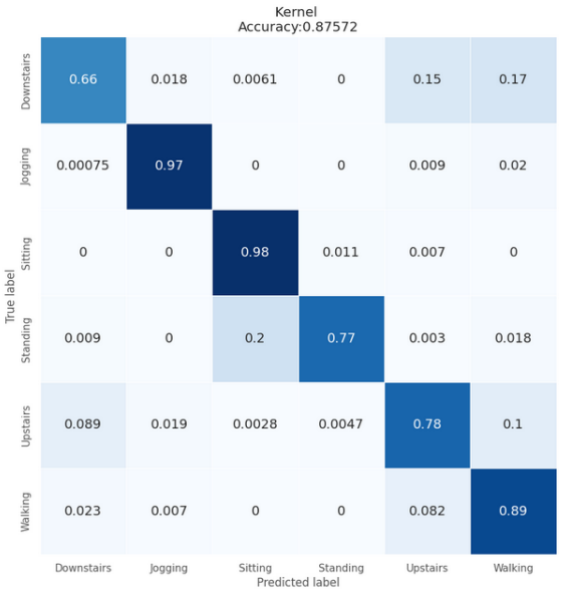


***Figuur 56:** Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite.*

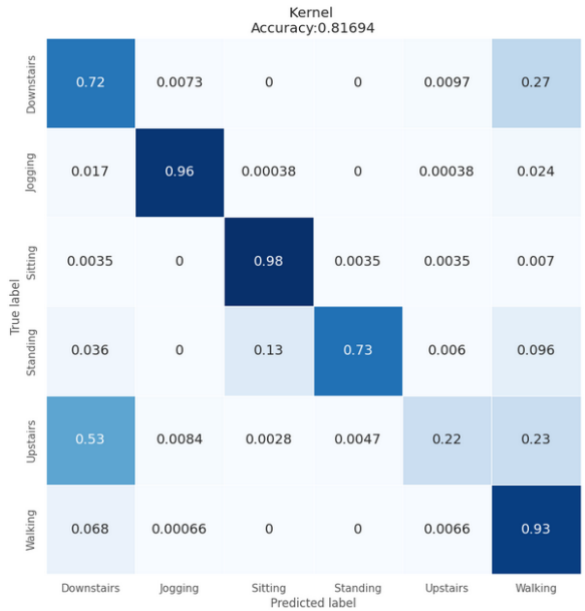
A.1.4. Kwantisatie en Pruning



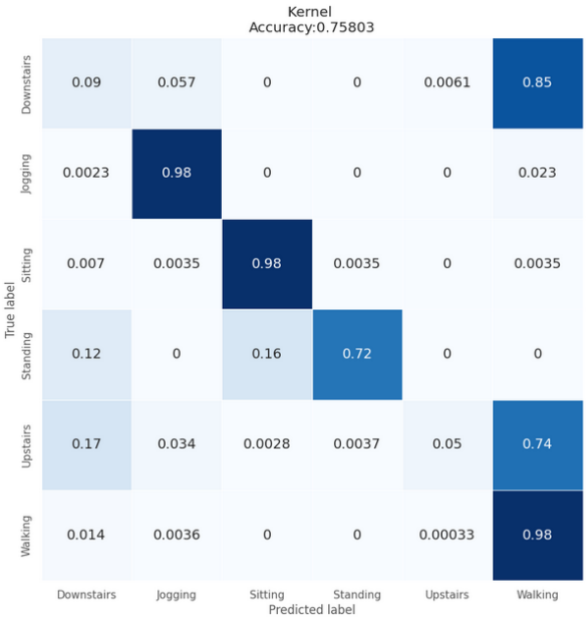
Figuur 57: *Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite met 25% uitdunning.*



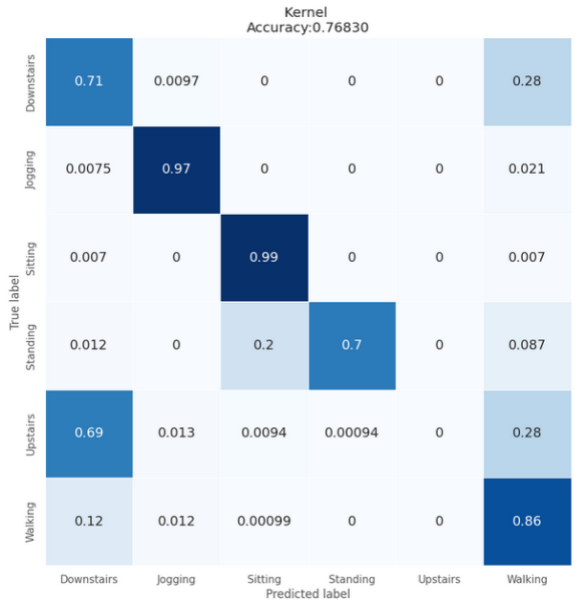
Figuur 58*: Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite met 50% uitdunning.*



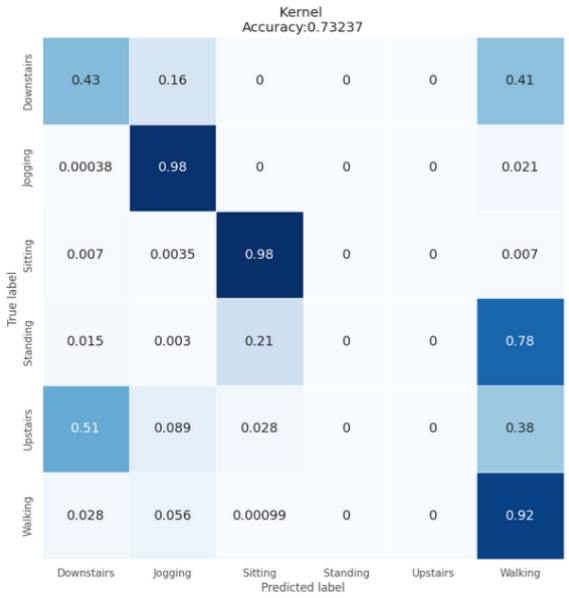
Figuur 59: *Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite met 75% uitdunning.*



Figuur 60: *Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite met 80% uitdunning.*

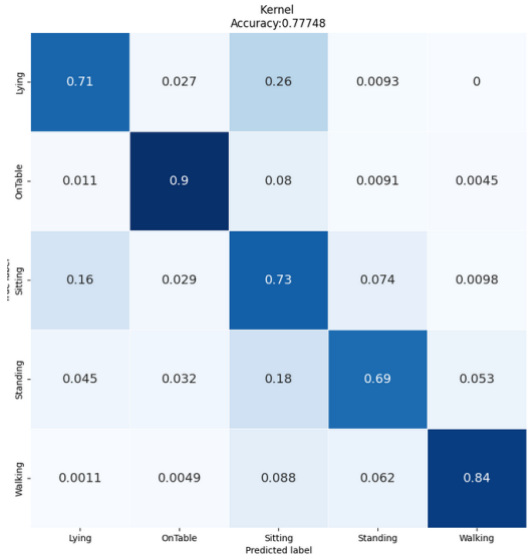


Figuur 61: Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite met 85% uitdunning.



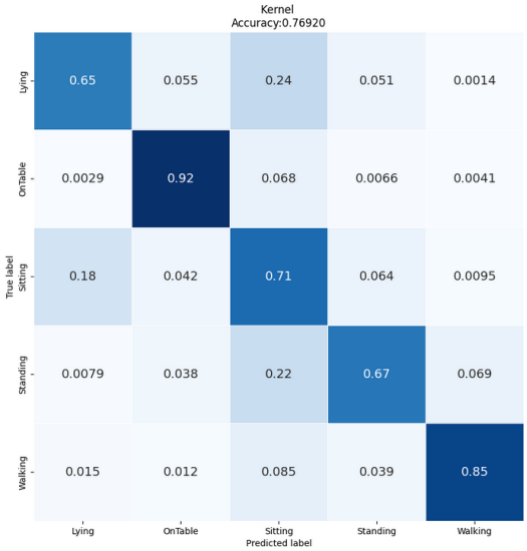
Figuur 62: Verwarringsmatrix TFLite-versie WISDM-model na de default optimizer van TFLite met 90% uitdunning.

A.2. mBrain-model



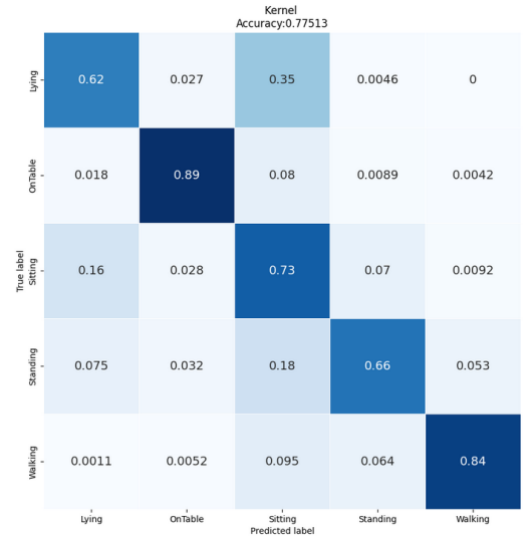
Figuur 63: *Verwarringsmatrix TFLite-versie mBrain-model.*

A.2.1. Depthwise Separable Convolutions

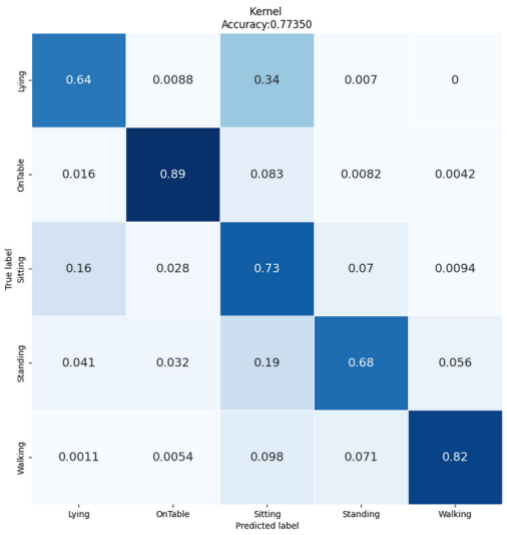


Figuur 64: *Verwarringsmatrix TFLite-versie mBrain-model met Depthwise Separable Convolutions.*

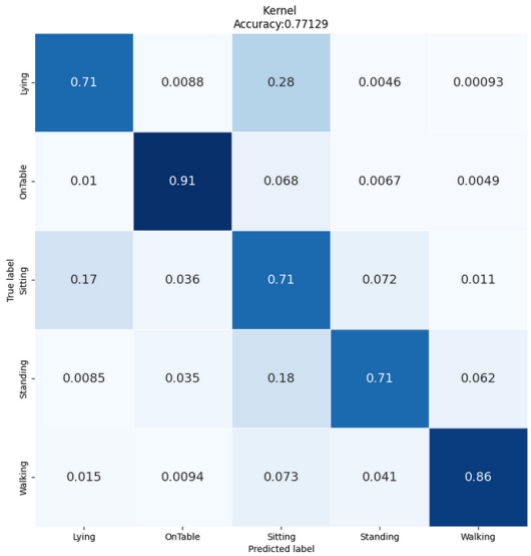
A.2.2. Pruning



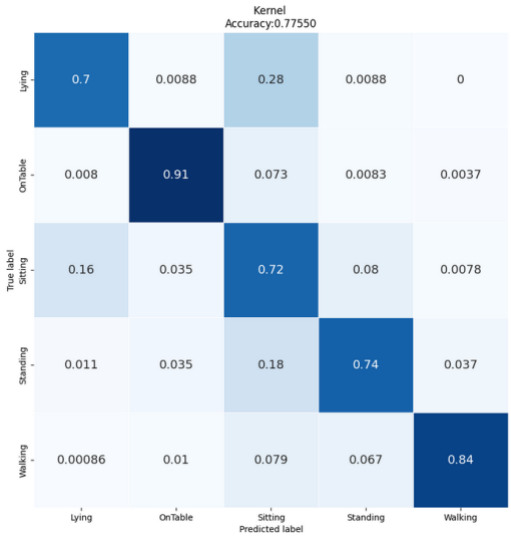
***Figuur 65**: Verwarringsmatrix TFLite-versie mBrain-model met 25% uitdunning.*



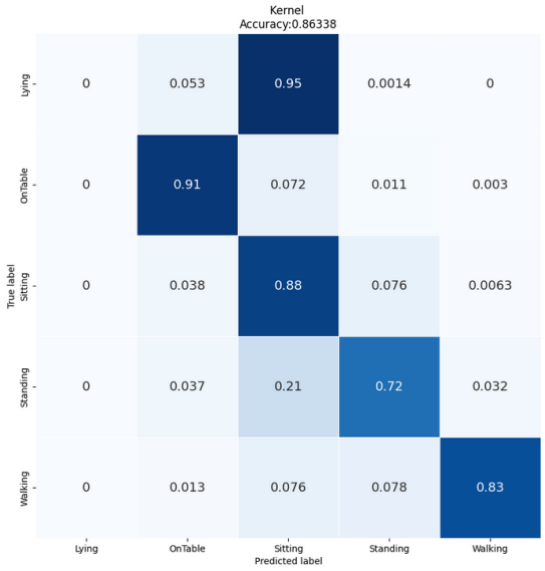
***Figuur 66:** Verwarringsmatrix TFLite-versie mBrain-model met 50% uitdunning.*



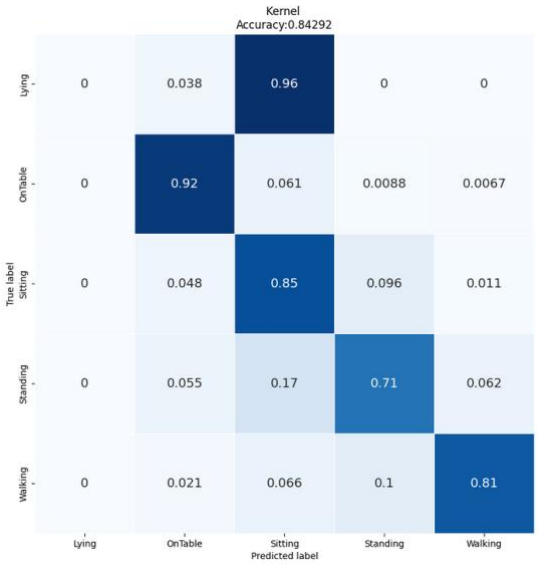
***Figuur 67:** Verwarringsmatrix TFLite-versie mBrain-model met 75% uitdunning*



***Figuur 68**: Verwarringsmatrix TFLite-versie mBrain-model met 80% uitdunning.*

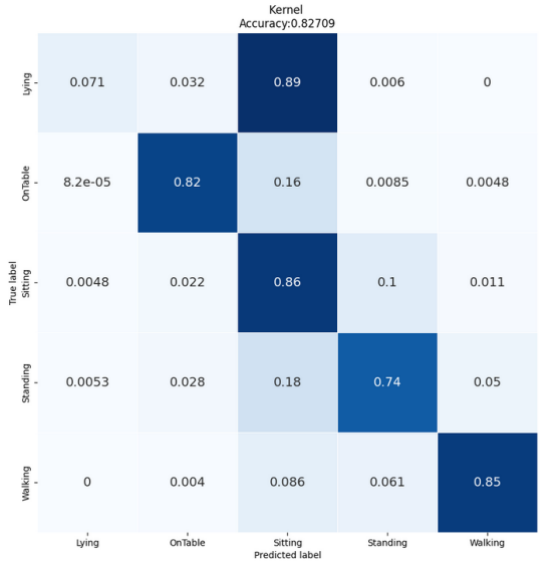


Figuur 69: *Verwarringmatrix TFLite-versie mBrain-model met 85% uitdunning.*

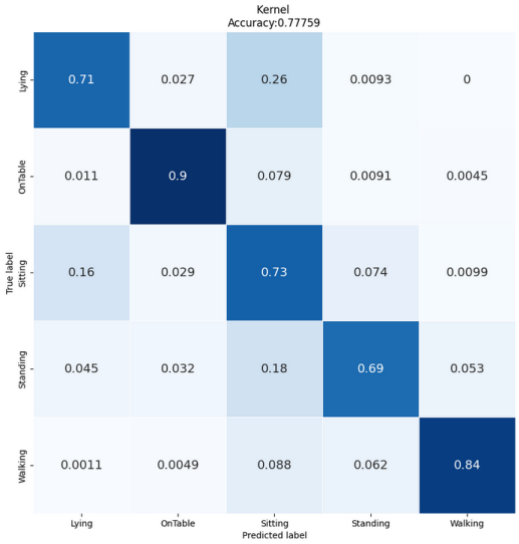


Figuur 70: *Verwarringsmatrix TFLite-versie mBrain-model met 90% uitdunning.*

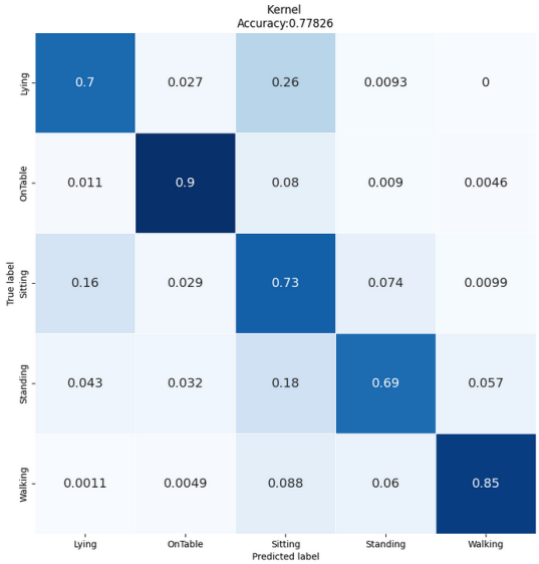
A.2.3. Kwantisatie



Figuur 71: *Verwarringsmatrix TFLite-versie mBrain-model na uint8-kwantisatie.*

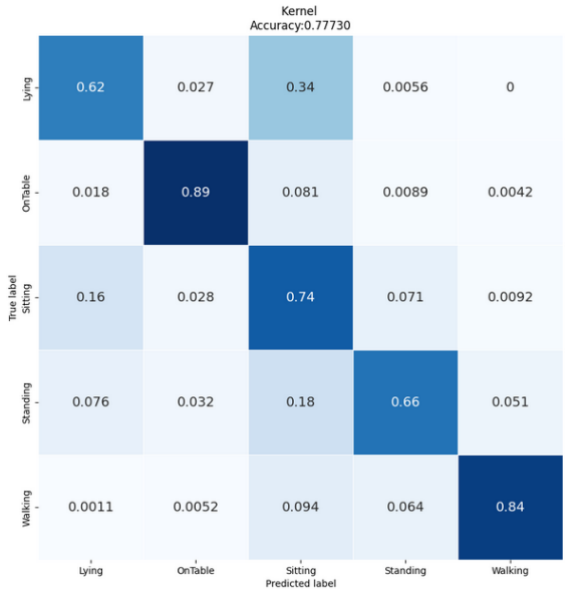


Figuur 72: *Verwarringsmatrix TFLite-versie mBrain-model na float16-kwantisatie.*

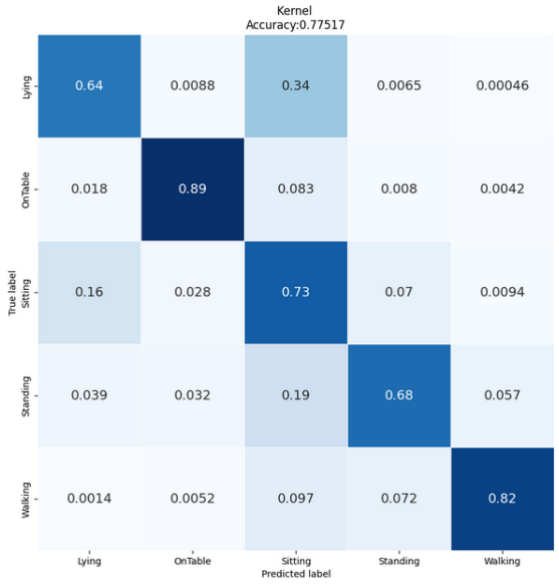


Figuur 73: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite.*

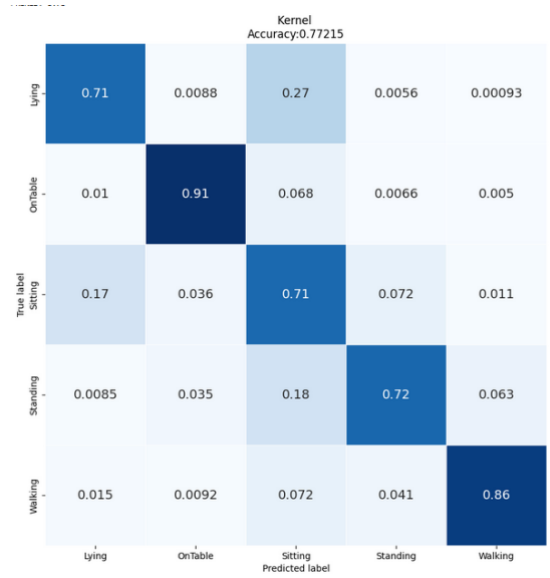
A.2.4. Kwantisatie en Pruning



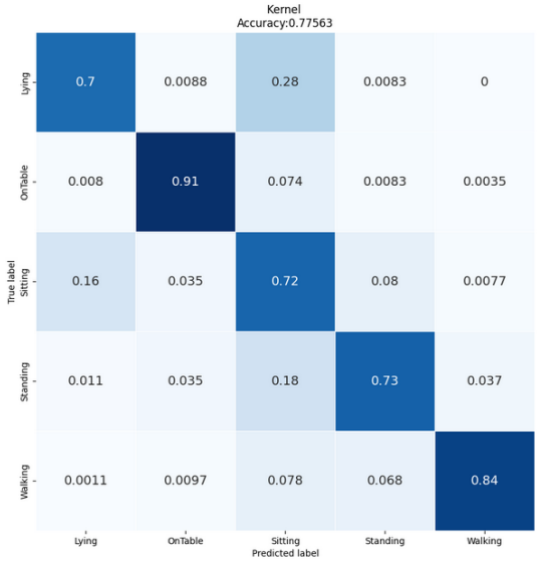
Figuur 74: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite met 25% uitdunning.*



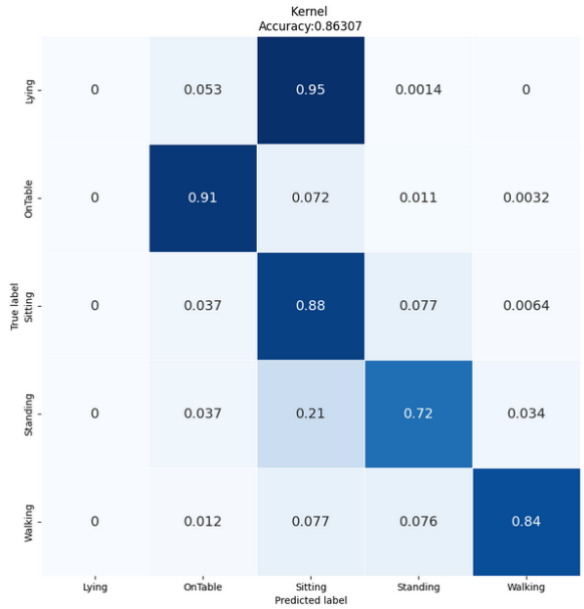
Figuur 75: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite met 50% uitdunning.*



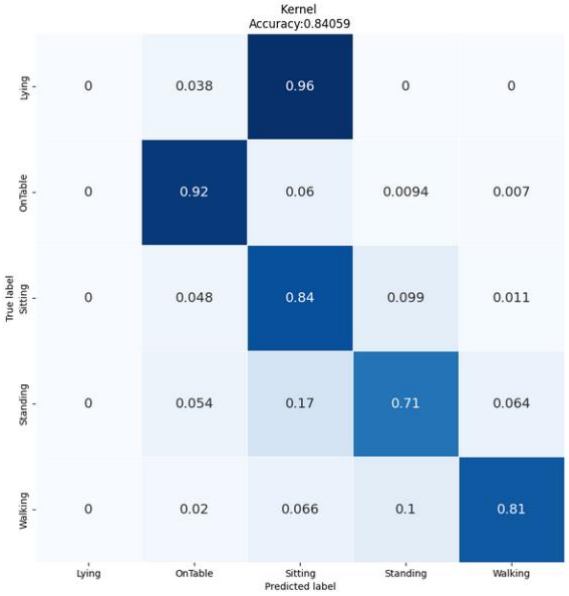
Figuur 76: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite met 75% uitdunning.*



Figuur 77: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite met 80% uitdunning.*



Figuur 78: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite met 85% uitdunning.*



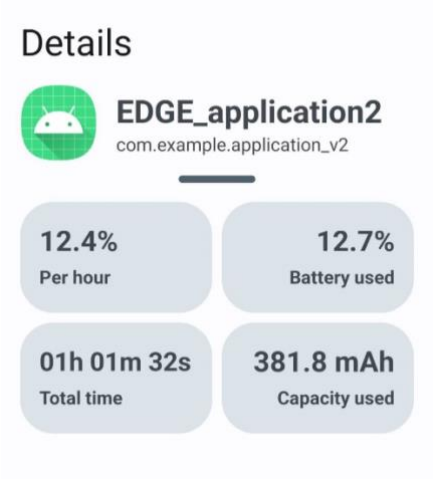
Figuur 79: *Verwarringsmatrix TFLite-versie mBrain-model na de default optimizer van TFLite met 90% uitdunning.*

B. Meetresultaten Batterijconsumptie

***Figuur 80**: Meetresultaat batterijconsumptie cloud-architectuur, scenario waarbij elke de data elke zes seconden wordt verwerkt.*

***Figuur 81:** Meetresultaat batterijconsumptie cloud-architectuur, scenario waarbij elke de data elke vijf minuten wordt verwerkt.*

***Figuur 82:** Meetresultaat batterijconsumptie cloud-architectuur, scenario waarbij elke de data elk uur wordt verwerkt.*



***Figuur 83**: Meetresultaat batterijconsumptie edge-architectuur, scenario waarbij elke de data elke zes seconden wordt verwerkt.*

***Figuur 84:** Meetresultaat batterijconsumptie edge-architectuur, scenario waarbij elke de data elke vijf minuten wordt verwerkt.*

***Figuur 85**: Meetresultaat batterijconsumptie edge-architectuur, scenario waarbij elke de data elk uur wordt verwerkt.*

Figuur 86: *Meetresultaten cloud-architectuur (tweede netwerk), scenario waarbij elke de data elke zes seconden wordt verwerkt.*

Figuur 87: *Meetresultaten cloud-architectuur (tweede netwerk), scenario waarbij elke de data elke vijf minuten wordt verwerkt.*

Figuur 88: *Meetresultaten cloud-architectuur (tweede netwerk) , scenario waarbij elke de data elk uur wordt verwerkt.*