# Less-Energy-Usage Network with Batch Power Iteration

**Hao Huang** [1] **Tapan Shah** [1] **Scott Evans** [1] **Shinjae Yoo** [2]

## Abstract

Large scale neural networks are among the mainstream tools of modern big data analytic. But their training and inference phase are accompanied by huge energy consumption and carbon footprint. The energy efficiency, running time complexity and model storage size are three major considerations of using deep neural networks in modern applications. Here we introduce **L**ess-**E**nergy-Us**a**ge **N**etwork, or ***LEAN***. Different from regular network compression (e.g. pruning and knowledge distillation) that transform a pre-trained huge network to a smaller network, our method is to build a lean and effective network during training phase. It is based on spectral theory and batch power iteration learning. This technique can be applied to almost any type of neural networks to reduce their sizes. Preliminary experiment results show that our *LEAN* consumes $30\%$ less energy, achieving $95\%$ of the baseline accuracy with $1.5\times$ speed-up and $90\%$ less parameters compared against the baseline *CNN* model.

## 1. Introduction

Large-scale deep neural networks have shown impressive performance in many domains e.g. computer vision and natural language processing (Neill, 2020). However, training and using deep neural networks often require immense amount of carbon footprint, as well as increasing running time and storage. In this work, we propose a **L**ess-**E**nergy-Us**a**ge **N**etwork, or ***LEAN*** that can be used to learn a lean and effective network in the training phase. It is a dimensional reduction technique that can be generally applied to any network architecture. We show that the *LEAN* version of neural network maintains high accuracy but at the meanwhile takes much less energy, model storage and computational time in both training and inference phase.

[1]General Electric Research, Niskayuna, NY, USA [2]Brookhaven National Laboratory, Upton, NY, USA. Correspondence to: Hao Huang <hao.huang1@ge.com>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

## 2. Related Work

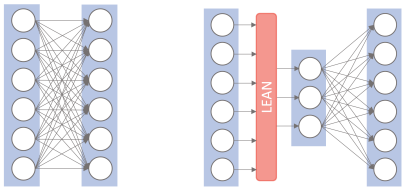
Recent years there has been a resurgence in model compression techniques, such as parameter pruning (Frankle & Carbin, 2018), low rank factorization (Swaminathan et al., 2020) and knowledge distillation (He et al., 2022). However, many of these methods are performed on a huge pre-trained network that already consumed large amount of energy during training. Quantization (Yang et al., 2019) reduces a neural network model's size by using fewer bits to represent its parameters. Many research focused on increasing computational efficiency in convolution operator, such as depth separable convolution (Chollet, 2017; Zhang et al., 2019; Kriman et al., 2020) and spatial separable convolution (Szegedy et al., 2016; Zhang et al., 2022). Some researchers applied Fast Fourier Transform (*FFT*) (Highlander & Rodriguez, 2016; Li et al., 2020) to implement convolution and they gain great speed up for large convolutional kernels. Scaling/balancing techniques were proposed on convolution network (Iandola et al., 2016; Tan & Le, 2019) to reduce the model size. Weight clustering-based model compression (Cho et al., 2021) try to reduce parameters by performing K-means on weights. However, these methods are either 1) not generally applicable to any type of neural network, 2) not for reducing energy consumption, or 3) with complexity that proportional to number of input channels $C_{in}$. Recently, energy efficient convolutional neural networks have been discussed in (Khatwani et al., 2018; Faraji et al., 2019), but they are either only applicable to the first layer of network or rely on specific hardware implementation.

## 3. Problem Setting and Methodology

As shown in Figure 1, our proposed *LEAN* is a dimensional reduction technique that can be generally applicable to almost any type of neural networks.

Simply, we explain it in the context of Convolutional Neural Network (*CNN*). The convolution operation is usually applied to two dimensional input data with multiple input channels $[N, H_{in}, W_{in}, C_{in}]$, and the size of output is $[N, H_{out}, W_{out}, C_{out}]$ where $N$ is the number of samples, $H$ and $W$ are height and width, and $C$ is the number of channels. The conventional computation of such convolution operator can be very heavy if both $C_{in}$ and $C_{out}$ are large, as its computational complexity is up to $O(C_{in}C_{out}K^2H_{in}W_{in})$

where $K$ is the kernel size.




(a) (b)

Figure 1: Original layer (1(a)) and its LEAN version (1(b)).

To reduce the computational cost, one way is to reduce the value of $C_{in}$. Here we propose a new technique to reduce $C_{in}$ to a much smaller dimensions $C'$ without introducing any new projection layer. Our idea is to extract lean embeddings from original dimension space with power iteration techniques (Lin & Cohen, 2010; Huang et al., 2014; 2015).

### 3.1. Power Iteration Embedding

To address the computational complexity of traditional spectral embedding construction (Ng et al., 2001), Lin et.al (Lin & Cohen, 2010) proposed power iteration clustering, which finds compact embedding (Power Iteration Embedding, or *PIE*) using truncated power iteration on a normalized affinity matrix that built upon input data.

Given an input data matrix $X \in \mathbb{R}^{n\times m}$, *PIE* is to reduce $X$ to $V \in \mathbb{R}^{n\times 1}$ by Algorithm 1. The theoretical foundation

**Algorithm 1** Power Iteration Embedding (*PIE*)

**Input:** $X \in \mathbb{R}^{n\times m}$, an initial embedding vector $V \in \mathbb{R}^{n\times 1}$
**Output:** embedding vector $V \in \mathbb{R}^{n\times 1}$
1: Construct the affinity matrix $A \in \mathbb{R}^{n\times n}$ by $XX^T$
2: Perform random walk normalization on $A$ so that each row of $A$ is $\ell_1$-normalized, denoted by $\tilde{A}$.
3: **repeat**
4: $\quad V \leftarrow \frac{\tilde{A}V}{\|\tilde{A}V\|_1}$
5: **until** $V$ converges.

of *PIE* is rooted in spectral theory (Ng et al., 2001). We denote the eigenvectors of $\tilde{A}$ (refer to line 2 in Algorithm 1) as $\Psi = \{\psi_1, ..., \psi_n\}$ that forms a basis in $\mathbb{R}^{n\times n}$. Therefore the initial vector $V$ in Algorithm 1 can be represented by a linear summation of $\Psi$:

$$V^{(0)} = a_1\psi_1 + a_2\psi_2 + ... + a_n\psi_n, \quad (1)$$

where $a_i$ is the coefficient/weight of the $i$-th eigenvector. The power iteration effect can be represented by:

$$V^{(t)} = \tilde{A}^t V^{(0)} = a_1\lambda_1^t\psi_1 + a_2\lambda_2^t\psi_2 + ... + a_n\lambda_n^t\psi_n$$
$$= a_1\lambda_1^t\psi_1 + \lambda_2^t\Big(\sum_{i=2}^{n} a_i(\frac{\lambda_i}{\lambda_2})^t\psi_i\Big), \quad (2)$$

where $\lambda$ are the eigenvalues of $\tilde{A}$ that have the order below:

$$1 = \lambda_1 > \lambda_2 > ... > \lambda_c \gg \lambda_{c+1} > ... > \lambda_n. \quad (3)$$

Here we assume the biggest eigengap exists between $c$ and $c+1$. Without early stopping, $V^{(t)}$ will finally converge to $a_1\psi_1$ (since $\lambda_1 = 1$) which is simply a constant vector. However, with a controlled early stopping, $V^{(t)}$ is a linear combination of the top $c$ informative eigenvectors, while all the other eigenvectors are decaying away due to eigengap. It has been proved that such embeddings have sufficient information that can be used in many tasks (Lin & Cohen, 2010; Huang et al., 2014; 2015).

### 3.2. Our proposed LEAN

Here we leverage the idea of *PIE* and propose *LEAN* that can be used in almost any type of neural network. As an example, we describe the usage of our *LEAN* on a *CNN* intermediate layer input with size $[N, H_{in}, W_{in}, C_{in}]$. In this case, the objective of *LEAN* is to reduce input channels $C_{in}$ to a much less number $C'$. The whole process is described in Algorithm 2. Specifically, we explain our *LEAN* in the following aspects:

1. In short, Algorithm 2 can be treated as a batch version of *PIE*. In line 1 of Algorithm 2, the original channels $C_{in}$ are split into preserved dimensions $C'$ and reducible dimensions $C''$ where $C_{in} = C'C''$. Similar to *PIE* that reduces dimension $m$ to 1, here our *LEAN* projects the original channels $C_{in}$ to $C'$ by reducing $C''$ to 1. In our implementation we use the function *view* in *pytorch* to split $C_{in}$. But any way of splitting, as long as it remains unchanged during training and inference phase, can be used. Our preliminary experiments show that the value of $C'$ can be set to $C_{in}/10$ or even $\sqrt{C_{in}}$ scale.

2. Different from $1\times1$ conv in inception module (Szegedy et al., 2015) or linear layer to project high dimensions to lower dimensions, our *LEAN* **doesn't introduce additional projection layers**. Instead, it is a self-dimensional-reduction technique.

3. Algorithm 2 can be understood as learning the corresponding Krylov subspace (Liesen & Strakos, 2013) of each pixel in each input sample based on its corresponding information in $\tilde{A}$. It is equivalent to finding the span of the Gramians but requires much less computations (Liesen & Strakos, 2013).

4. In practise, we simplify the repeat loop by setting the number of iteration as one and making the initial vector $V$ as learnable parameters that can be updated by back-propagation training. Specifically, setting number of iteration to be one has already been shown effective

and efficient in many power-iteration-based works such as (Yang et al., 2018). Setting the initial seed vector as learnable parameters is similar as setting the initial state trainable in Recurrent Neural Networks (*RNN*), that helps the model to start from a good default state.

5. Last but not least, our *LEAN* is **generally applicable to almost any type of networks**. For example, it can reduce any intermediate input in *RNN* $[N, L_{in}, C_{in}]$ to $[N, L_{out}, C']$ where $L_{in}$ and $L_{out}$ are the input and output sequence length. It can also be applied to simple fully connected network and reduce any intermediate input $[N, C_{in}]$ to $[N, C']$, where $C' \ll C_{in}$.

**Algorithm 2** Our proposed LEAN (on image data)

**Input:** $X \in \mathbb{R}^{N \times H_{in} \times W_{in} \times C_{in}}$, an initial embedding vector $V \in \mathbb{R}^{C' \times 1}$ where $C' \ll C_{in}$
**Output:** embedding vectors $V \in \mathbb{R}^{N \times H_{in} \times W_{in} \times C'}$

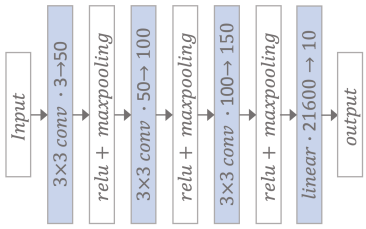
Split $C_{in}$ into two dimensions $[C', C'']$ (where $C_{in} = C'C''$), so the size of $X$ becomes $[N, H_{in}, W_{in}, C', C'']$
Construct the affinity matrix batch $A$ by batch matrix multiplication $XX^T$ where $X^T$ is transpose of $X$ between the last two dimensions. The size of $A$ is $[N, H_{in}, W_{in}, C', C']$
Perform random walk normalization on $A$ so that its second last dimension is $\ell_1$-normalized, denoted by $\tilde{A}$.
Expand the initial vector $V$ with dummy dimensions at the front, so the size of $V$ becomes $[N, H_{in}, W_{in}, C', 1]$.
**repeat**
  $V \leftarrow \frac{\tilde{A}V}{\|\tilde{A}V\|_1}$
**until** $V$ converges (in practice the $\#iteration = 1$).
return $V$ with the last dimension squeezed.

## 4. Experiments

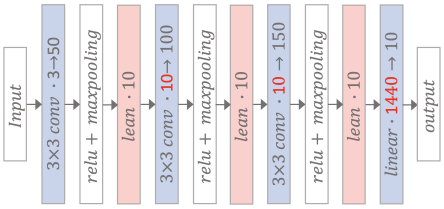
To discuss the trade-off between efficiency and accuracy from a Green AI (Schwartz et al., 2020) perspective, we consider prediction accuracy, energy-consumption, CO2-emission, running time and model size. Due to page limitation we focus on the comparison between our *LEAN* and a regular *CNN* model on *STL-10* dataset (Coates et al., 2011).

### 4.1. Network Structure and Model Size Comparison

Figure 2(a) illustrates the *CNN* baseline structure. It consists of three *conv2D* layers and one fully connected layer. The numbers of channels in the three *conv2D* layers are 50, 100, and 150, so the total number of parameters is about $400k$. In Figure 2(b), we apply our *LEAN* after each *conv2D* to reduce the $C_{in}$ in the subsequent layers. We set the preserved dimensions $C'$ to be 10, and our total number of parameters is about $40k$ that **reduces 90% parameters of the baseline *CNN* structure**.




(a) *CNN* Structure




(b) our *LEAN* on *CNN*

Figure 2: A *CNN* structure and its *LEAN* version.

### 4.2. Convergence and Prediction Accuracy Comparison

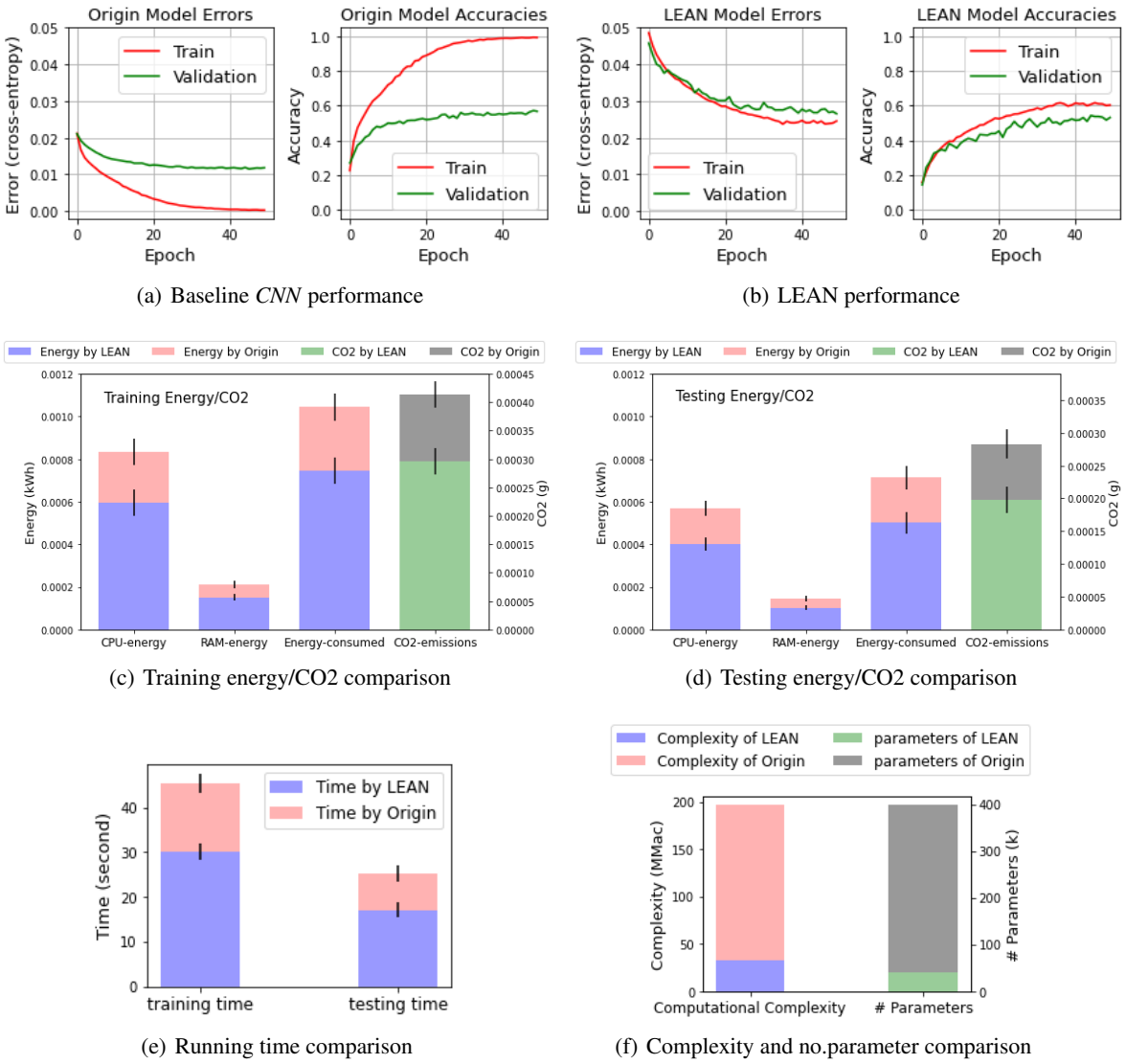
In Figure 3(a) and 3(b) we show the error convergence and prediction accuracy plot of *CNN* baseline and our *LEAN* along the first 50 training epochs. From error plots, we can see that both the training and validation loss decrease to a point of stability that indicate the two models converge well. The validation and testing accuracy by our *LEAN* **achieve 95% of the accuracy** by the baseline *CNN* model. The error and accuracy difference between training and validation are much smaller in our *LEAN* compared against baseline *CNN*. It suggests that our *LEAN* may suffer less from overfitting than baseline. More experiment and exploration will be conducted in our coming work to explore this direction.

### 4.3. Energy Consumption and CO2 Emission

We use *CodeCarbon*[1] to measure energy consumption and CO2 emission. *CodeCarbon* is an open-source Python library that estimates the produced CO2 and consumed energy while running the code. The experiments were performed on a local machine with 32GB Memory and 2.9 GHz Intel Core i9 without any GPU support, so the estimation is done by measuring the power consumption of CPUs and RAM.

The training and testing energy-consumption and CO2-emission (by one epoch) are shown in Figure 3(c) and 3(d). We can see that by using our proposed *LEAN*, we can **reduce 30% energy consumption and CO2 emission**. This is equivalent to 125 km travelled by a regular car if we train

[1]https://github.com/mlco2/codecarbon




(a) Baseline *CNN* performance

(b) LEAN performance

(c) Training energy/CO2 comparison

(d) Testing energy/CO2 comparison

(e) Running time comparison

(f) Complexity and no.parameter comparison

Figure 3: Overall comparison between the baseline *CNN* and our proposed *LEAN* on *STL-10*.

1000 models with 1000 epochs.

### 4.4. Running Time and Complexity Comparison

The running time and computational complexity comparison results are shown in Figure 3(e) and 3(f). Our *LEAN* is 1.5 times faster than baseline *CNN* in both training and testing phase. We also measure the Multiply-Accumulate Computations (MACs). Our *LEAN* is about one-sixth of the baseline *CNN* model.

## 5. Conclusion

We introduce **L**ess-**E**nergy-Us**a**ge **N**etwork, or ***LEAN*** that can reduce immediate network input. Different from regular network compression that transform a pre-trained huge network to a smaller network, our method is to build a lean and effective network in the training phase. Preliminary result showed that our *LEAN* preserves 95% accuracy of the baseline model, but at the meanwhile reduces 30% of energy consumption and CO2 emission. Future work will focus on 1) testing the energy consumption and CO2 emission using GPUs on server environment; 2) exploring the effect of our *LEAN* on more complicated network structures (e.g. AlexNet, VGG16, Resnet-152, etc.). Another interesting aspect is that from computational complexity comparison in Figure 3(f), we can see that our *LEAN* has potential to be be more efficient in terms of running time and energy consumption. We will further optimize our implementation and improve efficiency in saving energy.

## References


Cho, M., Vahid, K. A., Adya, S., and Rastegari, M. Dkm: Differentiable k-means clustering layer for neural network compression. *arXiv preprint arXiv:2108.12659*, 2021.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Faraji, S. R., Najafi, M. H., Li, B., Lilja, D. J., and Bazargan, K. Energy-efficient convolutional neural networks with deterministic bit-stream processing. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1757–1762. IEEE, 2019.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

He, H., Wang, J., Zhang, Z., and Wu, F. Compressing deep graph neural networks via adversarial knowledge distillation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 534–544, 2022.

Highlander, T. and Rodriguez, A. Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add. *arXiv preprint arXiv:1601.06815*, 2016.

Huang, H., Yoo, S., Yu, D., and Qin, H. Diverse power iteration embeddings and its applications. In *2014 IEEE International Conference on Data Mining*, pp. 200–209. IEEE, 2014.

Huang, H., Yoo, S., Yu, D., and Qin, H. Diverse power iteration embeddings: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 28(10): 2606–2620, 2015.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Khatwani, M., Hosseini, M., Paneliya, H., Mohsenin, T., Hairston, W. D., and Waytowich, N. Energy efficient convolutional neural networks for eeg artifact detection. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4. IEEE, 2018.

Kriman, S., Beliaev, S., Ginsburg, B., Huang, J., Kuchaiev, O., Lavrukhin, V., Leary, R., Li, J., and Zhang, Y. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6124–6128. IEEE, 2020.

Li, S., Xue, K., Zhu, B., Ding, C., Gao, X., Wei, D., and Wan, T. Falcon: A fourier transform based approach for fast and secure convolutional neural network predictions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8705–8714, 2020.

Liesen, J. and Strakos, Z. *Krylov subspace methods: principles and analysis*. Oxford University Press, 2013.

Lin, F. and Cohen, W. W. Power iteration clustering. 2010.

Neill, J. O. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.

Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

Swaminathan, S., Garg, D., Kannan, R., and Andres, F. Sparse low rank factorization for deep neural network compression. *Neurocomputing*, 398:185–196, 2020.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., and Hua, X.-s. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

Yang, P., Hsieh, C.-J., and Wang, J.-L. History pca: A new algorithm for streaming pca. *arXiv preprint arXiv:1802.05447*, 2018.

Zhang, R., Zhu, F., Liu, J., and Liu, G. Depth-wise separable convolutions and multi-level pooling for an efficient spatial cnn-based steganalysis. *IEEE Transactions on Information Forensics and Security*, 15:1138–1150, 2019.

Zhang, S., Zhang, X., Li, H., He, H., Song, D., and Wang, L. Hierarchical pyramid attentive network with spatial separable convolution for crowd counting. *Engineering Applications of Artificial Intelligence*, 108:104563, 2022.