(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 November 2020 (12.11.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/225772 A1

(51) International Patent Classification:
G06N 3/02 (2006.01) *G06N 3/08* (2006.01)
G06N 3/04 (2006.01)

(21) International Application Number:
PCT/IB2020/054350

(22) International Filing Date:
07 May 2020 (07.05.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/844,472 07 May 2019 (07.05.2019) US

(71) Applicant: **IMAGIA CYBERNETICS INC.** [CA/CA]; 100-6650 rue St-Urbain, Montral, Québec H2S 3G9 (CA).

(72) Inventors: **VARNO, Farsheed**; 100-6650 rue St-Urbain, Montreal, Québec H2S 3G9 (CA). **SOLEIMANI, Behrouz Haji**; 100-6650 rue St-Urbain, Montreal, Québec H2S 3G9 (CA). **SAGHAYI, Marzie**; 100-6650 rue St-Urbain, Montreal, Québec H2S 3G9 (CA). **DI JORIO, Lisa**; 100-6650 rue St-Urbain, Montreal, Québec H2S 3G9 (CA). **MATWIN, Stan**; 100-6650 rue St-Urbain, Montreal, Québec H2S 3G9 (CA).

(74) Agent: **FASKEN MARTINEAU DUMOULIN LLP**; 800 Square-Victoria, Suite 3500, Montreal, Québec H4Z 1E9 (CA).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV,

(54) Title: METHOD AND SYSTEM FOR INITIALIZING A NEURAL NETWORK

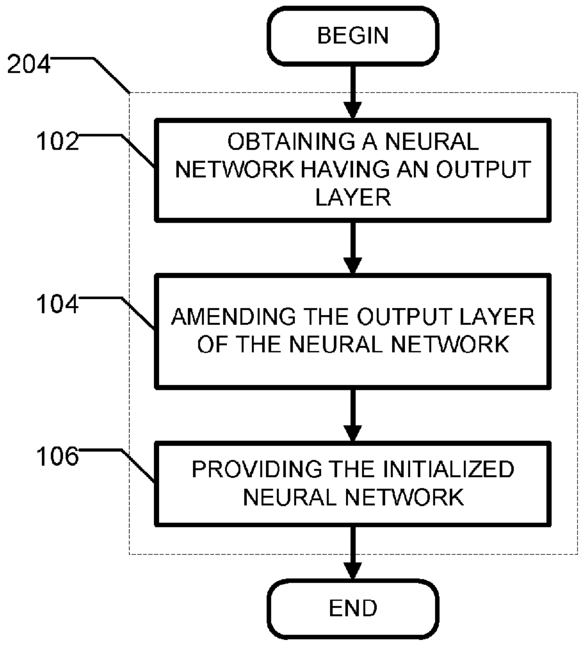

BEGIN
204
102
OBTAINING A NEURAL NETWORK HAVING AN OUTPUT LAYER
104
AMENDING THE OUTPUT LAYER OF THE NEURAL NETWORK
106
PROVIDING THE INITIALIZED NEURAL NETWORK
END

FIG. 3

(57) **Abstract:** A method and a system are disclosed for initializing a pre-trained neural network, the method comprising obtaining a pre-trained neural network having an output layer, amending the output layer of the pre-trained neural network, wherein the amending comprises updating each weight of the output layer according to a function that maximizes the entropy of the output classes probability, wherein the function depends on a parameter controlling a proportion of error of the output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

WO 2020/225772 A1

[Continued on next page]

MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *in black and white; the international application as filed contained color or greyscale and is available for download from PATENTSCOPE*

METHOD AND SYSTEM FOR INITIALIZING A NEURAL NETWORK

RELATED APPLICATION

The present application claims priority from US provisional application No. 62/844,472 filed on May 7, 2019, the content of which is incorporated herein in its entirety.

FIELD

One or more embodiments of the invention pertain to artificial intelligence. More precisely, one or more embodiment of the invention pertain to a method and a system for initializing a neural network.

BACKGROUND

Artificial Neural Networks (ANNs) have shown a great capacity in learning complicated tasks and have become the first contender to solve many problems in the machine learning community. However, a large training dataset is a key pre-requisite for these networks to achieve good performances. This limitation has opened a new chapter in neural network research, which attempts to make learning possible with limited amounts of data. So far, one of the most widely used techniques to cope with such hindrance is the initialization of parameters based on the prior knowledge obtained from already trained models.

To adapt a pre-trained model to a new task, usually task-specific, extraneous and random parameters are transplanted to a meaningful set of representations, resulting in a heterogeneous model [1,6,17,19]. Training these unassociated modules together may contaminate the genuinely learned representations and significantly degrade the maximum transferable knowledge. Current fine-tuning techniques slow down the training process to compensate for this knowledge leak [17] which undermines fast convergence of a model that suffers from data-shortage.

Previous studies on parameters initialization of ANNs [2, 7, 8, 14] focus on preserving the variance, or other statistics, of the flowing data along the depth. This stabilizes the model and makes training deeper networks possible. Arpit and Bengio [2] recently showed that the initialization introduced in [8] is the optimal one for a ReLU network trained from scratch. [2] recommended to use the fan-out mode which preserves the variance of the back-propagated error along the depth. He et al. [8], inconsistently exempted the last layer of the models used for their experiments from the distribution for weights that they have recommended. This layer's distribution is stated to be found experimentally and no justification has been provided for its outcome. Such a strategy could be traced down to the earlier practices in constructing deep neural networks [21].

Recent studies on transfer learning use variance preserving initialization techniques for fine-tuning [17, 20]. However, it can be shown that using such techniques, initially contaminates the transferred knowledge, resulting in unguided modification of valuable transferred features.

Careful initialization is also an inevitable part of self-normalized neural networks introduced in [14]. These networks use Scaled Exponential Units (SELUs) as their activation function.

In feature extraction [3, 4], the pre-trained features are only used in inference mode and corresponding parameters remain intact during training. This protects the learned representations from undesired contamination but also prevents the required new task-specific features to be learned.

Fine-tuning [6] lets the pre-trained features and augmented parameters learn the target task together. Fine-tuning usually performs better than feature extraction and training from scratch with random initialization [6]. However, the pre-trained features are substantially contaminated due to noise flowing from random layers to the loss and from there, back-propagated toward the features.

There is a need for at least one of a method and a system that will overcome at least one of the above-identified drawbacks.

SUMMARY

According to a broad aspect, there is disclosed a method for initializing a pre-trained neural network, the method comprising obtaining a pre-trained neural network having an output layer, amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

In accordance with one or more embodiments, the amending of the output layer of the pre-trained neural network further comprises z-normalizing features located right before the output layer prior updating each weight.

In accordance with one or more embodiments, the pre-trained neural network uses softmax logit in the output layer.

In accordance with one or more embodiments, there is disclosed a method for training a pre-trained neural network, the method comprising obtaining a pre-trained neural network to train; obtaining a dataset suitable for said training; initializing the pre-trained neural network using the method disclosed above; training the initialized pre-trained neural network using the obtained dataset; and providing the trained neural network.

In accordance with one or more embodiments, the training is a federated learning method.

In accordance with one or more embodiments, the training is a meta-learning method.

In accordance with one or more embodiments the training is a distributed machine learning method.

In accordance with one or more embodiments the training is a network architecture search using said pre-trained neural network as a seed.

In accordance with one or more embodiments, the pre-trained neural network comprises a generative adversarial network, wherein said initializing of the pre-trained neural network using the method disclosed above is performed at the discriminator.

In accordance with a broad aspect, there is disclosed method for training a neural network through federated learning, the method comprising obtaining a shared neural network to train; obtaining at least two datasets suitable for said federated learning, each of the at least two datasets for training a corresponding decentralized training unit; each decentralized training unit performing a first round of training using a corresponding dataset; for each subsequent round of training: each decentralized training unit initializing the shared neural network using the method disclosed above, each decentralized training unit training the initialized shared neural network using the corresponding dataset, globally federating the learning from all decentralized training units to a resulting global shared neural network, and until the global shared neural network converges to a good global model, providing the corresponding global shared neural network to the decentralized training units as the new shared neural network; and providing the trained shared neural network.

In accordance with a broad aspect, there is disclosed a method for training a neural network using a reptile meta-learning method, the method comprising obtaining a neural network to train; obtaining a dataset suitable for said reptile meta-learning method; for each iteration of the reptile meta-learning method: initializing the neural network using the method as disclosed above for each task sampled, and training the initialized neural network for said corresponding sampled task using the obtained dataset; and providing the trained neural network.

In accordance with one or more embodiments, the training of the initialized pre-trained neural network comprising training the initialized pre-trained neural network using a first training batch of the obtained dataset, wherein the first training batch is smaller than a number of features fed to said last layer of said initialized pre-trained neural network.

In accordance with a broad aspect, there is disclosed a method for using a pre-trained neural network trained in accordance with a method disclosed above.

In accordance with a broad aspect, there is disclosed a computer comprising a central processing unit; a graphics processing unit; a communication port; a memory unit comprising an application for initializing a pre-trained neural network, the application comprising: instructions for obtaining a pre-trained neural network having an output layer, instructions for amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and instructions for providing the initialized pre-trained neural network.

In accordance with a broad aspect, there is disclosed a computer program comprising computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising obtaining a pre-trained neural network having an output layer, amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

In accordance with a broad aspect, there is disclosed a non-transitory computer readable storage medium for storing computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising obtaining a pre-trained neural network having an output layer, amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

In accordance with a broad aspect, there is disclosed a method for initializing a neural network, the method comprising obtaining a neural network having an output layer, amending the output layer of the neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized neural network.

An advantage of one or more embodiments of the method disclosed is that they significantly decrease the initial noise that is back-propagated from randomly-initialized parameters toward layers that contain the transferred knowledge. As a consequence, a processing device used for training a neural network according to one or more embodiments of the method disclosed herein will use less resources for training a neural network resulting in more available resources available to complete other tasks. Moreover, one or more embodiments of the method disclosed herein may contribute to better performances overall compared to other traditional training methods, and does significantly contribute to bettering performances in training cases including a small number of training steps compared to other traditional training methods which is particularly useful to evaluate model potential during architecture search & design. Also, one or more embodiments of the method disclosed herein may contribute to decrease the negative impact of catastrophic forgetting, that may occur during model training across tasks, as it limits the impact of noise propagation.

In fact, experiments show that models trained by one or more embodiments of the method disclosed herein learn substantially faster than those using prior art fine-tuning or even more complicated tricks such as warm up [17].

Another advantage of one or more embodiments of the method disclosed is that they are easy to implement and can be beneficially applied to any pre-trained neural network that estimates output probabilities using softmax logits in one embodiment. As a consequence, a benefit is a broad applicability and integration across various deep learning frameworks offered by various vendors such as Google TensorFlow and Facebook PyTorch.

The optimal parameter initialization is derived for neural networks being fine-tuned on pre-trained models for classification and show that such optimal initial loss leads to a significant acceleration in adapting a pre-trained neural network to a new task.

Another advantage of one or more embodiments of the method disclosed is that they are independent of the choice of architecture and may be applied to transfer knowledge within any domain. As a consequence, a benefit is that one or more embodiments of the method disclosed herein do not increase the complexity of the overall architecture to be trained (e.g. no additional layer, no multi-stage training process like warm up methods)..

Another advantage of one or more embodiments of the method disclosed herein is that they show a significant practical impact on convergence. As a consequence, a processing device used for training a neural network according to one or more embodiments of the method disclosed herein will use less resources for training a neural network than with a conventional methods, resulting in more available resources available to complete other tasks. One or more embodiments of the method disclosed herein may contribute to better performances overall compared to other traditional training methods, and does significantly contribute to bettering performances in training cases including a small number of training steps compared to other traditional training methods which is particularly useful to evaluate model potential during architecture search & design. One or more embodiments of the method disclosed herein may contribute to decrease the negative impact of catastrophic forgetting, that may occur during model training across tasks, as it limits the impact of noise propagation.

BRIEF DESCRIPTION OF THE DRAWINGS

Figures 1a and 1b are graphs which show the negative effect of classical transfer learning techniques on the variance of the output layer on two benchmarks (MNIST and CIFAR), using state of the art models. Both graphs highlight the “noise injection” phenomenon.

Figures 2a and 2b are diagrams which show respectively an FNN architecture and last layer initialization in a base model and in accordance with one embodiment of the method disclosed.

Figure 3 is a flowchart which shows an embodiment of a method for initializing a pre-trained neural network.

Figure 4 is a flowchart which shows an embodiment of a method for training a pre-trained neural network which uses an embodiment of the method disclosed in Figure 3.

Figure 5 is a diagram which shows an embodiment of a processing device which may be used for initializing a pre-trained neural network in accordance with an embodiment.

Figure 6 is a table which illustrates an initial percentage of noise energy to total energy of back-propagated error at the output of the last layer, profiled for models fine-tuned using regular fine-tuning; 95% confidence interval is calculated over 24 seeds.

Figure 7 shows a plurality of graphs and illustrates test accuracy progress for fine-tuning models that are pre-trained on ImageNet dataset.

Figure 8a is a table which illustrates average initial test accuracy improvement by using one embodiment of the method disclosed herein.

Figure 8b is a table which illustrates convergence test accuracy of models trained on CIFAR10 dataset with 95% confidence.

Figure 8c is a table which illustrates convergence test accuracy of models trained on CIFAR100 dataset with 95% confidence.

Figure 9 is a table which illustrates convergence test accuracy of models trained on Caltech101 dataset with 95% confidence.

DETAILED DESCRIPTION

In the following description of the embodiments, references to the accompanying drawings are by way of illustration of an example by which the invention may be practiced.

Terms

The term "invention" and the like mean "the one or more inventions disclosed in this application," unless expressly specified otherwise.

The terms "an aspect," "an embodiment," "embodiment," "embodiments," "the embodiment," "the embodiments," "one or more embodiments," "some embodiments," "certain embodiments," "one embodiment," "another embodiment" and the like mean "one or more (but not all) embodiments of the disclosed invention(s)," unless expressly specified otherwise.

A reference to "another embodiment" or "another aspect" in describing an embodiment does not imply that the referenced embodiment is mutually exclusive with another embodiment (e.g., an embodiment described before the referenced embodiment), unless expressly specified otherwise.

The terms "including," "comprising" and variations thereof mean "including but not limited to," unless expressly specified otherwise.

The terms "a," "an" and "the" mean "one or more," unless expressly specified otherwise.

The term "plurality" means "two or more," unless expressly specified otherwise.

The term "herein" means "in the present application, including anything which may be incorporated by reference," unless expressly specified otherwise.

The term "whereby" is used herein only to precede a clause or other set of words that express only the intended result, objective or consequence of something that is previously and explicitly recited. Thus, when the term "whereby" is used in a claim, the clause or other words that the term "whereby" modifies do not establish specific further limitations of the claim or otherwise restricts the meaning or scope of the claim.

The term "e.g." and like terms mean "for example," and thus do not limit the terms or phrases they explain. For example, in a sentence "the computer sends data (e.g., instructions, a data structure) over the Internet," the term "e.g." explains that "instructions" are an example of "data" that the computer may send over the Internet, and also explains that "a data structure" is an example of "data" that the computer may send over the Internet. However, both "instructions" and "a data structure" are merely examples of "data" and other things besides "instructions" and "a data structure" can be "data."

The term "i.e." and like terms mean "that is," and thus limit the terms or phrases they explain.

Neither the Title nor the Abstract is to be taken as limiting in any way as the scope of the disclosed invention(s). The title of the present application and headings of sections provided in the present application are for convenience only, and are not to be taken as limiting the disclosure in any way.

Numerous embodiments are described in the present application, and are presented for illustrative purposes only. The described embodiments are not, and are not intended to be, limiting in any sense. The presently disclosed invention(s) are widely applicable to numerous embodiments, as is readily apparent from the disclosure. One of ordinary skill in the art will recognize that the disclosed invention(s) may be practiced with various modifications and alterations, such as structural and logical modifications. Although particular features of the disclosed invention(s) may be described with reference to one or more particular embodiments and/or drawings, it should be understood that such features are not limited to usage in the one or more particular embodiments or drawings with reference to which they are described, unless expressly specified otherwise.

With all this in mind, one or more embodiments of the present invention are directed to a method and a system for initializing a pre-trained neural network and its use for training a pre-trained neural network.

It will be appreciated that one or more embodiments of the method disclosed herein may be implemented according to various embodiments.

More precisely and now referring to Fig. 5, there is shown an embodiment of a processing device 500 which may be used for implementing a method for initializing a pre-trained neural network.

In fact, it will be appreciated that the processing device 500 may be any type of computer.

In one embodiment, the processing device 500 is selected from a group consisting of desktop computers, laptop computers, tablet PC's, servers, smartphones, etc.

In the embodiment shown in Fig. 5, the processing device 500 comprises a central processing unit (CPU) 502, also referred to as a microprocessor, a graphic processing unit (GPU) 503, input/output devices 504, an optional display device 506, communication ports 508, a data bus 510 and a memory unit 512.

The central processing unit 502 is used for processing computer instructions. The skilled addressee will appreciate that various embodiments of the central processing unit 502 may be provided.

In one embodiment, the central processing unit 502 comprises a CPU Intel i9-7920X manufactured by Intel$^{(TM)}$.

The graphics processing unit 503 is used for processing specific computer instructions. It will be appreciated that a memory unit 520 is operatively connected to the graphics processing unit 503.

In one embodiment, the graphics processing unit 503 comprises a GPU Titan V manufactured by Nvidia$^{(TM)}$.

The input/output devices 504 are used for inputting/outputting data into the processing device 500.

The optional display device 506 is used for displaying data to a user. The skilled addressee will appreciate that various types of display device 506 may be used.

In one embodiment, the optional display device 506 is a standard liquid crystal display (LCD) monitor.

The communication ports 508 are used for operatively connecting the processing device 500 to various processing devices.

The communication ports 508 may comprise, for instance, universal serial bus (USB) ports for connecting a keyboard and a mouse to the processing device 500.

The communication ports 508 may further comprise a data network communication port such as an IEEE 802.3 port for enabling a connection of the processing device 508 with another processing device.

The skilled addressee will appreciate that various alternative embodiments of the communication ports 508 may be provided.

The memory unit 512 is used for storing computer-executable instructions.

The memory unit 512 may comprise a system memory such as a high-speed random access memory (RAM) for storing system control program (e.g., BIOS, operating system module, applications, etc.) and a read-only memory (ROM). In one embodiment, the memory unit 512 has 128 GB of DDR4 RAM.

It will be appreciated that the memory unit 512 comprises, in one embodiment, an operating system module 514.

It will be appreciated that the operating system module 514 may be of various types.

In one embodiment, the operating system module 514 is Linux Ubuntu 18.04 + Lambda Stack.

In one embodiment, the memory unit 520 comprises an application 516 for initializing a pre-trained neural network.

In one embodiment, the memory unit 520 operatively connected to the graphics processing unit 503 and has a size of 24 GB of VRAM. The skilled addressee will appreciate that various alternative embodiments may be possible.

The memory unit 520 is further used for storing data 518. The skilled addressee will appreciate that the data 518 may be of various types.

In fact, it will be appreciated that the memory unit 520 of the graphics processing unit 503 is further used for storing at least a portion of data referred to as the batch size. The skilled addressee will appreciate that a larger batch size "may" improve the effectiveness of the optimization steps resulting in more rapid convergence of the model parameters, and a larger batch size can also improve performance by reducing the communication overhead caused by moving the training data to the graphics processing unit 503 - causing more compute cycles to run on the card with each iteration.

In one embodiment, the processing device 500 is a 4GPU Deep learning workstation manufactured by Lambda Quad.

Flow of data and error

It will be appreciated that a Feed-forward Neural Network (FNN) is usually built by stacking up a number of layers on top of each other. The input of a layer can be composed of any combination of the previous layers' outputs. Let the input and output of the l-th layer of an FNN having L layers be $\boldsymbol{X}^l$ and $\boldsymbol{A}^l$ respectively. They are related to each other through functions $g(.)$ and $h(.)$ as

$$\boldsymbol{X}^L = g^l(\boldsymbol{A}^1, \boldsymbol{A}^2, \ldots, \boldsymbol{A}^{l-1});\ \boldsymbol{A}^l = h^l(\boldsymbol{X}^l, \boldsymbol{W}^l, \boldsymbol{b}^l) \qquad (1)$$

where $\boldsymbol{W}^l \in \mathbb{R}^{U\times V}$ and $\boldsymbol{b}^l \in \mathbb{R}^{1\times V}$ are weights and biases from the l-th layer and V is the number of columns in $\boldsymbol{X}^l$.

If the target task is classification with C classes, then the last layer is usually a fully connected one with $A^L \in \mathbb{R}^{N\times C}$, where N is the number of examples passing through the network, known as *batch size*. Specifically for this layer

$$\boldsymbol{A}^L = \boldsymbol{X}^L \boldsymbol{W}^{L^T} + \vec{\mathbf{1}}\boldsymbol{b}^L, \tag{2}$$

where $\vec{1}$ is a column vector of ones.

Since many formulas are batch independent and could be easily broadcast, lower-case bold-face letters of corresponding introduced matrices are separately used to indicate a single example in the batch (N = 1). Therefore,

Equation 2 could be rewritten for a single example as

$$\boldsymbol{a}^L = \boldsymbol{x}^L \boldsymbol{W}^{L^T} + \boldsymbol{b}^L \tag{3}$$

The posterior of the last layer's neurons, corresponding to each class, is usually estimated using a softmax normalizer, defined as

$$\hat{y}^j = \frac{e^{a_j^L}}{\sum_{i=1}^{C} e^{a_j^{L}}}, \tag{4}$$

where a_j^L represents the j-th element of aL.

It will be appreciated that cross entropy is the most commonly used loss function for classification tasks and is equal to the Kullback-Leibler divergence between the labels and the estimates) $\mathcal{L} = -\mathrm{D}_{\mathrm{KL}}(\mathrm{Y}, \hat{\mathrm{Y}})$. To train the network using back-propagation [10], gradients of the loss with respect to each parameter are calculated. To make this easier using the chain rule, first, the gradients are computed with respect to the output of each layer as in

$$\delta^L = \nabla_{a^l}\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial a_1^l}, \frac{\partial \mathcal{L}}{\partial a_2^l}, \dots\right] \tag{5}$$

and from there the desired gradients are calculated;

$$\frac{\partial \mathcal{L}}{\partial w_j^l} = \delta_j^l \frac{\partial h^l}{\partial w_j^l}; \frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l \frac{\partial h^l}{\partial b_j^l} \tag{6}$$

where W_j^l is the *j*-th row of W^l

The gradients of the CE loss with respect to the output of the j-th neuron of the deepest layer are equal to:

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} = \frac{1}{N}(\hat{y}_j - y_j) \tag{7}$$

and since the last layer is fully connected, the back-propagated error to the previous layer is also easily calculated using the chain rule:

$$\delta_k^{L-1} = \sum_{j=1}^{C} \delta_j^L W_{j,k}^L \tag{8}$$

where $W_{j,k}^L$is the k-th element of W_j^L. Finally, the weights and biases are updated using gradient descend.

Specifically for the last layer, the gradients of loss with respect to the rows of the weight matrix are:

$$\nabla W_j^L \mathcal{L} = N\mathbb{E}_N[\delta_j^L x^L], \tag{9}$$

where $\mathbb{E}_N$ is the expectation over the examples in the batch. Likewise the derivative of the loss with respect to a single bias in the last layer is:

$$\frac{\partial \mathcal{L}}{\partial b_j^L} = N\mathbb{E}_N[\delta_j^L] \tag{10}$$

<u>Initialization</u>

During the back-propagation algorithm [10], each data entry is passed twice through each layer's weights except the layers fed directly by the raw input. The magnitude of weights in a layer may get affected by the energy of the input visited by the layer, and the error back-propagated up to its output. Mathematically, it means that in Equations 6, a term of $\boldsymbol{x}^l$ usually appears in the derivative of h^l with respect to the weights of the l-th layer. This is already shown for the last layer in Equation 9. Weights are distinguished from biases and called such since they involve multiplication. This operation can rapidly increase/decrease the energy of its result, compared to the operands. This intensified/lessened energy of the output may increase/decrease the energy of the weights themselves through the gradient updates as discussed. This loop can lead to numerical problems known as exploding/vanishing gradients. One way of facing these problems is to initialize the weights and biases such that the energy of the flowing data/error is preserved. Currently, energy preserving initialization [9] is known to be the optimal solution for training ReLU networks [2].

At the end of training a model on the source task, the magnitude of back-propagated errors goes toward zero. By switching the task and introducing randomly initialized layers, these errors are suddenly increased. Moreover, the optimization algorithm is usually restarted which causes updates to modify all the parameters with the same rate. These large back-propagated errors include considerable amounts of noise as shown in Figure 6. This noise is injected into pre-trained knowledge through the first update. Fig. 1 shows the sudden initial changes in the variance of the input to the last layer when pre-trained models are fine-tuned.

The two common approaches to reduce this contamination are to slow down the training and/or to include a warm-up (WU) phase. However, the former slows down the contamination rather than eliminating it [17] since the small learning rate also updates the augmented parameters slowly, which injects noise into pre-trained layers for a longer time. In the latter solution, the new parameters are updated for a number of steps before jointly training the entire network.

Fig. 1. Sudden initial change in variance of XL with initial learning rate equal to 0.0001, fine-tuned on (a) MNIST and (b) CIFAR100 datasets. The horizontal axes show the training steps. In each model the augmented parameters are initialized based on preserving the variance of gradient recommended in [8]. The color shadows represent the standard deviation through training with 24 different seeds.

The warm-up phase, the accuracy of the network is limited since most parts of the network are frozen. Additionally, the effective number of required training steps in the warm-up phase may be large, depending on the learning rate, initial values of augmented parameters and size of the dataset.

In a more effective approach, an initialization technique is disclosed herein for fine-tuning in which the noise is initially trapped only within the task-specific augmented parameters. In contrast to using the warm-up phase, in the method disclosed herein, the noise is always minimized after the first update and therefore the parameters can be trained altogether afterward. In addition, the method disclosed herein is easier to apply in the sense that the training process is not manipulated in any way.

Energy Components of Back-propagating Error

The energy consists of three components from which only one is directly correlated with the accuracy of the estimator and the two others are energies of the true labels and the estimates. The contribution of these components is disclosed and the lower and upper bounds for each one is found.

Let Φ_j be the sum of energy of δ_j^L through all examples of the batch, defined as:

$$\Phi_j = \mathbb{E}_N\big[\big(N\delta_j^L\big)^2\big] = \mathbb{E}_N\big[\big(\hat{y}_j - y_j\big)^2\big]. \tag{11}$$

Accordingly, using Equation 7, the total energy of the error over all examples in the batch and all C neurons of the last layer is equal to:

$$\Phi = \sum_{j=1}^{C} \Phi_j = \mathbb{E}_N[\hat{y}\hat{y}^T] + \mathbb{E}_N[yy^T] - 2\mathbb{E}_N[\hat{y}y^T] \tag{12}$$

Assuming the labels are one-hot encoded; the third term becomes the average probability assignment for the correct labels. The goal of training the model is to maximize this term which is bounded by $0 \leq \mathbb{E}_N[\hat{y}y^T] \leq 1$. The second term is the energy of the labels and is always equal to one. Finally, the first term is the energy of the estimates. The infimum of this term could be calculated using Cauchy-Schwarz inequality as follows:

$$(y \underset{1}{\rightarrow})^2 \leq \left(\underset{1}{\rightarrow}^T \underset{1}{\rightarrow}\right)(\hat{y}\hat{y}^T) \tag{13}$$

$$\hat{y} \underset{1}{\rightarrow} = \sum_{j=1}^{C} \hat{y}_j = 1; \underset{1}{\rightarrow}^T \underset{1}{\rightarrow} = C \tag{14}$$

$$\hat{y}\hat{y}^T \geq \frac{1}{C} \tag{15}$$

This could also be derived directly from the definition of the softmax (Equation 4),

$$\zeta = \sum_{j=1}^{C} \hat{y}_j^2 = \sum_{j=1}^{C} \frac{e^{2a_j^L}}{(\sum_{i=1}^{C} e^{a_i^L})^2} = \frac{\sum_{j=1}^{C} e^{2a_j^L}}{(\sum_{i=1}^{C} e^{a_i^L})^2}, \tag{16}$$

followed by taking the partial derivatives with respect to the inputs of the soft-max, setting it to zero and re-indexing gives:

$$\frac{\partial \zeta}{\partial a_i^L} = \frac{2e^{2a_k^L} \sum_{i=1}^{C} e^{a_i^L} - 2e^{a_k^L} \sum_{j=1}^{C} e^{2a_j^L}}{(\sum_{i=1}^{C} e^{a_i^L})^3} = 0; \tag{17}$$

that results in:

$$\sum_{i=1}^{C} e^{a_i^L a_k^L} = \sum_{j=1}^{C} e^{2a_j^L} \; \forall k, \tag{18}$$

which means $a_j^L = a_k^L; j,k \in \{1,2,..,C\}$. Reforming Equation 16 considering equal elements in a^L finds minimum of ζ as:

$$\zeta_{min} = \frac{Ce^{2a_j^L}}{(e^{a_j^L})^2} = \frac{1}{C}$$

The upper bound of the energy of the estimates equals to one and is achieved when their entropy is minimized per example, so $\frac{1}{C} \leq \mathbb{E}_N[\hat{y}\hat{y}^T] \leq 1$. All in all, the total energy of the back-propagated error is bounded between 0 and 2.

Here, the bounds for the energy of estimates are investigated. Using the definition of softmax, it is shown that to achieve the minimum of $\mathbb{E}_N[\hat{y}\hat{y}^T]$, all the neurons of the last layer should have per-example equal output.

Now referring to Fig. 3, there is now shown an embodiment of a method for initializing a pre-trained neural network 100.

According to processing step 102, a pre-trained neural network is obtained. It will be appreciated that the pre-trained neural network has an output layer. In one embodiment, the pre-trained neural network uses softmax logit.

It will be appreciated that the pre-trained neural network may be provided according to various embodiments.

In one embodiment, the pre-trained neural network is received from a processing device. In another embodiment, the pre-trained neural network is obtained from a memory unit of the processing device. In another embodiment, the pre-trained neural network is provided by a user interacting with the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the pre-trained neural network.

Still referring to Fig. 3 and according to processing step 104, the output layer of the pre-trained neural network is amended. It will be appreciated that the amending of the output layer of the pre-trained neural network comprises updating each weight of the output layer according to a function that maximizes the entropy of the output classes probability.

It will be appreciated that the function depends on a parameter controlling a proportion of error of the output classes probability such as it decreases the variance of the output classes probability.

In one embodiment, the amending of the output layer of the pre-trained neural network further comprises z-normalizing features located right before the output layer prior updating each weight.

It will be appreciated that in one embodiment, the initializing of the pre-trained neural network is performed to prevent adverse contamination during a training of the initialized pre-trained neural network.

According to processing step 106, the initialized pre-trained neural network is provided.

It will be appreciated that the initialized pre-trained neural network may be provided according to various embodiments. In one embodiment, the initialized pre-trained neural network is provided to a processing device. In another embodiment, the initialized pre-trained neural network is saved in a memory unit of the processing device. In another embodiment, the initialized pre-trained neural network is displayed to a user interacting with the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the initialized pre-trained neural network.

While it has been disclosed in Fig. 3 that the method is used for initializing a neural network which is pre-trained, it will be appreciated that in one or more alternative embodiments, the neural network is not pre-trained.

In such embodiments, there is disclosed a method for initializing a neural network. The method comprises obtaining a neural network having an output layer.

It will be appreciated that the neural network may be provided according to various embodiments.

In one embodiment, the neural network is received from a processing device. In another embodiment, the neural network is obtained from a memory unit of the processing device. In another embodiment, the neural network is provided by a user interacting with the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the neural network.

The method further comprises amending the output layer of the neural network. The amending of the output layer comprises updating each weight of the output layer according to a function that maximizes the entropy of the output classes probability. It will be appreciated that the function depends on a parameter controlling a proportion of error of the output classes probability such as it decreases the variance of the output classes probability.

The method further comprises providing the initialized neural network.

It will be appreciated that the initialized neural network may be provided according to various embodiments. In one embodiment, the initialized neural network is provided to a processing device. In another embodiment, the initialized neural network is saved in a memory unit of the processing device. In another embodiment, the initialized neural network is displayed to a user interacting with the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the initialized neural network.

It will be appreciated that an embodiment of the method disclosed in Fig. 3 is detailed herein below.

It will be appreciated that the initialized pre-trained neural network may be used when training the pre-trained neural network as disclosed for instance in Fig. 4.

More precisely and according to processing step 200, a pre-trained neural network to train is obtained.

As mentioned above, it will be appreciated that the pre-trained neural network may be obtained according to various embodiments.

According to processing step 202, a dataset suitable for the training is obtained.

It will be appreciated that the dataset suitable for the training may be obtained according to various embodiments.

In one embodiment, the dataset is obtained from a remote processing device operatively connected with a processing device.

According to processing step 204, the pre-trained neural network is initialized.

It will be appreciated that the pre-trained neural network may be initialized according to one or more embodiments of the method disclosed in Fig. 3.

According to processing step 206, the initialized pre-trained neural network is trained.

It will be appreciated that the initialized pre-trained neural network is trained using the dataset obtained.

In one or more embodiments, the training of the initialized pre-trained neural network comprises training the initialized pre-trained neural network using a first training batch of the obtained dataset, wherein the first training batch is smaller than a number of features fed to the last layer of the initialized pre-trained neural network.

It will be appreciated that in one or more embodiments, the training is a federated learning method. It will be appreciated that the federated learning method is disclosed at https://arxiv.org/pdf/1902.04885.pdf.

It will be appreciated that in one or more embodiments, the training is a meta-learning method. It will be appreciated that meta learning is disclosed for instance in the article "Human-level concept learning through probabilistic program induction" by Brenden M. Lake et al. Science 350, 1332 (2015).

It will be appreciated that in one or more embodiments, the training is a distributed machine learning method. It will be appreciated that the distributed machine learning method is disclosed at https://arxiv.org/abs/1810.06060.

It will be appreciated that in one or more other embodiments, the training is a network architecture search using the pre-trained neural network as a seed. It will be appreciated that the network architecture search is disclosed at https://arxiv.org/pdf/1802.03268.pdf.

It will be appreciated that in one or more embodiments, the pre-trained neural network comprises a generative adversarial network. In such embodiments, the initializing of the pre-trained neural network is performed at the discriminator.

Still referring to Fig. 4 and according to processing step 208, the trained neural network is provided.

It will be appreciated that the trained neural network may be provided according to various embodiments.

In one embodiment, the trained neural network is provided to a processing device. In another embodiment, the trained neural network is saved in a memory unit of the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the initialized neural network.

It will be appreciated that there is also disclosed a method for training a neural network through federated learning.

It will be appreciated that the method comprises obtaining a shared neural network to train.

The method further comprises obtaining at least two datasets suitable for the federated learning. Each of the at least two datasets is used for training a corresponding decentralized training unit.

The method further comprises each decentralized training unit performing a first round of training using a corresponding dataset.

The method further comprises, for each subsequent round of training, each decentralized training unit initializing the shared neural network using one or more embodiments of the method disclosed above for initializing a pre-trained neural network and each decentralized training unit training the initialized shared neural network using the corresponding dataset.

The method further comprises globally federating the learning from all decentralized training units to a resulting global shared neural network, and until the global shared neural network converges to a good global model, providing the corresponding global shared neural network to the decentralized training units as the new shared neural network.

Finally, the method comprises providing the trained shared neural network. It will be appreciated that the trained shared neural network may be provided according to various embodiments. In one embodiment, the trained shared neural network is provided to a processing device. In another embodiment, the trained shared neural network is saved in a memory unit of the processing device. In another embodiment, the trained shared neural network is displayed to a user interacting with the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the trained shared neural network.

It will be appreciated that there is also disclosed a method for training a neural network using a reptile meta-learning method. The method comprises obtaining a neural network to train.

The method further comprises obtaining a dataset suitable for the reptile meta-learning method. It will be appreciated that the reptile meta-learning method is disclosed at https://d4mucfpksywv.cloudfront.net/research-covers/reptile/reptile_update.pdf.

The method further comprises, for each iteration of the reptile meta-learning method, initializing the neural network using one or more embodiments of the method disclosed above for initializing a pre-trained neural network for each task sampled and training the initialized neural network for the corresponding sampled task using the obtained dataset.

Finally, the method comprises providing the trained neural network. It will be appreciated that the trained neural network may be provided according to various embodiments. In one embodiment, the trained neural network is provided to a processing device. In another embodiment, the trained neural network is saved in a memory unit of the processing device. In another embodiment, the trained neural network is displayed to a user interacting with the processing device. The skilled addressee will appreciate that various alternative embodiments may be provided for providing the trained neural network.

Detailed description of an embodiment of a method for initializing a neural network disclosed above

It will be appreciated that initially, the energy of the estimates contains pure noise, i.e. it lacks a meaningful relationship with either the inputs or the labels. Its infimum was calculated and it was shown that it can be achieved when all the estimates are exactly equal to each other for each example. This condition is intuitively appealing since it maximizes the entropy of the estimates prior to the training when $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are independent and/or unaligned. The entropy is exactly reflected in the CE loss which then becomes deterministically ln C regardless of $\hat{\boldsymbol{y}}$.

Another source of contaminating pre-trained layers is $\boldsymbol{W}^L$ itself. This is because $\boldsymbol{\delta}^{L-1}$ is affected by both the last layer's error and its weights (see Equation 8). To prevent the noise from contaminating pre-trained layers an efficient solution should consider both of these criteria. One or more embodiments of a method are therefore introduced which maximizes the initial entropy of estimates while preventing $\boldsymbol{\delta}^{L-1}$ to become contaminated by $\boldsymbol{W}^L$. The method can be described as follows.

First and in accordance with an embodiment, the method requires the features that are fed to the last layer to be normalized. This is done by applying z-normalization across the batch,

$$\bar{x}^L = \begin{cases} \frac{x^L - \mu_N}{\sqrt{\sigma_N^2 + \varepsilon}}, \text{ if } \textit{train} \\ \frac{x^L - \widehat{\mu}_N}{\sqrt{\widehat{\sigma}_N^2 + \varepsilon}}, \text{ if } \textit{inference} \end{cases} ; \boldsymbol{\mu}_N, \widehat{\boldsymbol{\mu}}_N, \boldsymbol{\sigma}_N^2, \widehat{\boldsymbol{\sigma}}_N^2 \in \mathbb{R}^{1 \times K}, \quad (20)$$

$$\boldsymbol{\mu}_N = \mathbb{E}_N[x^L]; \boldsymbol{\mu}_N = \text{detach}(\mu_N),$$

$$\boldsymbol{\sigma}_N^2 = \mathbb{E}_N[(\boldsymbol{x}_i^L - \boldsymbol{\mu}_N)^2]; \widehat{\boldsymbol{\sigma}}_N^2 = \text{detach}(\boldsymbol{\sigma}_N).$$

Figure 2 shows a FNN architecture and last layer's initialization in (a) base model and (b) EN- TAME. According to [8], *m* is 2 for ReLU networks.

This is similar to batch-normalization [12], except that it does not need any learnable parameters. The statistics used in inference mode of the simple z- normalization are detached from the computational graph version of the ones obtained in the corresponding training step.

Second, the method maximizes the entropy of the estimates by initializing the last layer's weights to values drawn from Independent and Identically Distributed (i.i.d.), zero centered normal distribution as follows

$$\boldsymbol{W}^L \sim \mathcal{N}(0, \phi_{\mathrm{w}}); \ \phi_{\mathrm{w}} = \frac{\gamma^2 \lambda^2}{C^2} \quad (21)$$

where ϕ_{w} is the energy of each element of $\boldsymbol{W}^L$, γ is the initial value of the learning rate (recommended default γ = 10^{-4}) and λ is a hyper-parameter that controls the proportion of noise energy over total energy of last layers weights, right after the first update (recommended range is 1 to 1000). ϕ_{w} is chosen to be a numerically small number (for example ϕ_{w} = 10−12 means that 95% of the values in $\boldsymbol{W}^L$ are initially between $-2 \times 10^{-6}$and 2×10^{-6}), but it can be seen that such small randomness may help the expressiveness of the model. If the biases are also initialized constantly to all zeros and if K is not extremely large, the energy of $\boldsymbol{a}^L$ would be initially very small as well. Concretely, from the distribution selected for $\boldsymbol{W}^L$, the value of each output neuron is approximately zero-centered, or $\mathbb{E}_N[\mathrm{a}^L] \approx \vec{0}$ and per-example energy of all output neurons together is

$$\mathbb{E}_N\left[\sum_{j=1}^{C} a_j^{L^2}\right] = \mathbb{E}_N\left[\boldsymbol{a}^L \boldsymbol{a}^{L^T}\right] = \mathbb{E}_N\left[\bar{x} W_j^T W_j \bar{x}^T\right] = K\phi_{\mathrm{W}} \mathbb{E}_N[\bar{x}\bar{x}^T] = \frac{K^2\phi_{\mathrm{W}}}{N} \qquad (22)$$

This is derived from $\mathbb{E}_N[\bar{x}\bar{x}^T] = \frac{1}{K}$, due to $\bar{\boldsymbol{X}}$ being normalized across the batch.

Moreover, the exponential function is close to linear when its input is close to zero. This could be easily shown by using the result of Equation 22, and the Taylor series approximation of the exponential function around zero, $e^{a_j^L} \approx 1 + a_j^L$. Plugging this into the softmax definition yields $\hat{y}_j \approx \frac{1}{C}$, which approximately maximizes the entropy as desired.

When the estimates are equal for each example, δ_j^L becomes only a function of the prior, i.e.,

$$\delta_j^L = \begin{cases} \frac{1}{NC}, & \text{if } y_j = 0 \\ \frac{1}{N}\left(\frac{1}{C} - 1\right), & \text{if } y_j = 1 \end{cases} ; j \in \{1, 2, \ldots, C\}. \qquad (23)$$

Accordingly, the gradients of the loss with respect to last layer's parameters are simplified to

$$\nabla \boldsymbol{W}_j^L \mathcal{L} = \mathbb{E}_N\left[\left(\frac{1}{C} - y_j\right)\bar{\boldsymbol{x}}^L\right];\ \nabla b_j^L \mathcal{L} = \mathbb{E}_N\left[\frac{1}{C} - y_j\right]. \qquad (24)$$

Applying the updates results in

$$\boldsymbol{W}_j^{L,1} = \boldsymbol{W}_j^{L,0} + \gamma \mathbb{E}_N\left[y_j \bar{\boldsymbol{x}}^L\right] - \gamma \frac{1}{C} \mathbb{E}_N[\bar{\boldsymbol{x}}^L];\ b_j^{L,1} = \gamma \mathbb{E}_N\left[y_j\right] - \gamma \frac{1}{C}, \qquad (25)$$

where γ is the initial learning rate and the second number in the superscripts represent the number of updates, e.g. $W_j^{l,u}$ indicates to the weights of the l-th layer after u updates. The error cannot move further backward at this point since ϕ_w is very small, making δ^{L-1} negligible.

After the first update, the outputs of each neuron of the last layer can get comparably high expected values. This may cause the estimates to have much lower entropy compared to the initial state. On the other hand, depending on $\boldsymbol{Y}$ and $\bar{\bar{X}}$, multiple rows and columns of weights and corresponding elements of biases from the last layer can possibly get identical first updates (see Equation 24).

This may cause the entropy of the estimates to stay comparably high for each example. Very small random numbers, used to initialize W^L, help these identical estimates to diverge and make different estimates. It will be appreciated that as the expectation of a_j^L gets away from zero toward positive values, the exponential functions in softmax make the small difference much larger. Therefore, the expressiveness of the last layer is preserved by initializing its weights to very small numbers instead of zero.

The first update makes the energy of W^L large enough to let the error of the next updates back-propagate through it and reach the pre-trained layers. In other words, this automatically opens up the stalled way and lets the error to back-propagate to the output of other layers. This is enough for correctly guiding pre-trained parameters with an advanced optimization algorithm like Adam [13] is used. Most of the noise is purified and the next back-propagated errors toward pre-trained features are meaningful and contain both prior and likelihood. In more details, the energy of the j-th row in $W^{L,1}$ becomes:

$$W_j^{L,1} W_j^{L,1^T} = K\phi_w + \gamma^2 \mathbb{E}_N \left[(\tfrac{1}{C} - y_j)^2 \ddot{x} \ddot{x}^T \right] \qquad (26)$$

$$= K\gamma^2 \frac{\lambda^2}{C^2} + K\gamma^2 \frac{1}{NC^2} + \gamma^2 \frac{C-2}{NC} \sum_{i=1}^{N} Y_i \overline{X}_i \overline{X}_i^T$$

which means:

$$\frac{K\gamma^2}{C^2}(\lambda^2 + \tfrac{1}{N}) \leq \boldsymbol{W}_j^{L,1} \boldsymbol{W}_j^{L,1^T} \leq \frac{K\gamma^2}{C^2}(\lambda^2 + \tfrac{1}{N} + \tfrac{C(C-2)}{N}). \tag{27}$$

Energy of $W_j^{L,1}$ is $\frac{K}{N}$times the energy of j-th bias in the last layer. Since $b_j^{L,1}$contains only information about the prior, we desire to make its energy initially smaller than $W_j^{L,1}$. For this to happen, initial N has to be chosen such that N < K (which usually is satisfied). On the other hand, λ should be chosen such that $\lambda^2 < \frac{1}{N}$ but not very small to numerically reduce the rank of $W_j^{L,1}$due to possible similar updates (see Equation 25), which may results in higher entropy of $\hat{\boldsymbol{y}}$. $N\lambda^2$ roughly determines the maximum proportion of remaining energy of noise to the total energy of elements of $\boldsymbol{W}_j^{L,1}$.

Role of Feature Normalization

As mentioned above and in accordance with one or more embodiments, a feature normalization is performed. It will be appreciated that applying z-normalization on top of features may increase or decrease the level of average feature-wise energy in X^L resulting in less need for tweaking the learning rate and ϕ_w for different tasks and even different models. If the values of $\boldsymbol{X}^L$ are too small, it may take a longer time for W^L to grow which leaves pre-trained features unchanged for a longer time.

In one or more embodiments, z-normalization is applied if the provided initialized pre-trained model is to be further trained on the provided data, where the provided data exhibits an important domain shift with respect to the data on which the model was pre-trained.

Z-normalization across batches plays a more important role than just equalization. To clarify, it is assumed that image classification is done and two images in a batch contain exactly the same pattern or visual object. If one of the columns of X^L represents a feature that recognizes said pattern, the feature is expected to reflect the presence of the pattern in both of the mentioned images equally. The problem is that raw inputs are usually normalized with statistics that are identically applied on all pixels in all examples. In the best case, such normalization is applied separately for different channels. Object-wise normalization does not seem to be feasible prior to detection which indirectly is done through training neural network classifier. Therefore, even if the same object is exactly copied in both images due to normalizing raw images, one object may get less intensified than the other one. This may directly be reflected to the values of the particular column of X^L responsible for showing the presence of the desired object. Z-normalization, compensates for this problem by normalizing the features after they are detected.

Batch-normalization layers used in between the hidden layers of some pre-trained models, usually need more training steps to adapt to the distribution of target task's data. Since we also care about the performance of the model in the first training steps, feeding normalized features to the last layer is vital. It will be appreciated that the simple z-normalization applied on X^L, directly influences the first update of W^L.

<u>Experiments</u>

ImageNet [18] ILSVRC 2012 is the source dataset used to pre-train the models. Each pre-trained model is fine-tuned on the following datasets: MNIST [16], CIFAR10, CIFAR100 [15] and Caltech101 [5]. The latter dataset is not originally separated into train and test nor is balanced in contrast to the other ones. Each Caltech101 category is split randomly into train and test subsets with 15 percent chance of drawing each image for test subset.

Prior to feeding the input to the models, each channel is normalized with its mean and standard deviation obtained from all pixels of that channel throughout the corresponding training subset. Training images are also augmented with random horizontal flip.

The set of used architectures are listed in the leftmost column of Figure 7. Among these models InceptionV3 requires all images to be sc$\overset{\times}{a}$led up to 229 229, so due to limitations in device's memory the batch size of 64 is chosen for this architecture. In addition, the other models that are trained on the Caltech101 dataset are also fed 64 images per batch owning to large image sizes. All other models and dataset use batch size of 256.

The initialization recommended by [8] is used for augmented layers in the base models. A try was made to unify the problem by applying similar conditions for training different models as much as possible. This by itself would show the impact of one or more embodiments of the method disclosed and how universally it could help the task adaptation, even without considering hyper-parameter tuning. Accordingly, learning rate is set to 0.0001 for all models and datasets and the value of φw is chosen to be 10−12 everywhere.

Figure 7 shows the progress of test accuracy of pre-trained models fine-tuned on each dataset. The smaller plot inside each larger one shows the same curves zoomed-in the first steps of training. The colorful shade around each curve shows the standard deviation across 24 different seeds. Each plot includes 4 curves color mapped as follows; blue: base, orange: base with a single Warm Up (WU) step, green: disclosed method's Maximum Entropy Initialization (MEI), red: full disclosed method or MEI + Feature Normalization (FN). Experiments were also done with only applying FN, but they mostly perform worse than all other cases, so they are not included to save space and make plots more readable.

To measure how the convergence is sped up initially, average progressing accuracy is compared over first ten training steps. Paired t-test suggests that one or more embodiments of the method disclosed herein significantly enhances the test accuracy compared to the base method for all architectures and datasets mentioned herein. Figure 8a shows the average increase in the accuracy of the first 10 training steps with 95% confidence. Further improvements have been observed by adjusting λ and the batch size but to show the robustness of the model the same setup has been kept as much as possible.

Finally, the converged accuracy of each curve shown in Fig. 7 is listed in Figures 8b, 8c and 9. The convergence test accuracy is recorded after training models for 10 epochs if target dataset is CIFAR10 or Caltech101 and 15 epochs if target dataset is CIFAR100. Further experiments have been done on ResNet [9], DenseNet [11] and VGG [21] with other popular sizes but similar results were obtained so results of the two most common sizes of each in the above-mentioned tables were only reported.

Now referring to Fig. 7, there is shown test accuracy progress for fine-tuning models that are pre-trained on ImageNet dataset. The horizontal axes on each plot show the number of training steps. Colorful shades show the standard deviation across different seeds. A superscript $*$ means that all models in corresponding row or column are trained with batch size of 64 instead of 256 to make the model fit into the device. The smaller plots inside the bigger ones are just zoomed-in version of the same curves for the first few steps.

Now referring to Fig. 8a, there is shown average initial test accuracy improvement by using an embodiment of the method disclosed herein instead of base method. The entries show increase in the mean of test accuracy over first 10 steps of training with 95% confidence calculated over 24 seeds.

Now referring to Fig. 8b, there is shown convergence test accuracy of models trained on CIFAR10 dataset with 95% confidence.

Now referring to Fig. 8c, there is shown convergence test accuracy of models trained on CIFAR100 dataset with 95% confidence.

Now referring to Fig. 9, there is shown convergence test accuracy of models trained on Caltech101 dataset with 95% confidence. It will be appreciated that although a focus was done on image classification, the reasoning behind one or more embodiments of the method disclosed herein's impressive performance is not tied to image datasets in any way.

An important outcome of the empirical results is that models fine-tuned on datasets with 100 or even more classes show initial test accuracy of over 40% by visiting only the first 64 images. This can open-up a whole new discussion about the power of few-shot learning algorithms.

It will be appreciated that the application 516 for initializing a neural network comprises instructions for obtaining a pre-trained neural network having an output layer.

The application 516 for initializing a pre-trained neural network further comprises instructions for amending the output layer of the pre-trained neural network. The amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability. The function depends on a parameter controlling a proportion of error of the output classes probability such as it decreases the variance of the output classes probability.

The application 516 for initializing a pre-trained neural network further comprises instructions for providing the initialized pre-trained neural network.

It will be appreciated that there is also disclosed a non-transitory computer readable storage medium for storing computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising obtaining a pre-trained neural network having an output layer, amending the output layer of the pre-trained neural network, wherein the amending comprises updating each weight of the output layer according to a function that maximizes the entropy of the output classes probability, wherein the function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

It will be appreciated that there is also disclosed a computer program comprising computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising obtaining a pre-trained neural network having an output layer, amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

It will be appreciated that there is also disclosed a method for using a pre-trained neural network trained in accordance one or more embodiments of the method disclosed herein.

It will be appreciated that one or more embodiments of the method disclosed herein are of great advantage for various reasons.

An advantage of one or more embodiments of the method disclosed is that they significantly decrease the initial noise that is back-propagated from randomly-initialized parameters toward layers that contain the transferred knowledge. As a consequence, a processing device used for training a neural network according to one or more embodiments of the method disclosed herein will use less resources for training a neural network resulting in more available resources available to complete other tasks. Moreover, one or more embodiments of the method disclosed herein may contribute to better performances overall compared to other traditional training methods, and does significantly contribute to bettering performances in training cases including a small number of training steps compared to other traditional training methods which is particularly useful to evaluate model potential during architecture search & design. Also, one or more embodiments of the method disclosed herein may contribute to decrease the negative impact of catastrophic forgetting, that may occur during model training across tasks, as it limits the impact of noise propagation.

In fact, experiments show that models trained by one or more embodiments of the method disclosed herein learn substantially faster than those using prior art fine-tuning or even more complicated tricks such as warm up [17].

Another advantage of one or more embodiments of the method disclosed is that they are easy to implement and can be beneficially applied to any pre-trained neural network that estimates output probabilities using softmax logits in one embodiment. As a consequence, a benefit is a broad applicability and integration across various deep learning frameworks offered by various vendors such as Google TensorFlow and Facebook PyTorch.

The optimal parameter initialization is derived for neural networks being fine-tuned on pre-trained models for classification and show that such optimal initial loss leads to a significant acceleration in adapting a pre-trained neural network to a new task.

Another advantage of one or more embodiments of the method disclosed is that they are independent of the choice of architecture and may be applied to transfer knowledge within any domain. As a consequence, a benefit is that one or more embodiments of the method disclosed herein do not increase the complexity of the overall architecture to be trained (e.g. no additional layer, no multi-stage training process like warm up methods)..

Another advantage of one or more embodiments of the method disclosed herein is that they show a significant practical impact on convergence. As a consequence, a processing device used for training a neural network according to one or more embodiments of the method disclosed herein will use less resources for training a neural network than with a conventional methods, resulting in more available resources available to complete other tasks. One or more embodiments of the method disclosed herein may contribute to better performances overall compared to other traditional training methods, and does significantly contribute to bettering performances in training cases including a small number of training steps compared to other traditional training methods which is particularly useful to evaluate model potential during architecture search & design. One or more embodiments of the method disclosed herein may contribute to decrease the negative impact of catastrophic forgetting, that may occur during model training across tasks, as it limits the impact of noise propagation.

Clause 1: A method for initializing a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network having an output layer,

amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized pre-trained neural network.

Clause 2: The method as claimed in clause 1, wherein the amending of the output layer of the pre-trained neural network further comprises z-normalizing features located right before the output layer prior updating each weight.

Clause 3: The method as claimed in clause 1, wherein the pre-trained neural network uses softmax logit in the output layer.

Clause 4: A method for training a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network to train;

obtaining a dataset suitable for said training;

initializing the pre-trained neural network using the method as claimed in any one of clauses 1 to 3;

training the initialized pre-trained neural network using the obtained dataset; and

providing the trained neural network.

Clause 5: A method as claimed in clause 4, wherein said training is a federated learning method.

Clause 6: A method as claimed in clause 4, wherein said training is a meta-learning method.

Clause 7: A method as claimed in clause 4, wherein said training is a distributed machine learning method.

Clause 8: The method as claimed in clause 4, wherein said training is a network architecture search using said pre-trained neural network as a seed.

Clause 9: The method as claimed in any one of clauses 4 to 8, wherein the pre-trained neural network comprises a generative adversarial network, wherein said initializing of the pre-trained neural network using the method as claimed in clause 1 is performed at the discriminator.

Clause 10: A method for training a neural network through federated learning, the method comprising:

obtaining a shared neural network to train;

obtaining at least two datasets suitable for said federated learning, each of the at least two datasets for training a corresponding decentralized training unit;

each decentralized training unit performing a first round of training using a corresponding dataset;

for each subsequent round of training:

each decentralized training unit initializing the shared neural network using the method as claimed in any one of clauses 1 to 3,

each decentralized training unit training the initialized shared neural network using the corresponding dataset,

globally federating the learning from all decentralized training units to a resulting global shared neural network, and until the global shared neural network converges to a good global model, providing the corresponding global shared neural network to the decentralized training units as the new shared neural network; and

providing the trained shared neural network.

Clause 11: A method for training a neural network using a reptile meta-learning method, the method comprising:

obtaining a neural network to train;

obtaining a dataset suitable for said reptile meta-learning method;

for each iteration of the reptile meta-learning method:

initializing the neural network using the method as claimed in any one of clauses 1 to 3 for each task sampled, and

training the initialized neural network for said corresponding sampled task using the obtained dataset; and

providing the trained neural network.

Clause 12: The method as claimed in any one of clauses 4 to 9, wherein the training of the initialized pre-trained neural network comprising training the initialized pre-trained neural network using a first training batch of the obtained dataset, wherein the first training batch is smaller than a number of features fed to said last layer of said initialized pre-trained neural network.

Clause 13: A method for using a pre-trained neural network trained in accordance with any one of clauses 4 to 9.

Clause 14: A computer comprising:

a central processing unit;

a graphics processing unit;

a communication port;

a memory unit comprising an application for initializing a pre-trained neural network, the application comprising:

instructions for obtaining a pre-trained neural network having an output layer,

instructions for amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

instructions for providing the initialized pre-trained neural network.

Clause 15: Computer program comprising computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network having an output layer,

amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and providing the initialized pre-trained neural network.

Clause 16: A non-transitory computer readable storage medium for storing computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network having an output layer,

amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized pre-trained neural network.

Clause 17: A method for initializing a neural network, the method comprising:

obtaining a neural network having an output layer,

amending the output layer of the neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized neural network.

References

1. Agrawal, P., Girshick, R., Malik, J.: Analyzing the performance of multilayer neural networks for object recognition. In: European conference on computer vision. pp. 329–344. Springer (2014)

2. Arpit, D., Bengio, Y.: The benefits of over-parameterization at initialization in deep relu networks. arXiv preprint arXiv:1901.03611 (2019)

3. Azizpour, H., Razavian, A.S., Sullivan, J., Maki, A., Carlsson, S.: Factors of transferability for a generic convnet representation. IEEE transactions on pattern analysis and machine intelligence 38(9), 1790–1802 (2016)

4. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition (2013)

5. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. Computer vision and Image understanding 106(1), 59–70 (2007)

6. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 580–587 (2014)

7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)

8. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human- level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)

9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

10. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks 2(5), 359–366 (1989)

11. Huang, G., Liu, Z., Maaten, L.v.d., Weinberger, K.Q.: Densely connected convolutional networks. 2017 IEEE Conference on Computer Vision and Pat- tern Recognition (CVPR) (Jul 2017). https://doi.org/10.1109/cvpr.2017.243, http://dx.doi.org/10.1109/CVPR.2017.243

12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)

13. Kingma, D., Ba, J.: Adam: a method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980 15 (2015)

14. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks (2017)

15. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)

16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)

17. Li, Z., Hoiem, D.: Learning without forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence 40(12), 2935–2947 (2018)

18. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision 115(3), 211–252 (2015)

19. Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: an astounding baseline for recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 806–813 (2014)

20. Shermin, T., Murshed, M., Lu, G., Teng, S.W.: Transfer learning using classification layer features of cnn (2018)

21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014)

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

CLAIMS:

1. A method for initializing a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network having an output layer,

amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized pre-trained neural network.

2. The method as claimed in claim 1, wherein the amending of the output layer of the pre-trained neural network further comprises z-normalizing features located right before the output layer prior updating each weight.

3. The method as claimed in claim 1, wherein the pre-trained neural network uses softmax logit in the output layer.

4. A method for training a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network to train;

obtaining a dataset suitable for said training;

initializing the pre-trained neural network using the method as claimed in any one of claims 1 to 3;

training the initialized pre-trained neural network using the obtained dataset; and providing the trained neural network.

5. A method as claimed in claim 4, wherein said training is a federated learning method.

6. A method as claimed in claim 4, wherein said training is a meta-learning method.

7. A method as claimed in claim 4, wherein said training is a distributed machine learning method.

8. The method as claimed in claim 4, wherein said training is a network architecture search using said pre-trained neural network as a seed.

9. The method as claimed in any one of claims 4 to 8, wherein the pre-trained neural network comprises a generative adversarial network, wherein said initializing of the pre-trained neural network using the method as claimed in claim 1 is performed at the discriminator.

10. A method for training a neural network through federated learning, the method comprising:

obtaining a shared neural network to train;

obtaining at least two datasets suitable for said federated learning, each of the at least two datasets for training a corresponding decentralized training unit;

each decentralized training unit performing a first round of training using a corresponding dataset;

for each subsequent round of training:

each decentralized training unit initializing the shared neural network using the method as claimed in any one of claims 1 to 3, each decentralized training unit training the initialized shared neural network using the corresponding dataset,

globally federating the learning from all decentralized training units to a resulting global shared neural network, and

until the global shared neural network converges to a good global model, providing the corresponding global shared neural network to the decentralized training units as the new shared neural network; and

providing the trained shared neural network.

11. A method for training a neural network using a reptile meta-learning method, the method comprising:

obtaining a neural network to train;

obtaining a dataset suitable for said reptile meta-learning method;

for each iteration of the reptile meta-learning method:

initializing the neural network using the method as claimed in any one of claims 1 to 3 for each task sampled, and

training the initialized neural network for said corresponding sampled task using the obtained dataset; and

providing the trained neural network.

12. The method as claimed in any one of claims 4 to 9, wherein the training of the initialized pre-trained neural network comprising training the initialized pre-trained neural network using a first training batch of the obtained dataset, wherein the first training batch is smaller than a number of features fed to said last layer of said initialized pre-trained neural network.

13. A method for using a pre-trained neural network trained in accordance with any one of claims 4 to 9.

14. A computer comprising:

a central processing unit;

a graphics processing unit;

a communication port;

a memory unit comprising an application for initializing a pre-trained neural network, the application comprising:

instructions for obtaining a pre-trained neural network having an output layer,

instructions for amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

instructions for providing the initialized pre-trained neural network.

15. Computer program comprising computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network having an output layer,

amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized pre-trained neural network.

16. A non-transitory computer readable storage medium for storing computer-executable instructions which, when executed, cause a computer to perform a method for initializing a pre-trained neural network, the method comprising:

obtaining a pre-trained neural network having an output layer,

amending the output layer of the pre-trained neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized pre-trained neural network.

17. A method for initializing a neural network, the method comprising:

obtaining a neural network having an output layer,

amending the output layer of the neural network, wherein said amending comprises updating each weight of said output layer according to a function that maximizes the entropy of the output classes probability, wherein said function depends on a parameter controlling a proportion of error of said output classes probability such as it decreases the variance of the output classes probability, and

providing the initialized neural network.

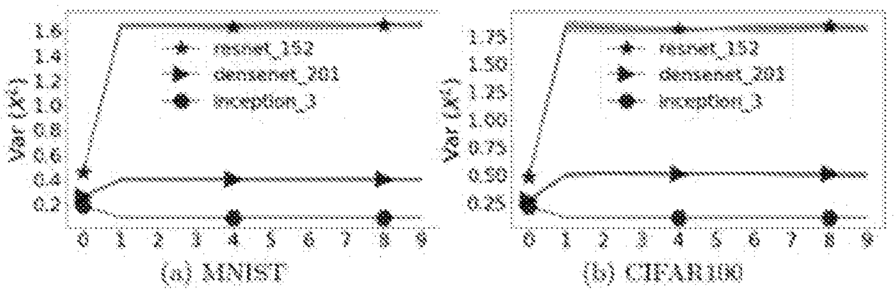


Fig. 1. Sudden initial change in variance of X^L with initial learning rate equal to 0.0001, fine-tuned on (a) MNIST and (b) CIFAR100 datasets. The horizontal axes show the training steps. In each model the augmented parameters are initialized based on preserving the variance of gradient recommended in [8]. The color shadows represent the standard deviation through training with 24 different seeds.

FIG. 1

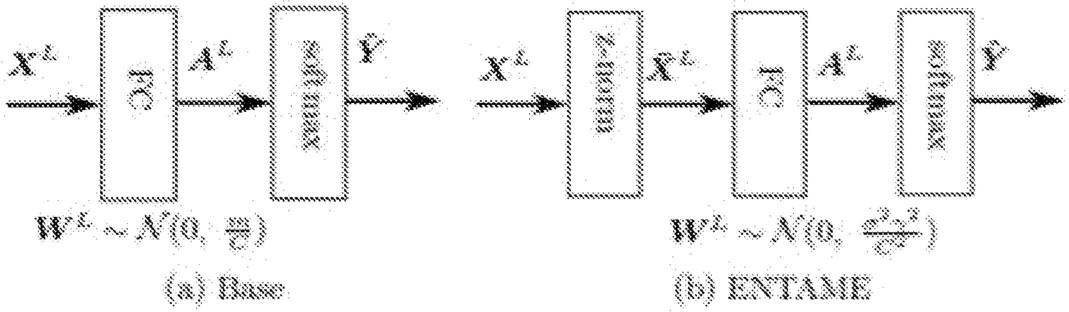


Fig. 2. FNN architecture and last layer's initialization in (a) base model and (b) ENTAME. According to [8], m is 2 for ReLU networks.

FIG. 2

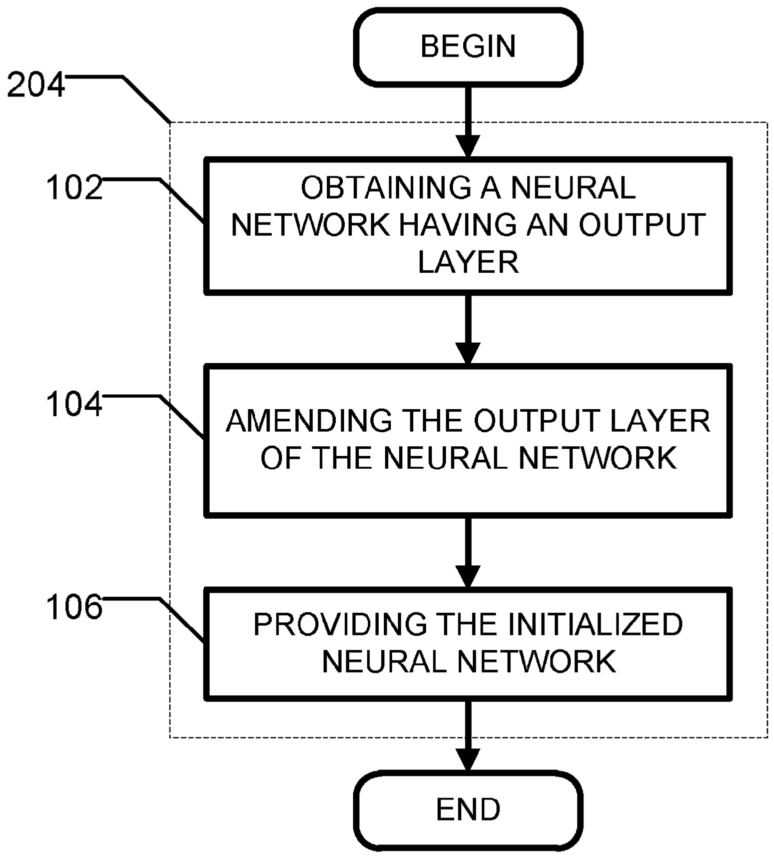

BEGIN
204
102
OBTAINING A NEURAL NETWORK HAVING AN OUTPUT LAYER
104
AMENDING THE OUTPUT LAYER OF THE NEURAL NETWORK
106
PROVIDING THE INITIALIZED NEURAL NETWORK
END

FIG. 3

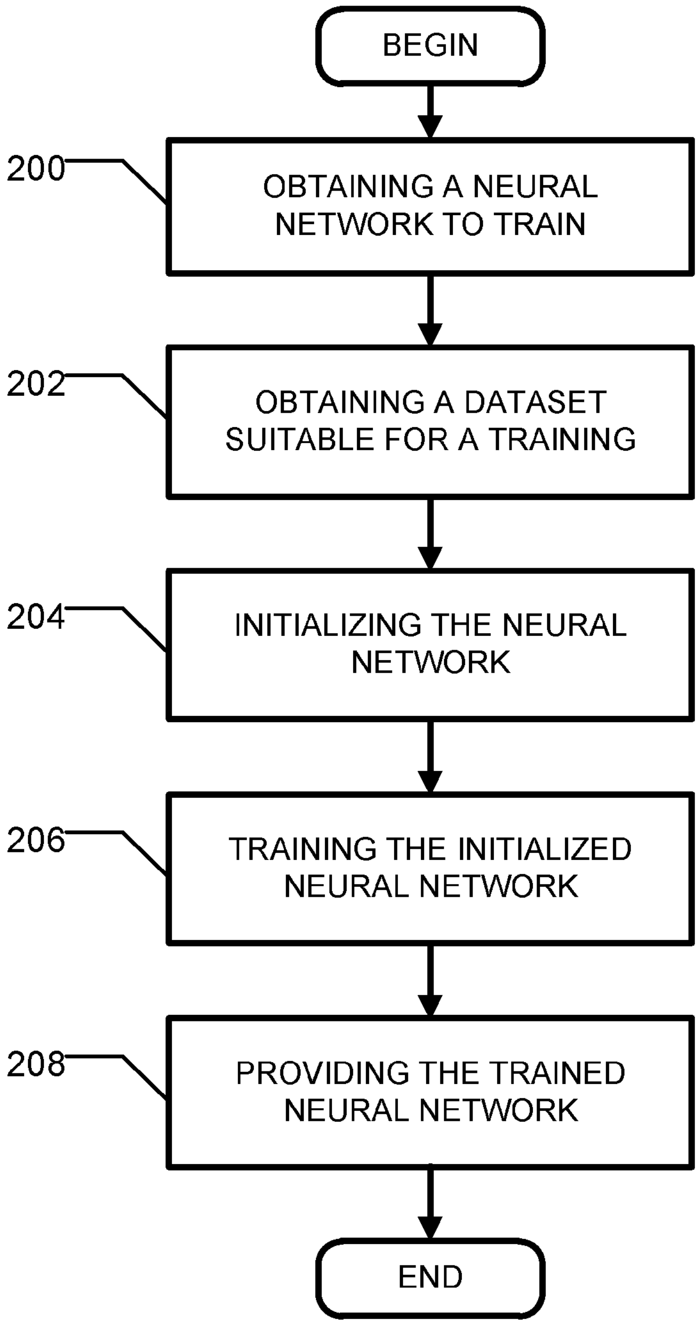

BEGIN
200
OBTAINING A NEURAL NETWORK TO TRAIN
202
OBTAINING A DATASET SUITABLE FOR A TRAINING
204
INITIALIZING THE NEURAL NETWORK
206
TRAINING THE INITIALIZED NEURAL NETWORK
208
PROVIDING THE TRAINED NEURAL NETWORK
END

FIG. 4

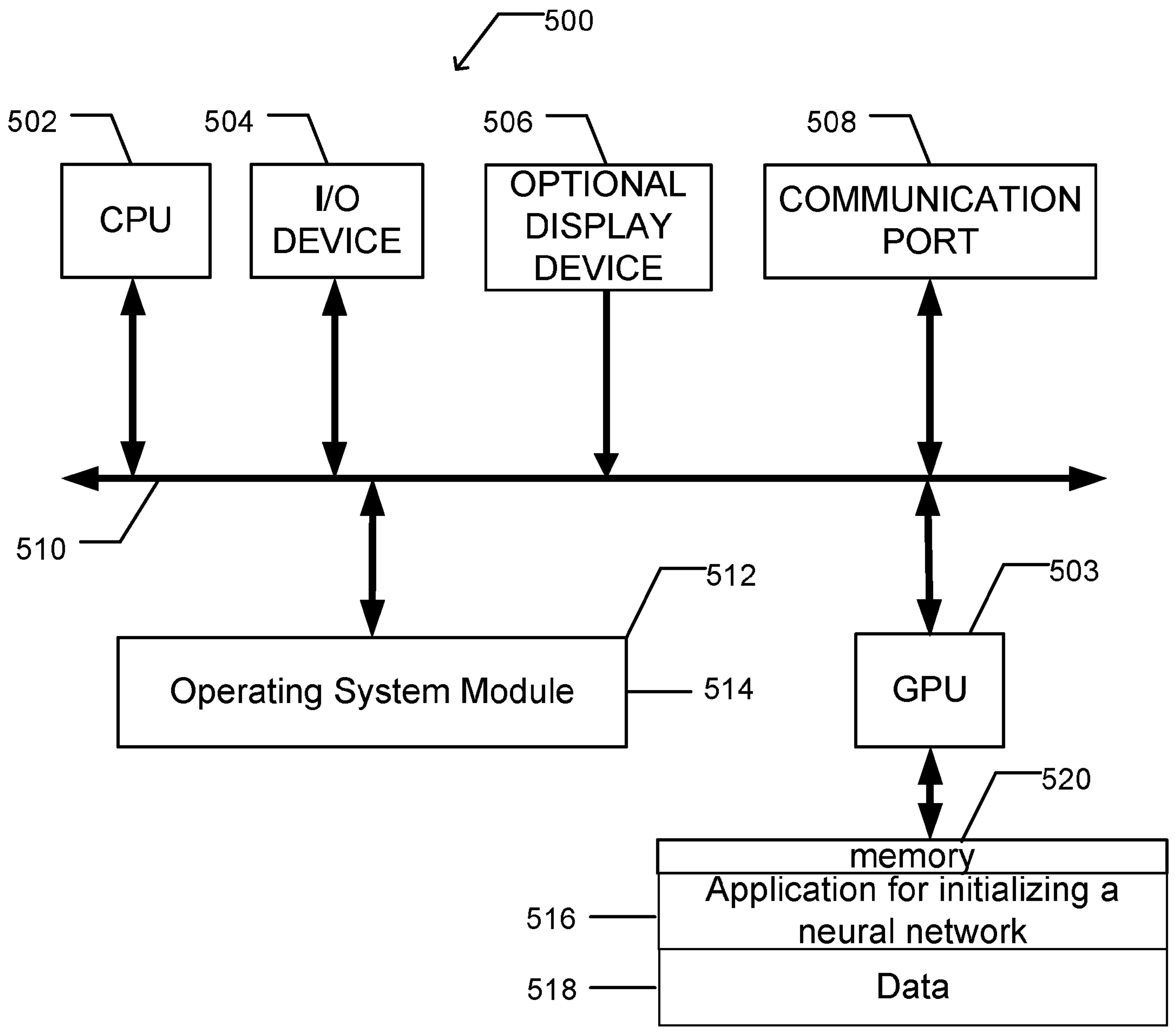
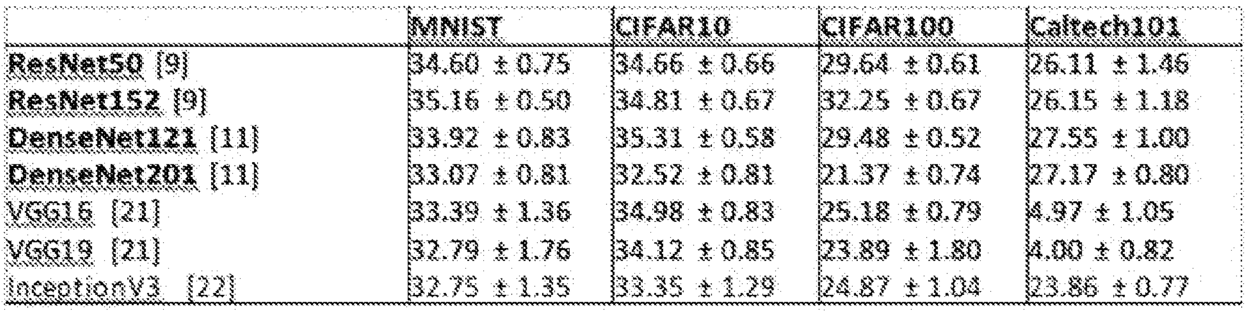

500
502
CPU
504
I/O DEVICE
506
OPTIONAL DISPLAY DEVICE
508
COMMUNICATION PORT
510
512
Operating System Module
514
503
GPU
520
memory
516
Application for initializing a neural network
518
Data

FIG. 5



	MNIST	CIFAR10	CIFAR100	Caltech101
ResNet50 [9]	34.60 ± 0.75	34.66 ± 0.66	29.64 ± 0.61	26.11 ± 1.46
ResNet152 [9]	35.16 ± 0.50	34.81 ± 0.67	32.25 ± 0.67	26.15 ± 1.18
DenseNet121 [11]	33.92 ± 0.83	35.31 ± 0.58	29.48 ± 0.52	27.55 ± 1.00
DenseNet201 [11]	33.07 ± 0.81	32.52 ± 0.81	21.37 ± 0.74	27.17 ± 0.80
VGG16 [21]	33.39 ± 1.36	34.98 ± 0.83	25.18 ± 0.79	4.97 ± 1.05
VGG19 [21]	32.79 ± 1.76	34.12 ± 0.85	23.89 ± 1.80	4.00 ± 0.82
InceptionV3 [22]	32.75 ± 1.35	33.35 ± 1.29	24.87 ± 1.04	23.86 ± 0.77

FIG. 6

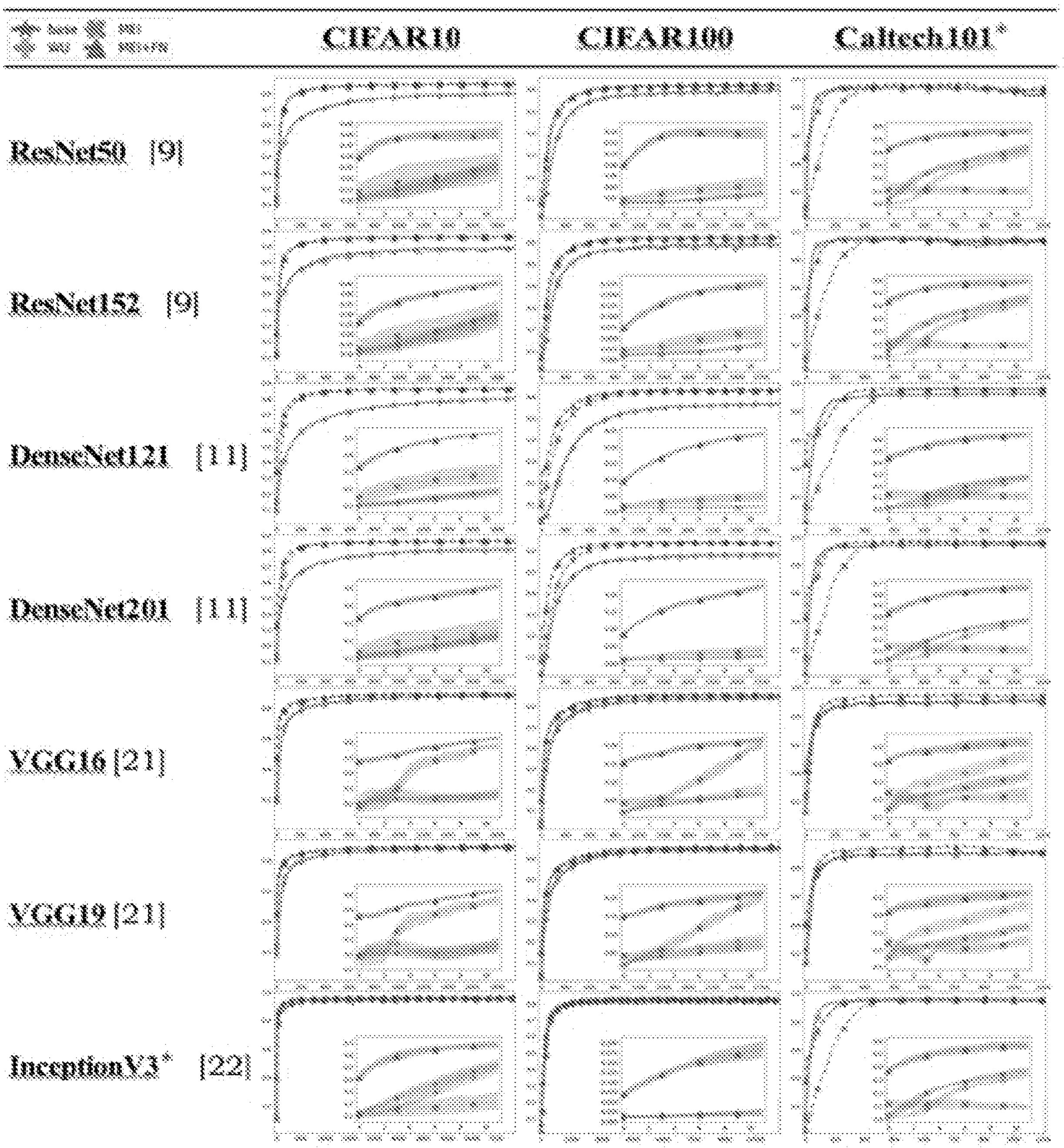

CIFAR10
CIFAR100
Caltech101*
ResNet50 [9]
ResNet152 [9]
DenseNet121 [11]
DenseNet201 [11]
VGG16 [21]
VGG19 [21]
InceptionV3* [22]

FIG. 7

	MNIST	CIFAR10	CIFAR100	Caltech101
ResNet50 [9]	10.86 ± 2.97	21.81 ± 1.10	10.19 ± 0.32	31.29 ± 1.21
ResNet152 [9]	4.52 ± 1.90	18.94 ± 1.22	9.74 ± 0.41	30.75 ± 1.09
DenseNet121 [11]	12.61 ± 1.55	28.38 ± 0.98	13.21 ± 0.26	43.95 ± 1.25
DenseNet201 [11]	17.90 ± 1.85	26.10 ± 1.08	11.99 ± 0.26	39.09 ± 1.95
VGG16 [21]	35.29 ± 2.40	29.14 ± 0.78	13.95 ± 0.38	25.86 ± 0.90
VGG19 [21]	33.04 ± 1.69	28.37 ± 1.30	13.38 ± 0.40	25.58 ± 1.31
InceptionV3 [22]	9.17 ± 1.38	33.21 ± 2.02	8.90 ± 0.38	31.94 ± 1.48

FIG. 8A

	Base	Base+WU	MEI	MEI+FN
ResNet50 [9]	79.73 ± 0.43	79.74 ± 0.76	**85.80 ± 0.35**	**85.54 ± 0.20**
ResNet152 [9]	79.18 ± 1.07	78.70 ± 1.16	**86.02 ± 0.32**	**86.01 ± 0.14**
DenseNet121 [11]	80.21 ± 0.23	80.39 ± 0.31	**86.20 ± 0.23**	**86.32 ± 0.27**
DenseNet201 [11]	81.11 ± 0.35	80.92 ± 0.46	**86.27 ± 0.20**	**86.38 ± 0.29**
VGG16 [21]	87.50 ± 0.37	87.70 ± 0.47	**88.79 ± 0.61**	**89.19 ± 0.25**
VGG19 [21]	87.94 ± 0.60	88.12 ± 0.25	**88.77 ± 0.22**	**89.12 ± 0.19**
InceptionV3 [22]	95.58 ± 0.47	95.50 ± 0.60	95.93 ± 0.27	95.91 ± 0.21

FIG. 8B

	Base	Base+WU	MEI	MEI+FN
ResNet50 [9]	59.38 ± 0.39	59.44 ± 0.41	**61.50 ± 0.46**	**61.54 ± 0.35**
ResNet152 [9]	58.71 ± 1.37	58.50 ± 0.89	**61.91 ± 0.63**	**61.85 ± 0.77**
DenseNet121 [11]	56.52 ± 0.46	56.70 ± 0.26	**62.52 ± 0.41**	**62.90 ± 0.27**
DenseNet201 [11]	58.27 ± 0.62	57.98 ± 0.58	**63.36 ± 0.15**	**63.64 ± 0.59**
VGG16 [21]	63.94 ± 0.24	63.77 ± 0.22	**65.19 ± 0.58**	**64.99 ± 0.40**
VGG19 [21]	64.30 ± 0.42	64.47 ± 0.54	65.11 ± 0.28	65.02 ± 0.21
InceptionV3 [22]	82.25 ± 0.37	82.19 ± 0.21	82.17 ± 0.32	81.75 ± 0.64

FIG. 8C

	Base	Base+WU	MEI	MEI+FN
ResNet50 [9]	89.69 ± 3.30	90.69 ± 1.12	**93.87 ± 0.61**	92.15 ± 1.61
ResNet152 [9]	93.12 ± 1.01	92.98 ± 1.04	93.19 ± 0.84	93.71 ± 1.34
DenseNet121 [11]	92.36 ± 0.67	92.41 ± 0.91	**95.13 ± 1.03**	**95.96 ± 0.19**
DenseNet201 [11]	93.95 ± 0.53	94.01 ± 0.50	94.87 ± 1.90	**96.50 ± 0.71**
VGG16 [21]	89.35 ± 1.75	91.02 ± 0.95	90.07 ± 0.68	**94.69 ± 1.19**
VGG19 [21]	90.50 ± 1.42	89.96 ± 0.90	89.70 ± 1.28	90.53 ± 4.15
InceptionV3 [22]	94.98 ± 0.54	95.07 ± 0.67	95.70 ± 0.41	95.50 ± 0.89

FIG. 9

INTERNATIONAL SEARCH REPORT	International application No. **PCT/IB2020/054350**

A. CLASSIFICATION OF SUBJECT MATTER
IPC: ***G06N 3/02*** (2006.01) , ***G06N 3/04*** (2006.01) , ***G06N 3/08*** (2006.01)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC: ***G06N 3/02*** (2006.01) , ***G06N 3/04*** (2006.01) , ***G06N 3/08*** (2006.01)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic database(s) consulted during the international search (name of database(s) and, where practicable, search terms used)
Databases: Questel Orbit, IEEExplore
Keywords: neural network, entropy, layer, federated learning, data set, decentralized training unit, error proportion, neural network share, output class probability, pre trained neural network, training neural network, variance, shared neural network

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X -- Y	US2018/0107926 A1 (CHOI et al.) 19 April 2018 (19-04-2018) *para. 4-5, 9, 60-66, 77-86, 89-95, 163, 249*	1-2, 4, 8, 13-17 -- 3, 5, 7, 9-10, 12
Y	US2018/0089587 A1 (SURESH et al.) 29 March 2018 (29-03-2018) *para. 6-10, 26-31*	5, 7, 10
Y	US2017/0148433 A1 (CATANZARO et al.) 25 May 2017 (25-05-2017) *para. 50, 54, 64, 139-141*	3, 12
Y	CA3022125 A1 (CAO et al.) 27 April 2019 (27-04-2019) *abstract*	9
A	US2019/0012594 A1 (FUKUDA et al.) 10 January 2019 (10-01-2019) *abstract*	1-17
A	EP2905722 A1 (DU et al.) 12 August 2015 (12-08-2015) *abstract*	1-17

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"D"	document cited by the applicant in the international application	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"E"	earlier application or patent but published on or after the international filing date	"&"	document member of the same patent family
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)		
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report 10 July 2020 (10-07-2020)
Name and mailing address of the ISA/CA Canadian Intellectual Property Office Place du Portage I, C114 - 1st Floor, Box PCT 50 Victoria Street Gatineau, Quebec K1A 0C9 Facsimile No.: 819-953-2476	Authorized officer Coralie Gill (819) 639-3176

INTERNATIONAL SEARCH REPORT	International application No. **PCT/IB2020/054350**

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	1 US2015/0072215 A1 (YU et al.) 12 March 2015 (12-03-2015) *abstract*	1-17

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/IB2020/054350

Patent Document Cited in Search Report	Publication Date	Patent Family Member(s)	Publication Date
US2018107926A1	19 April 2018 (19-04-2018)	CN107967515A	27 April 2018 (27-04-2018)
		CN107967517A	27 April 2018 (27-04-2018)
		KR20180043154A	27 April 2018 (27-04-2018)
		KR20180043172A	27 April 2018 (27-04-2018)
		TW201816669A	01 May 2018 (01-05-2018)
		TW201818302A	16 May 2018 (16-05-2018)
		US2018107925A1	19 April 2018 (19-04-2018)
US2018089587A1	29 March 2018 (29-03-2018)	CN107871160A	03 April 2018 (03-04-2018)
		DE102017122240A1	29 March 2018 (29-03-2018)
		DE202017105829U1	02 January 2018 (02-01-2018)
		EP3494522A1	12 June 2019 (12-06-2019)
		EP3494522B1	08 January 2020 (08-01-2020)
		EP3660754A1	03 June 2020 (03-06-2020)
		GB201715517D0	08 November 2017 (08-11-2017)
		GB2556981A	13 June 2018 (13-06-2018)
		US2019340534A1	07 November 2019 (07-11-2019)
		US10657461B2	19 May 2020 (19-05-2020)
		US2018089590A1	29 March 2018 (29-03-2018)
		WO2018057302A1	29 March 2018 (29-03-2018)
US2017148433A1	25 May 2017 (25-05-2017)	US10319374B2	11 June 2019 (11-06-2019)
		CN107408111A	28 November 2017 (28-11-2017)
		CN107408384A	28 November 2017 (28-11-2017)
		EP3245597A1	22 November 2017 (22-11-2017)
		EP3245597A4	30 May 2018 (30-05-2018)
		EP3245652A1	22 November 2017 (22-11-2017)
		EP3245652A4	30 May 2018 (30-05-2018)
		EP3245652B1	10 July 2019 (10-07-2019)
		JP2018513399A	24 May 2018 (24-05-2018)
		JP6629872B2	15 January 2020 (15-01-2020)
		JP2018513398A	24 May 2018 (24-05-2018)
		JP6661654B2	11 March 2020 (11-03-2020)
		KR20170106445A	20 September 2017 (20-09-2017)
		KR102008077B1	06 August 2019 (06-08-2019)
		KR20170107015A	22 September 2017 (22-09-2017)
		KR102033230B1	16 October 2019 (16-10-2019)
		US2017148431A1	25 May 2017 (25-05-2017)
		US10332509B2	25 June 2019 (25-06-2019)
		WO2017091751A1	01 June 2017 (01-06-2017)
		WO2017091763A1	01 June 2017 (01-06-2017)
CA3022125A1	27 April 2019 (27-04-2019)	US2019130266A1	02 May 2019 (02-05-2019)
US2019012594A1	10 January 2019 (10-01-2019)	US2019220747A1	18 July 2019 (18-07-2019)
		US10546238B2	28 January 2020 (28-01-2020)
EP2905722A1	12 August 2015 (12-08-2015)	CN104834933A	12 August 2015 (12-08-2015)
		CN104834933B	12 February 2019 (12-02-2019)
		US2015227816A1	13 August 2015 (13-08-2015)
		US9659233B2	23 May 2017 (23-05-2017)
US2015072215A1	12 March 2015 (12-03-2015)	CN203503745U	26 March 2014 (26-03-2014)
		CN203503747U	26 March 2014 (26-03-2014)
		DE202012010789U1	17 April 2013 (17-04-2013)
		DE202012010790U1	17 April 2013 (17-04-2013)
		EP2286484A1	23 February 2011 (23-02-2011)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2020/054350

Patent Document Cited in Search Report	Publication Date	Patent Family Member(s)	Publication Date
		EP2286484A4	14 November 2012 (14-11-2012)
		US2009326696A1	31 December 2009 (31-12-2009)
		US7945344B2	17 May 2011 (17-05-2011)
		US2012058377A1	08 March 2012 (08-03-2012)
		US8357464B2	22 January 2013 (22-01-2013)
		US2013059172A1	07 March 2013 (07-03-2013)
		US8492023B2	23 July 2013 (23-07-2013)
		US2013288084A1	31 October 2013 (31-10-2013)
		US8623543B2	07 January 2014 (07-01-2014)
		US2014072837A1	13 March 2014 (13-03-2014)
		US8889285B2	18 November 2014 (18-11-2014)
		US2012040233A1	16 February 2012 (16-02-2012)
		US8900743B2	02 December 2014 (02-12-2014)
		US2013266827A1	10 October 2013 (10-10-2013)
		US9065080B2	23 June 2015 (23-06-2015)
		US2015214573A1	30 July 2015 (30-07-2015)
		US9123969B2	01 September 2015 (01-09-2015)
		US2015244041A1	27 August 2015 (27-08-2015)
		US9350055B2	24 May 2016 (24-05-2016)
		US2015372346A1	24 December 2015 (24-12-2015)
		US9419303B2	16 August 2016 (16-08-2016)
		US2011202159A1	18 August 2011 (18-08-2011)
		US9666895B2	30 May 2017 (30-05-2017)
		US2016344065A1	24 November 2016 (24-11-2016)
		US9929440B2	27 March 2018 (27-03-2018)
		US2012058380A1	08 March 2012 (08-03-2012)
		US2012135292A1	31 May 2012 (31-05-2012)
		US2018248163A1	30 August 2018 (30-08-2018)
		WO2009155452A1	23 December 2009 (23-12-2009)