Next-Generation Intelligence in AIoT: A Comparative Analysis of Machine Learning, Deep Learning, and Spiking Neural Networks for Energy-Efficient Image Classification

BAI Haoran (22097746d), FU Tao (22097327d), NING Weichen (22097838d)

Group 1, COMP4436

Abstract— **This report presents a concise comparative analysis of various algorithms for Cats vs. Dogs image classification, including unsupervised methods (K-Means clustering, Cluster GAN), supervised machine learning (logistic regression), deep learning (CNN), and spiking neural networks (SNNs). For SNNs, we evaluate a directly trained convolutional SNN (CSNN) using SpikingJelly, an untrained SNN simulated with Nengo via temporal extension, and ANN-to-SNN conversion approaches based on both a two-layer CNN and a pre-trained MobileNet V2. Performance was assessed using precision, recall, F1-score, accuracy, and execution time. We note that the ANN-to-SNN conversion with MobileNet V2 achieved superior results, with 92.86% accuracy and 93.06% F1-score, which shows that leveraging pre-trained ANNs can effectively offset the high training cost of SNNs while presenting competitive performance.**

I. Introduction

MACHINE Learning (ML) and Deep Learning (DL) have played a pivotal role in advancing AIoT, offering effective approaches to complex problems. In ML and DL, both supervised and unsupervised methods form the foundation of predictive models and data processing. Supervised learning focuses on classification and regression based on labeled data, whereas unsupervised learning algorithms aim to perform clustering based on patterns and structures detected within unlabeled data.

Spiking Neural Networks (SNNs), a more recent addition to the deep learning landscape, are considered to be the third generation of neural networks. SNNs are distinguished not only for their bio-inspired architecture that mimics the behavior of neurons in the human brain, but also for their potential to enhance AIoT applications, particularly in real-time environments.

This report aims to provide a comparative analysis of various algorithms from the fields of ML and DL, with a focus on their application in AIoT. The experiment involves evaluating five distinct algorithms—one supervised ML algorithm, one unsupervised ML algorithm, one supervised DL algorithm, one unsupervised DL algorithm, and an SNN. By assessing these algorithms based on key evaluation metrics of accuracy, precision, recall, F1-score, and runtime efficiency, this report will highlight their relative performance and potential advantages within AIoT systems.

II. Dataset Loading and Processing

The dataset implemented is called *Cats and Dogs image classification* from https://www.kaggle.com/datasets/samuelcortinhas/cats-and-dogs-image-classification. The dataset contains 697 images of cats and dogs scraped off of Google images. It is initially generated into two folders, which are the training dataset containing 557 images, and the testing dataset containing 140 images. Each folder also contains pre-classified cats and dogs folders. The image sizes range from roughly 100x100 pixels to 2000x1000 pixels. The image format is jpeg. Duplicates have been removed.

A. General Preprocessing

The preprocessing steps aim to ensure the data is in the right format, properly scaled, and augmented to enhance model generalization.

Image Loading and Resizing

The first step in preprocessing involves loading the image data and resizing them to a uniform size, which is essential for feeding the data into a model. In this implementation, the images are resized to a target size of 32x32 pixels. This size was chosen as a balance between computational efficiency and the ability to capture sufficient image details for classification.

Image Augmentation

Image augmentation is a technique used to artificially expand the size of the dataset by applying random transformations to the images. This helps improve the generalization ability of the model and prevents overfitting. In our preprocessing pipeline, the ImageDataGenerator class from TensorFlow is used to apply several augmentation techniques to the training images, including, Rotation (Random rotations of up to 40 degrees), Shifting (Random horizontal and vertical shifts by 20% of the image width and height), Shearing and Zooming (Random transformations to simulate different perspectives of the images), and Flipping (Horizontal flipping of the images to simulate different orientations). These augmentations ensure that the model is exposed to a variety of transformations, thereby making it more robust when classifying unseen images.

Normalization

To ensure the input images are on the same scale as the model's expected input, all image pixel values are rescaled by a factor of 1/255.0, transforming the pixel values from the range of [0, 255] to [0, 1]. This normalization step is essential for speeding up the training process and improving the convergence of deep learning models.

B. *Specific Preprocessing for Detailed Algorithms*

Unsupervised Learning

Appending an extra step of joining training data and random shuffling. Since the dataset is generally presented as the structure of pre-classified and unsupervised learning requires a dataset without the label, it is necessary to add this particular preprocessing.

Spiking Neural Networks

In the context of Spiking Neural Networks (SNN), a temporal dimension must be incorporated into the data. Since SNNs are dynamic models that process data over time, we need to repeat the images in the dataset for some time steps to reflect a temporal sequence for each image.

III. Algorithms

This experiment selected algorithms that span both supervised and unsupervised learning categories, providing a comprehensive evaluation of their strengths and weaknesses in the classification task. Specifically, we focus on a Supervised ML Algorithm, Unsupervised ML Algorithm, Supervised DL Algorithm, Unsupervised DL Algorithm, and a Spiking Neural Network (SNN).

A. *Unsupervised Learning*

This approach is to apply unsupervised learning methods to supervise image classification problems. The misalignment of approach and task style is unconventional and normally inappropriate since clustering algorithms are not initially designed for classification. However, the uniqueness of unsupervised learning can also offer meaningful insight into the relationship between data separation and feature extraction, especially in AIoT environments where labels may be unavailable or hard to obtain due to data volume, velocity, timeliness, or data sharing restrictions. [1]

Machine Learning

In this aspect of unsupervised machine learning, K-means clustering is implemented to deal with the supervised problem. K-means clustering is the algorithm, typically adopting Lloyd's algorithm, that groups the data using their similarity. As shown in the pictures below, the algorithm functions by first randomly picking several central points based on the cluster number expected (Fig 1).

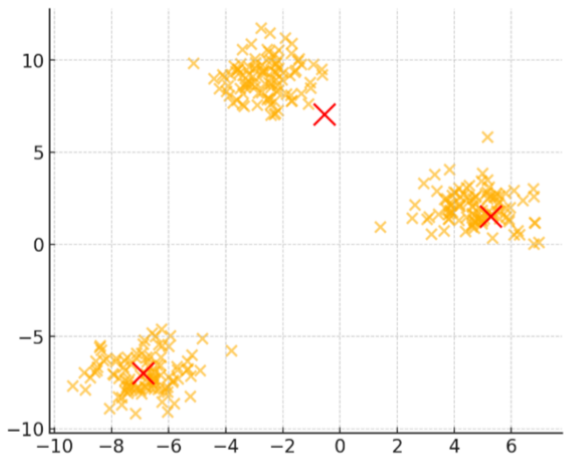


Fig. 1. Initial random centroids

Each data points are then assigned to clusters by associating with the nearest centroid (Fig 2). The assignment function is represented as

$$S_i^{(t)} = \left\{ x_p : \left| x_p - m_i^{(t)} \right|^2 \leq \left| x_p - m_j^{(t)} \right|^2 \forall j, 1 \leq j \leq k \right\}$$

where each point x_p is assigned to a cluster $S^{(t)}$.

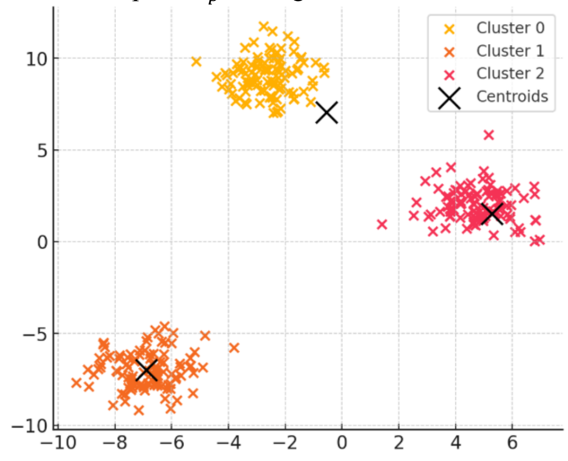


Fig. 2. Assign data points to clusters

After every point is generated to a group, the clusters update the centroids by calculating the relevant central position among all the points in the same group. The update function can be represented as,

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

This process iterates until the clusters cannot be further updated (Fig 3).

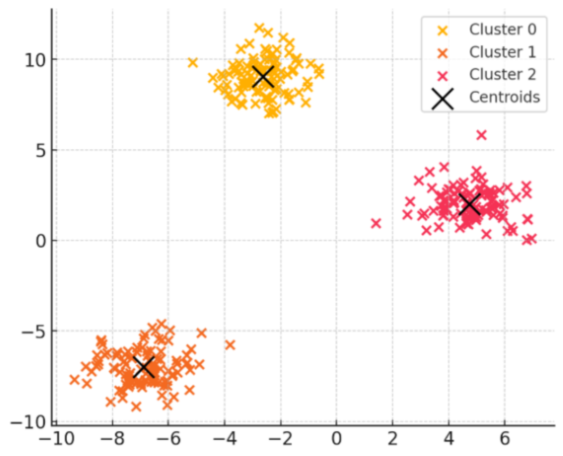


Fig 3: Update Centroids and Iteration

Deep Learning

In this scenario, the Generative Adversarial Networks (GANs) method is chosen to encounter the supervised task. GANs are unsupervised deep learning algorithms based on Convolutional Neural Networks (CNNs), which are typically used for image classification. The main structure for GANs is two parts, the Generator and the Discriminator. The Generator is an inverse Convolutional Neural Net, which receives random noise as inputs and outputs actual images. In simple terms, the Generator will try to fool the Discriminator with fake images. The loss function measuring how well the generator is fooling the discriminator is:

$$J_G = -\frac{1}{m}\sum_{i=1}^{m} \log D(G(z_i))$$

Where $\log D(G(z_i))$ represents log probability of the discriminator being correct for generated samples.

The Discriminator is functionally similar to CNNs, consisting of many hidden layers and one output layer. However, different from the CNNs generating outputs based on the input labels' number, the GANs only have two outputs, 1 representing real data and 0 representing fake data generated by the Generator. The algorithm for the Discriminator to reduce the negative log likelihood of correctly classifying the fake and real data can be shown as the following equation:

$$J_D = -\frac{1}{m}\sum_{i=1}^{m} \log D(x_i) - \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z_i)))$$

Where J_D presents the ability of the Discriminator's ability to separate real and fake data, $\log D(x_i)$ refers to the log probability of successfully categorizing real data, and $\log(1 - D(G(z_i)))$ log chance of accurate fake image identification.

In order to increase the ability of clustering, a particular model, Cluster GAN, has been adopted. [2] Apart from the typical part of the Discriminator and the Generator, the Cluster GAN introduces a Cluster Head module, which takes the generated image and outputs a set of cluster scores. These scores correspond to discrete cluster labels that are used to enforce the image generation process.

B. *Supervised Learning*

Supervised learning has key advantages, including high accuracy and interpretability. It uses labeled data to learn patterns, which helps models to better map inputs and output so that it leads to more precise predictions.

Machine Learning

For image classification tasks, logistic regression remains as a simple algorithm in supervised learning. However, it has both pros and cons.

In general, logistic regression predicts the probability of an image belonging to a specific class. This straightforward approach allows for easy interpretation of results. However, logistic regression works well when the dataset is not too large, and sometimes good feature extraction is required to enhance the accuracy. Besides, images might need to be preprocessed since logistic regression is not suitable for high-dimension data.

While it may not capture complex patterns as deep learning models do, logistic regression is efficient and requires less computational power. This makes it suitable for tasks where speed and simplicity are essential.

Deep Learning

When we consider image classification tasks, CNN stands in a well-known position in deep learning algorithms. Even though lots of new algorithms are derived and evolved over time, CNN still has its strength in image classification.

Using the convolution kernel, CNN becomes capable of local feature extraction as the kernel focuses on region by region. Besides the feature extraction, convolutional layers can help reduce parameters to enhance the efficiency of the model. Furthermore, more convolutional layers allow the model to learn more detailed and complex information.

C. *Spiking Neural Network*

Spiking neural networks (SNNs) are inspired by a paradigm rooted in biological systems where information is propagated via discrete spike events, unlike continuous-valued activation. SNNs are integrated with temporal dynamics. In this way, they could leverage internal state variables (e.g., membrane potentials) and rely on surrogate gradient methods to enable gradient-based training despite the intrinsic non-differentiability in generating spikes [3, 4]. In this section of the report, three distinct approaches of SNN in image classification are implemented and evaluated in Python scripts.

Trained Convolutional SNN with SpikingJelly

The fundamental implementation of our method uses the activation-based modules available in SpikingJelly [5] to design a Spiking Neural Network involving convolutional layers (CSNN). The network structure is a stack of three blocks, where each block consists of: a convolutional layer, batch normalization, a ReLU-based surrogate activation function, and a Leaky Integrate-and-Fire (LIF) neuron followed by max pooling. The LIF neurons are crucial in simulating the temporal dynamics produced by biological neurons because they contain a decay parameter (tau) that controls the temporal dynamics of the membrane potential.

In the process of the forward pass, the neural network is calculated over a specified duration, with outputs averaged over multiple timesteps. To handle the difficulties in backpropagation over time, we utilized a method called truncated backpropagation over time (TBPTT), where there is separation in the internal state with respect to output over distinct timesteps. We also included a feature where the internal state is reset on detecting differences in the shape of the expected tensors to handle situations where such approaches cause subtle shifts in dimensions. Thus, we could train the spiking neural network (SNN) with a direct implementation in binary cross-entropy loss with logits, ultimately resulting in a model able to learn from a sequence of images and enable satisfying predictions.

Untrained SNN with Simulated Output Using Nengo

In a later attempt, a basic SNN was built with the Nengo package, guided by laboratory resources and the respective documentation on Nengo and TensorFlow. The method was intended to test the ability of mimic spiking given the input of extracted features by pre-trained models, so the network was not trained. The primary difficulties encountered in the process were related to the fact that the network was not trained. The firing behavior of the neurons was close to a random pattern or simply failed to fire. These points emphasized the need to train the network or use complementary calibration methods in utilizing simulation tools such as Nengo.

ANN-to-SNN Conversion Using SpikingJelly

One viable method for mitigating the significant computational costs and difficulties associated with the training of spiking neural networks (SNNs) is the conversion from a pre-trained artificial neural network (ANN) to a SNN. The method exploits the existing training approaches and the vast library of pre-trained ANN models to construct SNN models. In this approach, we demonstrated such a conversion process in the process of training a CNN and a pre-trained MobileNet V2 model to the level of SNNs under the instructions provided in the SpikingJelly documentation. The procedure involves replacing conventional activation functions, like the ReLU, with spiking neuron modules that can mimic the temporal properties observed in neural cells. Simulating the adapted network over a specified temporal window makes estimating firing rates in the spiking neurons possible, which corresponds to the continuous activation in the original ANN models. The rate coding makes the adapted SNN models retain the performance properties of the originally trained ANN while having the ability to lower power usage in the inference phase.

IV. Experiment Settings

The performance on the test dataset was evaluated based on conventional metrics such as Accuracy, Precision, Recall, and F1-score. We also measured the inference time of models and algorithms on the test dataset to estimate the running efficiency. In addition to the numerical metrics, the training process was recorded and visualized as loss and accuracy curves over epochs or iterations. The experiments were conducted on the GPU platform (PQ605 Lab Computers) when possible, with a fallback strategy on the CPU of our laptops in case.

K-Means clustering algorithms

The K-Means clustering algorithm was implemented using the scikit-learn library. The dataset was divided into two sets: one for training and one for testing. For the training set, images were loaded without labels to facilitate unsupervised learning, while the test set included labeled images to evaluate the performance of the clustering model.

Cluster GAN algorithms

Implementing the PyTorch framework, the model architecture (As Fig 4) consists of three architectures, the Generator, the Discriminator, and the Cluster Head. Carried out 35 epochs, with the generator and discriminator trained alternately using a binary cross-entropy loss. In addition, the cluster head utilized a mean squared error loss to ensure the generated images were classified correctly. The optimizer used for both the generator and discriminator was the Adam optimizer, known for its ability to handle sparse gradients.

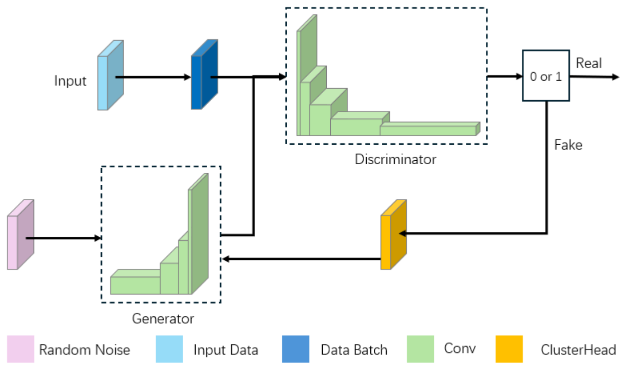


Fig. 4. Cluster Generative Adversarial Network Architecture

Logistic Regression

A logistic regression model was utilized for classification. The package used for the algorithm is scikit-learn. The model was trained using the flattened image data and their corresponding labels because logistic regression struggles when dealing with high-dimension data.

After fitting the training set and predict the test set. The model was evaluated using several performance metrics, including Precision, Recall, F1-Score, and Accuracy, and these variables are visualized through a bar chart for a clear view and easy comparison to other graphs.

Convolutional Neural Network

For the convolutional neural network, the setting of the environment and parameters of the model layers are more complex. The model was built based on the module from TensorFlow, and each pixel of each channel of the images is converted into the range [0, 1]. The layers are divided into 2 parts, the first part consists of several convolutional layers connected by pooling layers in between, and the second part is a dense layer shooting output using a 'sigmoid' activation function.

However, the dense layer only accepts one dimension input, so a Flatten() layer is required between the final pooling layer and the sense layer to flatten the high dimension data into the appropriate form.

Spiking Neural Network Training

In the trained convolutional Spiking Neural Network (CSNN), three distinct blocks existed in the structure, combining a convolutional layer, batch normalization, ReLU surrogate activation, LIF neurons, and max pooling in every block. The final fully connected layer was assigned to performing the binary classification. The network was simulated over a particular temporal window (20 timesteps), and the outputs over timesteps were averaged to obtain the prediction. The training process utilized the Adam optimizer with a learning rate of 5e-5, which was adjusted based on the number of trainable parameters and involved layers. The loss function is based on binary cross-entropy in combination with logits since this is a binary image classification task with binary labels. The method of truncated backpropagation through time was adopted by decoupling internal LIF states over a sequence of timesteps. The training process runs over 25 epochs, in which important metrics such as loss and accuracy were monitored systematically.

V. Results and Discussions

A. *Training Process Visualization*

The figures below (i.e., Fig. 5. ~ Fig. 9.) visualize the accuracy and loss of neural network algorithms over the training epochs and the metrics, including Accuracy, Precision, Recall, and F1-score for other algorithms over the test dataset (since no training epochs). The results of these metrics for all algorithms are summarized in subsection C. TABLE 1. Algorithms Performance Matrix.

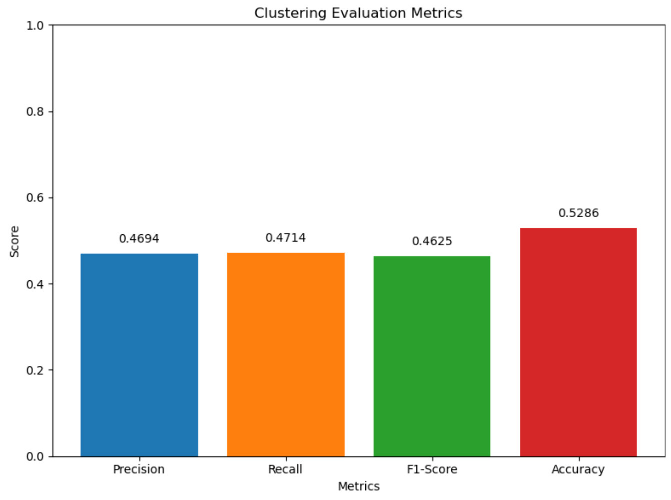


Fig. 5. K-Means Clustering Result

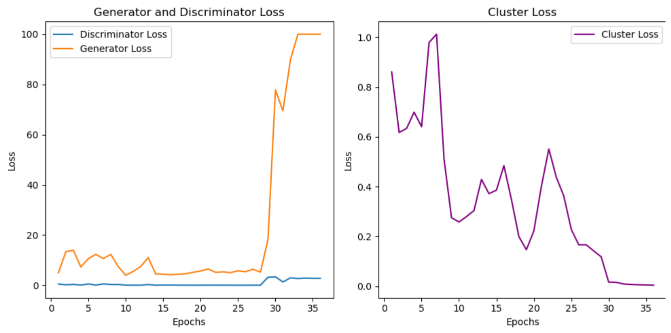


Fig. 6. Cluster GAN Result of Training Loss

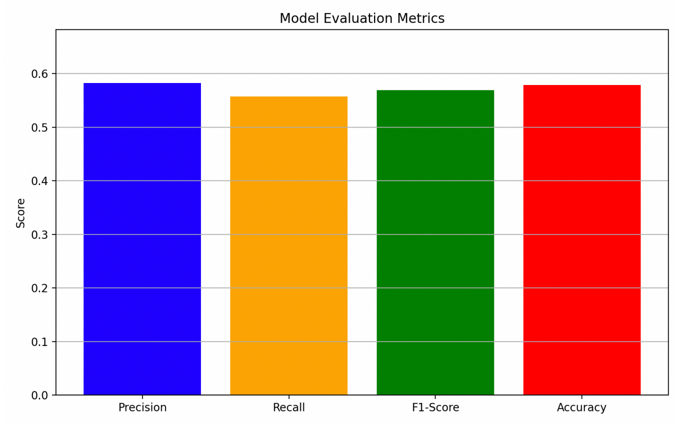


Fig. 7. Logistic Regression Result

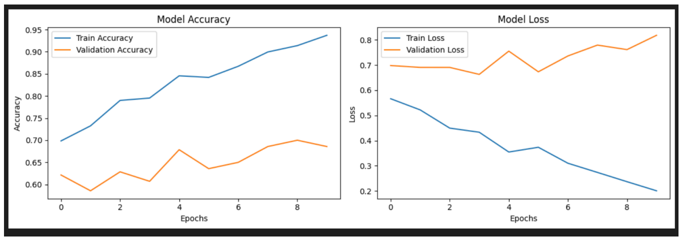


Fig. 8. CNN Result

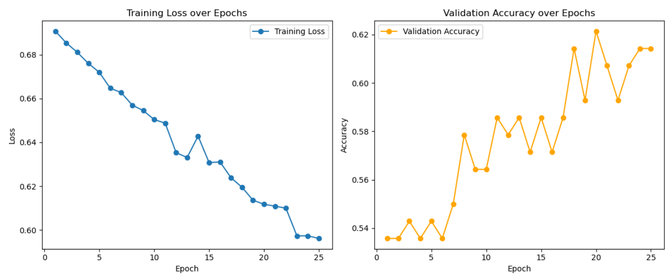


Fig. 9. CSNN Result

B. *Algorithm Performance Analysis*

K-Means clustering algorithms

The runtime for the K-Means algorithm was 0.3813 seconds, which indicates its computational efficiency. Although the runtime was fast, the classification performance, as shown by the evaluation metrics (Fig. 5), was modest. Specifically, the accuracy achieved by the K-Means algorithm was 0.5286, with a precision of 0.4694, a recall of 0.4714, and an F1-score of 0.4625. These results suggest that while K-Means is capable of clustering the data, it struggles with achieving high classification performance, likely due to the absence of labeled data during the training process.

Cluster GAN

The training runtime for the Cluster GAN was 1.9164 seconds, which was significantly longer than that of K-Means, reflecting the added complexity of its architecture and the need for iterative training. However, despite the longer training time, the performance in terms of classification metrics was slightly improved compared to K-Means. The accuracy achieved by Cluster GAN was 0.5357, with a precision of 0.5361, a recall of 0.5357, and an F1-score of 0.5346. These results, presented in Figures 6, demonstrate that Cluster GAN outperforms K-Means in terms of classification metrics, although both algorithms showed relatively limited performance in clustering real-world images.

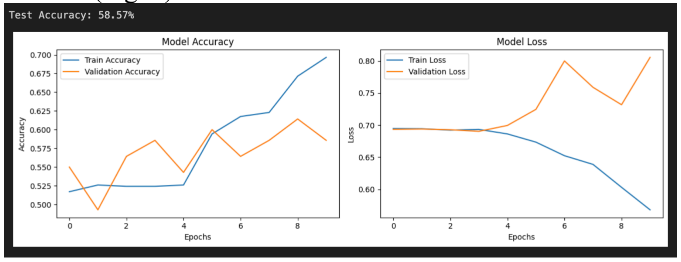
For unsupervised learning, both algorithms demonstrated their potential in AIoT applications, particularly in real-time image clustering tasks. While K-Means showed superior runtime efficiency, Cluster GAN provided slightly better classification performance. However, both algorithms would benefit from further refinement, particularly in handling more complex and diverse datasets.

Logistic Regression

After 0.038 seconds of execution, the logistic regression performs an accuracy of 57.86%, precision of 58.21%, recall of 55.71%, and f1 score of 56.93%. This algorithm is directly called through the scikit-learn package, and not much modification to the model was done as the algorithm is quite firm.

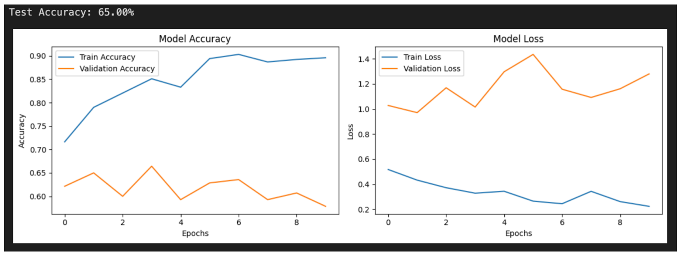
Convolutional Neural Network

By the original setting, the test result is 58.57% (Fig. 10), which is not appealing at all. After checking the tutorial file and finding that when the model is trained, checkpoints of best models at each time are created and an early stopping is activated to avoid overfitting problems. By creating this feature for the training, the accuracy is levitated to 65.00% (Fig. 11). However, the validation accuracy is getting lower even when the training accuracy increases, which means overfitting still exists. After checking the file again into the model, it appears that the dropout lay drops some links with a certain probability to overcome overfitting to some degree. After adding the dense layer, the overall structure is decided, and the next step is to manipulate the parameters of the layers including kernel size, strides, input size, etc. (Fig. 12) However, the accuracy of the test prediction still ranges from 58% to 66% through at least 10 trials of the different settings of the model. This is probably because of the small size of the dataset and not enough epochs to train the model. The best performance is given by this model, with the kernel size of 4x4, maxPool size of 4, and input size of 256x256x3 (Fig. 13). The result of prediction on the test set is 68.15% (Fig. 9).



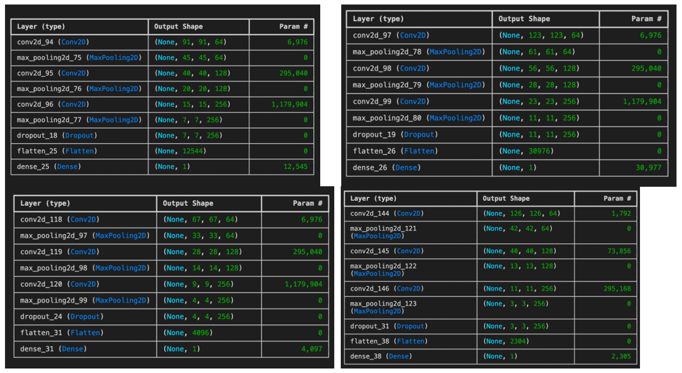
```
Test Accuracy: 58.57%
```

Fig. 10. CNN Initial Result



```
Test Accuracy: 65.00%
```

Fig. 11. CNN Drop-Out Version Result



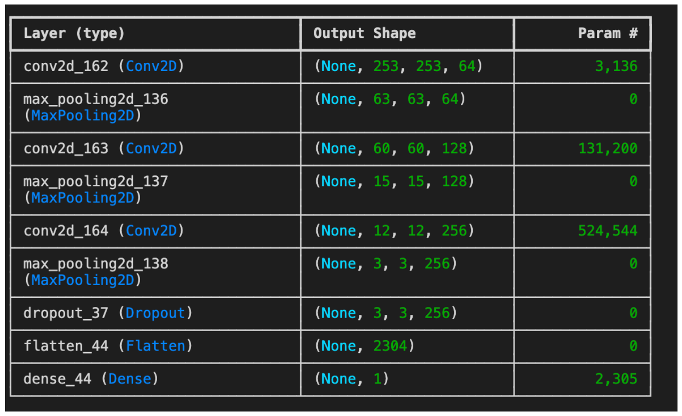
Layer (type)	Output Shape	Param #
conv2d_94 (Conv2D)	(None, 91, 91, 64)	6,976
max_pooling2d_75 (MaxPooling2D)	(None, 45, 45, 64)	0
conv2d_95 (Conv2D)	(None, 40, 40, 128)	295,040
max_pooling2d_76 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_96 (Conv2D)	(None, 15, 15, 256)	1,179,904
max_pooling2d_77 (MaxPooling2D)	(None, 7, 7, 256)	0
dropout_18 (Dropout)	(None, 7, 7, 256)	0
flatten_25 (Flatten)	(None, 12544)	0
dense_25 (Dense)	(None, 1)	12,545

Layer (type)	Output Shape	Param #
conv2d_97 (Conv2D)	(None, 123, 123, 64)	6,976
max_pooling2d_78 (MaxPooling2D)	(None, 61, 61, 64)	0
conv2d_98 (Conv2D)	(None, 56, 56, 128)	295,040
max_pooling2d_79 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_99 (Conv2D)	(None, 23, 23, 256)	1,179,904
max_pooling2d_80 (MaxPooling2D)	(None, 11, 11, 256)	0
dropout_19 (Dropout)	(None, 11, 11, 256)	0
flatten_26 (Flatten)	(None, 30976)	0
dense_26 (Dense)	(None, 1)	30,977

Layer (type)	Output Shape	Param #
conv2d_118 (Conv2D)	(None, 67, 67, 64)	6,976
max_pooling2d_97 (MaxPooling2D)	(None, 33, 33, 64)	0
conv2d_119 (Conv2D)	(None, 28, 28, 128)	295,040
max_pooling2d_98 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_120 (Conv2D)	(None, 9, 9, 256)	1,179,904
max_pooling2d_99 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_24 (Dropout)	(None, 4, 4, 256)	0
flatten_31 (Flatten)	(None, 4096)	0
dense_31 (Dense)	(None, 1)	4,097

Layer (type)	Output Shape	Param #
conv2d_144 (Conv2D)	(None, 126, 126, 64)	1,792
max_pooling2d_121 (MaxPooling2D)	(None, 42, 42, 64)	0
conv2d_145 (Conv2D)	(None, 40, 40, 128)	73,856
max_pooling2d_122 (MaxPooling2D)	(None, 13, 13, 128)	0
conv2d_146 (Conv2D)	(None, 11, 11, 256)	295,168
max_pooling2d_123 (MaxPooling2D)	(None, 3, 3, 256)	0
dropout_31 (Dropout)	(None, 3, 3, 256)	0
flatten_38 (Flatten)	(None, 2304)	0
dense_38 (Dense)	(None, 1)	2,305

Fig. 12. Trail Model Group



Layer (type)	Output Shape	Param #
conv2d_162 (Conv2D)	(None, 253, 253, 64)	3,136
max_pooling2d_136 (MaxPooling2D)	(None, 63, 63, 64)	0
conv2d_163 (Conv2D)	(None, 60, 60, 128)	131,200
max_pooling2d_137 (MaxPooling2D)	(None, 15, 15, 128)	0
conv2d_164 (Conv2D)	(None, 12, 12, 256)	524,544
max_pooling2d_138 (MaxPooling2D)	(None, 3, 3, 256)	0
dropout_37 (Dropout)	(None, 3, 3, 256)	0
flatten_44 (Flatten)	(None, 2304)	0
dense_44 (Dense)	(None, 1)	2,305

Fig. 13. CNN Best Result Model

Trained Convolutional SNN with SpikingJelly

This directly trained convolutional SNN had medium-level performance, with precision of 58.62%, recall of 72.86%, F1-score of 64.97%, and accuracy of 60.71%. Importantly, it had a very low execution time of 1.23 seconds, demonstrating its computational efficiency. The comparatively modest classification measures, however, suggest that the spiking dynamics and temporal integration of the network need to be better optimized to capture the discriminative features.

Untrained SNN with Simulated Output Using Nengo

The untrained SNN developed using Nengo that approximates the spiking behavior by repeating static inputs over time gave the worst performance on all the performance measures—precision of 53.49%, recall of 69.70%, F1-score of 60.53%, and accuracy of 53.12%. Apart from this, it also gave a significantly larger execution time of 11.02 seconds. These findings highlight the weakness of an untrained spiking network: without learning to update weights and tune firing thresholds, the network cannot generate consistent, discriminative spike patterns no matter how much increased simulation time.

ANN-to-SNN Conversion Using SpikingJelly

The SNN converted from a shallower, two-layer CNN produced a considerable improvement in performance. The method performed with 65.98% accuracy, 91.43% recall, 76.65% F1-score, and 72.14% accuracy, but at the cost of greater execution time of 18.80 seconds. The conversion leverages the use of a pre-trained CNN's feature representations; however, the relatively high simulation time suggests a requirement for additional calibration to optimize the spiking dynamics for real-time operation.

The best performance was obtained with the ANN to SNN conversion with a pre-trained MobileNet V2 model. This yielded great results, with 90.54% precision, 95.71% recall, 93.06% F1-Score, and 92.86% accuracy with an execution time of 6.77 seconds. With a mature ANN that has been widely tested in the literature, this conversion technique efficiently mitigated the expensive training cost generally concerned with SNNs. The high classification scores suggest that the MobileNet V2 conversion efficiently transferred the strong feature representations and decision boundaries of the ANN to spiking dynamics, although further calibration can still minimize simulation time.

The comparative analysis demonstrates that direct CSNN training is computationally efficient but limited in classification performance. Conversely, not fully trained but pre-trained SNNs with longer runtimes are not able to produce stable results. By contrast, pre-trained deep neural model-based conversion approaches, i.e., MobileNet V2, give a solution: the approaches greatly improve the classification performance with relatively small runtimes. The observation is to highlight the benefits of leveraging pre-trained models to balance the difficulty in initializing a spiking neural network from an uncalibrated perspective and to indicate the significance of careful calibration in the conversion process.

C. *Comparison and Discussion over all Algorithms*

TABLE I
ALGORITHMS PERFORMANCE MATRIX

Algorithms	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)	Execution Time (Seconds)
K-Means	46.94	47.14	46.25	52.86	0.38
Cluster GAN	53.61	53.57	53.46	53.57	1.92
Logistic Regression	58.21	55.71	56.93	57.86	**0.038**
CNN	63.64	50.00	56.00	68.15	3.84
SNN (CSNN)	58.62	72.86	64.97	60.71	1.23
SNN (Untrained)	53.49	69.70	60.53	53.12	11.02
SNN (Converted by Two-layer CNN)	65.98	91.43	76.65	72.14	18.80
SNN (Converted by MobileNet V2)	**90.54**	**95.71**	**93.06**	**92.86**	6.77

As shown in the table above, it presents the performance for various algorithms tested in this study. After comparison, the best-performing algorithm is SNN Converted by MobileNetV2, which significantly outperformed the other algorithms by precision, recall, F1-Score, and Accuracy. This indicates the model is well-calibrated for classification tasks, achieving a fine balance between identifying positive instances and minimizing false positives. The relatively longer execution time can be attributed to the complexity of the model, particularly the MobileNetV2 architecture, which requires more computational resources.

The worst-performing algorithm is K-Means. Although it is computationally efficient, its performance metrics reflect significant limitations in classification tasks. The low precision and recall suggest that the algorithm struggles to correctly classify data points similar to the test-given classification. The simplicity of the K-Means algorithm, which relies solely on distance metrics, may not be sufficient to capture the underlying patterns within such complex datasets.

The fastest-performing algorithm is Logistic Regression, which stood out as the fastest with an execution time of only 0.038 seconds. For applications such as real-time AIoT problems, where speed is crucial but performance can be compromised, Logistic Regression can be a suitable and practical choice.

VI. CONCLUSION

Our experiments and evaluation indicate that while traditional unsupervised and supervised algorithms offer computational efficiency, their classification performance is limited compared to deep learning methods. Directly trained SNNs are computationally efficient but achieve only moderate accuracy, and untrained Nengo-based SNNs perform poorly. In contrast, ANN-to-SNN conversion—especially using pre-trained MobileNet V2—delivers significantly higher precision, recall, and overall accuracy, validating its potential for energy-efficient, high-performance applications. Future work will focus on refining spiking dynamics and calibration techniques to further enhance the performance and applicability of SNNs in real-time AIoT systems.

REFERENCES

[1] H. Xu, Kah Phooi Seng, Li Minn Ang, and J. Smith, "Decentralized and Distributed Learning for AIoT: A Comprehensive Review, Emerging Challenges and Opportunities," *IEEE Access*, pp. 1–1, Jan. 2024, doi: https://doi.org/10.1109/access.2024.3422211.

[2] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "ClusterGAN: Latent Space Clustering in Generative Adversarial Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4610–4617, Jul. 2019, doi: https://doi.org/10.1609/aaai.v33i01.33014610.

[3] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[4] H. Neftci, S. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[5] W. Fang *et al.*, "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," *Science advances*, vol. 9, no. 40, pp. eadi1480-, 2023, doi: 10.1126/sciadv.adi1480