SSC24-WP2-03

Putting the "Space" in Hyperspace: Investigating Hyperdimensional Computing for Space Applications

Ian R. Peitzsch, Evan W. Gretok, Alan D. George
NSF Center for Space, High-Performance, and Resilient Computing (SHREC)
Suite 560, 4420 Bayard Street, Pittsburgh, PA 15213;
ian.peitzsch@nsf-shrec.org

ABSTRACT

Earth observation (EO) is currently a major application area of satellite operations. As more powerful imagers become more accessible and see increased use, these EO images will increase in size, which also increases the computational complexity of classifying them. Using neural networks for this task has limitations due to the small memory and compute capability possessed by embedded platforms. Instead, this research investigates a shift in machine-learning paradigms from neural networks to hyperdimensional computing (HDC). HDC uses very large vectors to represent and draw relations from data. HDC often has a lower latency, power usage, and memory footprint than neural networks. Using the EuroSAT dataset, this research achieved $> 1.4\times$ speedup and energy efficiency using an HDC model over a convolutional neural network. Though, this improvement came at the cost of 4% lower accuracy. These results indicate HDC is well suited for machine-learning tasks in space.

INTRODUCTION

In recent years, significant effort has been spent researching onboard machine learning for space applications, most notably apps using neural networks. However, neural networks often require significant onboard memory and lots of power. These challenges make neural networks nontrivial to use in space, but the benefits they offer, such as autonomy, make them a hot topic for space-computing research.

While there exist multiple solutions to overcome some of these challenges, one possible solution is shifting from neural networks to alternative paradigms, such as hyperdimensional computing (HDC). HDC generally offers smaller models, lower power consumption, and shorter training times compared to neural networks. These benefits often come at the cost of accuracy. This research investigates the trade offs between accuracy, latency, memory, power, and energy when using an HDC-based model instead of a neural network for the space-domain application of tile classification.

The main contribution of this research is evaluating the performance and tradeoffs of using an HDC model instead of a conventional neural network for tile classification. This contribution is achieved by collecting and comparing test accuracy, inference latency, memory usage, power consumption, and energy consumption on multiple compute platforms commonly used for space computing. This research does not make any algorithmic contributions. Rather, the goal of this research is to inform members of the space computing community of these tradeoffs, so they can better assess using HDC in their future missions.

BACKGROUND

This section first gives an introduction to HDC and how it can be applied to classification tasks. Then, commonly used deep-learning models are introduced. Finally, relevant Earth-observation datasets are given.

Hyperdimensional Computing

Hyperdimensional computing (HDC) is a method of computing which performs symbolic computation on data represented as very high dimensional vectors (*hypervectors*).[1] This high dimensionality makes it so independent and identically distributed randomly generated hypervectors are near orthogonal.[2] Thus, semantically different objects can be represented as different randomly generated hypervectors. In recent years, a machine-learning paradigm based on HDC has emerged as an alternative to traditional neural-network methods. HDC often achieves better energy efficiency, lower latency, and additional robustness to noise, making it of interest for embedded and near-sensor computing scenarios.[3] These benefits often come at a cost in accuracy. However,

more recent advancements in HDC techniques have significantly reduced this gap[4,5].

HDC uses a similarity function (δ) and a set of three data transformation operations: bundling ($\oplus$), binding ($\otimes$), and permutation (ρ). The similarity function compares the orthogonality between hypervectors. If for hypervectors A, B, the similarity $\delta(A, B) \approx 0$, then they are orthogonal and considered unrelated. Alternatively, if $\delta(A, B) >> 0$, then they are considered related. Commonly, this function is implemented as the cosine similarity function, $\delta(A, B) = \frac{A \cdot B}{|A||B|}$. The bundling operation combines the data represented by multiple hypervectors into a single hypervector. This resultant hypervector has high similarity to both input hypervectors, so $\delta(A, A \oplus B) >> 0$ and $\delta(B, A \oplus B) >> 0$. Bundling is often simply implemented as vector addition. The binding operation associates two hypervectors into a resultant hypervector that is dissimilar to its inputs, so $\delta(A, A \otimes B) \approx 0$ and $\delta(B, A \otimes B) \approx 0$. Binding is often implemented as XOR for binary hypervectors, and more complex operations like circular convolutions for real-valued hypervectors. The permutation operation rotates the input hypervector. This resultant hypervector is dissimilar to the input, so $\delta(A, \rho(A)) \approx 0$.

Using these operations and the similarity function, a clustering-based classification method can be built. Training a model using this method of classification starts by encoding the input features into their corresponding hypervectors. This mapping is achieved in many different ways, but often uses record-based encoding, matrix-vector multiplication, binding, and permutation. For each class, there is a corresponding hypervector which represents that class. After encoding, each encoded hypervector is bundled into its corresponding class hypervector, forming centroids for each class. Due to the similarity properties of bundling, these centroids will be similar to all the training data that make up each class.[2,6] Further fine-tuning can be performed in subsequent training passes. This fine-tuning is often performed by calculating the similarity of the encoded hypervector with each class hypervector. If the class corresponding to that encoded hypervector has the highest similarity of all the class hypervectors, then no updates are performed and the process continues with the next encoded hypervector. However, if the class does not have the highest similarity, then an update is performed where the encoded hypervector is subtracted from the class that has the highest similarity and bundled into the class the encoded hypervector is supposed to correspond to. This process is represented in the following equations:

$$C_{l'} = C_{l'} - \alpha A$$
$$C_l = C_l \oplus \alpha A$$

Where A is the encoded hypervector, $C_{l'}$ is the hypervector corresponding to the incorrect class, C_l is the hypervector corresponding to the correct class, and α is a scalar update value that adjusts how much to update the classes.[7,8] This method of classification has been applied to various tasks, such as language recognition,[9,10] gesture recognition,[11] seizure detection,[11,12] and simple image classification.[5,13]

Specific efforts into using HDC methods for classification on more complex image datasets, such as ImageNet, CIFAR-10, and CIFAR-100, have been more common in recent years. One such research culminated in PIONEER[13] which uses a learned encoding mapping to improve the accuracy of the HDC model. Using PIONEER lead to an increase from 4% accuracy on CIFAR-10 using vanilla HDC, to 50.10% accuracy. However, this increase in accuracy still lagged behind the 92% accuracy achieved by the ResNet-18 model that was used as a baseline. Though, for single epoch training, PIONEER outperformed ResNet-18, with PIONEER having 8.30% higher single-epoch accuracy than ResNet-18 (46.06% vs 37.76%). PIONEER achieved these accuracies while maintaining a traditional HDC model that consists of only an encoder and clustering. What seems to show more promise in this area, however, is methods that combine other machine-learning paradigms with HDC. An example of this concept is demonstrated in Dutta et al.'s[4] research which combined hyperdimensional and neural network (HDnn) methods for classification on the CIFAR-10, CIFAR-100, and FLOWERS datasets. Their HDnn model achieved 95.1%, 78.3%, and 88.8% accuracy on these datasets, respectively, while ResNet-18 achieved 94.6%, 78.7%, and 84.7% accuracy. Additionally, HDnn had fewer MAC operations and parameters than ResNet-18. Due to this high accuracy, HDnn is the main HDC method of interest in this research.

Deep-Learning Models

In order to compare HDnn to previous efforts, we also benchmark a MobileNetV2 model.[14] MobileNetV2 improved on the first MobileNet architecture[15] with architectural optimizations for improved accuracy and performance. MobileNetV2 has been featured on several previous onboard-classification studies by this group.[16,17] While MobileNetV3[18]

has since been released, further optimizations targeted GPU- and TPU-based accelerators, not CPU inference as featured in this paper.

The state-of-the-art method for feature extraction in image classification has arguably transitioned from convolutional neural networks to transformer backbones. However, mobile-optimized transformer models, such as MobileViT,[19] are comparable in parameter count and computational requirement to the MobileNetV2 model featured in this study. Extending this study to transformer models is a potential avenue of future research.

Earth-Observation Datasets

There are a number of Earth-observation datasets available for image-classification training. Among the first is the UC Merced dataset,[20] which includes 21 classes of land cover and 100 images per class at very low ∼0.3 m/px GRDs. The most widely used land cover dataset is NWPU-RESISC45.[21] This source includes 700 images in each of the 45 classes at variable GRDs also as low as ∼0.3 m/px. Cheng et al.'s research,[21] the origin of NWPU-RESISC45, also offers a comparison of deep-learning land-cover classification to previous methods. This comparison to previous methods also enables further comparison of HDnn to more classical options.

The dataset used most extensively in this study is EuroSAT.[22] This data is based on Earth-observation imagery from the Sentinel-2 satellite, a key component of the ESA Copernicus system. EuroSAT includes 2700 64-pixel square image tiles for each of ten land-cover classes. These images are much smaller than the typical 256-pixel square imagery in UC Merced and NWPU-RESISC45. The highest-resolution MobileNetV2 model is pre-trained on 224-pixel square images. This discrepancy sometimes makes it difficult to compare the datasets. Smaller image sizes can reduce inference runtimes, but may also result in lower accuracy. However, the smaller sizes of EuroSAT imagery are a better fit for the vector space of HDnn as the smaller feature space allows for greater benefit from the larger hyperdimensional space.

APPROACH

This research benchmarks various metrics of an HDC-based model and MobileNetV2 on multiple space-grade CPU architectures. The HDC model benchmarked is the HDnn method described in.[4] The HDnn models used in this research use the first four convolutional blocks of MobileNetV2 as a feature extractor. Two variants of HDnn are benchmarked in this research, one using 1000 hyperdimensions (HDs) and another using 4000 HDs. The HDnn model uses projection matrices for the encoding schemes. The MobileNetV2 model used is based on the model used in Evan W. Gretok and Alan D. George's research.[17] This model was trained using transfer learning, so it uses all of the convolutional layers with the pretrained Imagenet weights, and then has a single dense layer appended.

The metrics of interest in this research are accuracy, inference latency, and model size. The EuroSAT-RGB image dataset was the focus for this study. This dataset consists of 64×64 pixel color images of satellite imagery from the Sentinel-2 satellite.

Multiple test platforms were used for this research. The first platform was for training the HDnn models. This platform was implemented on Intel's DevCloud and used the TensorFlow2 library. The training was performed on an Intel Xeon Gold 6128 CPU. The other platforms were used to benchmark inference latency, memory usage, power consumption, and energy consumption. The platforms used were an NVIDIA Jetson AGX Orin (ARM Cortex-A78), a Raspberry Pi 4 (ARM Cortex-A72), and a Raspberry Pi 3 (ARM Cortex-A53). These platforms were chosen as they contain the same processor architectures representative of commercial-off-the-shelf compute platforms commonly considered for space applications, especially in the SmallSat domain. The models benchmarked on these platforms were converted to ONNX and made use of ONNX Runtime.

Model Architecture

The HDnn models used in this research were constructed with some of the convolutional layers from MobileNetV2 as the feature extractor. These layers were pretrained on the ImageNet dataset and frozen. This research uses the output of the `block_3_depthwise_relu` layer as the input feature vector to the HDC model. For the encoder, a series of randomly generated basis vectors $\vec{B_1}, ..., \vec{B_D}$ with D being the dimension of the hypervectors. Then, for each feature vector $\vec{F}$ from the feature extractor, the corresponding hypervector $\vec{H} = \{h_1, ..., h_D\}$ is calculated as follows:

$$h_i = \cos(\vec{B_i} \cdot \vec{F}) \times \sin(\vec{B_i} \cdot \vec{F})$$

To train the models, five passes through the training data are used, using the retraining method described previously. These extra training passes allow for the

model to achieve a higher accuracy than just a single pass.

The MobileNetV2 model used is based on the model used in related literature.[23] This model was trained using transfer learning. Thus, it uses all of the convolutional layers with the pretrained Imagenet weights, and then has a single dense layer appended to this pretrained model.

Benchmarking

Inference runtime data for each model type on each platform were collected using ONNX Runtime. Individual iterations were used to collect individual runtimes. Runtimes were averaged over three sets of 100 inferences.

Memory consumption data for each model type on each platform were collected using a combination of the free and process status. Free measures were conducted in kilobytes with the device idle and at peak inference load. The resident set sizes (RSS) in the process status output were also recorded. Memory measures were conducted using internal inference iterations of ONNX Runtime, as this method resulted in more stable, worst case, and easier to measure results.

Power consumption data for each model type on each platform was collected with a Ponnie PN2000 power meter. Minimum idle power and maximum load power were measured to ensure the worst-case values. Power was also measured using internal inference iterations of ONNX Runtime as this method provided similar improved stability in power consumption. Dynamic power is calculated by subtracting the idle measure from the load measure. Total and dynamic energy measures are calculated as the product of inference runtime with total and dynamic power measures, respectively.

RESULTS

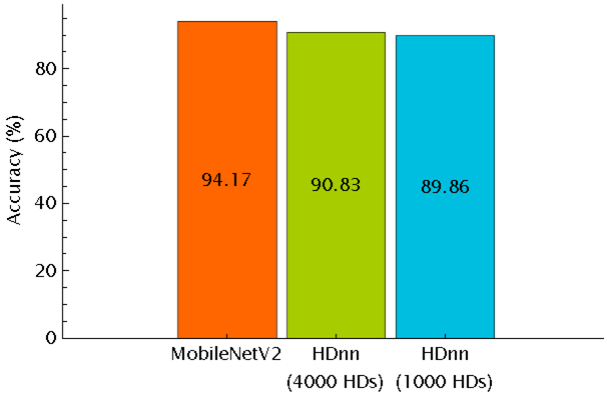

Figure 1: Test accuracy of each model on the EuroSAT-RBG dataset

Fig. 1 shows the accuracy of the tested models on the EuroSat-RGB dataset. MobileNetV2 achieves the highest accuracy at 94.17%, with the two variants of HDnn achieving accuracies near 90%. There is little difference in accuracy between the HDnn variant using 1000 hyperdimensions and the variant using 4000 hyperdimensions, so all further references to HDnn in this section are referring to the 1000 hyperdimension variant, as it has fewer parameters.

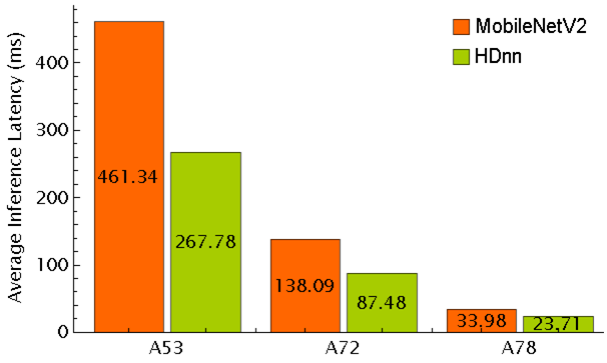

Figure 2: The average inference latency values across the tested devices.

Fig. 2 provides the average inference latency values for both the MobileNetV2 model and HDnn model on various ARM CPUs. HDnn consistently has lower latency, achieving speedup values of 1.72×, 1.58×, and 1.4× over MobileNetV2 on the ARM Cortex-A53, Cortex-A72, and Cortex-A78, respectively. While HDnn maintains lower latency values, the improvement over MobileNetV2 decreases as the performance of the CPU increases.

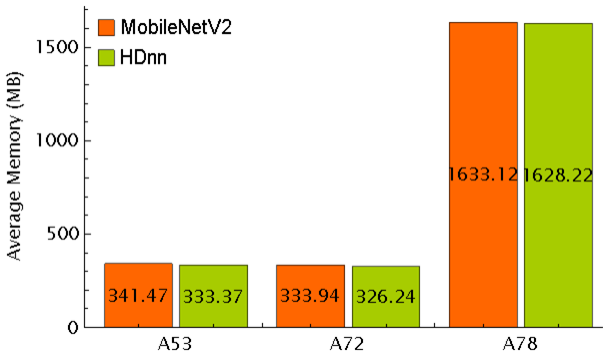

Figure 3: The peak memory consumption values across the tested devices.

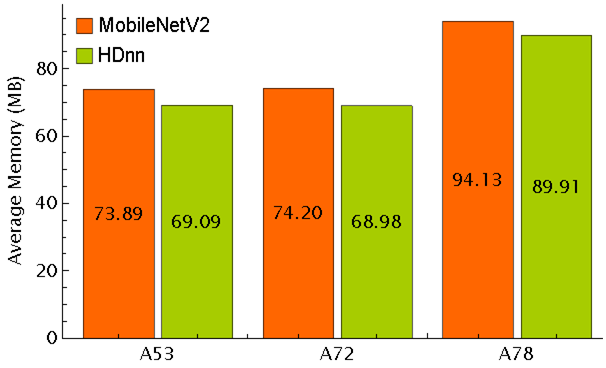

Figure 4: The peak resident set size across the tested devices.

Fig. 3 depicts the memory usage of both models. There is little difference between the memory usage between the two models across all tested hardware, though HDnn has consistently lower peak memory consumption. Fig. 4 shows the resident set memory allocated to the process. Here as well, HDnn generally has lower allocated memory than MobileNetV2, though the difference is not large.

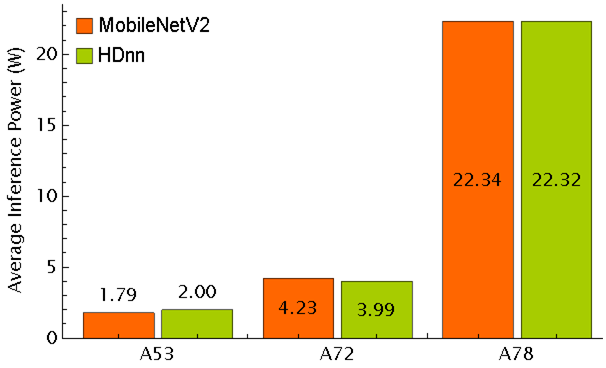

Figure 5: The dynamic power consumption values across the tested devices.

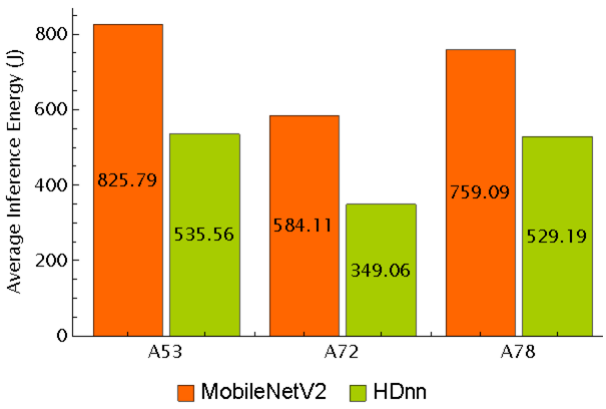

Figure 6: The dynamic energy consumption values across the tested devices.

Fig. 5 and Fig. 6 show the dynamic power consumption and the dynamic energy consumption. The difference between the dynamic power consumption of both models is very small and there is no consistent trend across all hardware. There is, however, a major difference between the dynamic energy consumption of the models. HDnn consistently has lower energy consumption than MobileNetV2, achieving 1.54×, 1.67×, and 1.43× energy improvement over MobileNetV2 on the tested ARM Cortex-A53, Cortex-A72, and Cortex-A78, respectively.

DISCUSSION

For the metrics of interest, MobileNetV2 performed best for accuracy while HDnn performed best for latency and energy consumption. From previous research, this result is mostly as expected. Below explanations and implications are given for each of the benchmarked metrics.

Accuracy

As previously stated, HDnn achieved lower accuracy than MobileNetV2. This reduction in accuracy is counter to the results presented in the original HDnn[4] paper. This result has multiple possible causes, the first being a suboptimal feature extraction method. The model used in this research uses the output of a single deep-convolutional block from MobileNetV2, which has previously been shown to perform well for image-classification tasks using HDnn.[4] However, there are other methods that could extract more information from the images, such as the method demonstrated in Wilson et al.'s research,[24] which encodes the outputs of multiple layers and bundles them together to form the query vector. HDnn's method of using the output

of a single layer was chosen instead of this bundling method, as it has performed well on relatively difficult datasets, such as CIFAR-100 and FLOWERS, but future research investigating other feature extraction methods is necessary to investigate the full capability of HDC-based methods.

The second likely cause is the the model used as the feature extractor. The original HDnn paper evaluated four models to uses as feature extractors: ResNet-18, ResNet-34, VGG-16, and MobileNetV2. This research evaluated only MobileNetV2 to give a fair comparison to prior research done using MobileNetV2 for tile classification,[17] but again further research investigating these other models as potential feature extractors would help optimize this model. The third cause is HDC generally having lower representation power than neural networks, especially on images. In spite of all these factors, HDnn is well suited for the task of tile classification, with it still achieving $> 90\%$ accuracy with $> 1.4\times$ inference speedup, especially in situations with strict latency/throughput and energy constraints.

Latency

As stated in the results section, HDnn exhibits lower inference latencies than the MobileNetV2 model on all platforms tested. This result is expected, as HDnn has a shorter critical datapath length[25] and fewer operations to perform, so it has both less work to do and is able to experience more speedup with multi-threading. Interestingly though, MobileNetV2's performance improves more with increasing compute power than HDnn's performance. This trend could simply be due to MobileNetV2 performing exceptionally poorly given the constraints of the A53. So, it has a greater capability of improving from the A78's increased clock speed, increased number of cores, and added out-of-order execution. Though, while the performance gap does shrink, HDnn still always has the lower latency. Thus, for low latency operations, and likely high throughput operations, it is the better option to use.

Memory Usage

Both the tested HDnn variant with 4000 hyperdimensions and MobileNetV2 had similar memory usage values. These results are surprising as HDnn had far fewer parameters than the MobileNetV2 variant, with 638,912 parameters (2.44 MB) and 2,270,794 parameters (8.66 MB), respectively. Thus, based solely on parameters, HDnn is $3.5\times$ more memory efficient than the MobileNetV2 variant; however, this difference is not observed with either the peak memory consumption or the peak resident set size. Added overhead from ONNX Runtime is likely dominating the memory usage for both models.

Power and Energy

Difference in overall power consumption between model types was not significant enough to draw conclusions. This behaviour was consistent across any and all platforms tested. As noted in the results section, overall energy consumption was reduced primarily due to the reduced inference time of the HDnn model. One item of concern is the relatively high power consumption of the A78 platform. The onboard GPU, DLA, vision system, and other components were not included in the ONNX Runtime tests performed, and no other active processes were running. Higher-power supporting components of the NVIDIA Jetson AGX Orin may have biased some of the power measures from this device. It is possible that a CPU-only A78 platform may provide lower power and energy consumption. Further investigation of this platform will be conducted as this study is extended to embedded GPU platforms. It is also worth noting that the A78 CPU has a greater focus on performance than energy efficiency compared to the A72 and A53. The AGX Orin also features a 12-core A78 compared to the quad-core A72 and A53 tested. However, investigation of how ONNX Runtime handles multithreading performance was beyond the scope of this study.

CONCLUSION

This research investigated the application of HDC for the space app of tile classification of satellite imagery. To achieve this goal, HDnn was leveraged and compared to a method using MobileNetV2 across accuracy, inference latency, memory usage, power consumption, and energy consumption. MobileNetV2 achieved the highest accuracy at 94.17%; however, HDnn saw only a $\approx$4% decrease in accuracy. HDnn still achieved speedup and reduced peak energy consumption when compared to MobileNetV2. Thus, there is potential for HDnn to be effectively applied for satellite tile classification for applications where speed and energy are major concerns.

FUTURE RESEARCH

This research could be extended to investigate additional feature extractors from the original HDnn

paper, including ResNet-18, ResNet-34, and VGG-16. Feature-extraction methods using transformers, such as MobileViT or similar, could also be investigated. Additionally, there is ongoing research into collecting similar performance metrics on both GPUs and FPGAs for both MobileNetV2 and HDnn.

ACKNOWLEDGEMENTS

This research was supported by SHREC industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783. The authors would like to especially thank Calvin Gealy, another student in the NSF SHREC Center, who provided invaluable benchmarking assistance and is helping to extend this research to embedded GPUs. The authors would like to thank additional fellow students from the NSF SHREC Center who provided peer review, including James Bickerstaff and Owen Lucas.

References

[1] Peer Neubert, Stefan Schubert, and Peter Protzel. An introduction to hyperdimensional computing for robotics. *KI - Künstliche Intelligenz*, 33(4):319–330, September 2019.

[2] Tao Yu, Yichi Zhang, Zhiru Zhang, and Christopher De Sa. Understanding hyperdimensional computing for parallel single-pass learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[3] Abbas Rahimi, Sohum Datta, Denis Kleyko, Edward Paxon Frady, Bruno Olshausen, Pentti Kanerva, and Jan M. Rabaey. High-dimensional computing as a nanoscalable paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2508–2521, 2017.

[4] Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, Weihong Xu, and Tajana Rosing. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, GLSVLSI '22, page 281–286, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Zhuowen Zou, Yeseong Kim, Farhad Imani, Haleh Alimohamadi, Rosario Cammarota, and Mohsen Imani. Scalable edge-based hyperdimensional learning system with brain-like neural adaptation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[6] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. A theoretical perspective on hyperdimensional computing. *J. Artif. Int. Res.*, 72:215–249, jan 2022.

[7] Lulu Ge and Keshab K. Parhi. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20(2):30–47, 2020.

[8] Mohsen Imani, Samuel Bosch, Sohum Datta, Sharadhi Ramakrishna, Sahand Salamat, Jan M. Rabaey, and Tajana Rosing. Quanthd: A quantization framework for hyperdimensional computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2268–2278, 2020.

[9] Aditya Joshi, Johan Halseth, and Pentti Kanerva. Language Recognition using Random Indexing. *arXiv e-prints*, page arXiv:1412.7026, December 2014.

[10] Geethan Karunaratne, Abbas Rahimi, Manuel Le Gallo, Giovanni Cherubini, and Abu Sebastian. Real-time language recognition using hyperdimensional computing on phase-change memory array. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–1, 2021.

[11] Abbas Rahimi, Pentti Kanerva, Luca Benini, and Jan M. Rabaey. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals. *Proceedings of the IEEE*, 107(1):123–143, 2019.

[12] Yipeng Du, Yuan Ren, Ngai Wong, and Edith C. H. Ngai. Hyperdimensional computing with multi-scale local binary patterns for scalp eeg-based epileptic seizure detection. *IEEE Internet of Things Journal*, pages 1–1, 2024.

[13] Fatemeh Asgarinejad, Justin Morris, Tajana Rosing, and Baris Aksanli. Pioneer: Highly efficient and accurate hyperdimensional computing using learned projection. In *2024 29th Asia*

and South Pacific Design Automation Conference (ASP-DAC), pages 896–901, 2024.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 4 2017.

[16] Jacob Manning, David Langerman, Barath Ramesh, Evan Gretok, Christopher Wilson, Alan D. George, James Mackinnon, and Gary Crum. Machine-Learning Space Applications on SmallSat Platforms with TensorFlow. *32nd Annual AIAA/USU Conference on Small Satellites*, 2018.

[17] Evan W. Gretok and Alan D. George. Onboard multi-scale tile classification for satellites and other spacecraft. In *2021 IEEE Space Computing Conference*, pages 110–121. IEEE, 8 2021.

[18] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Quoc Le, and Hartwig Adam. Searching for mobileNetV3. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-October, 2019.

[19] Sachin Mehta and Mohammad Rastegari. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. In *International Conference on Learning Representations*, 2022.

[20] Yi Yang and Shawn Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 2010.

[21] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote Sensing Image Scene Classification: Benchmark and State of the Art, 2017.

[22] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. *IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING*, 12(7), 2019.

[23] Evan W. Gretok and Alan D. George. Onboard multi-scale tile classification for satellites and other spacecraft. In *2021 IEEE Space Computing Conference (SCC)*, pages 110–121, 2021.

[24] Samuel Wilson, Tobias Fischer, Niko Sünderhauf, and Feras Dayoub. Hyperdimensional feature fusion for out-of-distribution detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2644–2654, January 2023.

[25] David Langerman, Alex Johnson, Kyle Buettner, and Alan D. George. Beyond Floating-Point Ops: CNN Performance Prediction with Critical Datapath Length. In *2020 IEEE High Performance Extreme Computing Conference, HPEC 2020*. Institute of Electrical and Electronics Engineers Inc., 9 2020.