Preprints.org

Article

Not peer-reviewed version

# Is Matrix Neural Network the Alternative of Convolutional Neural Network?

Loc Nguyen *

Posted Date: 2 April 2026

doi: 10.20944/preprints202604.0170.v1

Keywords: deep learning; artificial neural network (ANN); convolutional neural network (CNN); attention-based transformer; matrix neural network (MNN); image classification

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

*Article*

# Is Matrix Neural Network the Alternative of Convolutional Neural Network?

**Loc Nguyen**

Loc Nguyen's Academic Network Vietnam; ng_phloc@yahoo.com

**Abstract**

Currently (2025), deep learning is the most important and popular methodology in artificial intelligence (AI) and artificial neural network (ANN) is the foundation of deep learning. The main drawback of ANN is the boom problem of a huge number of parametric weights when ANN in deep learning establishes a large number of hidden layers. The boom problem can be alleviated by high-performance computer but will be serious in case of high-dimension input data like image. The excellent solution for image processing within context of deep learning is that large parametric weight vector is reduced into much smaller window encoded by a so-called filtering kernel which is often 3x3 matrix or 5x5 matrix which is convoluted over entire image data. ANN with support of such filtering kernel is called convolutional neural network (CNN). Many researches prove that CNN is feasible and effective in image processing. The hidden cause of the effectiveness of CNN is that the visionary structure of an image is aggregated in such a way that filtering kernel is ideal to extract image features. However, it is not asserted that matrix-based filtering kernel is appropriate to other high-dimension data that is not image. Another solution of the boom problem is that large parametric weight vector is organized as matrix that is the same structure of 2-dimension data like image, which leads to a so-called matrix neural network (MNN) whose parameters are weighted matrices. Computation cost of MNN is decreased significantly in comparison with ANN but it is necessary to test the effectiveness of MNN with respect to CNN. This is the main hypothesis "whether MNN is the alternative of CNN" which is tested in this research, hinted by the research title. Moreover, transformer which is the new trend (2025) in AI and deep learning, which aims to improve/replace traditional ANN by self-supervised learning, in which attention is the significant mechanism of self-supervised learning. Anyhow, attention which is the cornerstone of transformer is the representation of internal structure/relationship inside high-dimension data like image. Therefore, the implicit deep meanings of attention and filtering kernel are similar, which represents feature of data, which does not go beyond parametric weights too. In general, the research has two goals: 1) explaining and implementing ANN, CNN, and transformer (attention) and 2) applying analysis of variance (ANOVA) into evaluating the effectiveness of ANN, CNN, and transformer (attention) within context of image classification. The ultimate result is that it is not asserted that MNN is the alternative of CNN but MNN can be an optional choice for implementing ANN in context of image processing instead of focusing on the unique CNN solution. Moreover, the incorporation of MNN and attention in implementing transformer produces a compromising solution of high performance and computational cost.

**Keywords:** deep learning; artificial neural network (ANN); convolutional neural network (CNN); attention-based transformer; matrix neural network (MNN); image classification

## 1. Introduction

Machine learning is classified into three subjects such as supervised learning, unsupervised learning, and reinforcement learning, in which self-supervised learning is the intermediate one between supervised learning and unsupervised learning. Artificial intelligence (AI), which is the sub-domain of machine learning, can be considered as the core of machine learning, where *artificial neural*

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



*network* (*ANN*) is the preeminent approach built in AI. Fortunately, ANN supports fully the four subjects such as supervised learning, self-supervised learning, unsupervised learning, and reinforcement learning. ANN which simulates human neural network consists of one input layer, many enough hidden layers, and one output layer so that the information is entered input layer, then is propagated through hidden layers, and finally evaluated at output layer which is the result of mentioned machine learning approaches, according to a so-called *propagation rule*. For instance, given layer $k$ specified by vector variable $\boldsymbol{x}_k = (x_{k1}, x_{k2},\ldots, x_{kn})^T$ is evaluated by the previous layer $\boldsymbol{x}_{k-1}$ by propagation rule as follows:

$$\boldsymbol{x}_k = f(\hat{\boldsymbol{x}}_{k-1}) = f(W_k\boldsymbol{x}_{k-1} + \Theta_k)$$
$$\hat{\boldsymbol{x}}_{k-1} = W_k\boldsymbol{x}_{k-1} + \Theta_k$$

Or shortly,

$$\boldsymbol{x}_k = f(\hat{\boldsymbol{x}}_{k-1} = W_k\boldsymbol{x}_{k-1} + \Theta_k)$$

Note, $W_k$ and $\Theta_k$ are weight parameter and bias parameter at layer $k$, where $W_k$ is weight matrix and is $\Theta_k$ bias vector as usual. The function $f(.)$ is called activation function which is often squash function like sigmoid function. If it focuses on the particular $k^{\text{th}}$ layer, it will be denoted as $f_{\boldsymbol{x}_k}(\hat{\boldsymbol{x}}_{k-1})$. Training ANN is to estimate the parameters $W_k$ and $\Theta_k$ and a popular training method is the association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm, based on predefined likelihood function or predefined error function. When the number of hidden layers is large enough, ANN is called deep neural network (DNN) which is the basic of deep learning. The common representation of artificial neural network (ANN) is *feedforward network* (FFN) in which the input layer is fed forwards so as to evaluate the output layer and so, ANN mentioned in this research is FNN if there is no additional explanation. Given FFN with vector layers represented by a series of layers $\boldsymbol{x}_0$, $\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_K$ is called $K$-layer FFN which is represented as follows:

$$\boldsymbol{x}_k = f(\hat{\boldsymbol{x}}_{k-1} = W_k\boldsymbol{x}_{k-1} + \Theta_k), \forall k = \overline{1,K}$$

Classification is the most popular method belonging to supervised learning, which is implemented perfectly by artificial neural network (ANN). As usual, training FFN classifier (ANN classifier) aims to minimize the so-called *cross-entropy loss function* loss($\boldsymbol{x}$) given variable vector $\boldsymbol{x} = (x_1, x_2,\ldots, x_n)^T$ being the output layer $\boldsymbol{x}_K=\boldsymbol{x}$. For instance, given the last output vector $\boldsymbol{x}_K = (x_{K1}, x_{K2},\ldots, x_{Kn})^T$ computed from ANN and the real class probability $\boldsymbol{p} = (p_1, p_2,\ldots, p_n)^T$ of such vector $\boldsymbol{x}_K$, the cross-entropy loss function loss($\boldsymbol{x}_K$) is specified as follows:

$$\text{loss}(\boldsymbol{x}_K) = -\sum_{i=1}^{n} p_i \log(\hat{p}_i)$$

Where,

$$\hat{p}_i = \text{softmax}(x_{Ki}) = \frac{\exp(x_{Ki})}{\sum_{l=1}^{n}\exp(x_{Kl})}$$

So that:

$$W_K^* = \underset{W_K}{\text{argmin}}\,\text{loss}(\boldsymbol{x}_K)$$
$$\Theta_K^* = \underset{\Theta_K}{\text{argmin}}\,\text{loss}(\boldsymbol{x}_K)$$

The parameters $W_K$ and $\Theta_K$ at the last layer $K$ are estimated by stochastic gradient descent (SGD) algorithm as follows:

$$W_K = W_K + \gamma\nabla_{W_K} l$$
$$\Theta_K = \Theta_K + \gamma\nabla_{\Theta_K} l$$

Where $\gamma$ ($0 < \gamma \le 1$) is learning rate. Note, $\nabla_{W_K} l$ and $\nabla_{\Theta_K} l$ are gradients of loss($\boldsymbol{x}_K$) with respect to $W_K$ and $\Theta_K$, respectively, which are determined based on the cross-entropy gradient $\nabla$loss($\boldsymbol{x}$) that is calculated as following row vector:

$$\nabla\text{loss}(\boldsymbol{x}) = \left(\frac{\partial\text{loss}(\boldsymbol{x})}{\partial x_1}, \frac{\partial\text{loss}(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial\text{loss}(\boldsymbol{x})}{\partial x_n}\right) = \left(\frac{\partial\text{loss}(\boldsymbol{x})}{\partial x_j}\right)_{j=\overline{1,n}}$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Where $\frac{\partial \text{loss}(\boldsymbol{x})}{\partial x_j}$ is the partial derivative of loss($\boldsymbol{x}$) with respect to $x_j$ as follows:

$$\frac{\partial \text{loss}(\boldsymbol{x})}{\partial x_j} = \frac{\partial(-\sum_{i=1}^{n} p_i \log(\hat{p}_i))}{\partial x_j} = -\sum_{i=1}^{n} \frac{p_i}{\hat{p}_i} \frac{d\hat{p}_i}{dx_j}$$

Where $\frac{d\hat{p}_i}{dx_j}$ is derivative of real probability $\hat{p}_i$ with respect to $x_j$ as follows:

$$\frac{d\hat{p}_i}{dx_j} = \frac{d\left(\frac{\exp(x_i)}{\sum_{l=1}^{n}\exp(x_l)}\right)}{dx_j} = \frac{1}{(\sum_{l=1}^{n}\exp(x_l))^2}\left(\frac{d\exp(x_i)}{dx_j}\sum_{l=1}^{n}\exp(x_l) - \exp(x_i)\sum_{l=1}^{n}\frac{d\exp(x_l)}{dx_j}\right)$$

If $i=j$, we have:

$$\frac{d\hat{p}_i}{dx_j} = \frac{d\hat{p}_i}{dx_i} = \frac{\exp(x_i)}{\sum_{l=1}^{n}\exp(x_l)} - \left(\frac{\exp(x_i)}{\sum_{l=1}^{n}\exp(x_l)}\right)^2 = \hat{p}_i(1-\hat{p}_i)$$

If $i\neq j$, we have:

$$\frac{d\hat{p}_i}{dx_j} = \frac{d\hat{p}_i}{dx_i} = -\frac{\exp(x_i)}{\sum_{l=1}^{n}\exp(x_l)}\frac{\exp(x_j)}{\sum_{l=1}^{n}\exp(x_l)} = \hat{p}_i(0-\hat{p}_j)$$

This implies the derivative $\frac{d\hat{p}_i}{dx_j}$ of real probability $\hat{p}_i$ with respect to $x_j$ is specified as follows:

$$\frac{d\hat{p}_i}{dx_j} = \hat{p}_i(\delta_{ij} - \hat{p}_j)$$

Where,

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

Therefore, the partial derivative of loss($\boldsymbol{x}$) with respect to $x_j$ is determined as follows:

$$\frac{\partial \text{loss}(\boldsymbol{x})}{\partial x_j} = -\sum_{i=1}^{n}\frac{p_i}{\hat{p}_i}\hat{p}_i(\delta_{ij}-\hat{p}_i) = -\sum_{i=1}^{n} p_i(\delta_{ij}-\hat{p}_j)$$

As a result, cross-entropy gradient $\nabla$loss($\boldsymbol{x}$) is totally determined as follows:

$$\nabla\text{loss}(\boldsymbol{x}) = -\left(\sum_{i=1}^{n} p_i(\delta_{ij}-\hat{p}_j)\right)_{j=\overline{1,n}} = -\begin{pmatrix} \sum_{i=1}^{n} p_i(\delta_{i1}-\hat{p}_1) \\ \sum_{i=1}^{n} p_i(\delta_{i2}-\hat{p}_2) \\ \vdots \\ \sum_{i=1}^{n} p_i(\delta_{in}-\hat{p}_n) \end{pmatrix}^T$$

Where,

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

$$\hat{p}_j = \text{softmax}(x_j) = \frac{\exp(x_j)}{\sum_{l=1}^{n}\exp(x_l)}$$

So that:

$$\nabla_{W_K} l = (\nabla_{\Theta_K} l)\boldsymbol{x}_{K-1}^T$$

$$\nabla_{\Theta_K} l = -\left(f'_{x_K}(\hat{\boldsymbol{x}}_{K-1})\right)^T(\nabla\text{loss}(\boldsymbol{x}))^T = \left(f'_{x_K}(\hat{\boldsymbol{x}}_{K-1})\right)^T\left(\sum_{i=1}^{n} p_i(\delta_{ij}-\hat{p}_j)\right)_{j=\overline{1,n}}^T$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Note, the superscript "$T$" denotes transposition operator of vector and matrix. By association of SGD and backpropagation algorithm, all gradients $\nabla_{W_k} l$ and $\nabla_{\Theta_k} l$ of all layers from $k$=1 to $k$=$K$ are determined as follows:

$$\begin{aligned}
&\nabla_{W_k} l = \left(\nabla_{\Theta_k} l\right) \boldsymbol{x}_{k-1}^T, \forall k = \overline{1,K} \\
&\nabla_{\Theta_K} l = \left(f'_{x_K}(\hat{\boldsymbol{x}}_{K-1})\right)^T \left(\sum_{i=1}^{n} p_i(\delta_{ij} - \hat{p}_j)\right)^T_{j=\overline{1,n}} \\
&\nabla_{\Theta_k} l = \left(f'_{x_k}(\hat{\boldsymbol{x}}_{k-1})\right)^T W_{k+1}^T \nabla_{\Theta_{k+1}} l \text{ if } k = \overline{1,K-1}
\end{aligned} \tag{1.1}$$

So that ANN classifier are totally trained by estimating their parameters $W_k$ and $\Theta_k$ from $k$=1 to $k$=$K$ as follows:

$$\begin{aligned}
W_K &= W_K + \gamma \nabla_{W_k} l \\
\Theta_K &= \Theta_K + \gamma \nabla_{\Theta_k} l
\end{aligned}$$

Data classification become extremely hazard when the data is image which is 2-dimension data, which raises the problem of *image classification* consisting of two issues: 1) context meaning of image is too difficult to be caught and 2) high-resolution image causes the so-called boom problem of a huge number of parameters so that training ANN consumes a lot of time and resources. Convolutional neural network (CNN) is invented to solve such boom problem, in which an image is filtered via a so-called filter so as to extract the feature of such image, then such feature is fed as input of ANN for classification. The feature extraction has two goals: 1) dimension of image data space is reduced significantly by CNN filter so as to make ANN classifier to be feasible in training and 2) feature which is the essential aspect of image such as edge and pattern makes the image classification more accurate. Especially, parameter size in CNN which is the size of kernel matrix is much smaller than the size of weight matrix in ANN, which decreases significantly computational cost. In general, CNN is an excellent approach to image classification and so, it is necessary to sketch its methodology so that it is possible to compare matrix neural network (Gao, Guo, & Wang, 2016) and CNN. Essentially, convolutional neural network (CNN) has two connected parts: 1) the first part called *convolutional network* consists of a set of sequential convolutional layers which of them is filtered by a *k*x*k* filter and 2) the second part called fully connected network or dense network takes output of convolutional network as its input feature and then performs classification task on such feature with note that the output of convolutional network is image feature, essentially. Such classification task has just described but please pay attention that the main point of CNN is convolutional network (not dense network). Shortly, the input and output of convolutional network are image and image feature, respectively whereas the input and output of dense network are image feature and image class, respectively. Dense network is feedforward network (FFN) as usual whereas each convolutional layer is a 2-dimension data/feature that was filtered by a filter. Given an *m*x*n* image as *m*x*n* matrix and given *k*x*k* filter then the corresponding convolutional layer is reduced as (*m*/*k*)x(*n*/*k*) matrix after filtering, which means that image size is decreased $k^2$ times with *k*x*k* filter. For instance, given an image as 2-dimension *m*x*n* matrix denoted $\boldsymbol{X} = (x_{ij})_{mxn}$, a *filter* also called kernel or filtering kernel is a *k*x*k* square matric denoted $W = (w_{ij})_{kxk}$, which is often 3x3 or 5x5 matrix. Given pixel $\boldsymbol{X}[i][j] = x_{ij} = x_{i,j}$ at the $i^{th}$ row and $j^{th}$ column of matrix $\boldsymbol{X}$, the respective value $y$ which is a cell of convolutional layer $\boldsymbol{Y}$, which is called *convolutional value* or convolutional cell, is calculated by a so-called *filter operator* which is the sum of multiplications of neighbor pixels and $W = (w_{ij})_{kxk}$.

$$y_{ij} = \left(\sum_{u,v} w_{uv} x_{i+u,j+v}\right) + \theta = \left(\sum_{u=1}^{k}\sum_{v=1}^{k} w_{uv} x_{i+u,j+v}\right) + \theta$$

Both $W$ and $\theta$ are parameters of convolutional layer where $W$ is called filter weight matrix and $\theta$ is called filter bias. Activation function $f$(.) can be applied into filter operator like ANN does:

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$y_{ij} = f\left(\hat{x}_{ij} = \left(\sum_{u,v} w_{uv} x_{i+u,j+v}\right) + \theta\right)$$

But the common activation function for CNN is Rectified Linear Unit (ReLU) which limits its input in the interval $[a, b]$.

$$\text{ReLU}(x) = \max\big(a, \min(b, x)\big)$$

Indeed, the filter $W$ is slid over entire image $\boldsymbol{X}$ so as to complete convolutional layer $\boldsymbol{Y}$ and so, the movement of filter is jumped step-by-step where the step length is called *stride* denoted $s$. As usual, stride $s$ is in the range $[1, k]$ such that $1 \leq s \leq k$ where $k$ is often called *filter size* for filter $W = (w_{ij})_{k \times k}$. If image $\boldsymbol{X}$ is $n \times n$ matrix and filter $W$ is $k \times k$ matrix given stride $s$, then the size $o$ of convolutional layer $\boldsymbol{Y}_{p \times p}$ is:

$$o = 1 + \frac{n - k}{s}$$

When the filtering kernel $W$ move directionally to the edge (left column) of image, there are some residual pixels that are not fit totally to $W$ and so, these pixels are called padding pixels. As usual, padding pixel is filled by zero so as to form the so-called zero-padding technique. Let $p$ be the number of padding columns, the size $o$ of convolutional layer is modified a little bit as follows:

$$o = 1 + \frac{n - k + 2p}{s}$$

The effectiveness of a CNN depends on how to specify the set of sequential convolutional layers associated with their filters so that there are many scientific researches that defines very appropriate filters to extract (image) features as much as possible. However, this research focuses on specifying general filters and how to train such filters in order to generalize CNN in comparison with matrix neural network mentioned later. Therefore, it is necessary to skim over convolutional filter training (Gemini 2025). Given $K$-layer CNN whose layers are $\boldsymbol{X}_0, \boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_K$ where each convolutional layer $\boldsymbol{X}_k$ has parametric filter weight matrix $W^{(k)}$ and parametric filter bias $\theta^{(k)}$.

$$x_{ij}^{(k)} = f\left(\hat{x}_{ij}^{(k-1)} = \left(\sum_{u,v} w_{uv}^{(k)} x_{i+u,j+v}^{(k-1)}\right) + \theta^{(k)}\right)$$

Note,

$$x_{ij}^{(k)} \in \boldsymbol{X}_k$$
$$x_{i+u,j+v}^{(k-1)} \in \boldsymbol{X}_{k-1}$$
$$w_{uv}^{(k)} \in W^{(k)}$$

Let $l\big(x_{ij}^{(K)}\big)$ be the *likelihood function* which takes $x_{ij}^{(K)}$ as its input at the last layer $\boldsymbol{X}_K$, as usual, $l\big(x_{ij}^{(K)}\big)$ is the negative of error function:

$$l\big(x_{ij}^{(K)}\big) = -\varepsilon\big(x_{ij}^{(K)}\big)$$

Let $l(\boldsymbol{X}_K)$ be the entire likelihood function which is the mean of $l\big(x_{ij}^{(K)}\big)$ over all points $x_{ij}^{(K)}$ (s) belonging to $\boldsymbol{X}_K$.

$$l(\boldsymbol{X}_K) = \frac{1}{|\boldsymbol{X}_K|} \sum_{i,j} l\big(x_{ij}^{(K)}\big) = \frac{1}{|\boldsymbol{X}_K|} \sum_{i} \sum_{j} l\big(x_{ij}^{(K)}\big)$$

Or,

$$l(\boldsymbol{X}_K) = \sum_{i,j} l\big(x_{ij}^{(K)}\big) = \sum_{i} \sum_{j} l\big(x_{ij}^{(K)}\big)$$

Where the notation $|\boldsymbol{X}_K|$ denotes the size of layer $\boldsymbol{X}_K$, for instance, if $\boldsymbol{X}_K$ is $m \times n$ matrix, then its size $|\boldsymbol{X}_K|$ is $m^* n$. Training CNN within association of stochastic gradient descent (SGD) algorithm and

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



backpropagation algorithm is to estimate parameters $W^{(k)}$ and $\theta^{(k)}$ of all convolutional layers. The parameters $W^{(k)}$ and $\theta^{(k)}$ of the last layer $\boldsymbol{X}_K$ are estimated firstly:

$$\left(W^{(K)}\right)^* = \underset{W^{(K)}}{\operatorname{argmax}}\, l(\boldsymbol{X}_K)$$
$$\left(\theta^{(K)}\right)^* = \underset{\theta^{(K)}}{\operatorname{argmax}}\, l(\boldsymbol{X}_K)$$

According SGD, parametric filter $w_{uv}^{(K)}$ and parametric bias $\theta^{(K)}$ are estimated iteratively by addition of itself and gradients of likelihood function $l(\boldsymbol{X}_K)$ with respect to $w_{uv}^{(K)}$ and $\theta^{(K)}$, respectively, increased by step of learning rate $\gamma$. As a result, training CNN is essentially to calculate these gradients. For last layer $\boldsymbol{X}_K$, we have:

$$w_{uv}^{(K)} = w_{uv}^{(K)} + \gamma \nabla_{w_{uv}^{(K)}} l(\boldsymbol{X}_K)$$
$$\theta^{(K)} = \theta^{(K)} + \gamma \nabla_{\theta^{(K)}} l(\boldsymbol{X}_K)$$

Where,

$$\nabla_{w_{uv}^{(K)}} l = \nabla_{w_{uv}^{(K)}} l(\boldsymbol{X}_K) = \sum_{i:u,j:v} \nabla_{w_{uv}^{(K)}} l\left(x_{ij}^{(K)}\right)$$
$$\nabla_{\theta^{(K)}} l = \nabla_{\theta^{(K)}} l(\boldsymbol{X}_K) = \sum_{i:u,j:v} \nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right)$$

Please pay attention that the notation "$i{:}u$, $j{:}v$" implies that indices $i$ and $j$ are iterated given indices $u$ and $v$ where every product $\hat{x}_{ij}^{(K-1)} = \left(\sum_{u',v'} w_{u'v'}^{(K)} x_{i+u',j+v'}^{(K-1)}\right)$ is satisfied so that there is only one pair of $u{=}u'$ and $v{=}v'$. Let $\nabla_{w_{uv}^{(K)}} l\left(x_{ij}^{(K)}\right)$ and $\nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right)$ be gradients of likelihood function $l(z_{ij}^{(K)})$ with respect to $w_{uv}^{(K)}$ and $\theta^{(K)}$, respectively, as follows (Gemini 2025):

$$\nabla_{w_{uv}^{(K)}} l\left(x_{ij}^{(K)}\right) = \frac{dl\left(x_{ij}^{(K)}\right)}{dw_{uv}^{(K)}} = \frac{dl\left(x_{ij}^{(K)}\right)}{dx_{ij}^{(K)}} \frac{dx_{ij}^{(K)}}{d\hat{x}_{ij}^{(K-1)}} \frac{\partial \hat{x}_{ij}^{(K-1)}}{\partial w_{uv}^{(K)}} = l'\left(x_{ij}^{(K)}\right) f'\left(\hat{x}_{ij}^{(K-1)}\right) x_{i+u,j+v}^{(K-1)}$$

$$\nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right) = \frac{dl\left(x_{ij}^{(K)}\right)}{d\theta^{(K)}} = \frac{dl\left(x_{ij}^{(K)}\right)}{d\hat{x}_{ij}^{(K-1)}} = \frac{dl\left(x_{ij}^{(K)}\right)}{dx_{ij}^{(K)}} \frac{dx_{ij}^{(K)}}{d\hat{x}_{ij}^{(K-1)}} \frac{\partial \hat{x}_{ij}^{(K-1)}}{\partial \theta^{(K)}} = l'\left(x_{ij}^{(K)}\right) f'\left(\hat{x}_{ij}^{(K-1)}\right)$$

We obtain the equation to learn parametric filter $w_{uv}^{(K)}$ and parametric bias $\theta^{(K)}$ at output layer $\boldsymbol{X}_K$.

$$\nabla_{w_{uv}^{(K)}} l = \sum_{i:u,j:v} x_{i+u,j+v}^{(K-1)} \nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right)$$
$$\nabla_{\theta^{(K)}} l = \sum_{i,j} \nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right)$$
$$\nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right) = f'\left(\hat{x}_{ij}^{(K-1)}\right) l'\left(x_{ij}^{(K)}\right)$$

By generalization, parametric filter $w_{uv}^{(k)}$ and parametric bias $\theta^{(k)}$ at any layer $\boldsymbol{X}_k$ where $1 \le k \le K$ are totally determined as follows:

$$\nabla_{w_{uv}^{(k)}} l = \sum_{i:u,j:v} x_{i+u,j+v}^{(k-1)} \nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right)$$
$$\nabla_{\theta^{(k)}} l = \sum_{i,j} \nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right)$$

The quantity $\nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right)$ is also called *elemental parametric error* at any layer $\boldsymbol{X}_k$ where $1 \le k \le K$. Although it is determined given at the last layer $\boldsymbol{X}_K$ as:

$$\nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right) = f'\left(\hat{x}_{ij}^{(K-1)}\right) l'\left(x_{ij}^{(K)}\right)$$

It is necessary to generalize it at layer $\boldsymbol{X}_k$ where $1 \le k \le K$. Indeed, this elemental parametric error will be propagated backward from layer $\boldsymbol{X}_K$ back to layer $\boldsymbol{X}_{K-1}$ and so, let $\nabla_{x_{i+u,j+v}^{(K-1)}} l\left(x_{ij}^{(K-1)}\right)$ be the error of every element at layer $\boldsymbol{X}_{K-1}$ (Gemini 2025):

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$\nabla_{x_{i+u,j+v}^{(K-1)}} l\left(x_{ij}^{(K-1)}\right) = \frac{dl\left(x_{ij}^{(K)}\right)}{dx_{i+u,j+v}^{(K-1)}} = \frac{dl\left(x_{ij}^{(K)}\right)}{dx_{ij}^{(K)}} \frac{dx_{ij}^{(K)}}{d\hat{x}_{ij}^{(K-1)}} \frac{\partial \hat{x}_{ij}^{(K-1)}}{\partial x_{i+u,j+v}^{(K-1)}} = l'\left(x_{ij}^{(K)}\right) f'\left(\hat{x}_{ij}^{(K-1)}\right) w_{uv}^{(K)} = \nabla_{\theta^{(K)}} l\left(x_{ij}^{(K)}\right) w_{uv}^{(K)}$$

In practice, the error of every element $x_{ij}$ at layer $\boldsymbol{X}_{K-1}$ is the sum of $\nabla_{x_{i+u,j+v}^{(K-1)}} l\left(x_{ij}^{(K-1)}\right)$ over kernel $W^{(K)}$ as follows:

$$\nabla_{x_{ij}^{(K-1)}} l\left(x_{ij}^{(K-1)}\right) \cong \sum_{i':i,j':j} \nabla_{\theta^{(K)}} l\left(x_{i'j'}^{(K)}\right) w_{u'v'}^{(K)}$$

Please pay attention that the notation "$i':i, j':j$" implies that indices $i'$ and $j'$ are iterated given indices $i$ and $j$ where every product $\hat{x}_{i'j'}^{(K-1)} = \left(\sum_{u',v'} w_{u'v'}^{(K)} x_{i'+u',j'+v'}^{(K-1)}\right)$ is satisfied so that there is only one pair of $u$=$u'$ and $v$=$v'$ such that $i = i'+u'$ and $j = j'+v'$. The elemental parametric error is generalized at any layer $\boldsymbol{X}_k$ where $1 \le k \le K$ as follows:

$$\nabla_{x_{ij}^{(k)}} l\left(x_{ij}^{(k)}\right) \cong \begin{cases} l'\left(x_{ij}^{(K)}\right) \text{ if } k = K \\ \sum_{i':i,j':j} w_{u'v'}^{(k+1)} \nabla_{\theta^{(k+1)}} l\left(x_{i'j'}^{(k+1)}\right) \text{ if } k < K \end{cases}$$

In general, necessary gradients for training CNN are summarized from $k$=$K$ down $k$=1 as follows:

$$\begin{aligned} \nabla_{w_{uv}^{(k)}} l &= \sum_{i:u,j:v} x_{i+u,j+v}^{(k-1)} \nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right) \\ \nabla_{\theta^{(k)}} l &= \sum_{i,j} \nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right) \\ \nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right) &= f'\left(\hat{x}_{ij}^{(k-1)}\right) \nabla_{x_{ij}^{(k)}} l\left(x_{ij}^{(k)}\right) \\ \nabla_{x_{ij}^{(k)}} l\left(x_{ij}^{(k)}\right) &\cong \begin{cases} l'\left(x_{ij}^{(K)}\right) \text{ if } k = K \\ \sum_{i':i,j':j} w_{u'v'}^{(k+1)} \nabla_{\theta^{(k+1)}} l\left(x_{i'j'}^{(k+1)}\right) \text{ if } k < K \end{cases} \end{aligned} \tag{1.2}$$

Such that:

$$\begin{aligned} w_{uv}^{(k)} &= w_{uv}^{(k)} + \gamma \nabla_{w_{uv}^{(k)}} l \\ \theta^{(k)} &= \theta^{(k)} + \gamma \nabla_{\theta^{(k)}} l \end{aligned}$$

Where $\gamma$ ($0 < \gamma \le 1$) is learning rate.

There are special filters called pooling filters without parameters, in which *max-pooling filter* is a popular pooling filter. Although max-pooling filter is simple to find out maximum value in the bound of $u$x$v$ window, it is effective to extract image feature as a set of maximum-value pixel.

$$x_{ij}^{(k)} = \max_{u,v}\left(x_{i+u,j+v}^{(k-1)}\right)$$

It is only necessary to calculate the elemental parametric error for max-pooling filter so that backpropagation algorithm only sends backwards such error, as follows:

$$\nabla_{x_{ij}^{(k)}} l\left(x_{ij}^{(k)}\right) \cong \begin{cases} l'\left(x_{ij}^{(K)}\right) \text{ if } k = K \\ \sum_{i',j'} \begin{cases} \nabla_{x_{i'j'}^{(k+1)}} l\left(x_{i'j'}^{(k+1)}\right) \text{ if } x_{ij}^{(k)} \text{ is } x_{i'j'}^{(k+1)} \\ 0 \text{ otherwise} \end{cases} \text{ if } k < K \end{cases}$$

Please pay attention that the indices $i'$ and $j'$ are iterated so that the max-pooling operator $x_{i'j'}^{(k+1)} = \max_{u,v}\left(x_{i'+u,j'+v}^{(k)}\right)$ is satisfied.

CNN is extended with multi-channel layers in which each layer $\boldsymbol{X}_k$ has $C$ channels whereas filter weight matrix has $C$ channels too but parametric filter bias is kept intact so that the propagation is extended a little bit as follows:

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$x_{ijc}^{(k)} = f\left(\hat{x}_{ijc}^{(k-1)} = \left(\sum_{c=1}^{C}\sum_{u,v} w_{uvc}^{(k)} x_{i+u,j+v,c}^{(k-1)}\right) + \theta^{(k)}\right)$$

Where pixel values $x_{ijc}^{(k)}$ and weights $w_{uvc}^{(k)}$ are indexed more by channel $c$. The likelihood function $l\left(x_{ij}^{(K)}\right)$ is extended as follows:

$$l\left(x_{ij}^{(K)}\right) = \sum_{c=1}^{C} l\left(x_{ijc}^{(K)}\right)$$

Where,

$$l\left(x_{ijc}^{(K)}\right) = -\varepsilon\left(x_{ijc}^{(K)}\right)$$

Process of training CNN is not changed but its estimation equations are slightly changed, as follows:

$$\nabla_{w_{uvc}^{(k)}} l = \sum_{i:u,j:v} x_{i+u,j+v,c}^{(k-1)} \nabla_{\theta^{(k)}} l\left(x_{ijc}^{(k)}\right)$$

$$\nabla_{\theta^{(k)}} l = \sum_{i,j,c} \nabla_{\theta^{(k)}} l\left(x_{ijc}^{(k)}\right)$$

$$\nabla_{\theta^{(k)}} l\left(x_{ijc}^{(k)}\right) = f'\left(\hat{x}_{ijc}^{(k-1)}\right) \nabla_{x_{ijc}^{(k)}} l\left(x_{ijc}^{(k)}\right)$$

$$\nabla_{x_{ijc}^{(k)}} l\left(x_{ijc}^{(k)}\right) \cong \begin{cases} l'\left(x_{ijc}^{(K)}\right) \text{ if } k = K \\ \sum_{u:i,v:j} w_{uvc}^{(k+1)} \nabla_{\theta^{(k+1)}} l\left(x_{ijc}^{(k+1)}\right) \text{ if } k < K \end{cases}$$

The *core last convolutional error* $\nabla_{x_{ij}^{(K)}} l\left(x_{ij}^{(K)}\right) = l'\left(x_{ij}^{(K)}\right)$ at the output layer $\boldsymbol{X}_K$ is propagated backward from dense network. Let $\nabla_{\boldsymbol{X}_K} l(\boldsymbol{X}_K)$ denote the core last convolutional error (matrix) including all $\nabla_{x_{ij}^{(K)}} l\left(x_{ij}^{(K)}\right)$ at the last convolutional layer $\boldsymbol{X}_K$. Suppose the $N$-layer dense network has $N$ layers such as $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$,…, $\boldsymbol{y}_N$, the last convolutional error $\nabla_{\boldsymbol{X}_K} l(\boldsymbol{X}_K)$ is translated as the bias of the dense network at its first layer $\boldsymbol{y}_0$, propagated from the *core last bias* $b_{\boldsymbol{y}_N}(\boldsymbol{y}_N)$ as follows:

$$\nabla_{\boldsymbol{X}_K} l(\text{vec}(\boldsymbol{X}_k)) = \left(\prod_{n=1}^{N-1} \left(g'_{\boldsymbol{y}_n}(\hat{\boldsymbol{y}}_{n-1})\right)^T W_{n+1}^T\right) g'_{\boldsymbol{y}_N}(\hat{\boldsymbol{y}}_{N-1}) b_{\boldsymbol{y}_N}(\boldsymbol{y}_N)$$

Where vec(.) represents vectorization technique which makes a matrix flatten as column vector, $g(.)$ is activation function of dense network, and $W_n$ is parametric weight matrix of dense layer $n$. If dense network is trained based on minimizing squared norm function, the core last bias $b_{\boldsymbol{y}_N}(\boldsymbol{y}_N)$ is easily determined as follows:

$$b_{\boldsymbol{y}_N}(\boldsymbol{y}_N) = \boldsymbol{y}'_N - \boldsymbol{y}_N$$

Where $\boldsymbol{y}'_N$ is the real output of dense network from environment. Please pay attention that when $\nabla_{x_{ij}^{(K)}} l\left(x_{ij}^{(K)}\right) = l'\left(x_{ij}^{(K)}\right)$ is core last convolutional error, the gradient $\nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right)$ in general is called convolutional error, backward error, or backward bias at the $k^{\text{th}}$ convolutional layer, which is the cornerstone of association of backpropagation algorithm and stochastic gradient descent (SGD) algorithm because the gradient $\nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right)$ is also the first-order derivative of convolutional likelihood $l(x_{ij}^{(K)})$ with respect to $\hat{x}_{ij}^{(k-1)}$ at layer $\boldsymbol{X}_k$ where all elements $\hat{x}_{ij}^{(k-1)}$ represent the layer $\boldsymbol{X}_k$ as variable.

$$\nabla_{\theta^{(k)}} l\left(x_{ij}^{(k)}\right) = \frac{dl\left(x_{ij}^{(K)}\right)}{d\theta^{(k)}} = \frac{dl\left(x_{ij}^{(K)}\right)}{d\hat{x}_{ij}^{(k-1)}}$$

Although it is no doubt about the preeminence of convolutional neural network (CNN) in training image data, it is not meaningful enough to expend CNN into other high-dimension data

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



different from image because CNN filter is appropriate to extract image feature. Moreover, data is changed or distorted after it goes through CNN so that original data with complete or full meaning is not kept intact within CNN methodology. Therefore, this research focuses on evaluating the hypothesis "whether matrix neural network is the alternative of convolutional neural network" and so, image classification by CNN and matrix neural network called deep image classification is applied into testing the hypothesis. The next section mentions some other works related to deep image classification and the later section describes the main methodology of this research which focuses on matrix neural network.

## 2. Related Works

The important application of convolutional neural network (CNN) is image classification although there are some other applications of CNN like image retrieval, object detection, feature extraction, image recognition, etc. and so, related works described in this section are focused on association of CNN and image classification. Recall that standard and traditional CNN consists of two connected network such as convolutional network and dense network where convolutional network is the consequential set of convolutional layers filtered by small enough filter kernel and dense network is feedforward network (FFN) as normal artificial neural network (ANN) with note that dense network is also called fully connected network (FCN). A specific convolutional layer is *pooling layer* whose kernel operator is selection operation for selecting specific value in the range of the kernel instead of taking multiplication as usual. If the specific value is selected as maximum value with the range of kernel, pooling layer is called max-pooling layer. Pooling layer and its *pooling kernel* have an important role in CNN because pooling kernel aims to extract dominant pixels in image. As a convention, convolutional layer is called *c-layer* with filtering kernel (filter) and pooling layer with pooling kernel is called *p-layer* whereas traditional layer in dense network with parametric weights is simply called *w-layer*. CNN-based classification methods are grouped in three approaches (Chen, et al., 2021): 1) traditional CNN-based approach called *classic approach* conforms strictly definition of CNN including two networks such as convolutional network (convolutional layers + pooling layers) and dense network, 2) *Inception-based approach* follows the concept of Inception, 3) *residual approach* follows the concept of residual network aiming to solve overfitting problem due to another problem of gradient vanishing, 4) *attention-based approach* applies attention mechanism into CNN classifier, 5) *transformer-based approach* which is the most recent approach until now (2025) applies transformer developed in large language model (LLM) into CNN-based classifier with note that transformer is an improvement of both attention mechanism and pre-training model where pre-training model is originally developed for domain of computer vision.

In 1998, LeCun et al. (LeCun, Bottou, Bengio, & Haffner, 1998) developed LeNet model well-known as pioneering research which established fundamental principles of classic approach for CNN classifier. LeNet model also called LeNet-5 network consists of two successive network such as convolutional network and dense network where convolutional network has 4 c-layers and dense network has 3 w-layers, which means that LeNet-5 has 7 layers. Especially, each c-layer among 4 convolutional layers is a set of sub-convolutional layers (called *sub-c-layers*) where every sub-c-layer is resulted from (owns) one filter. There are 2 c-layers and 2 p-layers among such 4 convolutional layers, following the ordering (Chen, et al., 2021, p. 10): the first 28x28x6 c-layer denoted *conv*1 resulted from 6 5x5 filtering kernels (stride 1, zero-padding) on every MNIST 32x32x1 image → the second 14x14x6 p-layer denoted *maxpool*1 resulted from 2x2 pooling kernel (stride 2, zero-padding) on the first c-layer → the third 10x10x16 c-layer denoted *conv*2 resulted from 16 5x5x6 filtering kernels (stride 1, zero-padding) on the second p-layer → the fourth 5x5x16 p-layer denoted *maxpool*2 resulted from 2x2 pooling kernel (stride 2, zero-padding) on the third c-layer.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



**Table 2.1.** Convolutional network of LeNet-5.

| Layer | Input dimension | Filter dimension | Filter count | Output dimension |
|---|---|---|---|---|
| *conv*1 | 32 x 32 | 5 x 5 | 6 | 28 x 28 x 6 |
| *maxpool*1 | 28 x 28 x 6 | 2 x 2 | | 14 x 14 x 6 |
| *conv*2 | 14 x 14 x 6 | 5 x 5 x 6 | 16 | 10 x 10 x 16 |
| *maxpool*2 | 10 x 10 x 16 | 2 x 2 | | 5 x 5 x 16 |
| *conv*3 | 5 x 5 x 16 | 5 x 5 x 16 | 120 | 1 x 1 x 120 |

The fourth *maxpool*2 p-layer whose dimension is 5x5x16 is filtered again by 120 5x5x16 filtering kernels (stride 1, zero-padding) so as to be flatten as a 120-element vector denoted *conv*3 which is the first w-layer of dense network whereas the second w-layer is 84-element vector and the third w-layer is 10-element vector representing 10 classes. Within convolutional network, there are 6 5x5 filtering kernels plus 16 5x5 filtering kernels plus 120 5x5x16 filtering kernel, which issues requirement of 50,550 = 6*5*5 + 16*5*5*6 + 120*5*5*16 parameters. Among 3 w-layers of dense network, there are 120 weights multiplied with 84 weights plus 84 weights multiplied with 10 weights, which issues 11,014 = 120*84 + 84 (bias) + 84*10 + 10 (bias) parameters. In general, LeNet-5 needs 61,564 = 50,550 + 11,014 parameters. Pooling layers in LeNet-5 are max-pooling layers and activation function is sigmoid function. Moreover, LeNet-5 is the first CNN classifier that applied backpropagation algorithm into training convolutional layers when previous CNN applications may fix filters of c-layers like blurring filter, edge-detection filter, etc.

More than ten years later, in 2012, Krizhevsky et al. (Krizhevsky, Sutskever, & Hinton, 2012) proposed AlexNet model which gave better result in CNN classification consists of two successive networks such as convolutional network and dense network where convolutional network has 8 c-layers and dense network has 3 w-layers, which means that AlexNet has 11 layers. Because the architecture of AlexNet does not go too beyond LeNet-5, it still belongs to classic approach in CNN classification. However, there are many significant improvements of AlexNet in comparison with LeNet-5 (Chen, et al., 2021, p. 10) such as ReLU activation function, dropout technique to alleviate overfitting problem, data augmentation to alleviate overfitting problem, and overlapping pooling to also alleviate overfitting problem. The AlexNet architecture is more complicated than LeNet-5 when there are 5 c-layers and 3 p-layers among 8 convolutional layers of convolutional network, following the ordering (Chen, et al., 2021, p. 11): the first 55x55x96 c-layer denoted *conv*1 resulted from 96 11x11x3 filtering kernel (stride 4, zero-padding) on every 227x227x3 image → the second 27x27x96 p-layer denoted *maxpool*1 resulted from 3x3 pooling kernel (stride 2, zero-padding) on the first c-layer → the third 27x27x256 c-layer denoted *conv*2 resulted from 256 5x5x96 filtering kernels (stride 1, zero-padding) on the second p-layer → the fourth 13x13x256 p-layer denoted *maxpool*2 resulted from 3x3 pooling kernel (stride 2, zero-padding) on the third c-layer → the fifth 13x13x384 c-layer denoted *conv*3 resulted from 384 3x3x256 filtering kernels (stride 1, padding 1) on the fourth p-layer → the sixth 13x13x384 c-layer denoted *conv*4 resulted from 384 3x3x384 filtering kernel (stride 1, padding 1) on the fifth c-layer → the seventh 13x13x256 c-layer denoted *conv*5 resulted from 256 3x3x384 filtering kernel (stride 1, padding 1) on the sixth c-layer → the eighth 6x6x256 p-layer denoted *maxpool*3 resulted from 3x3 pooling kernel (stride 2, zero-padding) on the seventh c-layer.

**Table 2.2.** Convolutional network of AlexNet.

| Layer | Input dimension | Filter dimension | Filter count | Output dimension |
|---|---|---|---|---|
| *conv*1 | 227 x 227 x 3 | 11 x 11 x 3 (stride 4) | 96 | 55 x 55 x 96 |
| *maxpool*1 | 55 x 55 x 96 | 3 x 3 (stride 2) | | 27 x 27 x 96 |

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



| | | | | |
|---|---|---|---|---|
| *conv*2 | 27x27x96 | 5 x 5 x 96 (stride 1) | 256 | 27 x 27 x 256 |
| *maxpool*2 | 27 x 27 x 256 | 3 x 3 (stride 2) | | 13 x 13 x 256 |
| *conv*3 | 13 x 13 x 256 | 3 x 3 x 256 (stride 1) | 384 | 13 x 13 x 384 |
| *conv*4 | 13 x 13 x 384 | 3 x 3 x 384 (stride 1) | 384 | 13 x 13 x 384 |
| *conv*5 | 13 x 13 x 384 | 3 x 3 x 384 (stride 1) | 256 | 13 x 13 x 256 |
| *maxpool*3 | 13 x 13 x 256 | 3 x 3 (stride 2) | | 6 x 6 x 256 |

The eighth p-layer *maxpool*3 whose dimension is 6x6x256 (=1x1x9216) is flattened into 9216-element vector as input of dense network so that the first w-layer of dense network is 4096-element vector whereas the second w-layer is also 4096-element vector and the third w-layer is 1000-element vector corresponding to 1000 classes. Note, pooling kernel in AlexNet is max-pooling kernel. Within convolutional network, there are 96 11x11x3 filtering kernel plus 256 5x5x96 filtering kernel plus 384 3x3x256 filtering kernels plus 384 3x3x384 filtering kernels plus 3x3x256 filtering kernel, which issues requirement of 3,745,824 = 96*11*11*3 + 256*5*5*96 + 384*3*3*256 + 384*3*3*384 + 256*3*3*384 parameters. Within dense network, there are 9216 weights multiplied with 4096 weights plus 4096 weights multiplied with 4096 weights plus 4096 weights multiplied with 1000 weights, which issues 58,631,144 = 9216*4096 + 4096 (bias) + 4096*4096 + 4096 (bias) + 4096*1000 + 1000 (bias) parameters. In general, AlexNet needs at least 62,376,968 = 3,745,824 + 58,631,144 parameters.

The VGG model developed by Simonyan and Zisserman (Simonyan & Zisserman, 2015), which belongs to classic approach, aims to define a well-structured model by modular architecture. Exactly, VGG architecture still includes convolutional network and dense network but its convolutional network is modularized into five blocks and each *bock* has some c-layers and one p-layer (Gemini 2025). Output dimension is decreased doubly block-after-block, which is called *dimensionality reduction*, whereas the number of filters (filtering kernels) is increased doubly block-after-block, which is called *filter progression* (Gemini 2025). Moreover, the number of c-layers in each block is increased sequentially with note that size of filtering kernels is fixed by dimension 3x3 and size of pooling kernels is fixed by dimension 2x2 whereas stride of filtering kernels is fixed by 1 and stride of pooling kernels is fixed by 1 (Gemini 2025). For instance, the first block has two c-layers and one p-layer where the first 224x224x64 c-layer denoted *conv*1_1 is resulted from 64 3x3x3 filtering kernels (stride 1, zero-padding) on every 224x224x3 image and the second 224x224x64 c-layer denoted *conv*1_2 is resulted from 64 3x3x64 filtering kernels (stride 1, zero-padding) on the first c-layer *conv*1_1 whereas the third 112x112x64 p-layer denoted *maxpool*1 is resulted from 2x2 max-pooling kernel (stride 2) on the second c-layer *conv*1_2. Therefore, output dimension of the first block is 112x112x64 and the first block requires at least 38,592 = 64*3*3*3 (*conv*1_1) + 64*3*3*64 (*conv*1_2). Consequently, the second block which connects directly to the first block has two c-layers and one p-layer where the first 112x112x128 c-layer denoted *conv*2_1 is resulted from 128 3x3x64 filtering kernels (stride 1, zero-padding) on the 112x112x64 output of the first block and the second 112x112x128 c-layer denoted *conv*2_2 is resulted from 128 3x3x128 filtering kernels (stride 1, zero-padding) on the first c-layer *conv*2_1 whereas the third 56x56x128 p-layer denoted *maxpool*2 is resulted from 2x2 max-pooling kernel (stride 2) on the second c-layer *conv*2_2. Therefore, output dimension of the second block is 56x56x128 and the second block requires at least 221,184 = 128*3*3*64 (*conv*2_1) + 128*3*3*128 (*conv*2_2). Consequently, the third block which connects directly to the second block has three c-layers and one p-layer where the first 56x56x256 c-layer denoted *conv*3_1 is resulted from 256 3x3x128 filtering kernels (stride 1, zero-padding) on the 56x56x128 output of the second block, the second 56x56x256 c-layer denoted *conv*3_2 is resulted from 256 3x3x256 filtering kernels (stride 1, zero-padding) on the first c-layer *conv*3_1, and the third 56x56x256 c-layer denoted *conv*3_3 is resulted

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



from 256 3x3x256 filtering kernels (stride 1, zero-padding) on the second c-layer *conv*3_2 whereas the fourth 28x28x256 p-layer denoted *maxpool*3 is resulted from 2x2 max-pooling kernel (stride 2) on the third c-layer *conv*2_3. Therefore, output dimension of the third block is 28x28x256 and the third block requires at least 1,474,560 = 256*3*3*128 (*conv*3_1) + 256*3*3*256 (*conv*3_2) + 256*3*3*256 (*conv*3_3). Consequently, the fourth block which connects directly to the third block has three c-layers and one p-layer where the first 28x28x512 c-layer denoted *conv*4_1 is resulted from 512 3x3x256 filtering kernels (stride 1, zero-padding) on the 28x28x256 output of the third block, the second 28x28x512 c-layer denoted *conv*4_2 is resulted from 512 3x3x512 filtering kernels (stride 1, zero-padding) on the first c-layer *conv*4_1, and the third 28x28x512 c-layer denoted *conv*4_3 is resulted from 512 3x3x512 filtering kernels (stride 1, zero-padding) on the second c-layer *conv*4_2 whereas the fourth 14x14x512 p-layer denoted *maxpool*4 is resulted from 2x2 max-pooling kernel (stride 2) on the third c-layer *conv*4_3. Therefore, output dimension of the fourth block is 14x14x512 and the fourth block requires at least 5,898,240 = 512*3*3*256 (*conv*4_1) + 512*3*3*512 (*conv*4_2) + 512*3*3*512 (*conv*4_3). Consequently, the fifth block which connects directly to the fourth block has three c-layers and one p-layer where the first 14x14x512 c-layer denoted *conv*5_1 is resulted from 512 3x3x512 filtering kernels (stride 1, zero-padding) on the 14x14x512 output of the fourth block, the second 14x14x512 c-layer denoted *conv*5_2 is resulted from 512 3x3x512 filtering kernels (stride 1, zero-padding) on the first c-layer *conv*5_1, and the third 14x14x512 c-layer denoted *conv*5_3 is resulted from 512 3x3x512 filtering kernels (stride 1, zero-padding) on the second c-layer *conv*5_2 whereas the fourth 7x7x512 p-layer denoted *maxpool*5 is resulted from 2x2 max-pooling kernel (stride 2) on the third c-layer *conv*5_3. Therefore, output dimension of the fifth block is 7x7x512 and the fifth block requires at least 7,077,888 = 512*3*3*512 (*conv*5_1) + 512*3*3*512 (*conv*5_2) + 512*3*3*512 (*conv*4_3).

**Table 2.3.** Five blocks (convolutional network) of VGG.

| Block | Layer | Input dimension | Filter dimension | Filter count | Output dimension |
|---|---|---|---|---|---|
| 1 | *conv*1_1 | 224 x 224 x 3 | 3 x 3 x 3 | 64 | 224 x 224 x 64 |
| | *conv*1_2 | 224 x 224 x 64 | 3 x 3 x 64 | 64 | 224 x 224 x 64 |
| | *maxpool*1 | 224 x 224 x 64 | 2 x 2 | | 112 x 112 x 64 |
| 2 | *conv*2_1 | 112 x 112 x 64 | 3 x 3 x 64 | 128 | 112 x 112 x 128 |
| | *conv*2_2 | 112 x 112 x 128 | 3 x 3 x 128 | 128 | 112 x 112 x 128 |
| | *maxpool*2 | 112 x 112 x 128 | 2 x 2 | | 56 x 56 x 128 |
| 3 | *conv*3_1 | 56 x 56 x 128 | 3 x 3 x 128 | 256 | 56 x 56 x 256 |
| | *conv*3_2 | 56 x 56 x 256 | 3 x 3 x 256 | 256 | 56 x 56 x 256 |
| | *conv*3_3 | 56 x 56 x 256 | 3 x 3 x 256 | 256 | 56 x 56 x 256 |
| | *maxpool*3 | 56 x 56 x 256 | 2 x 2 | | 28 x 28 x 256 |
| 4 | *conv*4_1 | 28 x 28 x 256 | 3 x 3 x 256 | 512 | 28 x 28 x 512 |
| | *conv*4_2 | 28 x 28 x 512 | 3 x 3 x 512 | 512 | 28 x 28 x 512 |
| | *conv*4_3 | 28 x 28 x 512 | 3 x 3 x 512 | 512 | 28 x 28 x 512 |
| | *maxpool*4 | 28 x 28 x 512 | 2 x 2 | | 14 x 14 x 512 |
| 5 | *conv*5_1 | 14 x 14 x 512 | 3 x 3 x 512 | 512 | 14 x 14 x 512 |
| | *conv*5_2 | 14 x 14 x 512 | 3 x 3 x 512 | 512 | 14 x 14 x 512 |
| | *conv*5_3 | 14 x 14 x 512 | 3 x 3 x 512 | 512 | 14 x 14 x 512 |
| | *maxpool*5 | 14 x 14 x 512 | 2 x 2 | | 7 x 7 x 512 |

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Consequently, the output of the fifth final block whose dimension is 7x7x512 is flattened into 25,088 = 7*7*512 features as 25,088-element vector as input of dense network consisting of three w-layers. Therefore, the first w-layer, the second w-layer, and the third w-layer of dense network are 4096-element vector, 4096-element vector, and 1000-element vector (corresponding to 1000 image classes), respectively. Five blocks of convolutional network, there are 14,710,464 = 38,592 + 221,184 + 1,474,560 + 5,898,240 + 7,077,888 parameters. Among three w-layers of dense network, there are 25,088 weights multiplied with 4096 weights plus 4096 weights multiplied with 4096 weights plus 4096 weights multiplied with 1000 weights, which issues 123,642,856 = 25,088*4096 + 4096 (bias) + 4096*4096 + 4096 (bias) + 4096*1000 + 1000 (bias) parameters. In general, VGG needs at least 138,353,320 = 14,710,464 + 123,642,856 parameters. Although the number of parameters of VGG is much larger than LeNet-5 and AlexNet, VGG model can be easily customized because of its flexible architecture with modular blocks. The VGG model described here is also called VGG-16 because its c-layers are grouped in every block so that the total number of convolutional layers is 13 (*conv*1_1, *conv*1_2, *conv*2_1, *conv*2_2, *conv*3_1, *conv*3_2, *conv*3_3, *conv*4_1, *conv*4_2, *conv*4_3, *conv*5_1, *conv*5_2, *conv*5_3) plus 3 w-layers in order to issue 16 weighted layers (Gemini 2025). Another version of VGG model is VGG-19 including 16 convolutional layers plus 3 w-layers in order to issue 19 weighted layers (Gemini 2025).

Network-in-network (NiN) model proposed by Lin et al. (Lin, Chen, & Yan, 2014) which is the last generation one belonging to classic approach is an significantly methodology in CNN based classification because NiN decreases significantly complexity of linear filter operator in multi-channels although the filter operator decreases significantly complexity of weight multiplication in feed-forward network (FFN). Particularly, NiN filtering kernel is 1x1 window including only one weight $w$ so that the 1x1 NiN filtering operator is defined as follows:

$$y_{i,j} = \text{ReLU}\left(wx_{i,j} + \theta\right)$$

For multi-channel input data, NiN filtering kernel is generalized as follows:

$$y_{i,j,k} = \text{ReLU}\left(w_k x_{i,j,k} + \theta_k\right)$$

Given any two successive convolutional layers $\boldsymbol{X}^{(s)}$ and $\boldsymbol{X}^{(s+1)}$, there is a sub-network between the two c-layers, called multilayer perceptron convolutional network (*mlpconv*) including some sub-layers called *micro-layer* or micro-convolutional layer so that mlpconv is also called micro-network. Consequently, suppose a mlpconv has $n$ micro-layers $\boldsymbol{Y}^{(1)}$, $\boldsymbol{Y}^{(2)}$,…, $\boldsymbol{Y}^{(n)}$ whose input and output are the c-layer $\boldsymbol{X}^{(s)}$ and the c-layer $\boldsymbol{X}^{(s+1)}$, respectively, these micro-layers are determined by NiN filtering operator as follows (Lin, Chen, & Yan, 2014, pp. 3-4), (Gemini 2025):

$$\begin{aligned}
y_{i,j,1}^{(1)} &= \text{ReLU}\left(u_1^{(1)} x_{i,j,1}^{(s)} + \alpha_1\right) \\
y_{i,j,2}^{(1)} &= \text{ReLU}\left(u_2^{(1)} x_{i,j,2}^{(s)} + \alpha_2\right) \\
&\vdots \\
y_{i,j,k}^{(1)} &= \text{ReLU}\left(u_k^{(1)} x_{i,j,k}^{(s)} + \alpha_k\right) \\
y_{i,j}^{(1)} &= \max\left(y_{i,j,1}^{(1)}, y_{i,j,2}^{(1)}, \ldots, y_{i,j,k}^{(1)}\right) \\
y_{i,j}^{(2)} &= \text{ReLU}\left(v^{(2)} y_{i,j}^{(1)} + \beta^{(2)}\right) \\
&\vdots \\
y_{i,j}^{(n)} &= \text{ReLU}\left(v^{(n)} y_{i,j}^{(n)} + \beta^{(n)}\right) \\
x_{i,j,1}^{(s+1)} &= \begin{cases} y_{i,j}^{(n)} \\ \max\left(y_{i,j}^{(1)}, y_{i,j}^{(2)}, \ldots, y_{i,j}^{(n)}\right) \end{cases}
\end{aligned}$$

Where $x_{i,j,k}^{(s)}$ denotes an element of the input c-layer $\boldsymbol{X}^{(s)}$ at its $k^{\text{th}}$ channel and $x_{i,j,1}^{(s+1)}$ denotes an element of the output c-layer $\boldsymbol{X}^{(s+1)}$ at its first channel whereas $y_{i,j}^{(r)}$ denotes an element of the $r^{\text{th}}$ micro-layer $\boldsymbol{Y}^{(r)}$. Elements of the output c-layer $\boldsymbol{X}^{(s+1)}$ at other channel are calculated by the same way with the first-channel elements $x_{i,j,1}^{(s+1)}$. Note, $u$ and $v$ are parametric weights whereas $\alpha$ and $\beta$ are parametric biases. Obviously, pooling kernel in NiN is max-pooling kernel. The technique that NiN results out the maximum value after a series of filtering operators:

$$y_{i,j}^{(1)} = \max\left(y_{i,j,1}^{(1)}, y_{i,j,2}^{(1)}, \ldots, y_{i,j,k}^{(1)}\right)$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$x_{i,j,1}^{(s+1)} = \max\left(y_{i,j}^{(1)}, y_{i,j}^{(2)}, \dots, y_{i,j}^{(n)}\right)$$

Which is called *cross max-pooling*. Especially, NiN does not apply dense network into classifying feature from mlpconv, instead, NiN makes sequentially filtering operators and cross max-pooling so that the outputs of last mlpconv (s) are summed together to produce the aggregated output that is actually the image classes and then, soft-max function and cross-entropy loss function are optimized so as to perform classification task on the aggregated output (Chen, et al., 2021, p. 12). This technique is called global average pooling (GAP).

GoogleLeNet also called Inception V1 developed by Szegedy et al. (Szegedy, et al., 2015) improved CNN based classifier through a far distance by proposing the concept of Inception model where Inception inherits from both the modular mechanism of VGG and the micro-filtering operator of NiN so that every Inception module has some convolutional layers which include: 1) normal filtering kernel with matrix kernel, 2) 1x1 micro-filtering kernel with one weight, 3) max-pooling kernel, and 4) global average pool (GAP). Especially, the most important aspect of Inception is that these convolutional layers are operated in parallel so that the final layer called filter concatenation layer will concatenate outputs of such convolutional layers in depth according to predefined number of channels. Finally, such concatenated output can be fed into dense network for classification, however, GAP which produces aggregated output can be the alternative of dense network. Therefore, Inception model establishes basic principles of Inception-based approach on CNN classification. From practical improvements, there is an important recognition that any CNN-based framework need the sparsity in its structure in order to converge and group better classes together, thus, Inception (GoogleLeNet) and NiN aims to decrease such sparsity by 1x1 micro-filtering and GAP (Chen, et al., 2021, p. 14). Moreover, Inception improve more the structural sparsity by parallel-processing convolutional layers, besides, GAP and cross max-pooling decrease data dimension and the number of parameters. Indeed, Inception requires less parameters than classic approaches like AlexNet and VGG but reaches better performance. Particularly, an Inception module has two layers and four branches, in which, the two layers are the structural viewpoint in depth and the four branches are the structural viewpoint in width. For instance, the first branch includes 1x1 micro-filtering kernel (belonging to the first layer) followed by 5x5 filtering kernel (belonging to the second layer). The second branch includes 1x1 micro-filtering kernel (belonging to the first layer) followed by 3x3 filtering kernel (belonging to the second layer). The third branch includes 3x3 max-pooling kernel (belonging to the first layer) followed by 1x1 micro-filtering kernel (belonging to the second layer). The fourth branch includes 1x1 micro-filtering kernel (belonging to the first layer). All of these kernels have stride 1, which is an aspect of Inception in order to keep the size (width and height) of data intact for making a so-called *depth-wise concatenation* to concatenate outputs of branches into one layer. For instances, let $B_1$, $B_2$, $B_3$, and $B_4$ whose dimensions are $w$ x $h$ x $d_1$, $w$ x $h$ x $d_2$, $w$ x $h$ x $d_3$, and $w$ x $h$ x $d_4$ be outputs of the fourth branches where $d_1$, $d_2$, $d_3$, and $d_4$ are depths (also the numbers of kernels) of the fourth branches, then the depth-wise concatenation also called filter concatenation will merge $B_1$, $B_2$, $B_3$, and $B_4$ into one output B whose dimension is $w$ x $h$ x ($d_1$+$d_2$+$d_3$+$d_4$). Based on definition of Inception module, the first version of Inception framework called Inception V1 known as GoogleLeNet has three parts (Gemini 2025) such as stem part (part A), stacked Inception modules part (part B), and classification head (part C). Part A has three layers where the first layer is convolutional layer with 7x7 filtering kernel (stride 2), the second layer for dimensionality reduction is micro-convolutional layer with 1x1 micro-filtering kernel, and the third layer is convolutional layer with 3x3 filtering kernel. Part B consists of 9 Inception modules aforementioned and so, part B has 18 layers because every Inception module has 2 layers. Part C is actually the dense network for classification with only one layer having 1000 classes (neurons), thus, GoogleLeNet has 22 layers.

Deep neural network (DNN) with many hidden layers will obtain better evaluation with high convergence but there is an effect-side that too many hidden layers cause the problem of overfitting where DNN gets involved in very few of output neurons, which decrease significantly accuracy in learning by neural network. The overfitting problem which occurs frequently in machine learning is now known as degradation problem in neural network. The main cause of overfitting problem is the event that gradient of activation function become too small nearly zero after a chain of gradient

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



calculations by stochastic gradient descent (SGD) algorithm and backpropagation algorithm, which is known as *vanishing gradient* problem. There are two popular approaches to solve vanishing gradient problem such as dropout (dilution) technique and residual connection (RC) technique. Kaiming He et al. (He, Zhang, Ren, & Sun, 2016) proposed RC technique and applied RC into training convolutional filtering kernel (weights) so as to produce a so-called ResNet for CNN-based classification. The significant effectiveness of ResNet makes itself become an CNN-based image approach known as residual approach. In general, RC is applied into any artificial neural network (ANN) besides CNN. For instance, given deep neural network denoted by function $F(\boldsymbol{x} \mid \Theta)$ where $\boldsymbol{x}$ is input layer and $\Theta$ denotes parametric weights, RC technique adds the input vector (input layer) $\boldsymbol{x}$ into the $F(\boldsymbol{x} \mid \Theta)$ such that:

$$\boldsymbol{y} = \boldsymbol{x} + F(\boldsymbol{x}|\Theta)$$

Given the real output $\boldsymbol{y}'$ from environment, let $l(\boldsymbol{y})$ be the likelihood for training the deep neural network $F(\boldsymbol{x})$:

$$l(\boldsymbol{y}) = -\frac{1}{2}\|\boldsymbol{y}' - \boldsymbol{y}\|^2$$

The association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm is based on calculating the gradient of $l(\boldsymbol{y})$ with respect to $\boldsymbol{y}$.

$$\nabla l(\boldsymbol{y}) = (\boldsymbol{y}' - \boldsymbol{y})\frac{d\boldsymbol{y}}{d\boldsymbol{x}} = \big(\boldsymbol{y}' - \boldsymbol{x} - F(\boldsymbol{x}|\Theta)\big)\big(\boldsymbol{1} + F'(\boldsymbol{x}|\Theta)\big)$$

Where $F'(\boldsymbol{x} \mid \Theta)$ is the gradient / first-order derivative of $F(\boldsymbol{x} \mid \Theta)$ with respect to $\boldsymbol{x}$.

$$F'(\boldsymbol{x}|\Theta) = \frac{dF(\boldsymbol{x}|\Theta)}{d\boldsymbol{x}}$$

The expression $(\boldsymbol{1} + F'(\boldsymbol{x} \mid \Theta))$ alleviates the vanishing gradient problem because the gradient $\nabla l(\boldsymbol{y})$ is never zero due to the expression $(\boldsymbol{1} + F'(\boldsymbol{x} \mid \Theta))$ although $F'(\boldsymbol{x} \mid \Theta)$ may approach nearly zero. Note, $F'(\boldsymbol{x} \mid \Theta)$ is often non-negative because the derivative of activation function like sigmoid function or rectified linear unit (ReLU) is often non-negative.

Attention is the concept to indicate the self-structure of a data entity, which is the starting point of self-supervised learning that is the middle one between supervised learning and unsupervised learning because the compared object in self-supervised learning is the source entity itself whereas supervised learning requires explicit output as compared object and unsupervised learning does not require compared object. The self-structure which is the output feature of self-supervised learning is often internal relationship or hidden meaning, which is generalized as attention in artificial intelligence and moreover, attention focuses on the focus of deep neural network. The concept "attention" is refined by transformer (Vaswani, et al., 2017) which will be mentioned later where transformer is also applied into CNN-based image classification. Firstly, attention-based approach in image classification began with the Residual Attention Network (RAN) proposed by Wang et al. (Wang, et al., 2017) in which Wang et al. (Wang, et al., 2017) proposed attention as product of source entity and the mask. Given input data x, let $H(\boldsymbol{x})$ denote the output of *attention module* which is the composition (product) of *trunk branch* $T(\boldsymbol{x})$ and *mask branch* $M(\boldsymbol{x})$:

$$H(\boldsymbol{x}) = M(\boldsymbol{x}) * T(\boldsymbol{x})$$

When *residual learning* is applied into learning attention module, the attention output $H(\boldsymbol{x})$ is modified (Wang, et al., 2017) as follows:

$$H(\boldsymbol{x}) = \big(1 + M(\boldsymbol{x})\big) * T(\boldsymbol{x})$$

When trunk branch $T(\boldsymbol{x})$ is normal feed-forward network (FFN), mask branch $M(\boldsymbol{x})$ is structured as encoder-decoder architecture where the encoder is called *bottom-up mask branch* $M_1(\boldsymbol{x})$ mainly applying max-pooling kernel into extracting image feature and the decoder is called *top-down mask branch* $M_2(\boldsymbol{x})$ mainly applying bilinear interpolation into recovering image from such encoded feature. The process of bottom-up mask branch is called *down-sampling* and the process of top-down mask branch is called *up-sampling*. Note, $M_2(\boldsymbol{x})$ connects sequentially with $M_1(\boldsymbol{x})$ to form $M(\boldsymbol{x})$ where input of $M(\boldsymbol{x})$ is actually input of $M_1(\boldsymbol{x})$ so that output of $M_1(\boldsymbol{x})$ becomes input of $M_2(\boldsymbol{x})$ and output of $M_2(\boldsymbol{x})$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



is output of $M(\boldsymbol{x})$. The product of trunk branch and mask branch to produce attention as $H(\boldsymbol{x}) = (1 + M(\boldsymbol{x}))$ is concretized given position (row $i$, column $j$) and channel $c$ as follows (Gemini 2025):

$$H_{i,j,c}(\boldsymbol{x}) = \left(1 + M_{i,j,c}(\boldsymbol{x})\right) T_{i,j,c}(\boldsymbol{x})$$

It is necessary to describe a little bit about bilinear interpolation operator which is the main task of up-sampling of top-down mask branch $M_2(\boldsymbol{x})$. The main task of linear interpolation is to recover the low-resolution feature (output of $M_1(\boldsymbol{x})$) so as to be as near as possible to the high-resolution data representing image from trunk branch $T(\boldsymbol{x})$. Given $w$ and $h$ are width and height of the output of $M_1(\boldsymbol{x})$ which is also input of $M_2(\boldsymbol{x})$ with note the output of $M_1(\boldsymbol{x})$ is result of max-pooling operator on $M_1(\boldsymbol{x})$ and given $W$ and $H$ are width and height of the output of $M_2(\boldsymbol{x})$ which is the focused result of bilinear interpolation so that the width scale $s_w$ and height scale $s_h$ are calculated as follows:

$$s_w = w/W$$
$$s_h = h/H$$

Given target position $(i', j')$ on the output of $M_2(\boldsymbol{x})$ with plane of width $W$ and height $H$, bilinear interpolation estimates the pixel value of the output of $M_2(\boldsymbol{x})$ called target value $\boldsymbol{x}(i', j')$ from the pixel value of the input of $M_2(\boldsymbol{x})$ called source value $\boldsymbol{x}(i, j)$ where the source position $(i, j)$ on the plane of width $w$ and height $h$ is determined as follows:

$$i = i' * s_h$$
$$j = j' * s_w$$

The width fraction $f_w$ and height fraction $f_h$ are determined as follows:

$$f_w = j - \lfloor j \rfloor$$
$$f_h = i - \lfloor i \rfloor$$

Where the notation $\lfloor \rfloor$ denotes the low integer. The four neighbors of source pixel are determined as follows (Gemini 2025):

$$s_{00} = \boldsymbol{x}(\lfloor i \rfloor, \lfloor j \rfloor)$$
$$s_{01} = \boldsymbol{x}(\lfloor i \rfloor, \lfloor j + 1 \rfloor)$$
$$s_{10} = \boldsymbol{x}(\lfloor i + 1 \rfloor, \lfloor j \rfloor)$$
$$s_{11} = \boldsymbol{x}(\lfloor i + 1 \rfloor, \lfloor j + 1 \rfloor)$$

The top interpolation value $v_t$ and the bottom interpolation value $v_b$ are calculated as follows (Gemini 2025):

$$v_t = s_{00}(1 - f_w) + s_{01} f_w$$
$$v_b = s_{10}(1 - f_w) + s_{11} f_w$$

As a result, target value $\boldsymbol{x}(i', j')$ is estimated by bilinear interpolation as follows (Gemini 2025):

$$\boldsymbol{x}(i', j') = v_t(1 - f_h) + v_b f_h = (s_{00}(1 - f_w) + s_{01} f_w)(1 - f_h) + (s_{10}(1 - f_w) + s_{11} f_w) f_h$$

Note, target values $\boldsymbol{x}(i', j')$ move over the target plane of width $W$ and height $H$.

Transformer-based approach whose typical model is ViT developed by Dosovitskiy et al. (Dosovitskiy, et al., 2020) is the recent approach for image classification, which is described shortly here with note that transformer is also developed as self-supervised learning model where attention mechanism is improved more than the mechanism of product of mask branch and trunk branch aforementioned. Note, ViT is abbreviation of *vision transformer* which means the application of transformer into vision computer domain like image processing. Transformer-based approach with ViT is most different from other CNN-based classification methods because attention of transformer can replace convolutional layers. Transformer aforementioned is firstly applied into natural language processing (NLP), especially statistical translation machine (STM) and large language model (LLM) although transfer learning and pre-trained model are also important fundamental of LLM. On the other hand, pre-trained model is also applied perfectly into domain of vision computer. Therefore, it is totally natural to apply transformer into computer vision, especially image classification. The only problem of transformer-based image classification is that the huge dimension of image data prevents transformer from effective computation. There are two solutions of the huge dimension problem

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



(Gemini 2025): 1) image is divided into patches which are fed to transformer and 2) the auxiliary learnable parameter called *class token* is attached to the input of transformer (patch) and such digest which is actually weight vector becomes a digest of image after it is processed through transformer so that such *image digest* is input of classification network (dense network). That image digest is much smaller than image helps transformer to maintain effective computation. Note, image digest is actually evaluated class token and please pay attention that class token is randomly initialized only one time at the beginning of training process and it is not initialized again every time image patches are fed to transformer but it is updated (learned) through iterations of backpropagation algorithm like other weighted parameters.

For instance, given $A = A(\boldsymbol{X})$ is attention of input data $\boldsymbol{X}$ which the most important part of transformer:

$$A = A(\boldsymbol{X}) = \text{Attention}(\boldsymbol{X}) = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where $Q$, $K$, and $V$ are query matrix, key matrix, and value matrix, respectively.

$$Q = \boldsymbol{X}W^Q$$
$$K = \boldsymbol{X}W^K$$
$$V = \boldsymbol{X}W^V$$

Please pay attention that the three weight matrix $W^Q$, $W^K$, and $W^V$ consist of the first part of parameters of transformer. The second part of parameter is the class token which attaches to input data $\boldsymbol{X}$. Note that:

$$\boldsymbol{X} = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \dots, \boldsymbol{x}_m^T)^T$$

Which implies that class token is the weighted vector $\boldsymbol{x}_1$. Therefore, without loss of generality, we add class token $\hat{\boldsymbol{x}}_0$ right before $\boldsymbol{x}_1$ so that:

$$\boldsymbol{X} = (\hat{\boldsymbol{x}}_0^T, \boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \dots, \boldsymbol{x}_m^T)^T$$

It is possible to denote:

$$\boldsymbol{X} = (\hat{\boldsymbol{x}}_0^T, \boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \dots, \boldsymbol{x}_{m-1}^T)^T$$

Or,

$$\boldsymbol{X} = (\boldsymbol{x}_1^T = \hat{\boldsymbol{x}}_0^T, \boldsymbol{x}_2^T, \dots, \boldsymbol{x}_m^T)^T$$

Please pay attention that class token $\hat{\boldsymbol{x}}_0$ now is parametric weight vector whereas $\boldsymbol{x}_2$, $\boldsymbol{x}_3$,…, $\boldsymbol{x}_m$ are patches of image. Therefore, it is necessary to describe shortly how to dissolve image into these $n = m-1$ patches $\boldsymbol{x}_2$, $\boldsymbol{x}_3$,…, $\boldsymbol{x}_m$. Suppose an image has width $W$ and height $H$ and every patch has width $w$ and height $h$, then there are $W/w * H/h$ patches so that $n = W/w * H/h$. For instance (Gemini 2025), given 224 x 224 x 3 image and given every patch has the size of 16 x 16 x 3, there are 196 = (224/16) * (224/16) patches so that every 16 x 16 x 3 patch is flattened into 768 x 1 vector due to 768 = 16 x 16 x 3. As result, the input data $\boldsymbol{X}$ has 197 vector and every vector has size of 768 elements in which the first vector $\hat{\boldsymbol{x}}_0 = \boldsymbol{x}_1$is class token whose dimension is 768 x 1 too. Of course, $\boldsymbol{X}$ is 197 x 768 matrix.

In the next section, the backward bias/error $b(\boldsymbol{X})$ which is the gradient of the likelihood $L(A)$ with respect to input data $\boldsymbol{X}$ is determined as following sum:

$$b(\boldsymbol{X}) = \begin{pmatrix} \left(\frac{dl_1}{d\boldsymbol{x}_1}\right)^T \\ \left(\frac{dl_2}{d\boldsymbol{x}_2}\right)^T \\ \vdots \\ \left(\frac{dl_m}{d\boldsymbol{x}_m}\right)^T \end{pmatrix} + \left(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\right)^T (A' - A)(W^V)^T$$

Where,

$$\frac{dl_i}{d\boldsymbol{x}_i} = \frac{1}{\sqrt{d_k}}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i'(K(W^Q)^T + Q(W^K)^T)\right)^T$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$\left(\frac{dl_0}{d\widehat{\boldsymbol{x}}_0}\right)^T = \left(\frac{dl_1}{d\boldsymbol{x}_1}\right)^T$$

It is interesting that the first row of the backward $b(\boldsymbol{X})$ is actually the gradient of class token $\widehat{\boldsymbol{x}}_0$ and such first row is denoted $b(\boldsymbol{X})[0]$.

$$\frac{\partial L(A)}{\partial \widehat{\boldsymbol{x}}_0} = b(\boldsymbol{X})[0]$$

As a result, parametric lass token is iteratively updated like other parameters by stochastic gradient descent as usual:

$$\widehat{\boldsymbol{x}}_0 = \widehat{\boldsymbol{x}}_0 + \gamma b(\boldsymbol{X})[0]$$

Where $\gamma$ $(0 < \gamma \leq 1)$ is learning rate.

### 3. Matrix Neural Network

The common representation of artificial neural network (ANN) is feedforward network (FFN) in which the input layer is fed forwards so as to evaluate the output layer. If FFN has many hidden layers between input layer and output layer, it will be called deep neural network (DNN). As usual, every DNN layer is coded as a vector, which raises the so-called *boom problem* of a huge number of parameters, for example, given two successive layers are coded as $m$*$n$-element vector and $p$*$q$-element vector, then there are $m$*$n$*$p$*$q$ parametric weights, which is the problem of quadratic complexity, indeed. *Matrix neural network* (MNN) aims to solve the boom problem by coding layers as matrix and coding parameters as matrix too. The MNN mentioned here is FNN with matrix layers, which is represented by a series of matrix layers $\boldsymbol{X}_0$, $\boldsymbol{X}_1$, $\boldsymbol{X}_2$,…, $\boldsymbol{X}_K$ that is called $K$-layer FFN as follows:

$$\boldsymbol{X}_k = f\left(\widehat{\boldsymbol{X}}_{k-1} = U_k \boldsymbol{X}_{k-1} V_k + \Theta_k\right), \forall k = \overline{1, K}$$

Each matrix layer $\boldsymbol{X}_k$ represented by a matrix $\boldsymbol{X}_k$ has three parameters such as parametric weight matrix $U_k$, parametric weight matrix $V_k$, and parametric bias matrix $\Theta_k$, which aims to solve the boom problem of a huge number of parameters because weigh matrices are always keep intact their size in squared number. For emphasizing on the particular layer $\boldsymbol{X}_k$, the activation function $f\left(\widehat{\boldsymbol{X}}_{k-1}\right)$ is denoted as $f_{\boldsymbol{X}_k}\left(\widehat{\boldsymbol{X}}_{k-1}\right)$. Training MNN is to estimate parameters $U_k$, $V_k$, and $\Theta_k$ by maximizing the likelihood function $l(\boldsymbol{X}_K)$ at the last output layer $\boldsymbol{X}_K$ which is defined as the negative of squared Frobenius norm given the real output $\boldsymbol{X}_K'$.

$$l(\boldsymbol{X}_K) = -\frac{1}{2}\|\boldsymbol{X}_K' - \boldsymbol{X}_K\|^2$$

Where,

$$\|\boldsymbol{X}_K' - \boldsymbol{X}_K\|^2 = \sum_i \sum_j \left(x'_{Kij} - x_{Kij}\right)^2$$

$$l'(\boldsymbol{X}_K) = \nabla_{\boldsymbol{X}_K} l(\boldsymbol{X}_K) = \boldsymbol{X}_K' - \boldsymbol{X}_K$$

So that:

$$\begin{aligned} U_k^* &= \underset{U_k}{\operatorname{argmax}}\, l(\boldsymbol{X}_K) \\ V_k^* &= \underset{V_k}{\operatorname{argmax}}\, l(\boldsymbol{X}_K), \forall k = K, K-1, \ldots, 1 \\ \Theta_k^* &= \underset{\Theta_k}{\operatorname{argmax}}\, l(\boldsymbol{X}_K) \end{aligned}$$

The optimization problem is solved iteratively by association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm. Particularly, parametric weight matrices $U_k$, $V_k$ and parametric bias matrix $\Theta_k$ at $k^{\text{th}}$ layer are estimated iterative by SGD as follows:

$$\begin{aligned} U_k &= U_k + \gamma \nabla_{U_k} l \\ V_k &= V_k + \gamma \nabla_{V_k} l \\ \Theta_k &= \Theta_k + \gamma \nabla_{\Theta_k} l \end{aligned}, \forall k = \overline{1, K}$$

Where $\gamma$ $(0 < \gamma \leq 1)$ is learning rate and $\nabla_{U_k} l$, $\nabla_{V_k} l$, and $\nabla_{\Theta_k} l$ are gradients or first-order derivatives of likelihood $l(\boldsymbol{X}_K)$ with respect to $U_k$, $V_k$, and $\Theta_k$, respectively. Therefore, how to train MNN is

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



essentially to calculate the parametric weight gradients $\nabla_{U_k} l$, $\nabla_{V_k} l$ and parametric bias gradient $\nabla_{\Theta_k} l$. Note:

$$\nabla_{U_k} l = \nabla_{U_k} l(\boldsymbol{X}_k)$$
$$\nabla_{V_k} l = \nabla_{V_k} l(\boldsymbol{X}_k), \forall k = \overline{1,K}$$
$$\nabla_{\Theta_k} l = \nabla_{\Theta_k} l(\boldsymbol{X}_k)$$

Because high dimensions of gradients related to tensors, *vectorization* technique is applied into calculating these gradients so that matrix gradients are equivalent to vector gradients as follows:

$$\nabla_{U_k} l(\boldsymbol{X}_k) \equiv \nabla_{U_k} l\big(\text{vec}(\boldsymbol{X}_K)\big)$$
$$\nabla_{V_k} l(\boldsymbol{X}_k) \equiv \nabla_{V_k} l\big(\text{vec}(\boldsymbol{X}_K)\big), \forall k = \overline{1,K}$$
$$\nabla_{\Theta_k} l(\boldsymbol{X}_k) \equiv \nabla_{\Theta_k} l\big(\text{vec}(\boldsymbol{X}_K)\big)$$

As a convention, we denote:

$$\nabla_{U_k} l \equiv \nabla_{U_k} \text{vec}(l)$$
$$\nabla_{V_k} l \equiv \nabla_{V_k} \text{vec}(l), \forall k = \overline{1,K}$$
$$\nabla_{\Theta_k} l \equiv \nabla_{\Theta_k} \text{vec}(l)$$

Where the notation vec(.) denotes vectorization operator that convert a matrix into single column vector without information loss. Let $\nabla_{\boldsymbol{X}_K} l(\boldsymbol{X}_K)$ be the gradient of likelihood with respect to $\boldsymbol{X}_K$, which is determined as follows:

$$\nabla_{\boldsymbol{X}_K} l\big(\text{vec}(\boldsymbol{X}_K)\big) = \text{vec}\Big(\nabla_{\boldsymbol{X}_K} l(\boldsymbol{X}_K)\Big) = \text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)$$

Differential $dl$ of likelihood with respect to $\boldsymbol{X}_{K-1}$ within vectorization is:

$$d\text{vec}\big(l(\boldsymbol{X}_K)\big) = dl(\boldsymbol{X}_K) = dl\big(\text{vec}(\boldsymbol{X}_K)\big)$$
$$\big(\text{Due to } \|\boldsymbol{X}'_K - \boldsymbol{X}_K\|^2 = \|\text{vec}(\boldsymbol{X}'_K) - \text{vec}(\boldsymbol{X}_K)\|^2 = \text{vec}(\|\boldsymbol{X}'_K - \boldsymbol{X}_K\|^2)\big)$$
$$= \text{tr}\Big(\big(\nabla l\big(\text{vec}(\boldsymbol{X}_K)\big)\big)^T d\text{vec}(\boldsymbol{X}_K)\Big)$$

(This is the important property making the relationship between gradient and trace operator)

$$= \text{tr}\Big(\big(\text{vec}(\boldsymbol{X}'_K) - \text{vec}(\boldsymbol{X}_K)\big)^T d\text{vec}(\boldsymbol{X}_K)\Big)$$

$$\big(\text{Due to } \|\boldsymbol{X}'_K - \boldsymbol{X}_K\|^2 = \|\text{vec}(\boldsymbol{X}'_K) - \text{vec}(\boldsymbol{X}_K)\|^2\big)$$
$$= \text{tr}\big(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T \text{vec}(d\boldsymbol{X}_K)\big) = \text{tr}\Big(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T \text{vec}\Big(df_{\boldsymbol{X}_K}(\widehat{\boldsymbol{X}}_{K-1})\Big)\Big)$$
$$= \text{tr}\Big(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T d\text{vec}\Big(f_{\boldsymbol{X}_K}(\widehat{\boldsymbol{X}}_{K-1})\Big)\Big)$$

Note, the superscript "$T$" denotes transposition operator of vector and matrix whereas tr(.) denotes trace operator which takes the sum of diagonal elements of square matrix. Moreover, we can denote:

$$dl = d\text{vec}(l) = d\text{vec}\big(l(\boldsymbol{X}_K)\big)$$

Due to:

$$d\text{vec}\Big(f_{\boldsymbol{X}_K}(\widehat{\boldsymbol{X}}_{K-1})\Big) = d\text{vec}\Big(f_{\boldsymbol{X}_K}(U_K \boldsymbol{X}_{K-1} V_K + \Theta_K)\Big) = d\Big(f_{\boldsymbol{X}_K}\big(\text{vec}(U_K \boldsymbol{X}_{K-1} V_K + \Theta_K)\big)\Big)$$

$$= f'_{\boldsymbol{X}_K}\Big(\text{vec}\big(\widehat{\boldsymbol{X}}_{K-1}\big)\Big)(V_K^T \otimes U_K)\text{vec}(d\boldsymbol{X}_{K-1}) = f'_{\boldsymbol{X}_K}\Big(\text{vec}\big(\widehat{\boldsymbol{X}}_{K-1}\big)\Big)(V_K^T \otimes U_K) d\text{vec}(\boldsymbol{X}_{K-1})$$

Which implies:

$$d\text{vec}\big(l(\boldsymbol{X}_K)\big) = \text{tr}\Big(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T f'_{\boldsymbol{X}_K}\Big(\text{vec}\big(\widehat{\boldsymbol{X}}_{K-1}\big)\Big)(V_K^T \otimes U_K) d\text{vec}(\boldsymbol{X}_{K-1})\Big)$$

Note, the notation $\otimes$ denotes Kronecker product. Please pay attention that $f'_{\boldsymbol{X}_K}\Big(\text{vec}\big(\widehat{\boldsymbol{X}}_{K-1}\big)\Big)$ is the derivative of the activation function $f_{\boldsymbol{X}_K}\Big(\text{vec}\big(\widehat{\boldsymbol{X}}_{K-1}\big)\Big)$ with respective to $\text{vec}\big(\widehat{\boldsymbol{X}}_{K-1}\big)$ and the following property related to derivative of vec(.) variable, for instance, given matrix variable $\boldsymbol{X}$ and two matrix constants $A$ and $B$, we have the very important property as follows:

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$d\text{vec}(A\boldsymbol{X}B) = \text{vec}\big(d(A\boldsymbol{X}B)\big) = (B^T \otimes A)\text{vec}(d\boldsymbol{X}) = (B^T \otimes A)d\big(\text{vec}(\boldsymbol{X})\big)$$

On the other hand, we have:

$$d\text{vec}\left(f\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right) = d\text{vec}\left(f_{X_K}(U_K\boldsymbol{X}_{K-1}V_K + \Theta_K)\right) = f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right) d\text{vec}(\Theta_K)$$

Such that likelihood differential $dl$ with respect to last bias parameter $\Theta_K$ is:

$$d\text{vec}\big(l(\boldsymbol{X}_K)\big) = \text{tr}\left(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right) d\text{vec}(\Theta_K)\right)$$

In other words, the gradient of last bias parameter $\Theta_K$ is:

$$\nabla_{\Theta_K}\text{vec}(l) = \frac{d\text{vec}\big(l(\boldsymbol{X}_K)\big)}{d\text{vec}(\Theta_K)} = \left(f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)\right)^T \text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)$$

(This is the important property making the relationship between gradient and trace operator) Let $I_{r(U_K)}$ be the identity matrix whose dimension is $r(U_K)$ x $r(U_K)$ where $r(U_K)$ is the number of rows of $U_K$. We also have:

$$d\text{vec}\left(f\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right) = d\text{vec}\left(f_{X_K}(U_K\boldsymbol{X}_{K-1}V_K + \Theta_K)\right) = f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)(\boldsymbol{X}_{K-1}V_K)^T \otimes I_{r(U_K)} d\text{vec}(U_K)$$

Such that likelihood differential $dl$ with respect to last weight parameter $U_K$ is:

$$d\text{vec}\big(l(\boldsymbol{X}_K)\big) = \text{tr}\left(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)\left((\boldsymbol{X}_{K-1}V_K)^T \otimes I_{r(U_K)}\right) d\text{vec}(U_K)\right)$$

In other words, the gradient of last weight parameter $U_K$ is:

$$\nabla_{U_K}\text{vec}(l) = \frac{d\text{vec}\big(l(\boldsymbol{X}_K)\big)}{d\text{vec}(U_K)} = \left((\boldsymbol{X}_{K-1}V_K)^T \otimes I_{r(U_K)}\right)^T \left(f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)\right)^T \text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)$$

$$= \left((\boldsymbol{X}_{K-1}V_K)^T \otimes I_{r(U_K)}\right)^T \nabla_{\Theta_K}\text{vec}(l)$$

Let $I_{c(V_K)}$ be the identity matrix whose dimension is $c(V_K)$ x $c(V_K)$ where $c(V_K)$ is the number of columns of $U_K$. Similarly, likelihood differential $dl$ with respect to last weight parameter $V_K$ is:

$$d\text{vec}\big(l(\boldsymbol{X}_K)\big) = \text{tr}\left(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)\left(I_{c(V_K)} \otimes (U_K\boldsymbol{X}_{K-1})\right) d\text{vec}(V_K)\right)$$

In other words, the gradient of last weight parameter $V_K$ is:

$$\nabla_{V_K}\text{vec}(l) = \frac{d\text{vec}\big(l(\boldsymbol{X}_K)\big)}{d\text{vec}(V_K)} = \left(I_{c(V_K)} \otimes (U_K\boldsymbol{X}_{K-1})\right)^T \left(f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)\right)^T \text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)$$

$$= \left(I_{c(V_K)} \otimes (U_K\boldsymbol{X}_{K-1})\right)^T \nabla_{\Theta_K}\text{vec}(l)$$

Moreover, likelihood differential $dl$ is rewritten according to bias gradient as follows:

$$d\text{vec}\big(l(\boldsymbol{X}_K)\big) = \text{tr}\left(\text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)^T f'_{X_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)(V_K^T \otimes U_K) d\text{vec}(\boldsymbol{X}_{K-1})\right) =$$

$$= \text{tr}\left(\left(\nabla_{\Theta_K}\text{vec}(l)\right)^T (V_K^T \otimes U_K) d\text{vec}(\boldsymbol{X}_{K-1})\right)$$

By generalizing differential likelihood $dl$ with respect to any $k^{\text{th}}$ layer for all $1 \le k < K$ within vectorization, we obtain:

$$\nabla_{U_k}\text{vec}(l) = \left((\boldsymbol{X}_{k-1}V_K)^T \otimes I_{r(U_k)}\right)^T \nabla_{\Theta_k}\text{vec}(l)$$

$$\nabla_{V_k}\text{vec}(l) = \left(I_{c(V_k)} \otimes (U_k\boldsymbol{X}_{k-1})\right)^T \nabla_{\Theta_k}\text{vec}(l)$$

$$d\text{vec}(l) = \text{tr}\left(\left(\nabla_{\Theta_{k+1}}\text{vec}(l)\right)^T (V_{k+1}^T \otimes U_{k+1}) d\text{vec}(\boldsymbol{X}_k)\right)$$

Because the differential of vec($\boldsymbol{X}_k$) is always determined with respect to any bias parameter $\Theta_k$ for all $1 \le k < K$.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$d\text{vec}(\boldsymbol{X}_k) = d\text{vec}\left(f\left(\widehat{\boldsymbol{X}}_{k-1}\right)\right) = f'_{\boldsymbol{X}_k}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{k-1}\right)\right) d\text{vec}(\Theta_k)$$

Such that the likelihood differential $dl$ is rewritten with respect to any bias parameter $\Theta_k$ for all $1 \leq k < K$.

$$d\text{vec}\left(l(\boldsymbol{X}_K)\right) = \text{tr}\left(\left(\nabla_{\Theta_{k+1}}\text{vec}(l)\right)^T (V_{k+1}^T \otimes U_{k+1}) f'_{\boldsymbol{X}_k}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{k-1}\right)\right) d\text{vec}(\Theta_k)\right)$$

It is totally to determine any gradient of $\Theta_k$ for all $1 \leq k < K$.

$$\nabla_{\Theta_k}\text{vec}(l) = \frac{d\text{vec}\left(l(\boldsymbol{X}_K)\right)}{d\text{vec}(\Theta_k)} = \left(f'_{\boldsymbol{X}_k}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{k-1}\right)\right)\right)^T (V_{k+1}^T \otimes U_{k+1})^T \nabla_{\Theta_{k+1}}\text{vec}(l)$$

As a result of association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm for training matrix neural network (MNN), all gradients $\nabla_{U_k}\text{vec}(l)$, $\nabla_{V_k}\text{vec}(l)$, and $\nabla_{\Theta_k}\text{vec}(l)$ of Frobenius norm likelihood with respect to parameters $U_k$, $V_k$, and $\Theta_k$ within vectorization technique for all $1 \leq k \leq K$, respectively are calculated as follows:

$$\begin{aligned}
\nabla_{U_k}\text{vec}(l) &= \left((\boldsymbol{X}_{k-1}V_k)^T \otimes I_{r(U_k)}\right)^T \nabla_{\Theta_k}\text{vec}(l) \\
\nabla_{V_k}\text{vec}(l) &= \left(I_{c(V_k)} \otimes (U_k\boldsymbol{X}_{k-1})\right)^T \nabla_{\Theta_k}\text{vec}(l) \\
\nabla_{\Theta_k}\text{vec}(l) &= \left(f'_{\boldsymbol{X}_k}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{k-1}\right)\right)\right)^T (V_{k+1}^T \otimes U_{k+1})^T \nabla_{\Theta_{k+1}}\text{vec}(l) \\
\nabla_{\Theta_K}\text{vec}(l) &= \left(f'_{\boldsymbol{X}_K}\left(\text{vec}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right)\right)^T \text{vec}(\boldsymbol{X}'_K - \boldsymbol{X}_K)
\end{aligned} \tag{3.1}$$

Such that:

$$\begin{aligned}
\text{vec}(U_k) &= \text{vec}(U_k) + \gamma\nabla_{U_k}\text{vec}(l) \\
\text{vec}(V_k) &= \text{vec}(V_k) + \gamma\nabla_{V_k}\text{vec}(l) \\
\text{vec}(\Theta_k) &= \text{vec}(\Theta_k) + \gamma\nabla_{\Theta_k}\text{vec}(l)
\end{aligned}$$

Please pay attention that functions $r(A)$ and $c(A)$ returns the number of rows and the number of columns for matrix $A$. Therefore, $I_{r(U_k)}$ is identity matrix whose dimension is $r(U_k)$x$r(U_k)$ and $I_{c(V_k)}$ is identity matrix whose dimension is $c(V_k)$x$c(V_k)$. If $U_k$ is ignored, it is identity matrix $I_{r(\boldsymbol{X}_k)}$. If $V_k$ is ignored, it is identity matrix $I_{c(\boldsymbol{X}_k)}$. Although MNN is not new, please read the research paper "Matrix Neural Networks" (Gao, Guo, & Wang, 2016) for comprehending more about MNN called often matrix-valued neural network in literature, this research contributes to reproduce the succinct and practical equations above to train MNN.

In practical calculation, the estimation equations are not significantly simpler than traditional ANN training because of vectorization operator and Kronecker product. Fortunately, wise-multiplication $\odot$ is applied into reducing computational cost as follows:

$$\begin{aligned}
\nabla_{U_k} l &= \text{vec}^{-1}\left(\left((\boldsymbol{X}_{k-1}V_k)^T \otimes I_{r(U_k)}\right)^T \text{vec}\left(\nabla_{\Theta_k} l\right)\right) \\
\nabla_{V_k} l &= \text{vec}^{-1}\left(\left(I_{c(V_k)} \otimes (U_k\boldsymbol{X}_{k-1})\right)^T \text{vec}\left(\nabla_{\Theta_k} l\right)\right) \\
\nabla_{\Theta_k} l &= \left\{f'_{\boldsymbol{X}_k}\left(\widehat{\boldsymbol{X}}_{k-1}\right)\right\} \odot \left(\text{vec}^{-1}\left((V_{k+1}^T \otimes U_{k+1})^T \text{vec}\left(\nabla_{\Theta_{k+1}} l\right)\right)\right) \\
\nabla_{\Theta_K} l &= \left\{f'_{\boldsymbol{X}_K}\left(\widehat{\boldsymbol{X}}_{K-1}\right)\right\} \odot (\boldsymbol{X}'_K - \boldsymbol{X}_K)
\end{aligned}$$

Such that:

$$\begin{aligned}
U_k &= U_k + \gamma\nabla_{U_k} l \\
V_k &= V_k + \gamma\nabla_{V_k} l \\
\Theta_k &= \Theta_k + \gamma\nabla_{\Theta_k} l
\end{aligned}$$

Where notation $\odot$ denotes wise-multiplication and notation $\text{vec}^{-1}(.)$ denoted the inverse vectorization operator that converts reversely a single column vector into original matrix. Note, given two vectors (matrices) whose have the same dimension, the wise-multiplication operator between them produces a new vector (matrix) whose every element is result of multiplication of one element

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



from a vector (matrix) and one element from another vector (matrix) with constraint that the two elements have the same index in their own vectors (matrices). Please pay attention that $\{f'_{X_k}(\widehat{\boldsymbol{X}}_{k-1})\}$ is not the derivative of the activation function *f*(.) with respective to $\widehat{\boldsymbol{X}}_{k-1}$, indeed, which is the matrix of taking derivatives of all elements of $\widehat{\boldsymbol{X}}_{k-1}$, which means that:

$$\{f'_{X_k}(\widehat{\boldsymbol{X}}_{k-1})\} = \{f'(\widehat{\boldsymbol{X}})\} = \begin{pmatrix} f'(\hat{x}_{11}) & f'(\hat{x}_{12}) & \cdots & f'(x'_{1n}) \\ f'(x'_{21}) & f'(x'_{22}) & \cdots & f'(x'_{2n}) \\ \vdots & \vdots & \ddots & \vdots \\ f'(x'_{m1}) & f'(x'_{m2}) & \cdots & f'(x'_{mn}) \end{pmatrix}$$

For example, when *f*(.) is sigmoid function $f(\hat{x}_{ij}) = \frac{1}{1+e^{-\hat{x}_{ij}}}$, its derivative on $\hat{x}_{ij}$ is:

$$f'(\hat{x}_{ij}) = f(\hat{x}_{ij})\left(1 - f(\hat{x}_{ij})\right)$$

Moreover, Kronecker product $\otimes$ is easily optimized in calculation by programming technique. For example, given:

$$\nabla_{U_k} l = \text{vec}^{-1}\left(\left((\boldsymbol{X}_{k-1}V_k)^T \otimes I_{r(U_k)}\right)^T \text{vec}(\nabla_{\Theta_k} l)\right)$$

It is necessary to illustrate how to calculate:

$$\text{vec}^{-1}\left((A^T \otimes B)^T \text{vec}(C)\right)$$

Where *A*, *B*, and *C* are following matrices.

$$A = \boldsymbol{X}_{k-1}V_k$$
$$B = I_{r(U_k)}$$
$$C = \text{vec}(\nabla_{\Theta_k} l)$$

Indeed, we have:

$$\text{vec}^{-1}\left((A^T \otimes B)^T \text{vec}(C)\right) = \text{vec}^{-1}\left(A \otimes B^T \text{vec}(C)\right)$$

Suppose:

$$A = (a_{ij})_{mxn} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, B = (b_{ij})_{pxq} = \begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{pmatrix}$$

Let $P_i$ be the matrix whose every element is multiplication of matrix $B^T$ and every element of the $i^{\text{th}}$ row of matrix *A*:

$$P_i = A_i \otimes B^T = \begin{pmatrix} a_{11}B^T & \cdots & a_{1n}B^T \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & \cdots & a_{11}b_{p1} & \cdots & \cdots & a_{1n}b_{11} & \ldots & a_{1n}b_{p1} \\ \vdots & \ddots & \vdots & & & \vdots & \ddots & \vdots \\ a_{11}b_{1q} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{1q} & \cdots & a_{1n}b_{pq} \end{pmatrix}$$

Of course, dimension of $P_i$ is $q$ x $n*p$. It is easy to multiply every $P_i$ with vec(*C*) where vec(*C*) is the $n*p$ x 1 vector to produce every column vector $Q_i$ whose dimension is $q$ x 1.

$$Q_i = P_i \text{vec}(C)$$

The final result is the vertical concatenation of $m$ vectors $Q_i$, which is $q$ x $m$ matrix as follows:

$$(Q_1, Q_2, \ldots, Q_m) = \text{vec}^{-1}\left((A^T \otimes B)^T \text{vec}(C)\right) = \text{vec}^{-1}\left(\left((\boldsymbol{X}_{k-1}V_k)^T \otimes I_{r(U_k)}\right)^T \text{vec}(\nabla_{\Theta_k} l)\right) = \nabla_{U_k} l$$

The gradient $\nabla_{\Theta_k}\text{vec}(l)$ of the bias $\Theta_k$ at the layer $\boldsymbol{X}_k$ is called the $k^{\text{th}}$ bias is extended as follows:

$$b_{\boldsymbol{X}_k}(\text{vec}(\boldsymbol{X}_k)) = \nabla_{\Theta_k}\text{vec}(l)$$

$$= \left(\prod_{l=k}^{K-1}\left(f'_{X_l}\left(\text{vec}(\widehat{\boldsymbol{X}}_{l-1})\right)\right)^T (V_{l+1}^T \otimes U_{l+1})^T\right) * \left(f'_{X_K}\left(\text{vec}(\widehat{\boldsymbol{X}}_{K-1})\right)\right)^T b_{\boldsymbol{X}_K}(\text{vec}(\boldsymbol{X}_K))$$

Where the quantity $b_{\boldsymbol{X}_K}(\text{vec}(\boldsymbol{X}_K))$ is called the core last bias which was calculated by the likelihood based on squared Frobenius norm.

$$b_{\boldsymbol{X}_K}(\text{vec}(\boldsymbol{X}_K)) = \boldsymbol{X}'_K - \boldsymbol{X}_K$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



If MNN is applied into aforementioned classification task in order to derive MNN classifier, it is simpler to modify the core last bias to minimize the cross-entropy loss function as follows:

$$b_{\boldsymbol{X}_K}(\mathrm{vec}(\boldsymbol{X}_K)) = \left(\sum_{i=1}^{n} p_i(\delta_{ij} - \hat{p}_j)\right)^T_{j=\overline{1,n}}$$

Where,

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

Note, the vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)^T$ is the real class probability of the last layer $\boldsymbol{X}_K = \boldsymbol{x}_K$ such that

$$\hat{p}_i = \mathrm{softmax}(x_{Ki}) = \frac{\exp(x_{Ki})}{\sum_{l=1}^{n} \exp(x_{Kl})}$$

Please pay attention that it is totally possible to remove complexity of Kronecker product and vectorization technique by association of programming technique and independent aspect of activation function on matrix. Moreover, the $k^{\text{th}}$ bias $b_{\boldsymbol{X}_k}(\mathrm{vec}(\boldsymbol{X}_k)) = \nabla_{\Theta_k}\mathrm{vec}(l)$ which is also called backward bias or *backward error* at $k^{\text{th}}$ layer is the cornerstone of association of backpropagation algorithm and stochastic gradient descent (SGD) algorithm because the gradient (bias) $\nabla_{\Theta_k}\mathrm{vec}(l)$ is also the first-order derivative of training likelihood $l(\boldsymbol{X}_K)$ with respect to $\widehat{\boldsymbol{X}}_{k-1}$ at the layer $\boldsymbol{X}_k$.

$$\nabla_{\Theta_k}\mathrm{vec}(l) = \frac{dl(\boldsymbol{X}_K)}{d\mathrm{vec}(\Theta_k)} = \frac{dl(\boldsymbol{X}_K)}{d\mathrm{vec}(\widehat{\boldsymbol{X}}_{k-1})}$$

*Transformer* developed by Vaswani et al. (Vaswani, et al., 2017) is invented to improve deep neural network (DNN) so that it discovers the self-structure or the internal feature of input for DNN. Exactly, given input $\boldsymbol{X} = \boldsymbol{X}_0$ of DNN, self-structure of $\boldsymbol{X}$ which is internal structure of $\boldsymbol{X}$ is known as self-attention or attention in context of transformer. In other words, transformer extracts such attention denoted $A = A(\boldsymbol{X})$ so that DNN processes the finer feature $A$ instead of processing the coarser input $\boldsymbol{X}$. Extracting attention is indeed self-supervised learning which is the intermediate one between supervised learning and unsupervised learning. In practice, transformer is associated with feedforward network (FFN) in order to establish a complex transformer which aims to provide the self-supervised learning mechanism to artificial neural network, which is actually a revolution in domain of artificial intelligence (AI) when artificial neural network (ANN) supports fully four large subjects of machine learning such as supervised learning, self-supervised learning, unsupervised learning, and reinforcement learning. Note that ANN supports Q-learning which is a popular reinforcement learning method. As a convention, complex transformer which is the association of transformer and FFN where transformer is the first part and FFN is the second part is known as transformer as usual. At the other hand, transformer built in its core "attention" without FFN is also a different methodology from traditional ANN because the propagation rule of attentional transformer $A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ mentioned later is an improvement of the propagation rule of ANN $\boldsymbol{X}_k = f\left(\widehat{\boldsymbol{X}}_{k-1} = U_k\boldsymbol{X}_{k-1}V_k + \Theta_k\right)$. Actually, the transformer-based propagation rule $A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ is more complicated and clever than the ANN-based propagation rule $\boldsymbol{X}_k = f\left(\widehat{\boldsymbol{X}}_{k-1} = U_k\boldsymbol{X}_{k-1}V_k + \Theta_k\right)$ with note that the soft-max function $\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ is also a perfect squash activation function. Therefore, it is totally to connect many attentions as attentional layers together by the same way that ANN connects its neuron layers together. The reason that the FFN connects the attention $A(\boldsymbol{X})$ for constituting a complex transformer is to fine-tune the attention $A(\boldsymbol{X})$, which is the second viewpoint that is opposite to the aforementioned first viewpoint that the attention $A(\boldsymbol{X})$ aims to fine-tune the coarse input data $\boldsymbol{X}$ into finer feature $A(\boldsymbol{X})$.

The essence of convolutional neural network (CNN) is to extract feature of image whereas the essence of transformer-based attention discovering is to extract feature of general data in matrix form. That is the reason of the question "Is matrix neural network the alternative of convolutional neural network?" which is the main hypothesis of this research, which also implies that whether attention extracted by transformer can be the alternative of convolutional layer resulted by CNN when

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



attention of transformer is always based on matrix. The first hazard problem is that CNN processes fast convolutional layer because of the implication of filter operating by small size of kernel whereas there is the aforementioned boom problem of ANN depends on flattening high-dimensional matrix-form data like image into vector-form input layer which cause the aforementioned problem of quadratic complexity. Fortunately, the first hazard problem is solved exactly by matrix neural network (MNN) where parameters of MNN are in matrix form. As a result, the fact that FFN of transformer is MNN is application of MNN into transformer so that the research focuses on the effectiveness of MNN in comparison with CNN in order to evaluate the hypothesis "Is matrix neural network the alternative of convolutional neural network?" with note that convergence speed is also concerned when evaluating this hypothesis. Anyhow, it is necessary to sketch transformer and its extraction mechanism for extracting attention.

Given input data $\boldsymbol{X} = \boldsymbol{X}_0$ and three weight matrices $W^Q$, $W^K$, and $W^V$ which are parameters of transformer, then query matrix $Q$, key matrix $K$, and value matrix $V$ are specified as follows:

$$Q = \boldsymbol{X}W^Q$$
$$K = \boldsymbol{X}W^K$$
$$V = \boldsymbol{X}W^V$$

Query matrix $Q$ represents what the system necessarily searches and key matrix $K$ represents the information that sequence $\boldsymbol{X}$ may expectedly contain whereas value matrix $V$ represents the actual content of sequence $\boldsymbol{X}$ (Gemini 2025). As usual, the three parametric matrices $W^Q$, $W^K$, and $W^V$ are called query weight matrix, key weight matrix, value weight matrix, respectively. Attention $A = A(\boldsymbol{X})$ is calculated based on product of $Q$, $K$, and $V$ in association with soft-max function, as follows:

$$A = A(\boldsymbol{X}) = \text{Attention}(\boldsymbol{X}) = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Attention is learned by association of stochastic gradient descent (SGD) and backpropagation algorithm for maximizing the likelihood function which in turn is squared Frobenius norm of the distance between computed attention $A$ and environmental real attention $A'$.

$$L(A) = -\frac{1}{2}\|A' - A\|^2 = -\frac{1}{2}\sum_i\sum_j\left(a'_{ij} - a_{ij}\right)^2$$

Attention learning is the core of learning transformer and the essence of attention learning is to estimate the three parametric matrices $W^Q$, $W^K$, and $W^V$.

$$W^Q = \underset{W^Q}{\text{argmax}}\, L(A)$$
$$W^K = \underset{W^K}{\text{argmax}}\, L(A)$$
$$W^V = \underset{W^V}{\text{argmax}}\, L(A)$$

According to SGD algorithm, parameters $W^Q$, $W^K$, and $W^V$ are computed iteratively as follows:

$$W^Q = W^Q + \gamma\frac{\partial L(A)}{\partial W^Q}$$
$$W^K = W^K + \gamma\frac{\partial L(A)}{\partial W^K}$$
$$W^V = W^V + \gamma\frac{\partial L(A)}{\partial W^V}$$

Note, $\gamma$ ($0 < \gamma \leq 1$) is learning rate whereas $\frac{L(A)}{\partial W^Q}$, $\frac{L(A)}{\partial W^K}$, and $\frac{L(A)}{\partial W^V}$ are first-order derivatives (gradients) of $L(A)$ with respect to $W^Q$, $W^K$, and $W^V$, respectively. However, it is not easy to calculate directly calculating $\frac{L(A)}{\partial W^Q}$, $\frac{L(A)}{\partial W^K}$, and $\frac{L(A)}{\partial W^V}$ based only on $L(A)$.

Recall that SGD is applied into attention learning but calculating gradients in SGD is seemly infeasible in practice because derivative of the matrix-by-matrix soft-max function $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ is high-dimensional tensor. Fortunately, $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ is calculated by row-by-row way so that its rows are mutually independent so that the likelihood function $L(A)$ will be partitioned into partial

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



likelihood functions $l_i(\boldsymbol{a}_i)$ where every partial attention $\boldsymbol{a}_i$ is the column vector representing the $i^{\text{th}}$ row of $A$.

$$l_i(\boldsymbol{a}_i) = -\frac{1}{2}\|\boldsymbol{a}_i' - \boldsymbol{a}_i\|^2 = -\frac{1}{2}\sum_j \left(a_{ij}' - a_{ij}\right)^2$$

Such that differential of $L(A)$ are sum of partial differentials of $l_i(\boldsymbol{a}_i)$.

$$L(A) = \sum_{i=1}^{m} l_i(\boldsymbol{a}_i)$$

$$dL(A) = \sum_{i=1}^{m} dl_i(\boldsymbol{a}_i)$$

Note,

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = (\boldsymbol{p}_1^T, \boldsymbol{p}_2^T, \dots, \boldsymbol{p}_m^T)^T$$

$$R = (\boldsymbol{r}_1^T, \boldsymbol{r}_2^T, \dots, \boldsymbol{r}_m^T)^T = QK^T$$

$$\boldsymbol{p}_i = \text{softmax}\left(\frac{\boldsymbol{q}_i^T\boldsymbol{k}_1}{\sqrt{d_k}}, \frac{\boldsymbol{q}_i^T\boldsymbol{k}_2}{\sqrt{d_k}}, \dots, \frac{\boldsymbol{q}_i^T\boldsymbol{k}_m}{\sqrt{d_k}}\right)^T = (p_{i1}, p_{i2}, \dots, p_{im})^T$$

$$p_{ij} = \frac{\exp\left(\frac{\boldsymbol{q}_i^T\boldsymbol{k}_j}{\sqrt{d_k}}\right)}{\sum_{l=1}^{m}\exp\left(\frac{\boldsymbol{q}_i^T\boldsymbol{k}_l}{\sqrt{d_k}}\right)}$$

$$\boldsymbol{r}_i = (r_{i1}, r_{i1}, \dots, r_{im})^T = \left(\frac{\boldsymbol{q}_i\boldsymbol{k}_1^T}{\sqrt{d_k}}, \frac{\boldsymbol{q}_i\boldsymbol{k}_2^T}{\sqrt{d_k}}, \dots, \frac{\boldsymbol{q}_i\boldsymbol{k}_m^T}{\sqrt{d_k}}\right)^T$$

$$\boldsymbol{a}_i = \left(a_{i1}, a_{i2}, \dots, a_{id_v}\right)^T$$

Given

$$\boldsymbol{X} = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \dots, \boldsymbol{x}_m^T)^T$$

$$\boldsymbol{x}_i = \left(x_{i1}, x_{i2}, \dots, x_{id_m}\right)^T$$

Such that:

$$\text{softmax}(R) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

$$\boldsymbol{p}_i = \text{softmax}(\boldsymbol{r}_i)$$

Note, dimensions of weight matrices $W^Q$, $W^K$, and $W^V$ are $d_m$ x $d_k$, $d_m$ x $d_k$, and $d_m$ x $d_v$, respectively. Dimensions of matrices $Q$, $K$, and $V$ are $m$ x $d_k$, $m$ x $d_k$, and $m$ x $d_v$, respectively whereas dimension of input data $\boldsymbol{X}$ is $m$ x $d_m$. In literature, $d_m$ which is most important is called model dimension, which establishes length of input-output vector. Given the self-attention $\boldsymbol{a}_i = (a_{i1}, a_{i2}, \dots, a_{1d_v})^T$ of the $i^{\text{th}}$ token, its element $a_{ij} = \sum_{k=1}^{m} p_{ik}v_{kj}$ is the weighted sum of all $j^{\text{th}}$ terms (at the same $j^{\text{th}}$ column) over all tokens where every $k^{\text{th}}$ weight is specified as the probability $p_{ik}$ that measures the similarity of the $i^{\text{th}}$ token and the $k^{\text{th}}$ token. It is interesting that such probability $p_{ik}$ which is essentially the similarity is calculated as the association of dot product (non-normalized cosine) and soft-max function (for normalizing), which can be interpreted by another way as the frequency that the current $i^{\text{th}}$ token matches the $k^{\text{th}}$ token.

By partitioning $L(A)$ into many $l_i(\boldsymbol{a}_i)$, it is possible to calculate gradients $\frac{L(A)}{\partial W^Q}$, $\frac{L(A)}{\partial W^K}$, and $\frac{L(A)}{\partial W^V}$. Indeed, differential of $l_i(\boldsymbol{a}_i)$ with respect to $\boldsymbol{q}_i$ is:

$$dl_i = \text{tr}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T d(\boldsymbol{a}_i)\right) = \text{tr}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T d(\boldsymbol{p}_i^T V)^T\right) = \text{tr}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T d(V^T\boldsymbol{p}_i)\right)$$

$$= \text{tr}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' d(\boldsymbol{r}_i)\right) = \text{tr}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' \left(\frac{\partial r_{i1}}{\partial \boldsymbol{q}_i}, \frac{\partial r_{i2}}{\partial \boldsymbol{q}_i}, \dots, \frac{\partial r_{im}}{\partial \boldsymbol{q}_i}\right)^T d\boldsymbol{q}_i\right)$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Where $\boldsymbol{p}_i'$ is derivative of the soft-max function $\boldsymbol{p}_i$, which is mentioned later.

$$\boldsymbol{p}_i' = \text{softmax}'(\boldsymbol{r}_i) = \frac{d\text{softmax}(\boldsymbol{r}_i)}{d\boldsymbol{r}_i}$$

Due to:

$$\frac{\partial r_{ij}}{\partial \boldsymbol{q}_i} = \frac{\partial \frac{\boldsymbol{q}_i \boldsymbol{k}_j^T}{\sqrt{d_k}}}{\partial \boldsymbol{q}_i} = \frac{1}{\sqrt{d_k}} \boldsymbol{k}_j^T$$

We obtain:

$$dl_i = \frac{1}{\sqrt{d_k}} \text{tr}\big((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K d(\boldsymbol{q}_i)\big) = \frac{1}{\sqrt{d_k}} \text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K d\boldsymbol{q}_i)$$

Due to:

$$\boldsymbol{q}_i^T = \boldsymbol{x}_i^T W^Q$$

Differential of $l_i(\boldsymbol{a}_i)$ with respect to $W^Q$ is:

$$dl_i = \frac{1}{\sqrt{d_k}} \text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K d(\boldsymbol{x}_i^T W^Q)^T) = \frac{1}{\sqrt{d_k}} \text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K d(W^Q)^T \boldsymbol{x}_i)$$

$$= \frac{1}{\sqrt{d_k}} \text{tr}(\boldsymbol{x}_i (\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K d(W^Q)^T)$$

$$\big(\text{Due to } \text{tr}(BCD) = \text{tr}(CDB) = \text{tr}(DBC)\big)$$

Therefore, gradient of $l_i(\boldsymbol{a}_i)$ with respect to $W^Q$ is:

$$\frac{\partial l_i}{\partial W^Q} = \frac{1}{\sqrt{d_k}} \boldsymbol{x}_i (\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K$$

Note, every partial real attention $\boldsymbol{a}'_i$ is the column vector representing the $i^{\text{th}}$ row of the real attention $A'$. This means gradient of $L(A)$ with respect to $W^Q$ is:

$$\frac{\partial L(A)}{\partial W^Q} = \frac{1}{\sqrt{d_k}} \sum_{i=1}^{m} \boldsymbol{x}_i (\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K \tag{3.2}$$

Where $\boldsymbol{p}_i'$ is derivative of the soft-max function $\boldsymbol{p}_i$, which is mentioned later. Due to:

$$dL(A) = \sum_{i=1}^{m} dl_i(\boldsymbol{a}_i)$$

Similarly, gradient of $L(A)$ with respect to $W^K$ is:

$$\frac{\partial L(A)}{\partial W^K} = \frac{1}{\sqrt{d_k}} \sum_{i=1}^{m} \boldsymbol{x}_i (\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' Q \tag{3.3}$$

Recall that estimating $W^Q$ and $W^K$ is totally determined as follows:

$$W^Q = W^Q + \gamma \frac{\partial L(A)}{\partial W^Q}$$

$$W^K = W^K + \gamma \frac{\partial L(A)}{\partial W^K}$$

Consequently, it is necessary to calculate gradient of $L(A)$ with respect to $W^V$. Indeed, differential of $L(A)$ with respect to $W^V$ is:

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



$$dL = \mathrm{tr}\big((A' - A)^T d(A)\big) = \mathrm{tr}\left((A' - A)^T d\left(\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\right)\right) = \mathrm{tr}\left((A' - A)^T \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)dV\right)$$

$$= \mathrm{tr}\left((A' - A)^T \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)d(\boldsymbol{X}W^V)\right) = \mathrm{tr}\left((A' - A)^T \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\boldsymbol{X}d(W^V)\right)$$

Therefore, gradient of $L(A)$ with respect to $W^V$ is:

$$\frac{\partial L(A)}{\partial W^V} = \boldsymbol{X}^T\left(\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\right)^T (A' - A) \tag{3.4}$$

Such that $W^V$ is totally estimated as follows:

$$W^V = W^V + \gamma\frac{\partial L(A)}{\partial W^V}$$

Recall that parametric weight matrices of transformer-based self-attention are totally estimated by SGD algorithm as follows:

$$W^Q = W^Q + \gamma\frac{1}{\sqrt{d_k}}\sum_{i=1}^{m}\boldsymbol{x}_i(\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K$$

$$W^K = W^K + \gamma\frac{1}{\sqrt{d_k}}\sum_{i=1}^{m}\boldsymbol{x}_i(\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' Q$$

$$W^V = W^V + \gamma\boldsymbol{X}^T\left(\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\right)^T (A' - A)$$

The most important thing to estimate parametric matrices $W^Q$ and $W^K$ is to calculate the derivative $\boldsymbol{p}_i'$ of the probability $\boldsymbol{p}_i$ of soft-max function.

$$\boldsymbol{p}_i' = \mathrm{softmax}'(\boldsymbol{r}_i) = \frac{d\mathrm{softmax}(\boldsymbol{r}_i)}{d\boldsymbol{r}_i}$$

$$\boldsymbol{p}_i = \mathrm{softmax}(\boldsymbol{r}_i) = (p_{i1}, p_{i2}, \dots, p_{im})^T$$

$$\boldsymbol{r}_i = (r_{i1}, r_{i1}, \dots, r_{im})^T = \left(\frac{\boldsymbol{q}_i\boldsymbol{k}_1^T}{\sqrt{d_k}}, \frac{\boldsymbol{q}_i\boldsymbol{k}_2^T}{\sqrt{d_k}}, \dots, \frac{\boldsymbol{q}_i\boldsymbol{k}_m^T}{\sqrt{d_k}}\right)^T$$

Indeed, we have:

$$\boldsymbol{p}_i' = \mathrm{softmax}'(\boldsymbol{r}_i) = \begin{pmatrix} \frac{\partial p_{i1}}{\partial r_{i1}} & \frac{\partial p_{i1}}{\partial r_{i2}} & \cdots & \frac{\partial p_{i1}}{\partial r_{im}} \\ \frac{\partial p_{i2}}{\partial r_{i1}} & \frac{\partial p_{i2}}{\partial r_{i2}} & \cdots & \frac{\partial p_{i2}}{\partial r_{im}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_{im}}{\partial r_{i1}} & \frac{\partial p_{im}}{\partial r_{i2}} & \cdots & \frac{\partial p_{im}}{\partial r_{im}} \end{pmatrix}$$

Where,

$$\frac{\partial p_{ij}}{\partial r_{ik}} = p_{ij}(\delta_{jk} - p_{ik})$$

$$\delta_{jk} = \begin{cases} 1 \text{ if } j = k \\ 0 \text{ if } j \neq k \end{cases}$$

Because transformer is here estimated by association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm, it is necessary to calculate the backward bias (bias,

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



reward, backward reward, backward error, error) which is the gradient of the likelihood $L(A)$ with respect to input data $\boldsymbol{X}$:

$$b(\boldsymbol{X}) = \frac{dL(\boldsymbol{X})}{d\boldsymbol{X}}$$

Due to:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Let $dL(\boldsymbol{X})$ be the differential of likelihood $L(A)$ with respect to input data $\boldsymbol{X}$:

$$dL(\boldsymbol{X}) = dL(QK^T) + dL(V)$$

Where $dL(QK^T)$ is the differential of the likelihood with respect to $\boldsymbol{X}$ given $QK^T$ as intermediate variable and $dL(V)$ is the differential of the likelihood with respect to $\boldsymbol{X}$ given $V$ as intermediate variable, as a convention. Recall that, by the technique of likelihood partition, the partial differential $dl_i$ with respect to $\boldsymbol{q}_i$ is:

$$dl_i = \text{tr}\big((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' d(\boldsymbol{r}_i)\big) = \frac{1}{\sqrt{d_k}}\text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K d\boldsymbol{q}_i)$$

Similarly, the partial differential $dl_i$ with respect to $\boldsymbol{k}_i$ is:

$$dl_i = \text{tr}\big((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' d(\boldsymbol{r}_i)\big) = \frac{1}{\sqrt{d_k}}\text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' Q d\boldsymbol{k}_i)$$

This implies that partial differential $dl_i$ with respect to $\boldsymbol{x}_i$ which is the column vector representing the $i^{\text{th}}$ row of $\boldsymbol{X}$ is:

$$\begin{aligned} dl_i(\boldsymbol{x}_i) &= \frac{1}{\sqrt{d_k}}\text{tr}\big((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' (K d\boldsymbol{q}_i + Q d\boldsymbol{k}_i)\big) \\ &= \frac{1}{\sqrt{d_k}}\text{tr}\Big((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' (K d(\boldsymbol{x}_i^T W^Q)^T + Q d(\boldsymbol{x}_i^T W^K)^T)\Big) \\ &= \frac{1}{\sqrt{d_k}}\text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' K (W^Q)^T d\boldsymbol{x}_i) + \frac{1}{\sqrt{d_k}}\text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' Q (W^K)^T d\boldsymbol{x}_i) \\ &= \frac{1}{\sqrt{d_k}}\text{tr}((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' (K (W^Q)^T + Q (W^K)^T) d\boldsymbol{x}_i) \end{aligned}$$

Obviously, gradient of likelihood $l_i$ with respect to $\boldsymbol{x}_i$ is:

$$\frac{dl_i}{d\boldsymbol{x}_i} = \frac{1}{\sqrt{d_k}}\big((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i' (K (W^Q)^T + Q (W^K)^T)\big)^T$$

Therefore, gradient of likelihood $L(QK^T)$ with respect to $\boldsymbol{X}$ given $QK^T$ is determined by following concatenation:

$$\frac{dL(QK^T)}{d\boldsymbol{X}} = \left(\left(\frac{dl_1}{d\boldsymbol{x}_1}\right)^T, \left(\frac{dl_2}{d\boldsymbol{x}_2}\right)^T, \dots, \left(\frac{dl_m}{d\boldsymbol{x}_m}\right)^T\right)^T$$

Recall that the differential of $L(A)$ with respect to $V$ is:

$$dL = \text{tr}\big((A' - A)^T d(A)\big) = \text{tr}\left((A' - A)^T d\left(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\right)\right)$$

$$= \text{tr}\left((A' - A)^T \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) d(\boldsymbol{X}W^V)\right)$$

This implies that differential of $L(.)$ with respect to $\boldsymbol{X}$ given $A$ is:

$$dL(V) = \text{tr}\left((A' - A)^T \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) d(\boldsymbol{X})W^V\right) = \text{tr}\left(W^V (A' - A)^T \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) d(\boldsymbol{X})\right)$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Obviously, gradient of likelihood $L(A)$ with respect to $\boldsymbol{X}$ is:

$$\frac{dL(A)}{d\boldsymbol{X}} = \left(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\right)^T (A' - A)(W^V)^T$$

Due to:

$$dL(\boldsymbol{X}) = dL(QK^T) + dL(V)$$

The backward bias/error $b(\boldsymbol{X})$ which is the gradient of the likelihood $L(A)$ with respect to input data $\boldsymbol{X}$ is determined as following sum:

$$b(\boldsymbol{X}) = \frac{dL(\boldsymbol{X})}{d\boldsymbol{X}} = \frac{dL(QK^T)}{d\boldsymbol{X}} + \frac{dL(A)}{d\boldsymbol{X}} = \begin{pmatrix} \left(\frac{dl_1}{d\boldsymbol{x}_1}\right)^T \\ \left(\frac{dl_2}{d\boldsymbol{x}_2}\right)^T \\ \vdots \\ \left(\frac{dl_m}{d\boldsymbol{x}_m}\right)^T \end{pmatrix} + \left(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\right)^T (A' - A)(W^V)^T \quad (3.5)$$

Where,

$$\frac{dl_i}{d\boldsymbol{x}_i} = \frac{1}{\sqrt{d_k}}\left((\boldsymbol{a}_i' - \boldsymbol{a}_i)^T V^T \boldsymbol{p}_i'(K(W^Q)^T + Q(W^K)^T)\right)^T$$

Recall that this research focuses on evaluating the hypothesis "whether matrix neural network is the alternative of convolutional neural network", which means to test the preeminence of matrix neural network (MNN) in comparison with convolutional neural network (CNN). Moreover, the hypothesis in this research relates to transformer too because attention, which is the first part of transformer, is totally based on matrix although the second part of transformer, which is feed-forward network (FFN), can be matrix artificial neural network called MNN or vector artificial neural network called vector ANN. As a convention, the transformer whose FFN is MNN is called matrix transformer and the transformer whose FFN is vector ANN is called vector transformer. As a convention, vector ANN is called ANN in short if there is no additional explanation and matrix ANN is MNN. Among many applications related to CNN and transformer, this research focuses on application of image classification and thus, image classification is tested with ANN, MNN, matrix transformer and vector transformer. Therefore, the main hypothesis "*whether MNN is the alternative of CNN*" will lead to the first auxiliary hypothesis "*whether attention is the alternative of CNN too*" when attention plays the role of feature like convolutional layer although the attention-based feature is self-structure or internal structure, actually. Besides, if both MNN and attention are good enough in comparison with CNN although both of them cannot totally replace CNN, then FFN of transformer should be MNN because of the harmonization of both computational cost and accuracy. In other words, the second auxiliary hypothesis "*whether FFN of transformer should be MNN*" (whether matrix transformer is preferred to vector transformer) is tested.

An auxiliary problem needs to be solved in this research, which relates to convergence of ANN is to keep the accuracy stable when such accuracy in image classification decreases due to the ANN convergence. Indeed, although high convergence is a significantly strong point of ANN, it becomes inversely drawback of ANN because image input of ANN classifier is too highly dimensional data whereas class output of ANN classifier is too lowly dimensional data, which causes the side-effect that ANN classifier will focus/converge to a very few of classes. To overcome this drawback, this research proposes a so-called concept of *baseline* where baseline $\boldsymbol{b}$ consists of $n$ basepoints corresponding to $n$ classes. Each basepoint $b_i$ is the average probability that images belong to class $i$.

$$b_i = \frac{1}{N_i}\sum_{k=1}^{N_i} p_i(k) \text{ where } p_i(k) \text{ is the output probability of the classified image } k \text{ on class } i \text{ and such image } k \text{ really belongs to class } i \quad (3.6)$$

Where $N_i$ is the total number of images that really belong to class $i$ in training dataset. Note,

$$\boldsymbol{b} = (b_1, b_2, \ldots, b_n)^T$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Let $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)^T$ be the output probability vector of some image on $n$ classes, which is the output of ANN classifier, indeed. Let $\boldsymbol{d}$ be the probability deviation between baseline $\boldsymbol{b}$ and output $\boldsymbol{p}$.

$$\boldsymbol{d} = \boldsymbol{p} - \boldsymbol{b} = (d_1 = p_1 - b_1, d_2 = p_2 - b_2, \ldots, d_n = p_n - b_n)^T$$

Such image will be classified as class $k$ if $d_k$ is largest or $d_k$ is larger than some threshold. In general, the ANN-based image classification in this research is based on upper level from baseline (not from zero). In advanced research, the probability deviation $\boldsymbol{d}$ is improved by a so-called adjusted-line. For instance, the adjusted-line $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)^T$ consists of $n$ adjusted-points corresponding to $n$ classes. Each basepoint $a_i$ is the average output (average probability) corresponding to class $i$.

$$a_i = \frac{1}{N}\sum_{k=1}^{N} o_i(k) \text{ where } o_i(k) \text{ is the output of the image } k \text{ corresponding to class } i \quad (3.7)$$

Where $N$ is the total number of images in training dataset. Please pay attention that $o_i(k)$ is just the $i^{\text{th}}$ output of ANN classifier whose input is image $k$ but such image $k$ may not belong to class $i$. The advanced probability deviation $\boldsymbol{d}^*$ is wise-multiplication of probability deviation $\boldsymbol{d}$ and soft-max function of adjusted-line $\boldsymbol{a}$.

$$\boldsymbol{d}^* = (d_1^*, d_2^*, \ldots, d_n^*)^T = \boldsymbol{d} \odot \text{softmax}(\boldsymbol{a})$$

Such image will be classified as class $k$ if $d_k^*$ is largest or $d_k^*$ is larger than some threshold. It is easy to recognize that the probability deviation $\boldsymbol{d}$ is weighted by the concentrations (weights) on $C$ classes, which implies the convergence of leaning ANN. Experimental design for hypothesis testing on image classification with ANN classifiers in both matrix form and vector form is described in next section. The implementation source code of matrix neural network, transformer, and convolutional network is available at

https://github.com/ngphloc/ai/tree/main/3_implementation/src/net/ea/ann/mane

## 4. Experimental Design

The experiments in this research focus on image classification by matrix neural network (MNN) called *MNN classifier*, convolutional neural network (CNN) called *CNN classifier*, and transformer called transformer-based classifier (*TB classifier*). The main purpose of this experimental design is to evaluate the preeminence of matrix neural network in comparisons among MNN classifier, CNN classifier, and TB classifier, which is also related to diversity metric and convergence speed although accuracy metric in classification is also concerned because there are many excellent research resulting out very high accuracy in image classification by specific establishment of convolutional layers. In testing dataset, MNN classifier, TB classifier, and CNN classifier are represented by the "*feature*" field whose values are "*weight*", "*trans*", and "*conv*", respectively. Obviously, *weight* feature represents MNN classifier and *trans* feature represents TB classifier whereas *conv* feature represents CNN classifier. The aspect of traditional artificial neural network is that its layers are vectors, in other word, its aspect is vectorization. Therefore, testing dataset has the second field called "*vec*" field whose values are true and false representing whether or not MNN classifier, TB classifier, and CNN classifier are vectorized. When each classifier is vectorized, only its feedforward network (FFN) is vectorized because each classifier always has two parts: 1) the first part for feature extraction is always MNN, attention, or convolutional layers and 2) the second part for classification is always FFN. Note that the first parts of MNN classifier, TB classifier, and CNN classifier are MNN, attentional set (including only attentions), and convolutional set (including only convolutional layers), respectively. Experimental design also aims to two-way analysis of variance (two-way ANOVA) whose groups are *feature* field and *vec* field. Ultimately, the two-way ANOVA aims to test the three hypotheses: 1) whether MNN is the alternative of CNN, which focuses on testing the preeminence of MNN classifier in comparison with TB classifier and CNN classifier via testing the *feature* group, 2) whether attention is the alternative of CNN, which focuses on testing the preeminence of TB classifier in comparison with MNN classifier and CNN classifier via testing the *feature* group, and 3) whether FFN of transformer should be MNN via testing the *vec* group.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Recall that the three classifiers such MNN classifier, TB classifier, and CNN classifier have always two parts such as first part for image feature extraction and second part for feature classification. Therefore, MNN classifier has one MNN and one FFN and TB classifier has one *attentional set* and one FNN whereas CNN classifier has one *convolutional set* and one FFN. Note that *attentional set* is the set of consequential attentions and convolutional set is the network of convolutional layers. The depth of MNN, attentional set, and convolutional set exclude their input layers. The depth of attentional set is the number of its attentions and the depth of convolutional set is the number of its convolutional layers. MNN, attentional set, and convolutional set follow VGG architecture. As a convention, attention in attentional set is called attentional layer so that all elements of MNN, attentional set, and convolutional set are layers. Moreover, attentional set and convolutional set are called *attentional network* and *convolutional network*, as a convention. Therefore, MNN, attentional network, and convolutional network are indicated by the common term which is *feature extraction network* (*FEN*) so that FFN is called *feature classification network* (*FCN*). For distinguishing MNN, attentional network, and convolutional network, MNN is called *matrix FEN* whereas attentional network is called *attentional FEN* and convolutional network is called *convolutional FEN*. A FEN has some blocks and each block is a set of sequential layers. A (FEN) block of matrix FEN, attentional FEN, or convolutional FEN is called *matrix block*, *attentional block*, or *convolutional block*, respectively. A (FEN) layer of matrix block, attentional block, or convolutional block is called *matrix layer*, *attentional layer*, or *convolutional layer*, respectively. A layer of FCN is called *FCN layer*, which is always traditional neural network layer. Every FEN layer is always associated with a so-called *FEN filter* but please distinguish FEN filter from filtering kernel aforementioned. FEN filter of matrix layer is parametric weight matrix as usual, which is called *matrix filter*. FEN filter of attentional layer is attention itself, which is called *attentional filter*. FEN filter of convolutional layer is filtering kernel as usual, which is called *convolutional filter*. However, please pay attention that the last layer of any FEN block is the specific layer called *max-pooling layer* whose filter is aforementioned max-pooling kernel which is called *max-pooling filter*. In other words, max-pooing layer is always associated with max-pooling filter. Matrix filter, attentional filter, and convolutional filter can be called *matrix weight*, *attentional weight*, and *convolutional weight*, respectively because both weight matrix and filtering kernel are matrices except that weight matrix is applied into entire layer whereas filter kernel is applied into every small part (window) of layer.

The experimental design has two important configuration variable such as experimental variable "*base*" and experimental variable "*layer*" with respect to FEN. The current setting of base variable is 2 and there are two values of layer variable such as 2 and 3. The *base* variable (*base*=2) is the divisor of block size, which leads that all classifiers (MNN, TB, CNN) in the experimental design have two FEN blocks. *Layer* variable (2, 3) is the number of layers in each block. Given the number of layers is 2 (*layer*=2), the first block and the second block have two layers and three FEN layers, respectively, plus one max-pooling layer. Exactly, given *layer*=2, the first FEN block has two FEN layers and the second FEN block has three FEN layers and one max-pooling layer so that FEN has five layers. Similarly, given *layer*=3, the first FEN block has two FEN layers and the second FEN block has three FEN layers and one max-pooling layer so that FEN has seven layers. Suppose layer size follows three-dimension size like width x height x depth, all layers in the same block have the same width x height size and such width x height size is decreased according to block ordering by divisor *base*. Therefore, the width x height size of all layers in the same block is considered as width x height block size or *block size*. Recall that the *base* variable (*base*=2) is the divisor of block size so that block size is decreased according to block ordering by divisor *base*. Suppose the first block has size of 32 x 32, then the second block has size of 16 x 16. The depth of all layers in the same block is considered as *block depth*. Block depth is also the count of filters (*filter count*) in every layer in the same block. Block depth is increased according to block ordering by exponential function of variable *layer*. For instance, given *layer*=2 with respect to FEN, the first layer has $2 = 2^1$ filters, the second layer has $2 = 2^1$ filters, the third layer is max-pooling filter layer, the fourth layer has $4 = 2^2$ filters, and the fifth layer has $4 = 2^2$ filters with note that the first layer, the second layer, and the third belong to first block whereas the fourth layer and the fifth layer belong to second block. Given *layer*=3 with respect to FEN, the first layer has $3 = 3^1$ filters,

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



the second layer has $3 = 2^3$ filters, the third layer has $3 = 3^1$ filters, the fourth layer is max-pooling filter layer, the fifth layer has $9 = 3^2$ filters, the sixth layer has $9 = 3^2$ filters, and the seven layer has $9 = 3^2$ filters with note that the first layer, the second layer, the third layer, and the fourth layer belong to first block whereas the fifth layer, the six layer, and the seventh layer belong to second block. Following table summarizes the configuration of FEN for every classifier (MNN, TB, CNN) given *layer*=2.

**Table 4.1.** Two FEN blocks given layer variable is 2.

| Block | Layer | Input dimension | Filter dimension | Filter count | Output dimension |
|---|---|---|---|---|---|
| 1 | *layer*1_1 | 32 x 32 x 3 | 3 x 3 x 3 | 2 | 32 x 32 x 2 |
| | *layer*1_2 | 32 x 32 x 2 | 3 x 3 x 2 | 2 | 32 x 32 x 2 |
| | *maxpool*1 | 32 x 32 x 2 | 2 x 2 | | 16 x 16 x 2 |
| 2 | *layer*2_1 | 16 x 16 x 2 | 3 x 3 x 2 | 4 | 16 x 16 x 4 |
| | *layer*2_2 | 16 x 16 x 4 | 3 x 3 x 4 | 4 | 16 x 16 x 4 |

Following table summarizes the configuration of FEN for every classifier (MNN, TB, CNN) given *layer*=3.

**Table 4. 2.** Two FEN blocks given layer variable is 3.

| Block | Layer | Input dimension | Filter dimension | Filter count | Output dimension |
|---|---|---|---|---|---|
| 1 | *layer*1_1 | 32 x 32 x 3 | 3 x 3 x 3 | 3 | 32 x 32 x 3 |
| | *layer*1_2 | 32 x 32 x 3 | 3 x 3 x 3 | 3 | 32 x 32 x 3 |
| | *layer*1_3 | 32 x 32 x 3 | 3 x 3 x 3 | 3 | 32 x 32 x 3 |
| | *maxpool*1 | 32 x 32 x 3 | 2 x 2 | | 16 x 16 x 3 |
| 2 | *layer*2_1 | 16 x 16 x 3 | 3 x 3 x 3 | 9 | 16 x 16 x 9 |
| | *layer*2_2 | 16 x 16 x 9 | 3 x 3 x 9 | 9 | 16 x 16 x 9 |
| | *layer*2_3 | 16 x 16 x 9 | 3 x 3 x 9 | 9 | 16 x 16 x 9 |

The experimental design establishes FCN to have always two layers where the last layer (output layer) is the set of image classes. Recall that whether or not FCN is vectorized depends on "*vec*" field in dataset (*vec* group in ANOVA). Finally, each classifier (MNN, TB, CNN) including FEN and FCN has 7 layers given *layer*=2 and has 9 layers given *layer*=3.

Recall that the experimental design focuses on evaluating the preeminence of matrix neural network in comparing image classifiers and so, accuracy metric which measures the ratio of correct number of images to total number of all images is not so essential in this research when other deep image classification researches achieve very high accuracy by specific set of convolutional layers. Anyhow accuracy metric which is the ratio of the number of correctly classified images to the total number of images is still measured in this research, as follows:

$$\text{accuracy} = \frac{\text{the number of correctly classified images}}{\text{the total number of images}} \tag{4.1}$$

Because accuracy metric in this research is not larger than good enough level 0.6 (60%), it is necessary to survey the diversity in classification of the aforementioned classifiers when accuracy metric does not reach 0.11 (11%) with note that suppose a classifier does nothing except fixing one resulted class randomized among 10 classes, then such classifier still reaches accuracy 0.1 (10%) because the experimental dataset is CIFAR-10 dataset including 5 training folders and 1 testing folder

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



where each folder has 10000 images grouped in $n$=10 classes and every class has equal number of 1000 images, which means that the probability that an image belongs to a certain class is always 0.1 (10%) when every class in testing dataset containing 10000 images has always 1000 images. Therefore, given $n$=10 classes, the experimental design proposes $n$=10 pairs of precision metrics and recall metrics. For instance, given the $i^{th}$ class, the $i^{th}$ precision is the ratio of the number of correctly classified images in the $i^{th}$ class to the number of classified images in the $i^{th}$ class whereas the $i^{th}$ recall is the ratio of the number of classified images in the $i^{th}$ class to the number of images actually belong to the $i^{th}$ class. The $i^{th}$ precision of $i^{th}$ class is specified as follows:

$$\text{precision}_i = \frac{\text{the number of correctly classified images in } i^{th} \text{ class}}{\text{the number of classified images in } i^{th} \text{ class}}$$

The $i^{th}$ recall of $i^{th}$ class is specified as follows:

$$\text{recall}_i = \frac{\text{the number of classified images in } i^{th} \text{ class}}{\text{the number of images belong to } i^{th} \text{ class}}$$

The $i^{th}$ F1 metric which harmonizes $i^{th}$ precision and $i^{th}$ recall is specified as follows:

$$\text{F1}_i = 2\frac{\text{precision}_i * \text{recall}_i}{\text{precision}_i + \text{recall}_i}$$

The overall precision, recall, and F1 over $n$=10 classes are means of partial $i^{th}$ precisions, partial $i^{th}$ recalls, and $i^{th}$ F1 (s), respectively. These metrics concern classes and diversity in classes.

$$\begin{aligned} \text{precision} &= \frac{1}{n}\sum_{i=1}^{n}\text{precision}_i \\ \text{recall} &= \frac{1}{n}\sum_{i=1}^{n}\text{recall}_i \qquad \text{where } n = 10 \\ \text{F1} &= \frac{1}{n}\sum_{i=1}^{n}\text{F1}_i \end{aligned} \tag{4.2}$$

The internal meanings of precision metric and accuracy metric are very similar but they are still finely different. Within the same degree of accuracy, precision metric will be higher than accuracy metric if classifier obtain good enough diversity. For instance, the do-nothing classifier does nothing except fixing one resulted class randomized among $n$=10 classes always obtain accuracy metric 0.1 (10%) but its precision metric is always smaller than precision metric of the classifier that reaches the same accuracy metric 0.1 but covers more than one class. Although classification diversity is hidden under precision metric, recall metric, and F1 metric, the so-called class coverage is proposed to illustrate visually the classification diversity. For instance, class coverage denoted cc is the ratio of the number of classified classes to the number of total classes with note that a classified class is the class whose corresponding recall is positive. In other words, if any image is classified into a class by classifier, such class becomes classified class.

$$\text{cc} = \frac{\text{the number of classified classes}}{\text{the number of total classes}} \tag{4.3}$$

In general, there are 4 main testing metrics such as accuracy, precision, recall, and F1 and 1 additional metric such as class coverage, in which F1 is the final metric that evaluates preeminence of every classifier. The second important metric is accuracy metric because deep neural network turns forwards some convergence so that it often focuses on some certain class among $n$=10 classes, which decreases diversity but increases accuracy degree. Computational convergence is strong point of artificial neural network (ANN) but unfortunately, it becomes inversely drawback in image classification because image input is high-dimension data whereas class output is low-dimension data. If class output is large as corpus dictionary in natural language processing (NLP), such computational convergence aspect of ANN become much more strong point of deep neural network classifier. Other researches focusing on obtaining highly accurate image classification always try to

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



increase the accuracy metric and keep diversity stable in parallel by setting specific configuration of many enough filters and convolutional layers.

## 5. Results and Discussions

The experimental dataset is CIFAR-10 dataset including 5 training folders and 1 testing folder where each folder has 10000 images grouped in 10 classes and every class has equal number of 1000 images. Given experimental variable "layer" is 2, CIFAR-10 dataset is tested 10 times for 3 values ("weight", "trans", "conv") of *feature* group and 2 values ("true", "false") of *vec* group in order to derive 60 records of ANOVA dataset whose groups are "vec" and "feature" and whose treatments are "accuracy", "precision", "recall", and "F1" corresponding to accuracy metric, precision metric, recall metric, and F1 metric aforementioned. Given experimental variable "layer" is 3, CIFAR-10 dataset is tested again 10 times for 3 values ("weight", "trans", "conv") of *feature* group and 2 values ("true", "false") of *vec* group in order to derive other 60 records of ANOVA dataset. Following table illustrates 120 records of full ANOVA dataset.

**Table 5.1.** Illustration of ANOVA dataset.

| | layer | *vec* | *feature* | accuracy | precision | recall | F1 | cc |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | false | weight | 0.0826 | 0.0980 | 0.7623 | 0.1678 | 1.0000 |
| 2 | | false | trans | 0.0727 | 0.0841 | 0.7903 | 0.1350 | 1.0000 |
| 3 | | false | conv | 0.1000 | 0.0400 | 0.4000 | 0.0727 | 0.4000 |
| 4 | | true | weight | 0.0831 | 0.0872 | 0.7244 | 0.1524 | 1.0000 |
| 5 | | true | trans | 0.0995 | 0.1028 | 0.7543 | 0.1745 | 1.0000 |
| 6 | | true | conv | 0.0995 | 0.0896 | 0.4219 | 0.1090 | 0.9000 |
| | | … | … | … | … | … | … | … |
| 55 | | false | weight | 0.0782 | 0.0844 | 0.7532 | 0.1480 | 1.0000 |
| 56 | | false | trans | 0.0869 | 0.0861 | 0.8044 | 0.1502 | 1.0000 |
| 57 | | false | conv | 0.1018 | 0.1484 | 0.3899 | 0.1212 | 1.0000 |
| 58 | | true | weight | 0.1012 | 0.1078 | 0.8043 | 0.1843 | 1.0000 |
| 59 | | true | trans | 0.0866 | 0.0995 | 0.7281 | 0.1537 | 1.0000 |
| 60 | | true | conv | 0.0828 | 0.0838 | 0.7532 | 0.1476 | 1.0000 |
| | | | | | | | | |
| 61 | 3 | false | weight | 0.1010 | 0.1110 | 0.8207 | 0.1866 | 1.0000 |
| 62 | | false | trans | 0.0783 | 0.0815 | 0.8688 | 0.1450 | 1.0000 |
| 63 | | false | conv | 0.1000 | 0.0200 | 0.2000 | 0.0364 | 0.2000 |
| 64 | | true | weight | 0.0794 | 0.0833 | 0.7425 | 0.1466 | 1.0000 |
| 65 | | true | trans | 0.0800 | 0.0850 | 0.7450 | 0.1443 | 1.0000 |
| 66 | | true | conv | 0.1000 | 0.0300 | 0.3000 | 0.0545 | 0.3000 |
| | | … | … | … | … | … | … | … |
| 115 | | false | weight | 0.0712 | 0.0701 | 0.7380 | 0.1177 | 1.0000 |
| 116 | | false | trans | 0.0756 | 0.0872 | 0.7852 | 0.1446 | 1.0000 |
| 117 | | false | conv | 0.1000 | 0.0300 | 0.3000 | 0.0545 | 0.3000 |
| 118 | | true | weight | 0.0814 | 0.0889 | 0.8168 | 0.1480 | 1.0000 |
| 119 | | true | trans | 0.0858 | 0.0838 | 0.8438 | 0.1479 | 1.0000 |

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 120 | | true | conv | 0.0969 | 0.0924 | 0.5054 | 0.1203 | 1.0000 |

The ANOVA dataset is available at https://github.com/ngphloc/ai/blob/main/4_testing/mac/2026.01.22/cifar10-lr0.1-max-anova.csv

Recall that the two-way ANOVA aims to test the three hypotheses on the ANOVA dataset of size 120: 1) whether MNN is the alternative of CNN, which focuses on testing the preeminence of MNN classifier in comparison with TB classifier and CNN classifier via testing the *feature* group, 2) whether attention is the alternative of CNN, which focuses on testing the preeminence of TB classifier in comparison with MNN classifier and CNN classifier via testing the *feature* group, and 3) whether FFN of transformer should be MNN via testing the *vec* group. The research applies Gemini 2025 into making ANOVA test. Moreover, only baseline is applied into improving classification task and adjusted-line is not applied.

Firstly, the two-way ANOVA test focuses on groups "vec" and "feature" against treatments "accuracy" and "F1" in which accuracy metric measures the entire accuracy of classifiers over dataset whereas F1 metric makes the compromise between diversity and accuracy. The following table shows the ANOVA results for accuracy metric, which indicates how groups "vec" and "feature" affect on treatment "accuracy" (Gemini 2025).

**Table 5.2.** ANOVA results for accuracy metric.

| Source | SS | DF | F-statistic | P-value | Significant |
|---|---|---|---|---|---|
| vec | 6.31e-07 | 1 | 0.0121 | 0.9125 | No |
| feature | 0.00677 | 2 | 65.0845 | < 0.0001 | Yes |
| vec:feature | 6.87e-05 | 2 | 0.6604 | 0.5186 | No |
| Residual | 0.00593 | 114 | | | |

Note, SS is the abbreviation of *sum of squares* and DF is the abbreviation of *degree of freedom* whereas the source "vec:feature" represents the *interaction effect* between group "vec" and group "feature". From the result table above, group "feature" affects highly significantly on treatment "accuracy" in comparison with group "vec" because the F-statistic of group "feature" (65.0845) is highly larger than the F-statistic of group "vec" (0.0121) with very small P-value (< 0.0001). Moreover, there is no significant interaction between group "vec" and group "feature" because the F-statistic of the interaction effect "vec:feature" (0.6604) is small and its P-value (0.5186) is too larger than 0.05.

The following table shows the ANOVA results for F1 metric, which indicates how groups "vec" and "feature" affect on treatment "F1" (Gemini 2025).

**Table 5.3.** ANOVA results for F1 metric.

| Source | SS | DF | F-statistic | P-value | Significant |
|---|---|---|---|---|---|
| vec | 0.00074 | 1 | 1.7781 | 0.1850 | No |
| feature | 0.09846 | 2 | 118.3605 | < 0.0001 | Yes |
| vec:feature | 0.00139 | 2 | 1.6758 | 0.1917 | No |
| Residual | 0.04741 | 114 | | | |

From the result table above, group "feature" affects highly significantly on treatment "F1" in comparison with group "vec" because the F-statistic of group "feature" (118.3605) is highly larger than the F-statistic of group "vec" (1.7781) with very small P-value (< 0.0001). Moreover, there is no significant interaction between group "vec" and group "feature" because the F-statistic of the interaction effect "vec:feature" (1.6758) is small and its P-value (0.1917) is too larger than 0.05.

The two-way ANOVA test focuses again on groups "vec" and "feature" against treatments "precision" and "recall" in which precision metric leans forwards accuracy and recall metric leans

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



forward diversity with note F1 is the compromiser of precision and recall. The following table shows the ANOVA results for precision metric, which indicates how groups “vec” and “feature” affect on treatment “precision” (Gemini 2025).

**Table 5.4.** ANOVA results for precision metric.

| Source | SS | DF | F-statistic | P-value | Significant |
|---|---|---|---|---|---|
| vec | 4.8e-5 | 1 | 0.1180 | 0.7318 | No |
| feature | 0.01908 | 2 | 23.3842 | < 0.0001 | Yes |
| vec:feature | 0.00024 | 2 | 0.2909 | 0.7481 | No |
| Residual | 0.0465 | 114 | | | |

From the result table above, group “feature” affects highly significantly on treatment “precision” in comparison with group “vec” because the F-statistic of group “feature” (23.3842) is highly larger than the F-statistic of group “vec” (0.118) with very small P-value (< 0.0001). Moreover, there is no significant interaction between group “vec” and group “feature” because the F-statistic of the interaction effect “vec:feature” (0.2909) is small and its P-value (0.7481) is too larger than 0.05.

The following table shows the ANOVA results for recall metric, which indicates how groups “vec” and “feature” affect on treatment “recall” (Gemini 2025).

**Table 5.5.** ANOVA results for recall metric.

| Source | SS | DF | F-statistic | P-value | Significant |
|---|---|---|---|---|---|
| vec | 0.00809 | 1 | 1.5663 | 0.2133 | No |
| feature | 3.25747 | 2 | 315.5278 | < 0.0001 | Yes |
| vec:feature | 0.01207 | 2 | 1.1690 | 0.3144 | No |
| Residual | 0.58846 | 114 | | | |

From the result table above, group “feature” affects highly significantly on treatment “recall” in comparison with group “vec” because the F-statistic of group “feature” (315.5278) is highly larger than the F-statistic of group “vec” (1.5663) with very small P-value (< 0.0001). Moreover, there is no significant interaction between group “vec” and group “feature” because the F-statistic of the interaction effect “vec:feature” (1.1690) is small and its P-value (0.3144) is too larger than 0.05.

In general, within metrics accuracy, F1, precision, and recall, for the third hypothesis “FFN of transformer should be MNN (vector FEC is better than matrix FEC)”, it is possible to conclude that FFN/FEC of MNN classifier and TB classifier should be matrix because it is not necessary to vectorize FEC to obtain better classification while vectorized FEC (vector FEC) consumes much more computational resources. Exactly, there is significant evidence to reject null hypothesis of the third hypothesis or there is significant evidence to support the third hypothesis.

Although the experiment focuses on testing the three hypotheses by ANOVA tests on significant differences, it is necessary to summarize means of metrics “accuracy”, “F1”, “precision”, and “recall” with respect to groups “vec” and “feature” (Gemini 2025).

**Table 5.6.** Means of metrics “accuracy”, “F1”, “precision”, and “recall”.

| vec | feature | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|---|
| false | conv | 0.0991 | 0.0845 | 0.0614 | 0.4154 |
| false | trans | 0.0834 | 0.1508 | 0.0881 | 0.7751 |
| false | weight | 0.0816 | 0.1530 | 0.0887 | 0.7963 |
| true | conv | 0.0980 | 0.0983 | 0.0631 | 0.4601 |

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



| | | | | | |
|---|---|---|---|---|---|
| true | trans | 0.0857 | 0.1547 | 0.0926 | 0.7790 |
| true | weight | 0.0809 | 0.1502 | 0.0863 | 0.7970 |
| | | | | | |
| | conv | 0.0986 | 0.0914 | 0.0623 | 0.4377 |
| | trans | 0.0846 | 0.1527 | 0.0903 | 0.7771 |
| | weight | 0.0812 | 0.1516 | 0.0875 | 0.7966 |
| | | | | | |
| false | | 0.0880 | 0.1294 | 0.0794 | 0.6623 |
| true | | 0.0882 | 0.1344 | 0.0807 | 0.6787 |

Because feature "weight", "trans", and "conv" represent classifiers MNN, TB, and CNN, respectively, for the result table above, CNN classifier (0.0986) is better than both TB classifier (0.0846) and MNN classifier (0.0812) in accuracy metric but both MNN classifier and TB classifier are better than CNN classifier in remaining metrics such as F1, precision, and recall. Although both F1 and precision measure the accurate aspect in classification task, they reflect the diversity too. Therefore, *CNN classifier is the best in accuracy* because accuracy metric is the pure metric to measure the ratio of correct number of images to total number of all images in classifying entire dataset. Recall metric is the intermediate metric between accuracy and diversity but it leans forwards the diversity, which leads that TB classifier within context of transformer can harmonize MNN classifier and CNN classifier because recall metric of TB classifier (0.7771) is between recall metrics of CNN classifier (0.4377) and MNN classifier (0.7966). Moreover, accuracy metric of TB classifier (0.0846) is between accuracy metrics of CNN classifier (0.0986) and MNN classifier (0.0812). Especially, F1 metric of TB classifier (0.1527) is best and precision metric of TB classifier (0.0903) is best, which concludes that *TB classifier can be the preeminent one in compromising all factors*. Although there is significant evidence to support the third hypothesis which means it is not necessary to vectorize FEC to obtain better classification, vector FEC (accuracy=0.0880, F1=0.1294, precision=0.0794, recall=0.6623) is always slightly better than matrix FEC (accuracy=0.0882, F1=0.1344, precision=0.0807, recall=0.6787).

Secondly, because it is concluded that group "feature" is the most significant factor on accuracy metric and F1 metric, it is necessary to make Post-Hoc Tukey HSD (Honestly Significant Difference) test on values "weight", "trans", "conv" of independent variable "feature" in order to test the first hypothesis and second hypothesis. The following table shows the Post-Hoc Tukey HSD results for feature group within accuracy metric, which indicates which specific values ("weight", "trans", "conv") are significantly different among them with context of treatment "accuracy" (Gemini 2025).

**Table 5.7.** Post-Hoc Tukey HSD results for feature group within accuracy metric.

| Treatment 1 | Treatment 2 | Mean Difference | Adjusted P-value | Significant |
|---|---|---|---|---|
| conv | trans | –0.0140 | < 0.001 | Yes (conv > trans) |
| conv | weight | –0.0173 | < 0.001 | Yes (conv > weight) |
| trans | weight | –0.0034 | 0.0932 | No |

For the result table above, the feature "conv" is significantly much larger than features "weight" and "trans" in accuracy metric because its absolute mean differences with respect to feature "weight" and feature "trans" are larger (|–0.0173|, |–0.014|) with P-values (< 0.001) are too smaller than 0.05. Because these differences are negative, the feature "conv" is significantly much better than features "weight" and "trans". Conversely, there is no significantly difference between feature "weight" and feature "trans" because the P-value (0.0932) given their absolute mean difference (|–0.0034|) is larger than 0.05. Therefore, there is significant evidence to reject the first hypothesis "MNN is the alternative

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



of CNN" and there is no significant evidence to support the second hypothesis "attention is the alternative of CNN", within accuracy metric. Of course, CNN is better than both MNN and attention within accuracy metric. In other words, within accuracy metric, MNN should not be the alternative of CNN and it is not necessary to replace MNN by attention but attention is slightly better than MNN because the mean difference between feature "weight" and feature "trans" is negative (weight – trans) = –0.0034.

The following table shows the Post-Hoc Tukey HSD results for feature group within F1 metric, which indicates which specific values ("weight", "trans", "conv") are significantly different among them with context of treatment "F1" (Gemini 2025).

**Table 5.8.** Post-Hoc Tukey HSD results for feature group within F1 metric.

| Treatment 1 | Treatment 2 | Mean Difference | Adjusted P-value | Significant |
|---|---|---|---|---|
| conv | trans | 0.0613 | < 0.001 | Yes (trans > conv) |
| conv | weight | 0.0602 | < 0.001 | Yes (weight > conv) |
| trans | weight | –0.0011 | 0.9687 | No |

For the result table above, the feature "conv" is significantly much larger than features "weight" and "trans" in F1 metric because its absolute mean differences with respect to feature "weight" and feature "trans" are larger |0.0602|, (|0.0613|) with P-values (< 0.001) are too smaller than 0.05. Because these differences are positive, the feature "conv" is significantly much worse than features "weight" and "trans". Conversely, there is no significantly difference between feature "weight" and feature "trans" because the P-value (0.9687) given their absolute mean difference (|–0.0011|) is larger than 0.05. Therefore, there is significant evidence to support the first hypothesis "MNN is the alternative of CNN" and there is no significant evidence to support the second hypothesis "attention is the alternative of CNN", within F1 metric. Of course, CNN is worse than both MNN and attention within F1 metric. In other words, within F1 metric, MNN should be the alternative of CNN and it is not necessary to replace MNN by attention but attention is slightly better than MNN because the mean difference between feature "weight" and feature "trans" is negative (weight – trans) = –0.0011.

The following table shows the Post-Hoc Tukey HSD results for feature group within precision metric, which indicates which specific values ("weight", "trans", "conv") are significantly different among them with context of treatment "precision" (Gemini 2025).

**Table 5.9.** Post-Hoc Tukey HSD results for feature group within precision metric.

| Treatment 1 | Treatment 2 | Mean Difference | Adjusted P-value | Significant |
|---|---|---|---|---|
| conv | trans | 0.0280 | < 0.001 | Yes (trans > conv) |
| conv | weight | 0.0252 | < 0.001 | Yes (weight > conv) |
| trans | weight | –0.0028 | 0.806 | No |

For the result table above, the feature "conv" is significantly much larger than features "weight" and "trans" in precision metric because its absolute mean differences with respect to feature "weight" and feature "trans" are larger (|0.0252|, |0.0280|) with P-values (< 0.001) are too smaller than 0.05. Because these differences are positive, the feature "conv" is significantly much worse than features "weight" and "trans". Conversely, there is no significantly difference between feature "weight" and feature "trans" because the P-value (0.806) given their absolute mean difference (|–0.0028|) is larger than 0.05. Therefore, there is significant evidence to support the first hypothesis "MNN is the alternative of CNN" and there is no significant evidence to support the second hypothesis "attention is the alternative of CNN", within precision metric. Of course, CNN is worse than both MNN and

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



attention within precision metric. In other words, within precision metric, MNN should be the alternative of CNN and it is not necessary to replace MNN by attention but attention is slightly better than MNN because the mean difference between feature "weight" and feature "trans" is negative (weight – trans) = –0.0028.

The following table shows the Post-Hoc Tukey HSD results for feature group within recall metric, which indicates which specific values ("weight", "trans", "conv") are significantly different among them with context of treatment "recall" (Gemini 2025).

**Table 5.10.** Post-Hoc Tukey HSD results for feature group within recall metric.

| Treatment 1 | Treatment 2 | Mean Difference | Adjusted P-value | Significant |
|---|---|---|---|---|
| conv | trans | 0.3393 | < 0.001 | Yes (trans > conv) |
| conv | weight | 0.3589 | < 0.001 | Yes (weight > conv) |
| trans | weight | 0.0196 | 0.4481 | No |

For the result table above, the feature "conv" is significantly much larger than features "weight" and "trans" in recall metric because its absolute mean differences with respect to feature "weight" and feature "trans" are larger (|0.3589|, |0.3393|) with P-values (< 0.001) are too smaller than 0.05. Because these differences are positive, the feature "conv" is significantly much worse than features "weight" and "trans". Conversely, there is no significantly difference between feature "weight" and feature "trans" because the P-value (0.4481) given their absolute mean difference (|0.0196|) is larger than 0.05. Therefore, there is significant evidence to support the first hypothesis "MNN is the alternative of CNN" and there is no significant evidence to support the second hypothesis "attention is the alternative of CNN", within recall metric. Of course, CNN is worse than both MNN and attention within recall metric. In other words, within recall metric, MNN should be the alternative of CNN and it is not necessary to replace MNN by attention but attention is slightly worse than MNN because the mean difference between feature "weight" and feature "trans" is positive (weight – trans) = 0.0196.

In general, following table summarizes results of hypothesis testing on the three hypotheses: 1) MNN is the alternative of CNN, 2) attention is the alternative of CNN, and 3) FFN of transformer should be MNN.

**Table 5.11.** Results of hypothesis testing.

| | Hypothesis 1 | Hypothesis 2 | Hypothesis 3 |
|---|---|---|---|
| accuracy | *Not supported*. CNN is the best. | *Not supported*. TB is slightly better than MNN but worse than CNN. | *Supported*. Vector FEC is slightly better than matrix FEC. |
| F1 | *Supported*. CNN is worse than both TB and MNN. | *Not supported*. TB is slightly better than both MNN and CNN. | *Supported*. Vector FEC is slightly better than matrix FEC. |
| precision | *Supported*. CNN is worse than both TB and MNN. | *Not supported*. TB is slightly better than both MNN and CNN. | *Supported*. Vector FEC is slightly better than matrix FEC. |
| recall | *Supported*. CNN is worse than both TB and MNN. | *Not supported*. TB is slightly worse than MNN and better than CNN. | *Supported*. Vector FEC is slightly better than matrix FEC. |

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



Please pay attention that supporting is not acceptance, which means that supporting is slightly stronger than not rejecting. The third hypothesis "MNN is the alternative of CNN" is not supported by only accuracy metric but accuracy metric is the most important metric about accuracy aspect of classification method and so, CNN classifier is the best classifier in accuracy. The second hypothesis "attention is the alternative of CNN" is always not supported but TB is slightly better than MNN (except recall metric) and better than CNN. Moreover, CNN classifier is worse than both TB classifier and MNN classifier in F1 metric which is the compromiser of accuracy and diversity in image classification. This implies that TB classifier is the best classifier in harmonizing all factors (metrics). The third hypothesis "FFN of transformer should be MNN" is always supported although vector FEC is slightly better than matrix FEC, which implies that it is possible to replace traditional neural network (vector ANN) by matrix neural network (MNN) with note that vector ANN consumes much more computational resources due to boom problem of a huge number of parameters. Moreover, attention of transformer can overcome drawback of vector ANN in accuracy.

## 6. Conclusions

In general, although it is not asserted that matrix neural network (MNN) is the alternative of convolutional neural network (CNN), transformer is surely preeminence approach in deep learning because of three reasons: 1) its attention is an excellent mechanism to extract internal structure / relationship / feature of any kind of high-dimension data according to self-supervised learning, 2) transformer is different from the methodology of traditional ANN consisting of simple layers equipped by parametric weights when attention of transformer concerns both explicit information via value matrix and implicit information via query matric and key matrix, and 3) ANN and attention are incorporated into transformer so that deep learning still always takes advantages of excellent aspects of ANN. Again, although it is not asserted that MNN is the alternative of CNN, it is not strict to implement FFN of transformer by vector ANN. Therefore, it is totally possible that MNN and attention are incorporated into transformer with note that MNN is the ANN implemented by parametric weight matrices in order to reduce significantly computational resources so that its accuracy (performance) is not decreased significantly. Attentional mechanism built in transformer can alleviate the drawback of MNN in performance, especially, in case of huge data. Moreover, for high-dimension data whose dimension is larger than 2, MNN is a possible technique to process such tensor-based data.

## References


1. Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., & Miao, Y. (2021, November 21). Review of Image Classification Algorithms Based on Convolutional Neural Networks. (S. Paheding, Z. Alom, M. Maimaitijiang, & M. Maimaitiyiming, Eds.) *Remote Sensing, 13*(22), 4712-4763. doi:10.3390/rs13224712
2. D'Ascoli, S., Touvron, H., Leavitt, M. L., Morcos, A. S., Biroli, G., & Sagun, L. (2021). ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases. *Proceedings of the 38th International Conference on Machine Learning. 139*, pp. 2286-2296. PMLR. Retrieved from https://proceedings.mlr.press/v139/d-ascoli21a.html
3. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., . . . Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv Computer Vision and Pattern Recognition*. doi:10.48550/arXiv.2010.11929
4. Gao, J., Guo, Y., & Wang, Z. (2016, December 9). Matrix Neural Networks. *arXiv Preprints*, 1-28. doi:10.48550/arXiv.1601.03805
5. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)* (pp. 770-778). IEEE. doi:10.1109/CVPR.2016.90
6. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. (F. Pereira, C. Burges, L. Bottou, & K. Weinberger, Eds.) *Advances in Neural Information Processing Systems 25 (NIPS 2012)*. doi:10.1145/3065386


© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 April 2026 doi:10.20944/preprints202604.0170.v1



7. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998, November 30). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE, 86*(11), 2278-2324. doi:10.1109/5.726791
8. Lin, M., Chen, Q., & Yan, S. (2014, March 4). Network In Network. *arXiv Neural and Evolutionary Computing*. doi:10.48550/arXiv.1312.4400
9. Simonyan, K., & Zisserman, A. (2015, April 10). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv Computer Vision and Pattern Recognition*. doi:10.48550/arXiv.1409.1556
10. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going Deeper With Convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)* (pp. 1-9). IEEE. Retrieved from https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html
11. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention Is All You Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, & S. Vishwanathan (Ed.), *Advances in Neural Information Processing Systems (NIPS 2017). 30.* Long Beach: NeurIPS. Retrieved from https://arxiv.org/abs/1706.03762
12. Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., . . . Tang, X. (2017, April 23). Residual Attention Network for Image Classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)* (pp. 3156-3164). IEEE. Retrieved from https://openaccess.thecvf.com/content_cvpr_2017/html/Wang_Residual_Attention_Network_CVPR_2017_paper.html

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.