Performance Analysis of Event Based Recognition Networks

GDL 2023
Group id: 9
Project id: AEGNN

Anthony Bugatto, Filippo Casari, Alessandro De Grandi
{anthony.bugatto, filippo.casari, alessandro.degrandi}@usi.ch

Abstract

In recent years neuromorphic computing has emerged as a new paradigm for information processing. Modeled after the human retina, event cameras offer low latency, low power consumption, and high dynamic range compared with regular cameras but produce an asynchronous and irregularly structured continuous data stream rather than an evenly spaced grid at discrete time steps. To process this new form of data efficiently for computer vision tasks new neural models such as graph neural networks have emerged as the current state of the art. In this reproduction study we choose one such class of networks, Asynchronous Event-based Graph Neural Networks, and analyze their characteristics, efficiency, and design decisions. Finally we improve the reproducibility of their code base, test against multiple datasets, compare results with competing models, and suggest some potential future modifications. We find that the model does in fact have significant efficiency gains for highly accurate recognition yet is no longer state of the art in accuracy due to recent advances in graph convolutional networks, event transformers, and spiking neural networks.

1 Introduction

Neuromorphic computing, or computing inspired by the brain, is becoming increasingly important in the fields of photography, computer vision, robotics, and real time computing systems. Like biological neural networks, these information processing systems typically function using asynchronous spikes, mimicking biological neurons, rather than traditional synchronous digital logic processing. Described by a dynamical system due to their time dependence [5], these networks not only have the potential to allow online learning that could rewire the network during runtime [hebbian learning] similar to biological neural networks but have the potential to be orders of magnitude faster and less power hungry than traditional computers. However, they are much more challenging to train than classical neural networks [5], have only recently achieved state of the art accuracy in tasks such as robotics [5] and vision [5], and require neuromorphic sensor data to achieve their full potential.

Though it is possible to convert traditional data into spikes for spiking neural networks [24], to produce native neuromorphic data suitable for spiking neural networks, neuromorphic sensors have been developed for vision, audio, and actuation [5]. For photography and computer vision, event cameras were designed to closely mimic biological human retina function [6], resulting in cameras that output a sparse, asynchronous ”event” stream which tracks changes in pixel intensity while delivering higher temporal resolution, lower latency, lower power, and higher dynamic range. Due to the challenges of spiking neural networks in practice, the majority of state of the art research in event based signal processing is done using classical neural networks, with graph neural networks recently emerging as a suitable candidate to handle this form of data efficiently and easily [5]. One recent and popular solution for visual recognition problems is the Asynchronous Event-based Graph Neural Network, or AEGNN [1], which takes advantage of the sparse and asynchronous nature of event camera streams to produce a network that is orders of magnitude more efficient in training and inference time while retaining the same state of the art accuracy as previous models.

In this work we run a reproduction study on both the theoretical and experimental components of the network, building on both the original research as well as more recent github forks. We show that the network does indeed run with orders of magnitude higher efficiency while retaining near state of the art accuracy in recognition and detection tasks. However we also show that more recent models on the same datasets can achieve both better accuracy and efficiency, culminating in an analysis of potential modifications that could be made to AEGNN package in order to once again have state of the art performance. Additionally, we have modified the code base to include pretrained models and a training script for both the cloud and personal computers. We believe that this will lower the barrier to future research and applications of AEGNN’s while helping increasing transparency in the research community.

2 Related works

Event based deep learning has been attempted with a myriad of different architectures, ranging from graph neural networks to spiking neural networks. Due to the nature of event streams, methods can be grouped by four different kinds of input data assumptions: spikes [24,25], time surfaces [27], event histograms [28], and spatiotemporal event graphs [1]. Generally, the field seems to be moving towards more event native models that don’t have as much preprocessing, with time surfaces and event histograms being the furthest awat, event graphs being closer, and spiking nets being event native. The more natively the network works

with events the easier it will be for researchers to create models and solve event based problems.
On the experimental side, there have been informal additions to the AEGNN package in the form of github forks. The github fork done by user Ercbunny [2] includes not only replication experiments to build off, but a training script that wasn omitted from the original repository.

3 Prerequisites

3.1 Event Based Vision

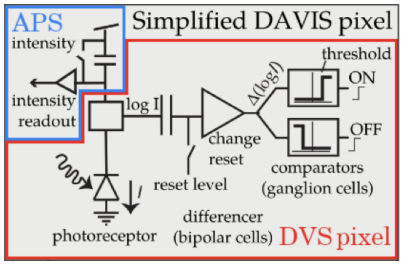


Figure 1. DAVIS Sensor Pixel Schematic [5, DAVIS]

Event cameras are designed much like a regular camera but instead of an APS [5] circuit detecting pixel intensity at each shutter iteration, it has a DVS [8] circuit that detects changes in the intensity outside a threshold at each pixel asynchronously. Because of the asynchronous nature of the DVS sensor we have significantly higher temporal resolution compared to the APS sensor, with asynchronous readout coming in at roughly 1 MHz [5] compared to 40Hz on a high end APS camera [5]. This allows the camera to capture very fast motion compared to a traditional APS camera. DVS event cameras, due to the asynchronous nature of events have extremely low latency between events with as low as $t_latency = 10\mu s$ [5]. Because of the DVS's ability to adapt to very dark as well as very light brightness conditions there is significantly higher dynamic range, with greater than 120 dB compared to 60dB in an APS camera [5]. Additionally, since event cameras only record brightness changes they are over 100x more efficient than APS cameras, with $10\mu W$ DVS vs $10mW$ APS [5]. These effects have the potential to solve for camera abberations, enable more efficient and accurate real time vision, and significantly improve photo and video quality. More modern event cameras not only have per-pixel DVS circuitry but also greyscale APS circuitry to enable the camera to capture not only asynchronous events but synchronous video [DAVIS, ATIS]. Examples of this are the DAVIS sensor [figure 1] and the ATIS sensor. The combination of these two data streams allows reconstruction of traditional images and video with significantly higher dynamic range and significantly lower motion blur. Examplse of an event streams are given in figures 1 and 6.

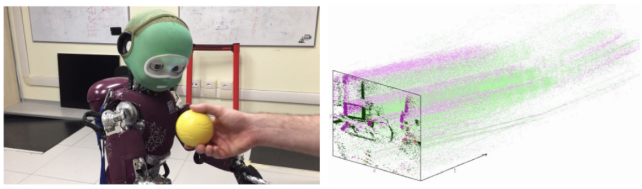


Figure 2. Event stream from IIT iCub [5]

From a theoretical standpoint we can consider the each event camera to be producing a tuple of coordinates r_i, timestep t_i, and polarity value p_i:

$$e_i = (r_i, t_i, p_i)$$

. Using the brightness $L(r_i, t_k)$ we can interpret an event as a temporal brightness derivative above a certain threshold C:

$$\frac{d}{dt}L(r_i, t_i) = \frac{p_i C}{\delta t_i}$$

Conceptually, we can think of event cameras as measuring the optical flow of their scene, concluding that moving edges cause events.

3.2 Graph Neural Networks

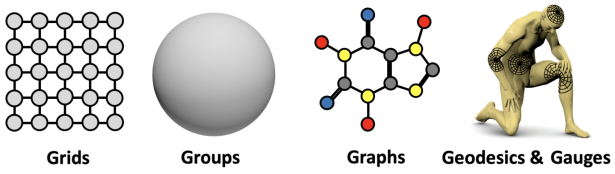


Figure 3. Deep Learning on Different Symmetry Groups [bronstein]

Graph neural networks (GNNs) [bronstein] were developed to overcome the limitation of convolutional networks on irregularly structured data without requiring a prohibitively large number of parameters using multilayer perceptrons [deep learning]. In order to decrease the number of model parameters there must be an inductive bias imposed according to the symmetry groups identified in the training and inference datasets as well as the desired manifold structure of the network [conv1, conv2, bronstein]. This ensures that the model both learns within the desired structure and does not overfit. For example, convolutional neural nets (CNNs) are designed under the translational symettry group due to the equidistant grid structure of images which allows parameters to be shared in convolution filters [bronstein]. Long short term memory networks, or (LSTMs) are designed under temporal symmettry which allows parameters to be shared each timestep [bronstein]. Graph neural networks do not have equidistant grids of nodes and may have loops, however, graphs do have the property of permutation invariance under permutation group Σ_n for all nodes, meaning that we can define a linear equivariant operator (or convolution), a nonlinearity operatorn $(\sigma(x))(u)$ on each node u in the graph, a local pooling operator P, and a global pooling operator A to construct arbitrary neural networks from [bronstein].

The general formula for a graph convolution is:

$$z_i = \sum_{j \in N(i)} \psi_\Theta(x_i, x_j, e_{ij})$$

and

$$x_i^{i+1} = \gamma_\Theta(x_i, z_i)$$

where z_i is the intermediate convolution layer representation, x_i^{i+1} is the pre or post layer representation, $N(i)$ is the neighborhood of node i, and ψ_{Theta} with γ_Θ make up the graph convolution [aegnn, bspline, articles]. Graph convolutions can be defined in a variety of ways depending on the initial assumptions, typically as spatial convolutions, attention convolutions, message passing convolutions, or spectral convolutions [figure 4].

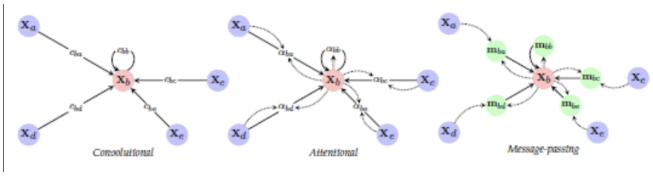


Figure 4. Concolutional, Attentional, and Message Passing Graph Operators [bronstein]

Spatial convolutions work by computing a inner product of the node features and learned weights but require a fixed number of neighbors for each node [articles]. Spectral convolutions fix this problem by taking the graph G laplacian:

$$L = \nabla G = D - A$$

where D is the degree matrix and A is the adjacency matrix of G. By taking the n principle eigenvectors of the graph laplacian we can treat the graph convolution as a linear combination of the eigenvectors with learned weights [articles]. Though this solves the limitations of spatial convolutions the eigenvector computation is significantly more computationally intensive, rendering training and inference times much higher. More commonly attention or message passing networks are used. Attention convolutions work similarly to spatial convolutions but use attention to weight the feature space at each node, then averaging over all features. This can be done for either edges, nodes, or global data [articles]. Message passing, however, is very similar to attention but allows for edge features, node features, and global features to be used in each convolution [articles].

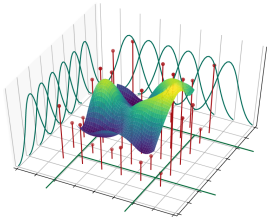


Figure 5. B-Spline Convolution Kernel in 1D [aegnn, bspline]

For event data we can assume no edge features, each node having coordinates, and each node having a polarity value. In order to take advantage of the low computation time of spatial convolutions while being able to allow for node coordinates to lie along a spatial continuum we can use spline convolutions. B-spline convolutions learn a parameterized b-spline instead of a node vector, allowing the linear convolution to use the b-spline values at the given coordinates of each node. This is expressed as:

$$(g_n * f)(i) = \sum_{l=1}^{M_{in}} \sum_{j \in N(i)} f_l(j) g_{n,l}(u(i,j))$$

for the ith node where $g_{n,l}$ is the nth b-spline kernel and $u(i,j)$ are the coordinates [1, spline].
Graph local and global pooling is typically defined using a mean, max, or sum of the node and edge features [bronstein, pooling]. For cases where we can assume coordinate data we can use point cloud sampling algorithms for our pooling layer such as voxelgrid filters [voxel], KD-trees, and octrees.

3.3 Asynchronous Event-Based Graph Neural Networks

Asynchronous event-based graph neural networks (AEGNNs) [1] assume that only a small number of events in the input image are be updated at each timestep. This means that we can run the network on the entire graph once and then we only need to compute a convolution of the subgraph containing new nodes and neighborhoods of event nodes in each successive timestep. As the event propagates upward through the network, the receptive field becomes larger, thus updating more and more of the network, and finally culminating into a fully updated output vector.

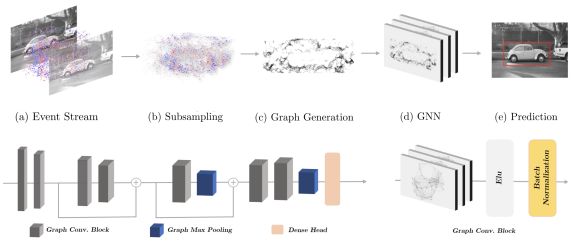


Figure 6. AEGNN Dataset and Model from [aegnn]

We can see in figure 6 that the DAVIS sensor stream [figure 6a-b] initially has a raw video and event cloud. In order to process the event stream using a GNN we must first convert to an event graph G. Due to the high number of events produced, this is done by sampling events with a uniform factor K and then connecting edges whenever the distance between node i and node j is less than a threshold R [1, figure 6c]. This event graph is computed over the entire dataset before training and during inference time each new event is given edges to it's R-radius neighborhood. Having much of this processing done before training makes the training significantly faster. Because the purpose of the AEGNN is to recognize and detect objects from event streams, the network can be trained in a supervised manner on the full event stream, label pair. This ensures that the network learns to recognize and detect objects, such as cars, by looking at a full and correct event stream each time. During inference time the recognition and detection probabilities will update with each new asynchronous update and become more accurate according to how long the event

stream has covered the object.
The AEGNN, due to it's event data input, uses b-spline convolutions as well as max value voxelgrid pooling layers. This means that the convolution layers have an efficient and accurate convolution that accounts for their irregularly spaced event nodes. It also means that when pooling to a courser graph, each voxel in the spatio-temporal T time window will be downsampled to a single node corresponding to the maximum value event in the voxel [1,voxel,pooling]. In the original paper seven convolution layers with an ELU nonlinearity and batch normalization were used, along with two pooling layers and a dense head for prediction. Due to their asynchronous operation, AEGNN's achieve between 1.5x to 8x faster computation than a synchronous CNN, even without highly optimized GPU implementations [1]. Additionally, they are able to achieve near state of the art recognition accuracy from only 5000 events [1].

3.4 Spiking Neural Networks

Spiking neural nets [figure 7] are widely considered to be the future of deep learning due to their low latency, power consumption, high spatial and temporal sparsity, closer modeling of biological neural networks, and their natural integration with biologically inspired hardware such as neuromorphic chips, event cameras, event based audio sensors, and event based actuators [5, 6, 24, 25].

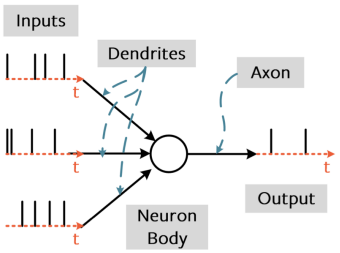


Figure 7. Spiking Neuron [spiking]

We can describe a spiking neuron using the leaky integrate and fire model as an ordinary differential equation. For training n neurons we can convert this to a matrix equation and use backpropogation. We can see that the description as a dynamical system significantly increases the backpropogation training complexity. Models such as the hodgkin-huxley or Izhikevich models more accurately model the brain but are significantly more difficult to train due to nonlinearity. These models, however, can be easily implemented in hardware as part of a neuromorphic chip. Instead of using neuroscience inspired neurons there are kernel based, recurrence based, and deep learning based models that have all achieved success in niche applications [24]. Alternatively, we can take any given neural network and convert it to a spiking neural network [spiking].

3.5 Current State of the Art

Since the publication of AEGNNs, the state of the art in event-based recognition has improved [30, 31, 32], and though AEGNN's remain one of the most efficient in inference and training times, they lag behind in parameter count and accuracy [figure 8]. Notably, the event dynamic graph convolutional network (EDGCN) [32], event transformer, and neuromorphic data auugmentation with spiking nets have improved the accuracy from up to 20%.

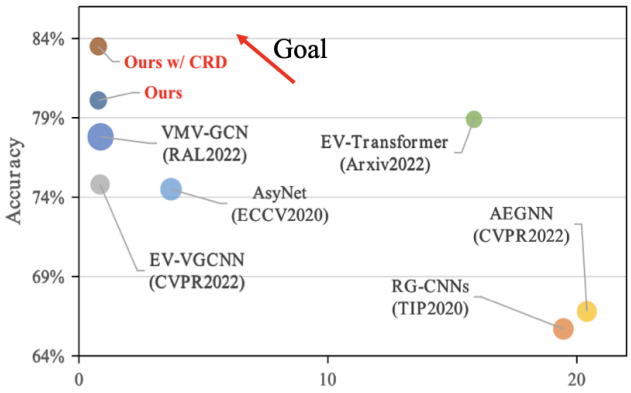


Figure 8. Comparison of Architecture Accuracies from [dgcnn]

The event transformer improves the accuracy and ease of use by using a custom transformer block to input the events in their native tensor format [30]. Though the accuracy is not as good as other models, there is less preprocessing and fewer parameters. The EDGCNN improves it's accuracy by using a frame based convolutional neural net in addition to the event based graph convolutional net to add extra supervision in the loss function at both intermediate steps and the output of the model [32]. Perhaps the most surprising and interesting of the current methods is the neuromorphic data augmentation (NDA) model [25] since it not only uses spiking neural networks but recent training advances such as contrastive learning using siamese networks. Contrastive learning is a popular self supervised learning technique that is accomplished by augmenting the dataset during training with different synthetic abberations and transformations and then training the network via a saimese network [26] to tell whether the same picture with different randomized augmentations can still be detected as the same picture. This implicitly teaches the network to differentiate between classes. Furthermore according to our experiments, they outperform AEGNN in efficiency, training time, parameter count, and accuracy.

4 AEGNN Design Verification and Future

4.1 Accuracy, Efficiency, and Reproduction Experiments

AEGNN was created to provide an event based recognition and detection deep learning model with significantly better efficiency, power consumption, and state of the art accuracy. According to the authors, the AEGNN is from 1.5 to 8 times faster at inference time, can achieve near state of the art accuracy defined as ¿90% using only 5000 events at inference time, and have given a formula for the runtime of the algorithm. In order to verify these claims we have run a reproduction study for the N-Cars [3] and N-Caltech [4] datasets in order to verify the per layer runtimes, total runtimes, and accuracy. Additionally we have run the AEGNN model against a promising new model, the

NDA spiking neural net, in order to experimentally test it's accuracy and efficiency against the accuracy data in figure 8. From here we have done a survey of the current state of the art and in the next section will give a taxonomy of potential future research directions.

4.2 Future Directions

As the authors mentioned in the AEGNN paper, the effect of the clustering method in the pooling layer has not been verified beyond the fact that we know it works. Because event camera pixels approximate the brightness derivative and on large scales seem to follow edges, we can conclude that if we used a clustering algorithm that was more sensitive to edges we could achieve better results by being able to preserve the edges more than the noise, assumed to be in the low density areas of the clustering tesselation. For example kd-trees, octrees, or BSPs [figure 9] could be used instead of the voxelgrid filter to keep the highly dense points in their own quadrant while keeping the noise in sparse large quadrants. We could then modify the percentage of max points taken from each quadrant inversely proportional to the volume, thus keeping more edge points.

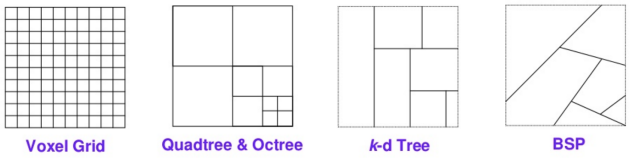


Figure 9. VoxelGrid, Octree, K-D Tree, and BSP

Despite the great success of graph neural networks for event based information processing, the preprocessing step is computation intensive and if it were possible to use data native event transformers or spiking neural networks that would be more ideal. Because of the superior performance in the NDA paper [25], it would be a logical next step to implement NDA [25, figure 10] and siamese network [26, figure 11] to train the event transformer, EDCGNN, and AEGNN and see which one performs the best.

Figure 10. Dataset Augmentation from [25]

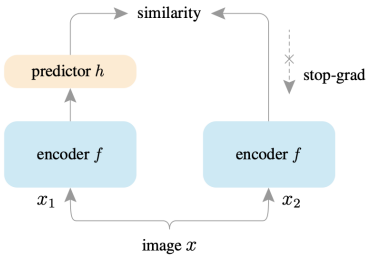


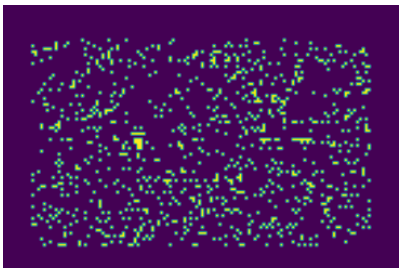
Figure 11. Contrastive Self-Supervised Learning using Siamese Networks [26]

Additionally, it may be a good idea to implement NDA with an additional convolutional neural net to add extra "supervision" to the loss function during training, thus opening the door to potential accuracy increases.

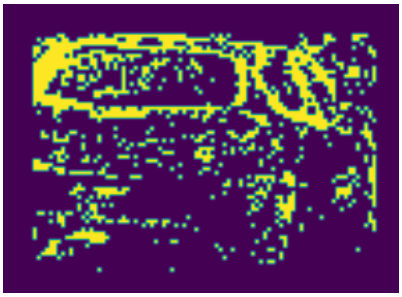
5 Experiments

5.1 Dataset Visualization

We implemented a simple python script to plot (through authors' functions) the Ncars dataset for visualizing both Event Histograms (Fig. 12) and Graphs representation (Fig. 13).

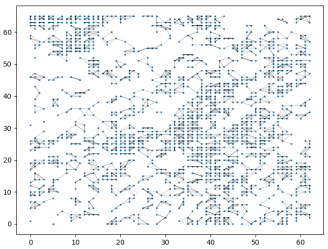


(a) Event Histogram background class

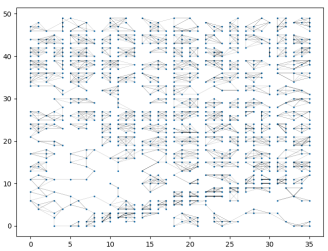


(b) Event Histogram car class

Figure 12. Ncars Visualization



(a) Example graph



(b) Example graph

Figure 13. Graph representation

Additionally, we created a script to display a chosen class from the NCaltech dataset in a grid (Fig. 14). For this, we employ the library's function of tonic. Additionally, we utilized a function that creates a voxel grid from a sequence of events using bilinear interpolation in the time domain and presents it as an animation.

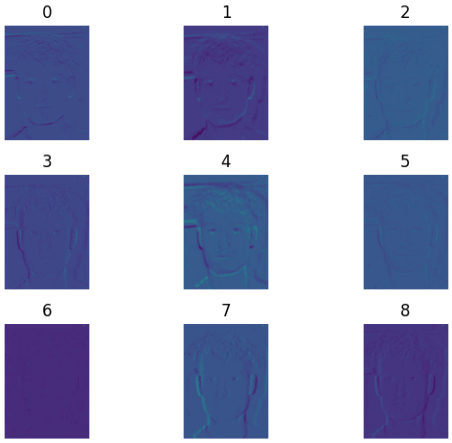


Figure 14. Plot events accumulated as frames

—

For the experiments, we used a repository forked from the author's code. We used this repo instead of the original one because the authors did not implement the training part (train.py). Moreover, the developers of the fork repo used **wandb** for tracking the accuracy and the losses during the training and the evaluation. Wandb is a tool that allowed us to plot the statistics/metrics, and keeps in log files the output of the program, allowing us to debug easily.

5.1.1 NDA and SNN To run the experiments with Neuromorphic Data Augmentation and Spike NN (from this paper) on the same dataset (N-Caltech), we used a MAC book Pro with MPS support (M1 chip). We used these techniques (and model) to have a comparison with our model. We run the training on a Spike NN (model VGG11) after doing a preprocessing on the dataset using spikejelly library . It turns out that after "only" 50 epochs (8 mins per epoch) the accuracy score on the validation set was 75 percent. We are not going to explain further SNN because it is not the main focus of our challenge. However, it is crucial to underline that by exploring this new preprocessing technique (i.e NDA) may be useful for future development. Indeed, NDA can be used for augmenting our dataset and for avoiding overfitting. Unfortunately, we could not use it because it requires a massive effort only to make it suitable for our AEGNN. NDA applies transformations on the dataset in a format which is not directly compatible with our model.

5.2 Datasets

We used neuromorphic datasets as the authors did:

- **NCars**: consisting of 12,336 car samples and 11,693 non-cars samples (background) for binary classification. There are 24'029 events in total, 100 ms long each.
- **NCaltech**: 101 classes. It is a "spiking version of the original frame-based Caltech101 dataset". These categories consist of 8,246 event sequences, 300 ms long each.

However, we could not use directly these raw datasets since they are not suitable for our GNN. Moreover, during preprocessing we sampled from it to avoiding overfitting.
For what concerns the code itself, we kept the modifications suggested by this fork. Indeed, functions such as max_pool_x and voxel_grid were not in the right submodule of pytorch geometric. Furthermore, original_num_nodes variable has been added in order to not having troubles during the run. Quoting the developers: "it stores the initial number of nodes in the input data" and "This change makes the code more robust and avoids potential issues due to changing data.num_nodes during the loop". Other info about modifications are available in their README.

5.3 Hyper parameters

Preprocessing parameters:

- 80%,10%,10% Training,Validation,Test split (original not known).
- subsampling of 10000 samples for N-Cars dataset (same as original).
- subsampling of 5000 samples for N-Caltech dataset (reduced from 25000 due to hardare limitations).

The hyper parameters we used were:

- Adam Optimizer
- Batch size of 64 for Ncars and 8 for Ncaltech dataset
- Initial learning rate equal to 10^{-3}. Every 20 epochs it is divided by 10.
- Gradient Accumulation of 2

5.4 Experimental setup

To run our experiments we got some issues because the cluster CSCS was busy all the time. Every time we tried to get access, there were some students or professors with higher priority, and our scripts could not be even launched. In addition, our Macs computers were not able to run the code properly because the python libraries such as pytorch and pytorch geometric required cuda cores. To solve these issues we used a computer with Linux Ubuntu OS and NVIDIA GeForce GTX 1050. This component is relatively old, and this is the reason why our experiments required a lot of time for completing. To summarize we used:

- CPU: Intel Core i7
- RAM: 16 GB
- GPU: NVIDIA GeForce GTX 1050, 4 GB dedicated
- OS: Ubuntu 22.04 LTS

5.5 Computational requirements

To train the model on NCaltech Dataset (20 epochs) it took 48 hours, and 4 hours on NCars Dataset (50 epochs).
We used as batch size:

- 8 (instead of 16) for NCaltech101
- 128 (instead of 64) for NCars

We reduced the number of batches from 16 to 8 for Ncaltech in order to run the code without any issues regarding the GPU memory, ours had only 4 GB of RAM.

6 Results

6.1 Expected results

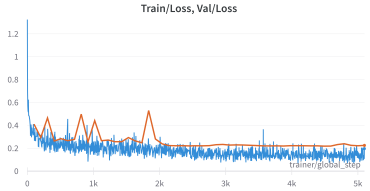
Methods	Representation	Async.	N-Caltech101 Accuracy ↑	N-Caltech101 MFLOP/ev ↓	N-Cars Accuracy ↑	N-Cars MFLOP/ev ↓
H-First [40]	Spike	✓	0.054	-	0.561	-
HOTS [29]	Time-Surface	✓	0.210	54.0	0.624	14.0
HATS [32]	Time-Surface	✓	0.642	4.3	0.902	0.03
DART [44]	Time-Surface	✓	0.664	-	-	-
YOLE [7]	Event-Histogram	✓	0.702	3659	0.927	328.16
EST [17]	Event-Histogram	✗	**0.817**	4150	0.925	1050
SSC [20]	Event-Histogram	✗	0.761	1621	0.945	321
AsyNet [36]	Event-Histogram	✓	0.745	202	0.944	21.5
NVS-S [32]	Graph	✓	0.670	7.8	0.915	5.2
EvS-S [32]	Graph	✓	0.761	11.5	0.931	6.1
Ours	Graph	✓	0.668	**7.31**	**0.945**	**0.47**

Table 1. Comparison with several asynchronous and dense methods for object recognition. Our graph-based method has the lowest computational complexity overall while achieving state-of-the-art performance. Especially, it obtains the best accuracy on N-Cars [32] with 20 times lower computational complexity, compared to the second-best asynchronous method.

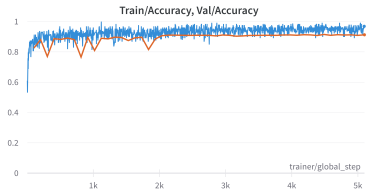
Figure 15. Expected results

6.2 Training/Validation

We trained 2 models for the recognition task.
For the ncars datset we trained for 50 epochs and achieved 0.91 accuracy on the validation set, this model took approximately 4 hours to train. Our result is very similar to the 0.945 stated by the authors.



(a) NCars Loss



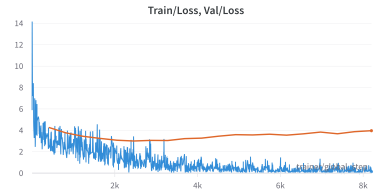
(b) NCars accuracy

Figure 16. NCars Training/Validation

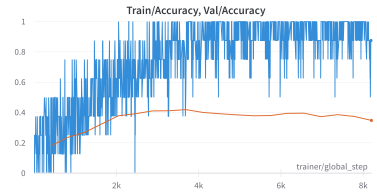
3 runs NCars	**mean**	**std**
Top 1 Val Accuracy	0.91233	0.00104
Validation Loss	0.23906	0.00989

Table 1. Mean and standard deviation

For the N-Caltech101 dataset we trained a recognition model for 20 epochs and achieved 0.41 accuracy on the validation set, this model took approximately 2 days to train. Our results are far from the 0.668 stated by the original authors. It is unlikely that this difference is caused by a different number of training epochs, and that training for longer would improve the model performance, because from the training curves it is clear that the model is over-fitting. This might be caused by the reduced batch size, from 16 to 8. But due to a lack of computational resources we cannot increase it without running out of GPU memory, so we used gradient accumulation of 2, to get an effective batch size of 16. But probably the main cause is the fact that due to hardware limitations we used an event subsampling of 5000 instead of 25000 as the original paper. In conclusion we could not accurately confirm the authors results on this dataset.



(a) N-Caltech101 Loss



(b) N-Caltech101 accuracy

Figure 17. N-Caltech101 Training/Validation

For comparison, Figure 18 shows training and validation accuracy of the NDA-VGG11 spike NN model, over just 10 epochs and approximately 1 hour of training, it already achieves 0.52 on validation accuracy on the N-Caltech101 dataset.

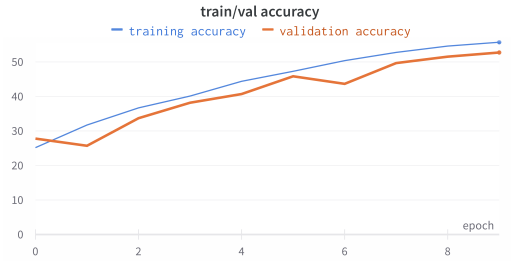


Figure 18. Train/Val accuracy NDA-VGG11

6.3 Efficiency

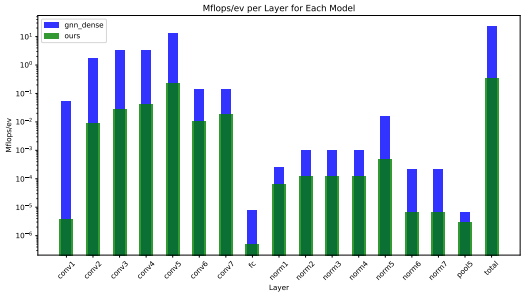


Figure 19. MFlops per event

Figure 19 shows a comparison for each layers on MFLOPs/ev, between a dense GNN model and it's asynchronous version with local update rules. This demonstrates the remarkable efficiency gains obtained by the asynchronous GNN variant, which requires considerably fewer MFLOPS to perform its computations.

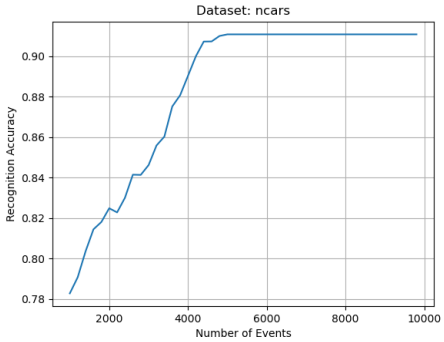


Figure 20. Accuracy per Event with Ncars

Figure 20 shows that the model we trained on the NCars dataset using a subsampling of 10000 events can achieve very high accuracy during inference also on data with subsampling as low as 5000 events.

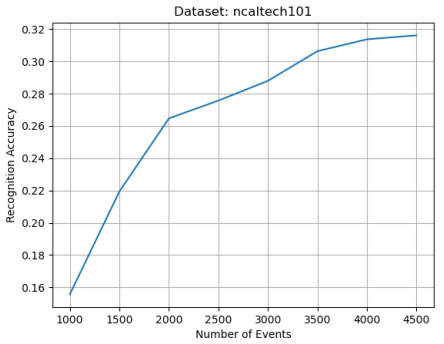


Figure 21. Accuracy per Event with NCatech101

Figure 21 shows that subsampling of 5000 events or less is not enough data to train a state-of-the-art model on the N-Caltech101 dataset.

7 Discussion and conclusion

We believe that we were faithfully able to reproduce the AEGNN paper [1] while building on the previous reproduction work [2] by further increasing the usability of the package, comparing against more recent results, and analyzing promising future research directions. To increase the usability we will include all of our pretrained models in the repository as well as our all of our training scripts. With the addition of docker and microsoft azure cloud training integration, we expect future researchers will be able to not only easily train and use the AEGNN model, but modify the model based on our future research directions to create new state of the art models. Our paper and github repository have been made to make it as easy as possible to learn about AEGNN and improve on it.

References

[1] Davide Scaramuzza Simon Schaefer, Daniel Gehrig. ”aegnn: Asynchronous event-based graph neural networks”.

[2] Justas Andriuskevicius Dequan Ou Yueqian Liu. ”replication results for aegnn: Asynchronous event-based graph neural networks”.

[3] Amos Sironi1 Manuele Brambilla Nicolas Bourdis Xavier Lagorce Ryad Benosman. ”hats: Histograms of averaged time surfaces for robust event-based object classification”.

[4] ”n-caltech dataset”. . URL `http://www.vision.caltech.edu/datasets/`.

[5] Garrick Orchard Chiara Bartolozzi Brian Taba Andrea Censi Stefan Leutenegger Andrew Davison Joerg Conradt Kostas Daniilidis Davide Scaramuzza Guillermo Gallego, Tobi Delbruck. ”event-based vision: A survey”.

[6] Bernabe Linares-Barranco Christoph Posch, Teresa Serrano-Gotarredona and Tobi Delbruck. ”retinomorphic event-based vision sensors: Bioinspired cameras with spiking output”.

[7] ”event-based robot vision”. . URL `https://sites.google.com/view/guillermogallego/teaching/event-based-robot-vision`.

[8] P. Lichtsteiner and T. Delbruck. ”64x64 event-driven logarithmic temporal derivative silicon retina”.

[9] Raphael Berner; Christian Brandli; Minhao Yang; Shih-Chii Liu; Tobi Delbruck. ”a 240×180 10mw 12us latency sparse-output vision sensor for mobile applications”.

[10] Daniel Matolin Rainer Wohlgenannt Michael Hofstätter Peter Schön Martin Litzenberger Daniel Bauer Heinrich Garn. ”asynchronous time-based image sensor (atis) camera with full-custom ae processor”.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. ”deep learning”.

[12] Michael M. Bronstein Joan Bruna Taco Cohen Petar Veličković. ”geometric deep learning grids, groups, graphs, geodesics, and gauges”.

[13] Adam Pearce Emily Reif Benjamin Sanchez-Lengeling Alexander B. Wiltschko. ”a gentle introduction to graph neural networks”.

[14] Ameya Daigavane Balaraman Ravindran Gaurav Aggarwal. ”understanding convolutions on graphs”.

[15] Chuang Liu1 Yibing Zhan2 Chang Li2 Bo Du1 Jia Wu3 Wenbin Hu1† Tongliang Liu4 Dacheng Tao2. ”graph pooling for graph neural networks: Progress, challenges, and opportunities”.

[16] Hai Liu Youfu Li Yongjian Deng, Hao Chen. ”a voxel graph cnn for object classification with event cameras”. .

[17] ”registration”. . URL https://towardsdatascience.com/neighborhood-analysis-kd-\trees-and-octrees-for-meshes-and-\point-clouds-in-python-19fa96527b77.

[18] Haijun Yu Shanshan Tang, Bo Li. ”chebnet: Efficient and stable constructions of deep neural networks with rectified power units using chebyshev approximations”.

[19] Max Welling Thomas N. Kipf. ”semi-supervised classification with graph convolutional networks”.

[20] Arantxa Casanova Adriana Romero Pietro Liò Yoshua Bengio Petar Veličković, Guillem Cucurull. ”graph attention networks”.

[21] David Gregg Maria Francesca, Arthur Hughes. ”spectral convolution networks”.

[22] Frank Weichert Heinrich Muller Matthias Fey, Jan Eric Lenssen. ”splinecnn: Fast geometric deep learning with continuous b-spline kernels”.

[23] Justin Gilmer Samuel S. Schoenholz Patrick F. Riley Oriol Vinyals George E. Dahl. ”neural message passing for quantum chemistry”.

[24] Mohammed Bennamoun Max Ward Gregor Lenz Doo Seok Jeong Emre Neftci Girish Dwivedi Wei D. Lu Jason K. Eshraghian, Xinxin Wang. ”training spiking neural networks using lessons from deep learning”.

[25] Yuhang Li Youngeun Kim Hyoungseob Park Tamar Geller Priyadarshini Panda. ”neuromorphic data augmentation for training spiking neural networks”.

[26] Xinlei Chen Kaiming He. ”exploring simple siamese representation learning”.

[27] Bangbang Yang Ye Zhang Zhaopeng Cui Hujun Bao Guofeng Zhang Yijin Li, Han Zhou. ”graph-based asynchronous event processing for rapid object recognition”.

[28] Nico Messikommer Daniel Gehrig Antonio Loquercio Davide Scaramuzza. ”event-based asynchronous sparse convolutional networks”.

[29] Zongming Guo Amin Zheng Gusi Te, Wei Hu. ”rgcnn: Regularized graph cnn for point cloud segmentation”.

[30] Bochen Xie; Yongjian Deng; Zhanpeng Shao; Hai Liu; Youfu Li. ”vmv-gcn: Volumetric multi-view based graph cnn for event stream classification”.

[31] Ana C. Murillo Alberto Sabater, Luis Montesano. ”event transformer. a sparse-aware solution for efficient event data processing”.

[32] Bochen Xie Hai Liu Youfu Li Yongjian Deng, Hao Chen. ”a dynamic gcn with cross-representation distillation for event-based learning”. .

[33] ”pytorch”. . URL https://pytorch.org.

[34] ”pytorch lightning”. . URL https://www.pytorchlightning.ai/index.html.

[35] ”pytorch geometric”. . URL https://pytorch-geometric.readthedocs.io/en/latest/.

[36] ”spiking jelly”. . URL https://github.com/fangwei123456/spikingjelly.

[37] ”microsoft azure”. . URL https://azure.microsoft.com/.

[38] ”docker”. . URL https://www.docker.com.

[39] ”wandb”. . URL https://wandb.ai/home.

[40] ”our github repo aegnn”. . URL https://github.com/filippocasari/GDL.git.

[41] ”our github repo nda plus vgg11, and visualization part”. . URL https://github.com/filippocasari/Spike_NDA_and_Visualization_Dataset.git.

[42] Yuhang Li, Youngeun Kim, Hyoungseob Park, Tamar Geller, and Priyadarshini Panda. Neuromorphic data augmentation for training spiking neural networks. *arXiv preprint arXiv:2203.06145*, 2022.

[1] [2]
[3] [4]
[5] 5 [6] [7] [8] [9] [10]
[11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23]
[24] [25] [26]
[27] [28] [29]
[30] [31] [32]
[33] [34] [35] [36] [37] [38] [39] [40] [41] [42]