Article

Deep ConvNet: Non-Random Weight Initialization for Repeatable Determinism, examined with FSGM

Richard Rudd-Orthner [1]* and Lyudmila Mihaylova[2]

[1] University of Sheffield; RNMRudd-Orthner1@sheffield.ac.uk
[2] University of Sheffield; L.S.Mihaylova@sheffield.ac.uk
* Correspondence: ruddorthner@gmail.com; Tel.: (optional; +966 (0)54 192 2312)

Abstract: This paper presents a non-random weight initialization method in convolutional layers of neural networks examined with the Fast Gradient Sign Method (FSGM) attack. This paper's focus is convolutional layers, and are the layers that have been responsible for better than human performance in image categorization. The proposed method induces earlier learning through the use of striped forms, and as such has less unlearning of the existing random number speckled methods, consistent with the intuitions of Hubel and Wiesel. The proposed method provides a higher performing accuracy in a single epoch, with improvements of between 3-5% in a well known benchmark model, of which the first epoch is the most relevant as it is the epoch after initialization. The proposed method is also repeatable and deterministic, as a desirable quality for safety critical applications in image classification within sensors. That method is robust to Glorot/Xavier and He initialization limits as well. The proposed non-random initialization was examined under adversarial perturbation attack through the FGSM approach with transferred learning, as a technique to measure the affect in transferred learning with controlled distortions, and finds that the proposed method is less compromised to the original validation dataset, with higher distorted datasets.

Keywords: Repeatable Determinism; Weight Initialization; Convolutional Layers; Adversarial Perturbation Attack; FSGM, Transferred Learning, Machine Learning, Smart Sensors.

1. Introduction

Convolutional layers in neural networks have been used in Artificial Intelligence (AI) applications, and led to the use of multiple layers separated by non-linearity functions. This layering of hidden layers are said to be deep, and the successes of that architecture led to the Deep Learning research thread. It is generally accepted that convolutional layers may have translation to brain anatomy with respect to Hubel and Wiesel [1] [2]. Whom examined spider monkey's and cat's brain activity when under a light anaesthetic, while stimulating the retina with images of spots, stripes and patterns. Convolutional layers have had biological inspirations, and are generally accepted as providing hierarchical feature extraction in a deep Convolutional Network (ConvNet) [3] [4]. Later in 2012, Alex Krizhevsky's paper [5] would prove to become an influential paper, and demonstrated better than human performance within image categorization in the image net challenge using deep convolutional networks. Convolutional Neural Networks (CNN) have played an influential role ever since. Although, applications of convolutional layers provide some important human level capabilities, but they have not been embraced into mission critical applications [6] [7] [8] [9], owing in part to learning session accuracy variations, and certification of the network content as complete and correct. Currently, random initialization methods provide a cross-validation variation in accuracy that is visible over regularisation, that is to say that different random initialization states provide a variation in the prediction accuracy when cross-validated.

To this end, the previous published background work [10] to this paper, also examined repeatable determinism, but in perceptron layers only, and proposed a method,

although was tightly coupled to the perceptron layers. Although tightly coupled, that previous paper, resolved a numerical stability issue for repeatability, and proposed a non-random method for determinism, that achieved an almost equal performance in accuracy proving viability of a non-random method. That work was furthered in a journal published version, which used the Glorot/Xavier limit values [11] with the non-random method, and this time achieved an equal performance in accuracy to the random method, now proving equality in performance as an alternative augmented approach. That non-random method [11] was an augmentation to the established existing initialization method, rather than a replacement. As it proposed and alternative to the random numbers only, rather than the limit values used in those methods, and as such is a complimentary approach. The previous work [11] also had the benefit of ordering the weights after learning, into a structured form along the number of neurons axis and highlighted the correlation in structure at pixel indexes. See Figure (1, left) for the weight matrix as an image of learnt weights of the existing random method, and Figure (1, right) for the weights using the non-random method [11], both in a perceptron layer.

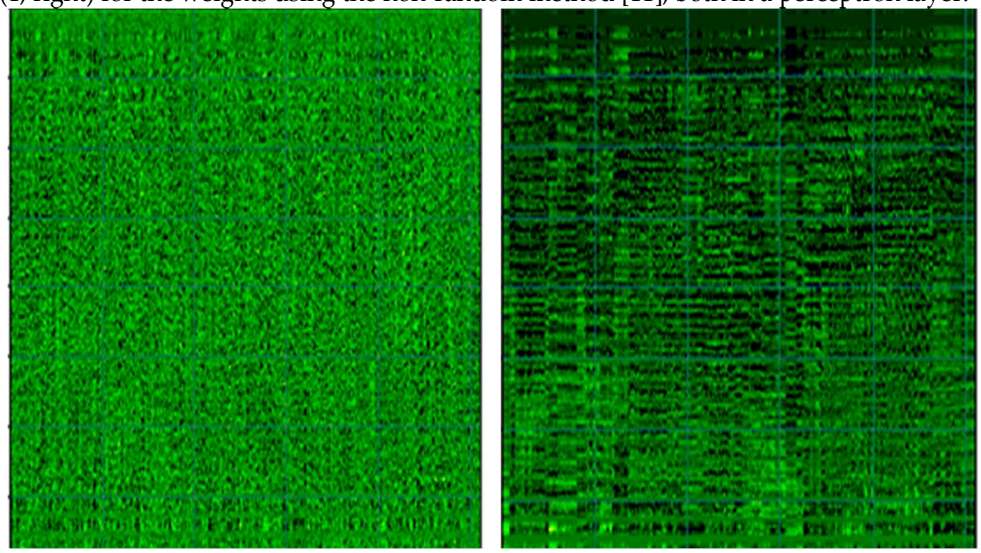

Figure **1.** Weight matrix after learning, results from a perceptron only network, left is an existing random method (Glorot/Xavier), and right a non-random method from the previous work [11].

It was noted in that previous work [11], that both weight sequences have an equivalence in performance, but the non-random method (in Figure 1 right) has a structure that may have a benefit for *rule extraction*. As the weights have been grown in an ordered sequence along the number of neurons (in the x plane), and shows activation correlations at pixel positions (in the y plane), and that helps to generalise in a rule extraction approach, as the pixel activations have been clustered to neighbouring weights. However, both those previous papers [10] [11] were confined to perceptron layers and this paper furthers that work into convolutional networks. That earlier work [10] [11] in perceptron layers did prove that an equal performance of a non-random weight initialization method was viable, and that random numbers for the initialization is not necessary. Which is the same assertion of Blumenfeld et al. too in 2020 [12], in an experiment of zeroing of some of the weights in a convolutional layer. However, the zeroing of weights is not this paper's approach. Furthermore, the order of weights in the background perceptron work [10] [11] was not significant, due to the fully connected links of nodes, and in convolutional layers the weights relate to convolved spatial filters, and so the order is significant. So the previous perceptron form [10] [11] is not applicable within convolutional layers of networks directly.

A. Structure of the paper

This papers structure is as follows: Section 1 is the introduction to the area and the background work. Section 2 introduces recent contemporary related works. Section 3 introduces the novelty of this work. Section 4 is the benchmark baseline model, and is comparable to the background work in perceptron layers [10] [11]. It presents the baseline results of that model, reusing the *critical-sections* defined from the background work [10] to re-produce a 'repeatable' learning session. Section 5 compares the existing random initialization method as a benchmark to the proposed method. Section 6 explains the number of weights and image size in each layer of the benchmark model, so the proposed method can be explained and the Glorot/Xavier limit values calculated. Section 7 shows the weight differences of the existing random method and the proposed method, before and after learning, with an illustration of what has been learnt. Section 8 presents the proposed non-random method that achieved a higher accuracy score and explains the design, and also verifies robustness to He et al. initialization limits as the current *state of the art*. Section 9 makes a comparison under adversarial perturbation attack using the Fast Sign Gradient Method (FSGM) with transferred learning. This is a convenient method for examining transferred learning compromise with a controlled distortion through the epsilon value *(ε)*. Section 10 presents the discussion of results and concludes this paper.

II. Related Work

Ding et al. in 2020 [13], proposed a *shuffle leap frog algorithm* approach, for the update and initialization with random Gaussian forms in the area of fundus lesions images. The approach presented in that paper contains random numbers, initially in a Gaussian distribution optimised with the *shuffle leap frog algorithm*, where as the approach presented in this paper, does not contain random numbers. Wang et al. in 2020 [14], proposed a *2D Principle Component Analysis (2DPCA)* approach to the initialization of convolutional networks to adjust the weight difference values to promote back propagation. This approach avoids the use of random numbers, and uses samples of the dataset instead. making it convergent to the sample data seen. However, in this paper the approach is not coupled to the sample data, only the architecture in terms of layers and is adaptive for filter geometries and layer types used. Ferreira et al. in 2019 [15], examined weight initialization using a *De-noising Auto-Encoder (DAE)* in the field of classifying tumour samples through dataset sampling, but this is also a data sample convergent approach, unlike the finite number sequence in the proposed method.

III. Contribution and Novelty

This paper's contribution is a proposed alternative initialization method, for generating a non-random number sequence for the initialization of convolutional and perceptron layer mixes, rather than perceptron layer only networks as in the previous work [10] [11]. The proposed non-random number sequence has formations of stripes and curves in that initialization state, and as such is predisposed to the application of image categorization. It is more generic and independent of the dataset utilised. The proposed method also allows earlier learning, to lower the loss quicker, and arrives at a higher performing accuracy in the first learning session, that is repeatable and deterministic, supporting a value of dependable systems in mission and safety critical applications of smart sensing. With the existing random number methods several learning sessions are required to establish which learning session's random sequence has provided the best accuracy from a variation of random initialization states. In comparison to Blumenfeld et al. [12], Ding et al. [13], Wang et al. [14] and Ferreira et al. [15] approaches, that required to adjust weights, use random numbers or sampling the dataset for convergence, the proposed method, is without data sampling or random sequences as a more general case, and is a complimentary approach to both Glorot/Xavier, and He et al. initialization limit values [16]. The proposed method substitutes only the use of random numbers for a deterministic non-random finite number sequence, and retains the number range limits

of the original methods [16]. The proposed method also after model defence with transferred learning with an FSGM dataset, has less compromised the original dataset training with larger epsilon *(ε)* value distortions of the FSGM attack. The intuition for this is that the existing random initialization method provides an unintentional noise source, which causes noise re-colorization, when combined with the noise sources in the images delaying learning. The FSGM with transferred learning approach provides a convenient method for examining the transferred learning compromise with a controlled distortion via the epsilon *(ε)* value. Although, neither method is immune to FSGM attacks, the proposed method has the advantage of been less compromised to the original dataset, with transferred learning in model defence.

IV. Benchmark Baseline Model and Method

In the previous work [10] [11], perceptron layers and the MNIST dataset [17], were used as it is familiar to researchers. So to demonstrate non-random weight initializations the same application and dataset is used, but in a convolutional form. This benchmark is also used such that comparisons can be made from the previous background work [10] [11] as well. Figure (2) presents the architecture of the benchmark model in a convolutional layer form.

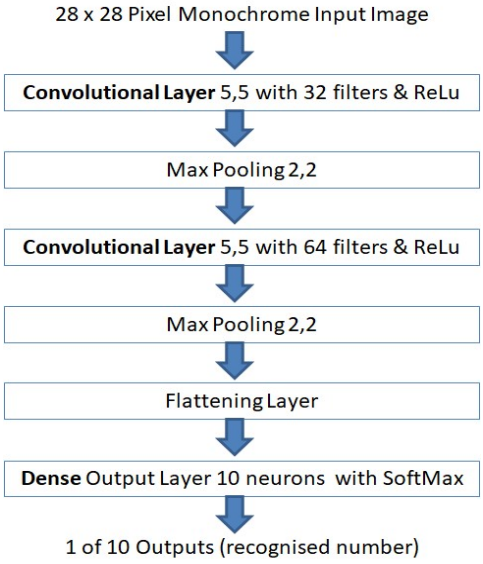

Figure **2.** Architecture of the benchmark model, by Torres.

The model architecture is the equivalent of the perceptron layer foundation work's benchmark [10] [11], but in a convolutional layer form, and as such forms a comparison bridge to the background work [10] [11]. Using the repeatable *critical-sections* defined from the background work [10], that removed a source of numerical instability in learning session variations, the convolutional layer benchmark results are in Table 1. The benchmark model is using the Glorot/Xavier random number method initialization as per its' definition within Keras by Torres [18] and has a stated accuracy of about ~97%. Although it should be noted, that there are higher scoring models using the MNIST dataset in a convolutional form, a high accuracy score of 99.8% by Kassem [19], provides little-head room to show an improvement. That model also requires 50 epochs, and that is along learning duration beyond the initial condition in this context, where the random shuffle may be a more dominant random effect. Table 1 forms the experiment control results using the Torres model [18] baseline, as the benchmark.

Table 1. Torres benchmark with the existing random initialization method results.

Random Seeded	**Epochs**	**Accuracy** *(Cross-Validation)*	**Loss** *(Cross-Validation)*

Yes	**5 Shuffled**	**96.85%**	**0.105400465**
No	5 Shuffled	96.87%	0.105643138
No	5 Shuffled	96.81%	0.112332284
No	5 Shuffled	97.13%	0.104294091
No	5 Shuffled	96.78%	0.111712694
No	5 Shuffled	96.95%	0.103740148
No	5 Shuffled	97.02%	0.102941796
No	5 Shuffled	96.67%	0.110733353
No	5 Shuffled	96.94%	0.104514159
No	5 Shuffled	96.99%	0.103553012
Averages:		96.901%	0.106486514
Yes	**1 Shuffled**	**88.98%**	**0.352222472**
No	1 Shuffled	90.44%	0.319229484
No	1 Shuffled	91.79%	0.294954896
No	1 Shuffled	91.31%	0.300388008
No	1 Shuffled	90.71%	0.309825629
No	1 Shuffled	91.9%	0.288508445
No	1 Shuffled	91.95%	0.281219631
No	1 Shuffled	91.34%	0.289107263
No	1 Shuffled	91.81%	0.290958554
No	1 Shuffled	91.05%	0.295696706
Averages:		91.128%	0.302211109
Yes	**1 No Shuffle**	**89.3%**	**0.34017086**
No	1 No Shuffle	89.42%	0.334787607
No	1 No Shuffle	89.29%	0.337718517
No	1 No Shuffle	89.5%	0.333430499
No	1 No Shuffle	89.8%	0.3234815
No	1 No Shuffle	89.84%	0.324871719
No	1 No Shuffle	90.02%	0.31683287
No	1 No Shuffle	89.17%	0.342095852
No	1 No Shuffle	89.34%	0.338885903
No	1 No Shuffle	90.07%	0.314374834
Averages:		89.575%	0.330665016

The results in Table 1 show the benchmark results with the full 5 epochs shuffled, and reaching the approximate stated accuracy of that model, but also shows just 1 epoch, as the 1st epoch after initialization is of interest. Those 1st epoch runs are also in two forms which are: with or without the shuffle, as there are two random effects (weight initialization and shuffle order), and this allows those effects to be distinguished. So that when no shuffle is used, there is only the effect of the random weight initialization, and the shuffled version shows the equivalence to the 5 epoch results where only shuffle forms are appropriate, to have reordering in each of the 5 epochs. When the random number generator uses a *seeded value* (shown in bold), the results are completely repeatable between learning sessions. But also more results have been added, not using the seeding of the random number generator, to show the accuracy variation that different random number initialization sequences have, that are visible over regularisation. As random number generator seeding has two effects: the weights initialization values, and also the shuffle reorganization of the dataset. So for this reason, Table 1 shows results from three configurations: 5 epochs with shuffles, a single epoch with the shuffle and a single epoch with no shuffle as that is the effect of the random number initialization in the weights alone, disregarding the original dataset order. The results in Table 1. show an average accuracy of 89.575% in a single epoch with no shuffle, an average of 91.128% (+1.553%) when a shuffle is used, and an average accuracy of 96.901% (+5.773% greater) with the use of 5 shuffled epochs). The Table 1 results forms the benchmark performance of the Torres model [18].

V. Comparison of the Benchmark with the Proposed (Non-Random) Method

The presented proposed non-random initialization method achieves 93.28% in a single epoch with no shuffle, +3.705% better than the existing random method. 93.77% accuracy is achieved when a shuffle is used in a single epoch, again using the proposed non-random method, which is a 2.642% gain over the benchmark of the existing random method. Then 97.5% (+0.599% over the existing random method) when 5 epochs are used. Those results are within Table 2, shown in bold and are repeatable and deterministic.

Table 2. Gains of the proposed non-random initialization method over the existing benchmark.

Epochs	Accuracy *(Cross-val.)*	Loss *(Cross-val.)*	Gains over existing (random) method
5 Shuffled	**97.5%**	**0.085728347**	**+0.599%** *(Cross-validation gain)*
4 Shuffled	97.11%	0.097854339	N/A
3 Shuffled	96.85%	0.114757389	N/A
2 Shuffled	95.96%	0.141269892	N/A
1 Shuffled	**93.77%**	**0.230065033**	**+2.642%** *(Cross-validation gain)*
1 No Shuffle	**93.28%**	**0.230725348**	**+3.705%** *(Cross-validation gain)*

From Table 2 the best gains are achieved in the first epoch, which is the epoch that occurs after the weight initialization. Less relative gain is achieved in further epochs, as the learning is occurring longer after the initialization in the subsequent epochs, diminishing its' influence but inheriting the earlier learning. An interpretation is the subsequent learning is more equivalent but earlier learning has a higher benefit as it may be using more of the dataset more effectively in the first epoch. This could be because the initial learning in the proposed initialization method has stripes and curves rather than dots and speckles, and is more predisposed to the application of image categorization for feature extraction. That is also consistent with the biological intuitions of Hubel and Wiesel [1] [2]. The approach is also repeatable and deterministic, as a value of dependable systems for smart sensing in mission critical applications. As the model derived from training is repeatable, it supports testing and verification with different environments and conditions in development and testing.

VI. Understanding the Weights and Image sizes

To understand the presented proposed non-random weight method, the structure of the weights and the image sizes, in the benchmark model need to be understood clearly. To understand how the weights are used is critical, to understanding how convolutional layers use the weights and affect the image size. As this is quite different from perceptron layers, and as in the findings of the journal version of the perceptron repeatable determinism paper [11]. When using the Glorot/Xavier limits [20], the results were enhanced with the non-random method [11], over the results initially presented in the conference paper [10]. This was because of the tolerance and matching to the model architecture in terms of propagation values and limits. Convolutional layers use the weights for the filters, and not the pixels directly, as such their dimensions of each filter is: width by height, then by depth (channels), where that depth may be inherited from the previous convolutional layer's filters. Perceptron layers in a ConvNet use the image size by previous layer's filters, as the previous layers filter would have translated to depth (channels) in activations, and those activations are connected to each neuron. Thus, the Gloror/Xavier limits need to be calculated, and Figure 3 illustrates the adjustment of image and weights sizes necessary in the Torres [18] benchmark model.

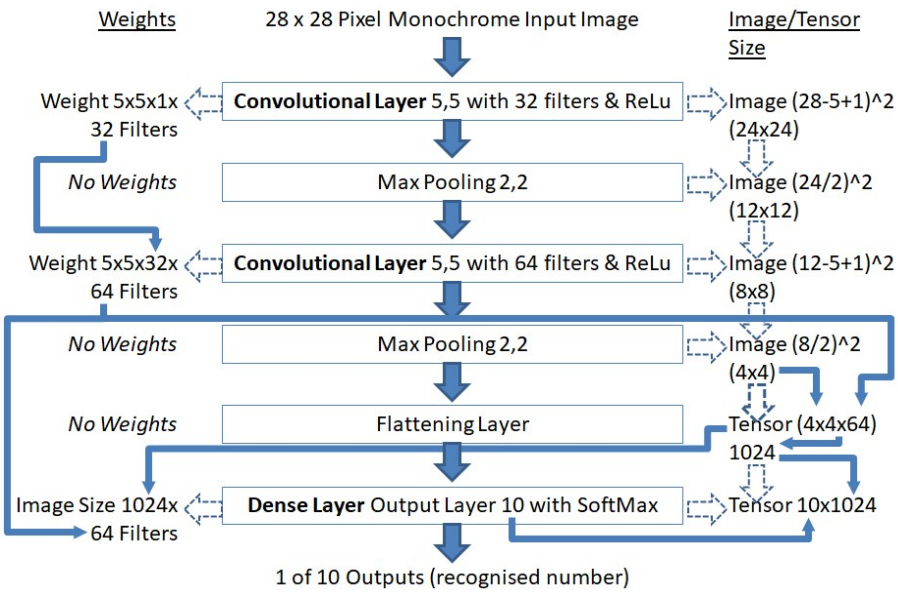

Figure **3.** Weight & image size adjustments in the Torres benchmark model.

Extracted from Figure 3, Table 3 shows the weights and image sizes in each layer in the benchmark model, such that the Glorot/Xavier limit values can be calculated to support the number ranges in the Glorot/Xavier approach, and later in the He et al. approach too.

Table 3. Weights and image sizes in each layer of the benchmark model by Torres.

Layer	Filter/Pool/Neurons	Depth	Image/Tensor Size	Weights
Input 28x28x1	N/A	1 (B/W image)	28x28 (748)	N/A
Conv Layer 1	**5 by 5 by 32 filters**	**1**	**24x24 (576)**	**800**
Max Pooling	2 by 2	32	12x12 (144)	N/A
Conv Layer 2	**5 by 5 by 64 filters**	**32**	**8x8 (64)**	**51200**
Max Pooling	2 by 2	64	4x4 (16)	N/A
Flatten Layer	N/A	1	1x(4x4x64) 1024	N/A
Dense Layer	**10**	**1**	**10x1024 (10240)**	**10240**

To calculate the Glorot/Xavier limits, See Equations (1), (2) and (3), but note that the calculated values have been rounded to 8 decimal places (as a rule of thumb for precision [21]), and are used as such and are shown as such in the Equations (1), (2) and (3):

$$ConvLayer1 = \sqrt{\frac{6}{(5*5*1 + 5*5*1x32)}} = 0.08528029, \quad (1)$$

$$ConvLayer2 = \sqrt{\frac{6}{(5*5*1*32 + 5*5*64)}} = 0.05 \quad \text{and} \quad (2)$$

$$DenseLayer = \sqrt{\frac{6}{(4*4*64 + 10)}} = 0.07617551. \quad (3)$$

Alongside the Glorot/Xavier limits, the structure of the weight initialization sequence also requires to have positional stripes and curves variations in each filter. Those stripes and curves positional variations, are also to be aligned to feature extraction, in a Hubel and Wiesel [1] [2] stripes intuition. That structure, is ideally to be more allied to edge detection than a random value placement as the start condition. This is to predisposed the initial condition more generically to the application of image classification. Thus outperform the random methods, by inducing earlier learning, with less unlearning of the initial state, and using the early data in the dataset more effectively.

VII. Comparison of the Weights before and after Learning

In Figure (4, left), is the existing random method benchmarks weights before learning of the first convolutional layer, and in Figure (4, right), are the same filter weights but after learning.

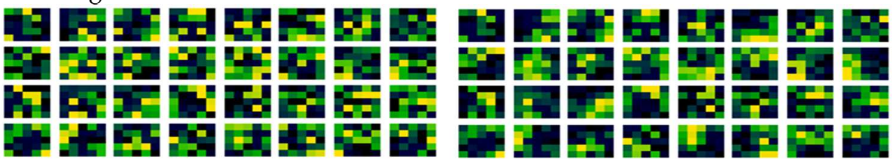

Figure **4.** Initial and learnt weights in the first conv layer with the existing random method.

In Figure (4), the existing random initialization method form has a speckled appearance in each filter, and it may also be noticed that there are high similarities in the filters, between the before and after learning. Suggesting that the initial condition has a dominant affect in the subsequent learning even over a large dataset. Which is perhaps why the variation in accuracy with different random sequences are visible over regularisation. As the initial speckled positions in the filter relate to positions in image features, affecting the resultant performance of that filter from the outset. Meaning that the filter organization is important to the performance, rather than just statistical equivalence. Furthermore, looking carefully, adaption can be notice between the before in Figure (4, left) and after learning in Figure (4, right). Illustrated by when the initial weights are subtracted from the learnt weights, the adaption can be seen more clearly in Figure (5).

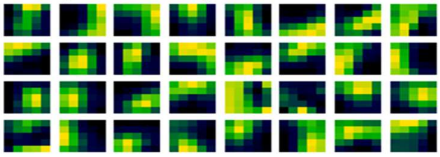

Figure **5.** Learnt filter weight adaption updates of the existing random method.

Consistent with Figure (5), convolutional filters examined after learning may be expected to have stripes, spots and perhaps curves, that may be used in edge detection of feature extraction, that will be hierarchically connected and organized to form shapes in later layers. Also convolutional layers, offer the ability to change the image resolution, at which a filter and the subsequent layers operate. Combined with this, striped and curved patterns may be more conducive in shape detection, that vary from filter to filter. In filter generation stripes and curves from a modulation may orientate directions with different selections of width bounding values as they wrap by the maximum width value of the filter too. So considering the proposed method, in Figure (6, left) is the proposed non-random method weight filter initialization, again produced for the first convolutional layer. In Figure (6, right) is those same weights of the filters but after learning has been conducted, as the update to those filters.

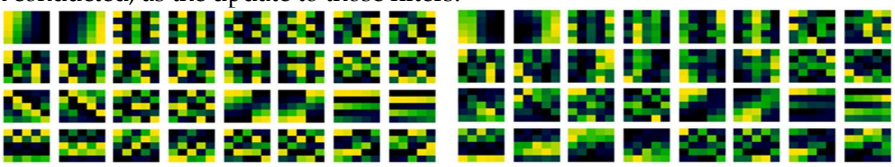

Figure **6.** Initial and learnt weights in the first conv layer with the proposed non-random method.

It may be noted that there are also some similarities between the before in Figure (6, left) and after learning in Figure (6, right) with the proposed method, reinforcing the dominance of the initial condition assertion. But those similarities are less, and this method is higher performing then the existing random method. When subtracted to expose the learnt adaption in Figure (7), there are similarities with Figure (5). The sub-

traction of the before learning initialization from the learnt weights, might be thought of as what has been learnt, but is actually more formally what has been adapted from the initial condition in nudges of values in optimisation iterations.

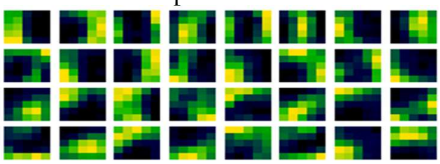

Figure 7. Learnt filter weight adaption updates with the proposed method.

Thus it may also be noticed that there are also some similarities in what has been relatively adapted, between the random and non-random methods in Figure (5) and Figure (7). In fact some equivalent relative filter adaption's between them can be noticed.

So it follows, that the initial condition therefore has an effect originating from their arrangement from the outset of learning, and different arrangements will affect the after learning result. However, the relative adaptation from the methods (both random and non-random) have some equivalence indicating that it is a similar design implementation arrived at from the same dataset, rather than a different implementation. Which is reassuring as the dataset, model architecture and algorithms are unchanged, and it is the initialization alone that has been modified, and that is what is responsible for the increased accuracy.

VIII. The Proposed (Non-Random) Method

To explain the design of the proposed method, and how it is derived. The intention was that the proposed initialization method would make filter arrangements that have stripes, spots and curves that are different in each filter, such that the subsequent learning adapts to the dataset quicker being pre-disposed to the application. These arrangements are also different in each filter providing a filter diversity of edge detection, in different orientations, as the positional variation of values is important as it relates to a filter sweeping across the pixels. If the modulation of arrangement position is based on a filter cell multiple (like two) as in a matrix, then alternations may relate to stripes in a 2D matrix when different maximum width values are used, that would be controlled from a hyper-parameter for that layer: filter width. This would also connect the filter to the resolution in that filter in the model layer. That striping will then be controlled by the convolutional layer's hyper-parameters (filter height, width and number of filters). As also the stripe orientation in different filter arrangements is important, a diversity is required in each filter over the number of filters.

An algorithm published in the papers [22], that produced a least adjacent arrangement based on a modulation of two for dataset shuffling, has some attractive properties to this application. It was originally an alternative to the established random dataset shuffle approach. This non-random shuffle approach, rearrange the dataset to produce a sequence with the first half of the input, that was output at a stride of two and then in filled the gaps with the remaining vector, also at a stride of two but in reverse order. This resulted in a placement with smallest and largest numbers neighbouring each other at the start of the vector. Figure (8) shows, the number sequence with a *vector length* of 10, and with the unordered in row 1 and the reordered in row 2.

Number sequence with a vector length of 10 unordered and reordered									
0	1	2	3	4	5	6	7	8	9
0	9	1	8	2	7	3	6	4	5

Figure 8. Least neighbour shuffle with a vector length of 10.

This process can be repeated iteratively as an in place operation, and provides an number of filter variations that are deterministic. When iteratively repeated as an in place operation, the original sequence order will repeat nominally, at a maximum number of iterations of: *vector length -1* or less, and is an iterative numerical sequence. See Figure (9), where row 1 and row 10 (the unordered, and 9th iteration of reordering), are the same due to that repeating nature of the numerical sequence.

Number sequence with a vector length of 10, unordered and then 9 reordering iterations									
0	1	2	3	4	5	6	7	8	9
0	9	1	8	2	7	3	6	4	5
0	5	9	4	1	6	8	3	2	7
0	7	5	2	9	3	4	8	1	6
0	6	7	1	5	8	2	4	9	3
0	3	6	9	7	4	1	2	5	8
0	8	3	5	6	2	9	1	7	4
0	4	8	7	3	1	5	9	6	2
0	2	4	6	8	9	7	5	3	1
0	1	2	3	4	5	6	7	8	9

Figure **9**. Least neighbour shuffle over 1st unordered and 9 reordering iterations.

This algorithm is to be used as a readdressing method of the initialization weights pertaining to the filters to provide stripes and curves in that initialization method. Referring to Figure (9), it can be noted that there is a sliding shift in value placement with iterations, and that those shifts occur in both diagonal slants, providing an influence from height and width bounding differences of the selected height and width of the filters.

Another attractive quality of this reordering algorithm, is illustrated with a linear ramp of values. Where a reorganisation from a modulated ramp to a saw tooth can occur in addressing shown in this test case at *vector length -2* number of iterations, and is the iteration before the sequence is repeated, at *vector length -1* number of iterations. Figure (10) shows, the reorganisation from the 1st iteration sequence in black and the penultimate iteration (*vector length -2*) in red, which is the iteration before the repeat for a *vector length* of 10 in the sequence at (*vector length -1* iterations).

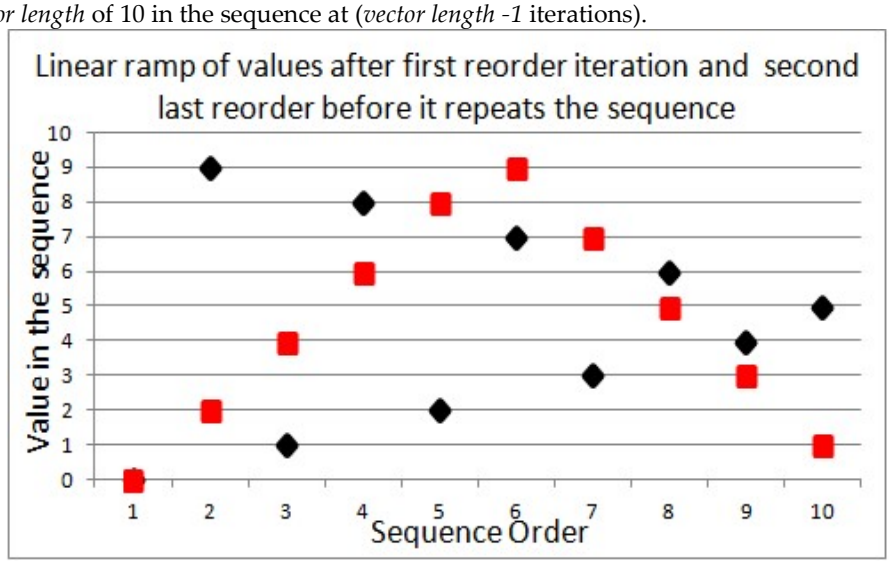

Figure **10**. Least neighbour re-addressing at iterations: 1 and 8.

As the sequence repeats, that algorithm also has a maximum number of combinations up to *vector length -1*, although some times less iterations depending on the *vector length* used, and a subject for further research is to extend the number of filters on offer.

The algorithm from the papers [22], is further enhanced here to deal with odd number length vectors. The formulas for the algorithm are in Equations (4), (5), (6) and (7) as amended and use zero indexing: The output of the function *shuffle* is a set y, with a re-ordering of the set x as defined by the subscripts α and β. As in logically moving from address position α to position β. The *shuffle* function declared in Equation (4) is recursive, where the number of recursions is defined by the number of filters (*nFilter*) in that layer as a subscript ($_{LayerNo}$).

$$y = shuffle\left(x, nFilter_{(LayerNo)}\right) \ . \tag{4}$$

The *shuffle* function is defined in Equation (5) using equation guards, that if the $nFilter_{(LayerNo)}$ (or i) is greater than one filter, then the recursion is still made with the subscript reordering (α), while decrementing the i value by 1 on each recursion iteration. These recursions occur until the last recursion that will return the unordered subscript location (β) which will then be subject to all the subscript reorder recursions as the function shuffling iterations prior are applied to complete the shuffle definition pattern.

$$shuffel(x_{\{\beta\}}, i) = \begin{cases} shuffel(x_{\{\alpha\}}, i-1) & if\ i > 1 \\ x_{\{\beta\}} & otherwise \end{cases}, \tag{5}$$

where, the index subscript set for 'β' is in Equation (6), and where n defines the set size (and again is zero indexed):

$$\beta_{\{0..n-1\}} = \left\{ \beta \in \mathbb{N} \middle| \begin{array}{c} \{0 \le 2\beta \le 2\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)\} \cup \\ \{1 \le (2\beta+1) \le \left(2\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)+1\right)\} \cup \\ \{ = \begin{cases} (n-1) & if\ n(mod\ 2) \ne 0 \\ \{\} & otherwise \end{cases} \} \end{array} \right\}, \tag{6}$$

and where the last set union is included if the set length *(n)* is an odd number defined by *n (mod 2)≠0*, as the modulo division of the set length *(n)* by modulus 2. This was the amendment from the papers [22].

The index subscript set for 'α' is in Equation (7), and naturally has the same set size *(n)* and forms the initial order for the re-ordering displacement subscripting in the shuffle pattern:

$$\alpha_{\{0..n-1\}} = \left\{ \alpha \in \mathbb{N} \middle| \begin{array}{c} \{0 \le \alpha \le \left\lfloor \frac{n}{2} \right\rfloor - 1\} \cup \\ \{n-1 \le n-1-\alpha \ge \left\lfloor \frac{n}{2} \right\rfloor\} \cup \\ \{ = \begin{cases} \left\lfloor \frac{n}{2} \right\rfloor & if\ n(mod\ 2) \ne 0 \\ \{\} & otherwise \end{cases} \} \end{array} \right\}, \tag{7}$$

In illustration, x at this point can be thought of as a sequence of numbers as in Equation (8), and when *n=10* will provide the address shuffle sequences as in Figure (9). But however, the intended set values for x will be further defined later in this paper in the *valSet* function:

$$x_{\{0..n-1\}} = \{0..(n-1)\} \text{ and where } n \text{ is the length of the tenser.} \tag{8}$$

This algorithm will always have the same value in the first location, and although this was not significant in the dataset shuffle application in the papers [22], it is significant in this application of convolutional filters. Experiments were conducted with pre-placement shift offsets in the data, and also with a data direction alternation of this algorithm. These experiments proved to not be as high performing, although did provide a higher number of unique filters.

As with convolutional layers the order of filter values is significant, so a pre-alternation of the data is conducted instead. So that every second filter is reversed (or flipped) and the memory is addressed through width, height and depth for the odd filter numbers, and vector address reversed as then depth, height, width in reverse order

for the even filters. This provides two different filters of alternating direction placement for the same shuffle iteration, doubling the number of unique filter iterations on offer, with also a crucial disruption variation to the first position value as its' primary intention.

As well as the address order placement, the value distribution of the values within the initialization sequence can be significant, as images are less likely to be uniformly distributed in pixel values. Experiments were conducted with linear ramps, as these had been the highest performing in perceptron layers [10] [11]. In these experiment cases the application of a linear ramp was higher performing with dense perceptron layers, and sinusoidal slopes in the convolutional layers. This might be because of the *cos(x)* content is a partial distribution of a *sine* function (bath-tub), or at least it's distribution has a match to convolutional layers and the image data it processes. As such the sinusoidal slope and linear ramp are selected based on the layer type within the network model architecture, because of the direct image processing in the convolutional layers.

The formulas that call the addressing shuffle function (*shuffle*) is in Equations (9) - (30), and includes the addressing alternation in the definition. Note that it also calls a function (called *valSet*) that provides a response based on the value ratio (*cnt*/*m*), and network layer type (*t*) to select a sinusoidal slope or linear ramp value form, altering the value distribution between layer types. The initialization tensor length ($InitTensor_{Length}$) for a layer is based on a number of filters *(maxFilters)* of that layer and the number of weights in the filter (*m*) is as in Equation (9):

$$InitTensor_{Lengt} = m \cdot maxFilters \ . \tag{9}$$

Where, each convolutional layer's filter tensor length can be calculated as (*m*), which is from the convolutional layer's filter: height, width and depth as a 3D matrix size and is defined as in Equation (10). The value of *m* provides a maximum scale value (as the denominator of a ratio) for a numerator value *cnt* (as a progressive weight *count* in the filter), within each filter of a convolutional layer, Where *m-1* is the maximum value that the value that *cnt* as part of a ratio can achieve as defined in Equation (11).

$$m = Height_{Filter} \cdot Width_{Filter} \cdot Depth_{Filter} \ , \tag{10}$$

$$cnt = \{0..m-1\}, indexed\ as: cnt_{(Widt\ _{Filter}, Heigh\ _{Filter}, Depth_{Filter})} \ , \tag{11}$$

Note that the weight calculation algorithm of the dense layer is dependent on the other layers, and as such the dense layer weight calculation algorithm may vary depending on the prior layer type in the architecture. This is because the activations could be the number of neurons of a proceeding dense layer, and in that case the subsequent shuffle reordering and flips may not be necessary as the layers are fully connected as in the papers [10] [11]. However, if there is a proceeding convolutional layer, then the activations map to the receptive fields of the convolved image filters, which is the case in this benchmark model. So in this case the height and width are the image size and the depth is inherited as the channel depth (or filters from the previous layer), as in Equation (12), and the value set of *cnt* is in Equation (13).

$$m = Height_{Image} \cdot Width_{Image} \cdot Depth_{image} \ , \tag{12}$$

$$cnt = \{0..m-1\}, indexed\ as: cnt_{(Widt\ _{image}, Height_{image}, Depth_{image},)} \ . \tag{13}$$

Where *nFilter* (or *nNeurons* for a dense layer) and *nDepth* are number sets that are zero indexed, and the limit of *nFilter* is *maxFilters-1* (or *maxNeurons-1* for a dense layer), and *nDepth* is *maxDepth-1* of which those sets are defined as in the Equations (14), (15) and (16).

$$nFilter = \{nFilter \in \mathbb{N} \mid 0 \leq nFilter < maxFilters\} \ , \quad for\ a\ convolutional\ layer, \tag{14}$$

$$nNeurons = \{nNeurons \in \mathbb{N} \mid 0 \leq nNeurons < maxNeurons\} \ , \quad for\ a\ dense\ layer\ and \tag{15}$$

$$nDepth = \{nDepth \in \mathbb{N} \mid 0 \leq nDepth < maxDepth\} \ . \tag{16}$$

As such each filter vector (or *neurons vector* in a dense layer) will be a vector of values with a vector length *MaxFilters (or MaxNeurons)*, and with repeating values in the set in Equation (17):

$$nSet = \{nSet \in \mathbb{N} \mid 0 \leq cnt < m - 1\}. \quad (17)$$

The initialization tensor is a 4D tensor matrix of a *nHeight* 3D matrix tensor that comprises a 2D matrix of *nWidth*, and that is a *nDepth* 1D vector of *nFilter* length as the subscripts illustrated in the Equation (18):

$$InitTensor = {}_{(nHeig\ \ ,nWidt\ ,nDe\ \ \ ,nFilter)}\ , \textit{for a convolutional layer}. \quad (18)$$

In this test case, the dense perceptron layers' initialization tensor is re-indexed from the receptive field mapping of the convolved filters in a previous layer to a matrix of *activations* and *neurons* as the subscripts in the Equation (19):

$$InitTensor_{(Activations,Neurons)} = {}_{((nHeight,nWidt\ ,nDepth),nNeurons)}\ , \textit{for a dense layer.} \quad (19)$$

Where the set for the subscripts *nHeight* and *nWidth* are given as in Equations (20) (21), and is the filter geometry in convolutional layers, or the image geometry mapped to the convolved filters in dense perceptron layers, following a convolutional layer:

$$nHeight = \{nHeight \in \mathbb{N} \mid 0 \leq nHeight < maxHeight\}\ , \quad (20)$$

$$nWidth = \{nWidth \in \mathbb{N} \mid 0 \leq nWidth < maxWidth\}\ . \quad (21)$$

A convolutional layer illustrative example of the *cnt* values (convolved filter addressing) is given in Equation (22), in the case of 5 filters with a channel depth of 4 and the filter dimensions of width 3 and a height of 2.

$$set_{of_{cnt}} = \left\{ \left\{ \left\{ \begin{matrix} \{0,0,0,0,0\}, \\ \{6,6,6,6,6\}, \\ \{12,12,12,12,12\}, \\ \{18,18,18,18,18\} \end{matrix} \right\}, \left\{ \begin{matrix} \{1,1,1,1,1\}, \\ \{7,7,7,7,7\}, \\ \{13,13,13,13,13\}, \\ \{19,19,19,19,19\} \end{matrix} \right\}, \left\{ \begin{matrix} \{2,2,2,2,2\}, \\ \{8,8,8,8,8\}, \\ \{14,14,14,14,14\}, \\ \{20,20,20,20,20\} \end{matrix} \right\} \right\}, \left\{ \left\{ \begin{matrix} \{3,3,3,3,3\}, \\ \{9,9,9,9,9\}, \\ \{15,15,15,15,15\}, \\ \{21,21,21,21,21\} \end{matrix} \right\}, \left\{ \begin{matrix} \{4,4,4,4,4\}, \\ \{10,10,10,10,10\}, \\ \{16,16,16,16,16\}, \\ \{22,22,22,22,22\} \end{matrix} \right\}, \left\{ \begin{matrix} \{5,5,5,5,5\}, \\ \{11,11,11,11,11\}, \\ \{17,17,17,17,17\}, \\ \{23,23,23,23,23\} \end{matrix} \right\} \right\} \right\} \textit{. for Conv layer} \quad (22)$$

The Equation (22) shows vectors of 5 filters are provided for each of the 4 depth channels, and those in turn are for each of the filter dimensions of width (3) and height (2). The values of *cnt* are counting through values of width then height and then depth as an indexing order and at this point all filter weight values of *cnt* are matching, as latterly they will be shuffled and alternated towards the final filter permutations for linier striping.

The tensor of values of *cnt* with respect to *m* are applied to the slope alternatives that are as in the Equation (23), and that can select between the convolutional and perceptron layer type *(t)*, also with the chosen calculated limit value as *(l)* for either uniform He et al. or Glorot/Xavier limit values.

$$valSet(cnt, m, l, t) = \begin{cases} cos\left(\frac{cnt}{m-1}\pi\right) l & if\ t = Convolutional \\ \frac{cnt}{m-1} 2l - l & if\ t = Perceptron \end{cases} . \quad (23)$$

The *valSet* function provides two value sequence types depending on the layer type, this provides two distributions of values that are either uniformly distributed for dense layers, or bathtub distributed in nature for the convolutional layers.

As the shuffle reordering does not shift the address of the first filter value every second filter is reversed in order, for convenience this is done in a matrix transpose form, and forms a transposed matrix as in Equation (24).

$$\text{TsprMat}_{(nFilter,\ nDepth,\ nWidth,\ nHeight,)} = T_{(initvalues_{nHeight,\ nWi\quad,\ nDepth,\ nFilter})} \tag{24}$$

Every second filter is order reversed (flipped) as a contiguous vector tenser as in Equation (25):

$$flipMat = \begin{cases} TsprMat_{(nFilter,(m-1)..0)} & (nFilter+1)\ (mod 2) = 0 \\ TsprMat_{(nFilter,0..(m-1))} & otherwise \end{cases}. \tag{25}$$

Then the vector is re-indexed back to the matrix subscripts as in Equation (26):

$$TsprMat2[nFilter] \ = \ flipMat\ (depth, width, height) \tag{26}$$

Again for illustration, if the matrix (*TsprMat2)* is also transposed back and using the *cnt* values, rather than the *valSet* function response values as intended, so as to provide a clear illustration in comparison with the previous example in Equation (22), then the matrix becomes as in Equation (27):

$$set_{of_{cnt}} = \left\{ \left\{ \left\{ \begin{matrix} \{0,23,0,23,0\}, \\ \{6,17,6,17,6\}, \\ \{12,11,12,11,12\}, \\ \{18,5,18,5,18\} \end{matrix} \right\}, \left\{ \begin{matrix} \{1,22,1,22,1\}, \\ \{7,16,7,16,7\}, \\ \{13,10,13,10,13\}, \\ \{19,4,19,4,19\} \end{matrix} \right\}, \right. \right. \left. \left. \left\{ \begin{matrix} \{2,21,2,21,2\}, \\ \{8,15,8,15,8\}, \\ \{14,9,14,9,14\}, \\ \{20,3,20,3,20\} \end{matrix} \right\} \right\} \left\{ \left\{ \left\{ \begin{matrix} \{3,20,3,20,3\}, \\ \{9,14,9,14,9\}, \\ \{15,8,15,8,15\}, \\ \{21,2,21,2,21\} \end{matrix} \right\}, \left\{ \begin{matrix} \{4,19,4,19,4\}, \\ \{10,13,10,13,10\}, \\ \{16,7,16,7,16\}, \\ \{22,1,22,1,22\} \end{matrix} \right\}, \right. \right. \left. \left. \left\{ \begin{matrix} \{5,18,5,18,5\}, \\ \{11,12,11,12,11\}, \\ \{17,6,17,6,17\}, \\ \{23,0,23,0,23\} \end{matrix} \right\} \right\} \right\}. \tag{27}$$

Equation (27) shows the filter flips in every second filter when compared with Equation (22).

Furthermore, to apply the *shuffle* as an address re-order on the alternated vector reversed matrix (flipped) as a contiguous vector, Equation (28) is used:

$$shuffleMat(nFilter) = Shuffle(flipMat_{(nFilter,0..(m-1))}, \lfloor nFilter/2 \rfloor) \tag{28}$$

Then the vector is re-indexed back to the matrix subscripts as in Equation (29):

$$TsprMat3[nFilter] \ = \ shuffelMat_{(dep\quad,width,height)} \tag{29}$$

Yet again for illustration, if the matrix (*TsprMat3*) is again transposed back and using the *cnt* values for a clear illustration rather than the *valSet* function response values, so as to be in comparison to previous examples in Equations (22) and (27), the matrix becomes as in Equation (30):

$$set_{of_{cnt}} = \left\{ \left\{ \left\{ \begin{matrix} \{0,23,0,23,0\}, \\ \{6,17,4,19,20\}, \\ \{12,11,6,17,4\}, \\ \{18,5,10,13,21\} \end{matrix} \right\}, \left\{ \begin{matrix} \{1,22,3,20,23\}, \\ \{7,16,2,21,1\}, \\ \{13,10,9,14,19\}, \\ \{19,4,8,15,5\} \end{matrix} \right\}, \right. \right. \left. \left. \left\{ \begin{matrix} \{2,21,1,22,3\}, \\ \{8,15,5,18,22\}, \\ \{14,9,7,16,2\}, \\ \{20,3,11,12,18\} \end{matrix} \right\} \right\} \left\{ \left\{ \left\{ \begin{matrix} \{3,20,23,0,12\}, \\ \{9,14,19,4,8\}, \\ \{15,8,17,6,16\}, \\ \{21,2,13,10,9\} \end{matrix} \right\}, \left\{ \begin{matrix} \{4,19,20,3,11\}, \\ \{10,13,21,2,13\}, \\ \{16,7,14,9,7\}, \\ \{22,1,22,1,22\} \end{matrix} \right\}, \right. \right. \left. \left. \left\{ \begin{matrix} \{5,18,5,18,5\}, \\ \{11,12,11,12,11\}, \\ \{17,6,17,6,17\}, \\ \{23,0,23,0,23\} \end{matrix} \right\} \right\} \right\}. \tag{30}$$

Equation (30) shows the *shuffle* reordering of the filters when compared with the example in Equation (27), although the *valSet* value for the *cnt* value would be used in the actual implementation.

The *cnt* values have been shown here for illustration to compare with Equations (22) (27), and show the address reordering *cnt* values for clearer understanding, rather than the intended *valSet* function distribution response.

A. Summary of the proposed method

In summary the resulting initialization weights, that were higher performing provided a sinusoidal bathtub distribution in convolutional layers, and a uniform distribution in the perceptron layer. Where the reordering provides uniquely reordered filters in each filter, with alternating vector directions and that reordering provides a two position base shift least neighbour arrangement. That least neighbour arrangement has a pattern reorganisation from a linear ramp to a saw tooth that is quasi-progressive in each filter, with a matching filter direction alternative provided from the numerical reverse order arrangement.

The nominal maximum number of unique filters is as in Equation (31), although with some weight geometries of a filter the pattern repeats earlier through aliasing. This is a subject of research to extend the number of useful filters on offer, and optimise the filters, given the filter geometries and number of them.

$$n_{filter} = 2(row_{filter} \cdot column_{filter} \cdot depth_{filter} - 1). \quad (31)$$

When the losses are compared during learning, of the first epoch, as the epoch after initialization, were the initial learning occurs. Then the loss does reduce quicker with the proposed non-random method, owing to the stripes and curves in the initial condition, being pre disposed to the application of categorization. Note that losses are shown as they are the optimisation objective. See Figure (11, left) for the existing random method, and Figure (11, right) for the proposed non-random method when shuffled.

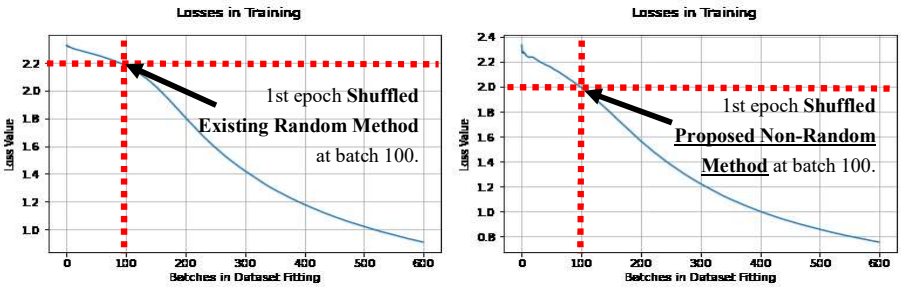

Figure **11**. Losses over batches in fitting when shuffled (existing method left, and proposed method right).

Figure (12) also makes the comparison with the existing random method in Figure (12, left), and the proposed non-random method in Figure (12, right). In comparison to the dataset shuffling in Figure (11), and the un-shuffled dataset results in Figure (12), the proposed non-random method has achieved the lower loss quicker in learning in both cases, and is noted at the batch 100 point. Thus earlier learning has benefited relatively in the proposed non-random method, using more of the dataset more effectively from the outset of learning, regardless of the dataset shuffling.

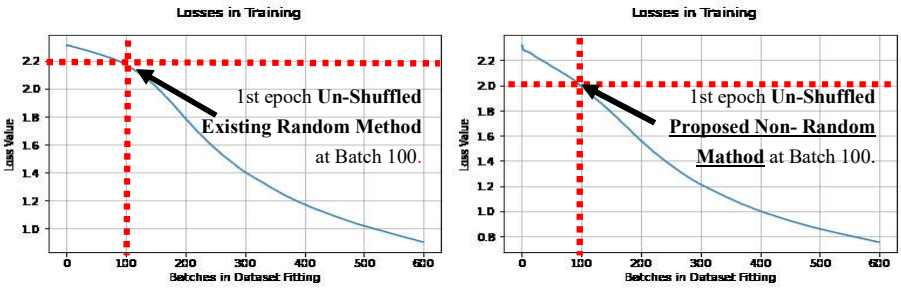

Figure **12**. Losses over batches in fitting when un-shuffled (existing method left, and proposed method right).

There is however an enquiring question raised, and that is: although it is a departure from the benchmark model, would the proposed method be robust to He et al. [16] ini-

tialisation limits instead. As He et al. initialisation is regarded as the *state of the art*. The uniform He et al. initialisation limit values are calculated in Equations (32), (33) and (34):

$$ConvLayer1 = \sqrt{\frac{6}{(5*5*1)}} = 0.48989795\ , \tag{32}$$

$$ConvLayer2 = \sqrt{\frac{6}{(5*5*32)}} = 0.08660254 \quad \text{and} \tag{33}$$

$$DenseLayer = \sqrt{\frac{6}{(4*4*64)}} = 0.07654655. \tag{34}$$

As before with the Glorot/Xavier limits the He et al. limit values are rounded to 8 decimal places [21], so as to be compatible in comparison. The cross-validation results using the He et al. initialisation limits are presented in Table 4 in columns: 'Loss' and 'Accuracy. Also the relative gain percentage of the proposed non-random method, over the Glorot/Xavier limits, is within the column 'He/Glorot (Non-Rand)'. Those percentages are the He et al and Glorot/Xavier initialisation limit values gain results both using the proposed non-random method. Also within Table 4, for completeness is a comparison gain using the proposed non-random method and the existing random method, both with He et al. initialisation, which is in column the 'He (Rnd/Non-Rnd)'.

Table 4. He et al. limits with the proposed initialization method, and gain comparisons.

Epochs	He et al. (Non Rnd) measure		Gains Over the Baseline	
	Loss	Accuracy	He/Glorot (Non-Rand)	He (Rnd/Non-Rnd)
5 Shuffled	0.082669578	97.55%	+0.05%	+0.7%
4 Shuffled	0.093996972	97.19%	+0.08%	+0.91%
3 Shuffled	0.10997723	96.97%	+0.12%	+1.49%
2 Shuffled	0.134461805	96.15%	+0.19%	+1.83%
1 Shuffled	0.214723364	94.11%	**+0.34%**	**+5.13%**
1 No Shuffle	0.217569217	93.57%	**+0.29%**	**+4.27%**

In all cases in Table 4 the proposed method offers a positive accuracy gain advantage in cross-validation. However, the greatest increase in accuracies are in both the first epoch cases shown in bold. Suggesting that He et al. initialisation limits also benefited again in extra earlier learning, over the earlier learning gains of the Glorot/Xavier limits previously demonstrated. The proposed method also offers an advantage gain in comparison with the existing random form using He et al. initialization, with again the greatest advantage in the initial epoch.

In all cases, the proposed method has an advantage in cross-validation accuracy when either applied to Glorot/Xavier, or the current *state of the art* of He et al limit values, and is repeatable and deterministic. That is an additional advantage for development of dependable safety critical applications, and also is an advantage within smart sensors using image classification.

IX. Fast Sign Gradient Method (FSGM) Perturbation Attack

This section will further examine the proposed non-random and existing random methods, using a transferred learning and FSGM approach. FSGM with transferred learning is a convenient approach to control distortions in a transferred learning dataset from the FSGM's epsilon *(ε)* value. The FSGM model defence with transferred learning approach, is used rather than other approaches, as it will demonstrate in influence of how compromised the transferred learning is, over the updated further learning, and the transferred learning approach will provide an indicator of those influences with a varying controlled magnitude of error distortion via the FSGM's epsilon *(ε)* value.

A. The FSGM transferred learning approach

A modern theme in neural networks is the area of perturbation attacks using the Fast Sign Gradient Method (FSGM) attack proposed by Ian Goodfellow [23] [24]. This is

an attack that can cause miss-classifications with an effect that may not be humanly perceivable. The attack itself can have a strength value of the attack controlled by an error magnitude value epsilon (ε) [23] [24] as in Equation (36):

$$x' = x + \varepsilon \cdot sign\left(\nabla_x J(\theta, x, y)\right) \quad (36)$$

A modification to the FSGM attack equation is made to avoid out of scale numbers in the image after perturbation, by clipping the perturbation image pixel values between 0 and 1, as in Equation (37). This is to be compatible in comparison to the non-perturbed image pixel scales, that were also scaled between values 0 and 1 in the Torres model [18]:

$$x' = \max\left(\min\left(x + \varepsilon \cdot sign\left(\nabla_x J(\theta, x, y)\right), 1.0\right), 0.0\right) \quad (37)$$

When the epsilon (ε) value is small the attack can be a deception (or *spoofing*) coursing miss-classification to another number assignment other than a human would. Also when the epsilon (ε) value is large it can cause a *denial of service* (*DoS*) to the human while still having a classification in the computer. It might be noted, that this might also have applications to encryption and hidden messages.

The FSGM transferred learning approach used is by Theiler [25], and it explains the attack with examples, and importantly uses the same MNIST dataset. See Figure (13) for the Theiler and Torres architecture as integrated, with three experiment test points.

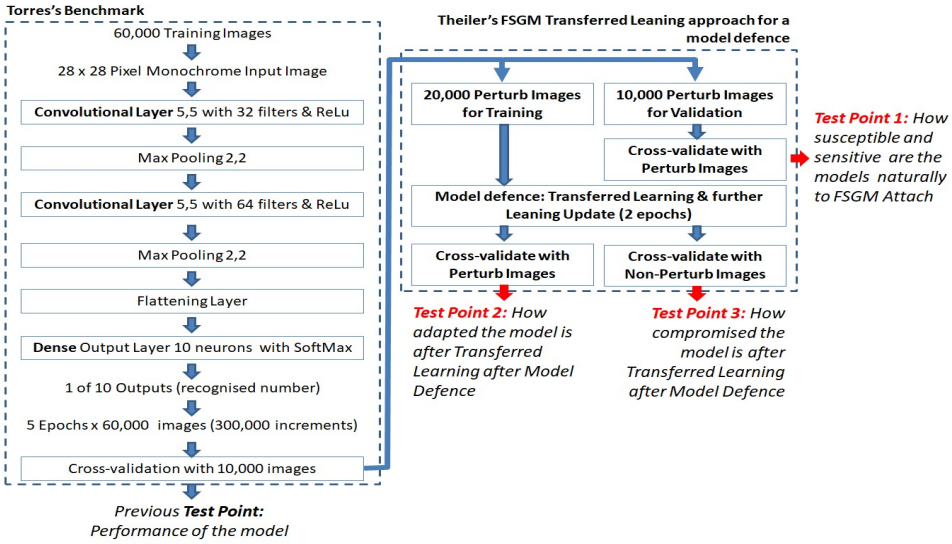

Figure **13**. Theiler's FSGM transferred learning experiment model [25], as added to the Torres's benchmark model.

The Theiler attack dataset sizes [25] are thus used as the dataset is the same, although is adapted to the Torres benchmark's number of epochs [18] instead. The epochs used in transferred learning are set to be half that of the Torres baseline benchmark case, instead of half the Theiler number of epochs, as is the case from Theiler's initial learning model. Such that the amount of back propagation is relatively similar to the transferred learning from Theiler's model but applied to the Torres model.

The relevance of the FSGM attack to this paper is a hypotheses that FSGM attacks could be effective partially from the less humanly perceivable noise content in an image dataset rather than the useful information in the dataset alone. Furthermore, that that noise could be noise re-colourised by the random numbers as a noise source in the initialisation state when the weights are multiplied with the activations. It follows that if an initialisation state has less random content that could be thought of as noise, then the subsequent learning may have less opportunity for unintentional noise re-colourisation as a result. Thus the effect could affect the compromise of the original training after defence if the adversarial attack dataset is used to retrain the model to protect it. The FSGM approach also provides a controlled distortion via the epsilon (ε) value.

In Figure (14, left and right) are the 1st 20 images of the FSGM perturbation attack datasets. They are generated from the sign of the gradient of the loss of a true prediction and an image, In Figure (14, left) is the generation from the existing random form, and in Figure (14, right) from the proposed non-random method instead. Along the columns axis are the 1st 20 images of each dataset, and in each row as perturbed with images with an increasing epsilon value of 0.0 to 1.0 in steps of 0.05. In green are the perturbation images that are correctly classified against their original tag, and in red are the perturbation images that are miss-classified.

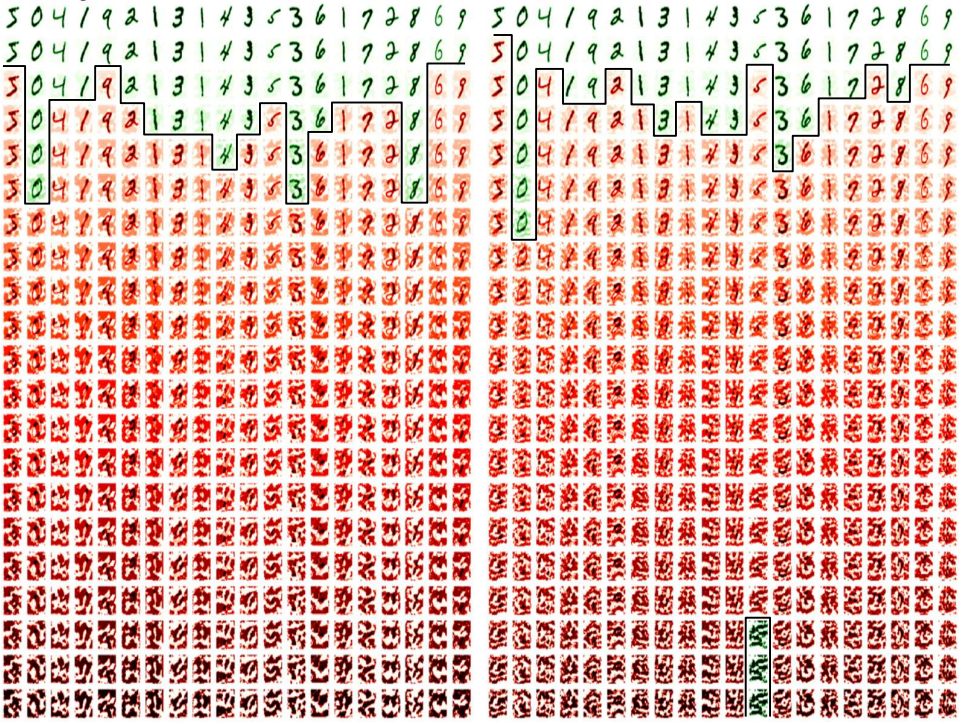

Figure **14**. FSGM attack image examples: existing method (left) proposed method (right).

Figure (14) shows that the low perturbation epsilon error values images (at the 1st rows) are humanly recognisable and the high epsilon error values images (at the last rows) are not humanly recognisable. At the point that the miss-classifications occur, the strength of the competing images pixels become more apparent, and traditional de-noising and image value clipping would enhance the image classification in many cases of both methods. The subject of noise and noise injection, but as a method for defence from FSGM is proposed in 2020 by Schwinn et al. [26], but their approach requires a learning regularisation step that couples it to the dataset. Although the Schwinn et al. [26] approach is of interest, but in a non-dataset coupled form, and perhaps as an augmentation to the epoch scheme. However, this paper's approach is purely to examine the possible resistance and compromise with a reduction in noise re-colorization opportunities.

Thus applying the same perturbed FSGM attack method with different epsilon error strengths, a comparison in accuracy and loss between the existing (random) and proposed (non-random) methods are presented in Figures (15), (16) and (17).

B. The undefended model, attacked by FSGM

Figure (15, left) and Figure (15, right) are results (at the experiment test point 1) in accuracy and loss, from cross-validation with a generated validation adversarial attack perturbation dataset of 10,000 images using the FSGM approach, and applied to an undefended model. In Figure (15), both the existing (random shown in red) and proposed

(non-random shown in blue) methods are using the higher performing He et al. initialisation limit values, and with controlled steps of the epsilon (ε) distortion value.

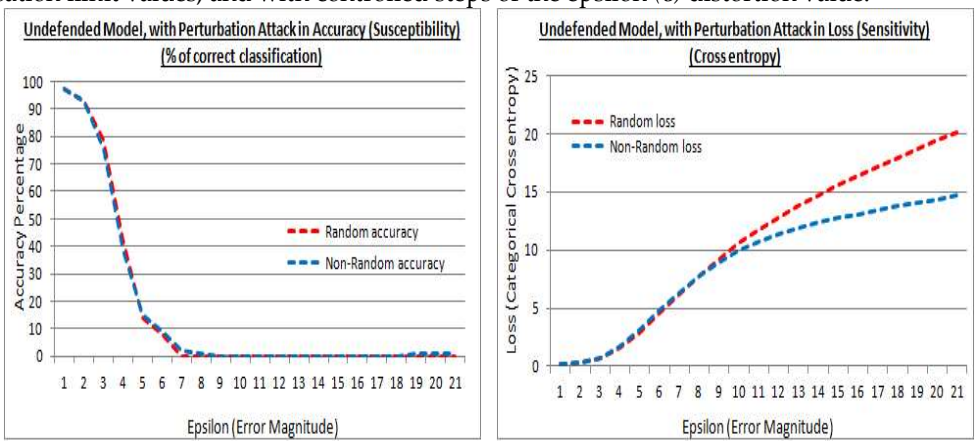

Figure **15**. Accuracy and losses of an undefended model with epsilon increments under attack (test point 1).

In Figure (15, left) in both initialization cases, the results are almost identical in accuracy, with an undefended model, signalling a similar generalisation was achieved under attack. Ideally the accuracy would be higher, to correctly classify the images with their original tag despite the perturbation attack. Both the existing (random) and proposed (non-random) methods are susceptible to FSGM attack, and in both cases the susceptibility is greater as the epsilon value gets larger, signalled by the cross-validation accuracy lowering. In Figure (15, right) ideally the losses may be low showing regularisation is still effective, i.e. it reduces the variance, despite the perturbation attack. However, in this attack the sign of the gradient is taken from a calculated loss of a true prediction and an input image, that gradient is then applied to an image as a perturbation image, therefore the loss rises as the epsilon value is raised, as the controlled distortion. The proposed (non-random) method has a lower loss then the existing (random) method, at larger epsilon values. Suggesting that the perturbation of features in the attacks are less sensitive to regularisation in the proposed method, although this is at epsilon values where the generalisation (accuracy) has diminished in the attack.

C. Defending a model from FSGM

The Theiler [25] approach to defending a model from an attack is to transfer learning and include adversarial perturbation attack examples into a further training dataset, and generate a further attack validation dataset. Then cross-validate with both validation datasets (i.e. both attack and non-attack datasets). This makes a comparison in accuracy and loss after model defence. As such the original model is further trained with 20,000 training adversarial perturbation attack images to defend it over 2 epochs, examined with controlled increments of the epsilon (ε) value. In the experiment approach the epsilon (ε) value is stepped, providing a controlled distortion in the transferred learning. Such that the controlled distortions can be examined after transferred learning, and reveal how compromised both methods are after the learning transfer.

D. Examining the transferred learning adaption

The results from the existing (random) and proposed (non-random) methods are shown for experiment test point 2. In Figure (16, left) and Figure (16, right) the performance against the validation attack perturbation dataset is shown. Such that the transferred learning of the original dataset is transferred with an attack dataset to evaluate the performance.

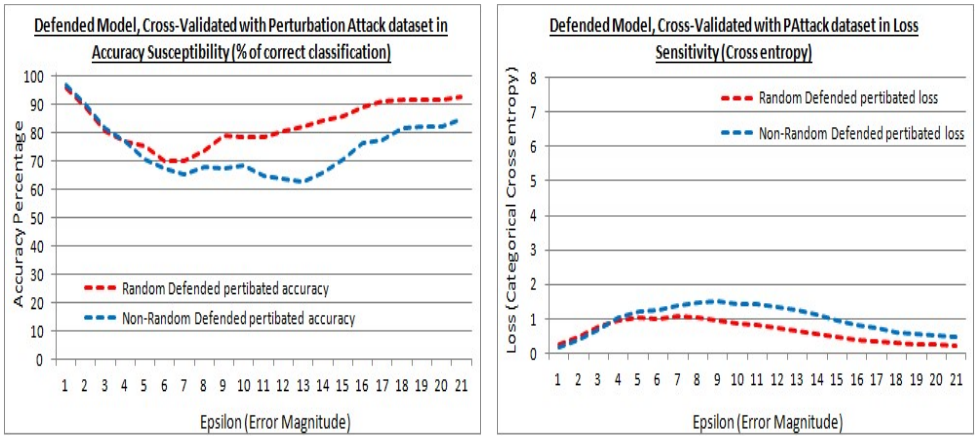

Figure 16. Accuracy and losses of a defended model with epsilon increments against the validation attack dataset (test point 2).

Figure (16, left) and Figure (16, right) show the accuracy and losses of the defended model cross-validated with the validation attack dataset, and shows the existing (random) and proposed (non-random) methods, have as similar performance with low epsilon values. Figure (16, left) shows that with larger epsilon values, the cross-validation accuracy of the attack dataset is higher with the existing method, showing the existing method prefers the further generalisation from retraining after model defence at higher distortions. In Figure (16, right) the losses are also lower in the existing method indicating that they are slightly more regularised in the existing method.

E. Examining the model compromise in defence to the original cross-validation dataset

In Figure (17, left) and Figure (17, right) are the results after model defence with the adversarial FSGM perturbation attack dataset, but cross-validated with the original non-attack dataset (at experiment test point 3). This is to show how compromised the models are to the original non-attack dataset after model defence via transferred learning.

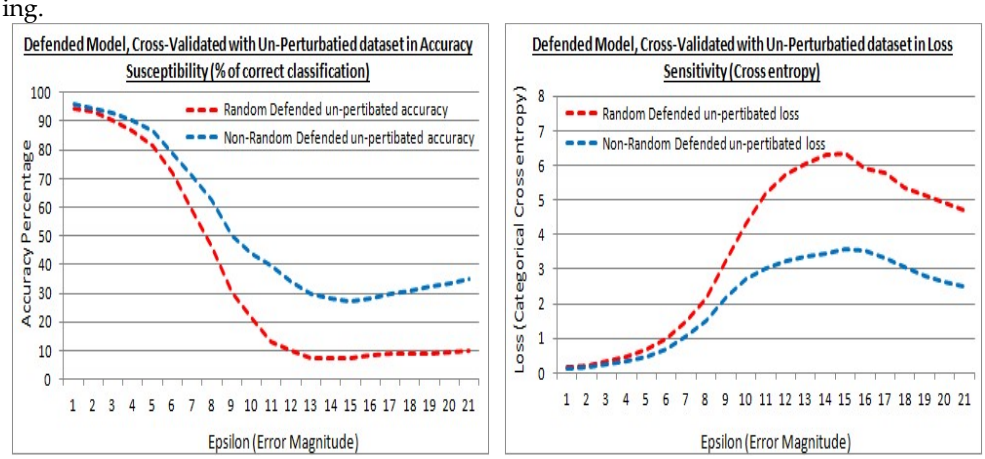

Figure 17. Accuracy and losses of a defended model with epsilon increments against the original non-attack cross-validation dataset (test point 3).

Figure (17, left) and Figure (17, right) shows the defended model with the original dataset cross-validation in accuracy and loss. Figure (17, left) and Figure (17, right) shows that at greater epsilon error magnitudes the difference between the existing (random) and proposed (non-random) methods are divergent. With the proposed (non-random) method being the higher accuracy in Figure (17, left) and lower loss in Figure (17, right). Meaning that the proposed (non-random) method has suffered less in accuracy when re-trained with the perturbed attack dataset to defend it. i.e. the attack

defence has less affected the accuracy to the original dataset with the proposed method, most notably with larger distortions in that further training attack dataset.

F. *FSGM attacks with varying epsilon values*

In all cases included in Figures (15), (16) and (17), each perturbation attack dataset, used a constant epsilon value in each measurement, that was progressively increased. In Table 5 are the results from both the existing (random) and proposed (non-random) methods, but with a randomly varying epsilon value in both the cross-validation and training datasets.

Table 5. Results from a single dataset that included random epsilon values.

Initialization method used prior to model defence via transferred learning.	**Non-Attack cross-validation dataset**		**Attack cross-validation dataset**	
	Loss *(Cross Val)*	Accuracy *(Cross Val)*	Loss *(Cross Val)*	Accuracy *(Cross Val)*
Proposed (Non-Random) Method	**0.9854**	**67.01%**	1.3331	61.82%
Existing (Random) Method	1.2736	61.05%	**0.8366**	**78.41%**

In Table 5 the cross-validation accuracy with the proposed (non-random) method is greater with a lower loss, when cross-validated with the original non-attack validation images. This is shown in Bold, within the column: *'Non-Attack cross-validation dataset'*. However, the cross-validation accuracy is higher with a lower loss with the existing method using the attack cross-validation dataset. This is also shown in bold, within the column: *'Attack cross-validation dataset'*.

Table 5 demonstrates that the proposed non-random method has been less compromised to the original learning that was transferred. Also that the existing method by contrast, had a higher accuracy and lower loss with the retraining attack dataset, thus may have embraced the subsequent attack dataset more, but has been more compromised to the original training to a greater extent. These findings were also shown in Figures (15), (16) and (17), and those results also showed a greater difference with constant epsilon values, and the most extreme difference was with the larger epsilon (ε) values, rather than smaller values.

X. Discussion and Conclusions

This work focused on repeatable determinism to support mission and safety critical systems that use convolutional networks with mixes of both perceptron and convolutional layers.

A. *Proposed method in neural networks*

The proposed method, is applicable to deep convolutional networks for repeatability, however also achieves a higher accuracy in a single learning session, with a computational initialization state number sequence that has been designed to be more conducive to image classification. The proposed method has a finite number sequence set that is not coupled to the dataset via sampling, which may be closer to a general case. The losses during learning show a quicker reduction using the proposed form, and result in a higher accuracy. The repeatable deterministic property also provides no variation in learning sessions, aiding the speed of development of a model. The proposed method is complimentary to existing methods, replacing only the random numbers in those methods. The proposed non-random method provided a higher cross-validation accuracy against the existing random number method, and when used with Glorot/Xavier limits of the benchmark models achieved an extra 3.705% un-shuffled, and 2.642% shuffled in the first epoch. Thus dataset order is still a significant effect, but the proposed method was tolerant. The proposed method is also robust to He et al. initialization value limits as the current *state of the art*, and when used with the proposed method offered an accuracy gain of 5.13% shuffled and 4.27% un-shuffled in the first epoch.

B. *Proposed method in FSGM with transferred learning*

When applied to FSGM attacks both methods offer little resistance to perturbation attacks without transferred learning, but the proposed (non-random) method also has an advantage of being less compromised, noted with larger epsilon values. These results may be biased to the order that the datasets where trained in, and transferred from, as a bias to the original programming. However it could also be that the higher epsilon values have less suitable application to the proposed method, and further study would be required. The results do show that the proposed method when transferred in this order, can provide a lower compromise to the original programming, and achieved an accuracy of ~31%, compared with ~9% of the existing method. Moreover, the FSGM attack was used with transferred learning to examine the compromise of the learning from cross-validation datasets, under controlled distortions. This found that with higher epsilon distortions the proposed (non-random) method was less compromised to the original cross-validation datasets that had no distortions. By contrast, the existing random method with higher attack distortions was less compromised with the attack dataset, but much more compromised to a dataset with no distortions. This also provides support to the notion that high epsilon values can be used to cause a DoS attack to a human while still being readable by a computer at 63%-85%, and that the proposed method can still provide a higher level of performance in generalisation in non-perturb cases of ~31%, instead of just ~9% with the existing method. The bias to original training is favoured, illustrated by, that if the transferred learning was to extend the training to a letter set from A to F as the hexadecimal set. Then the existing random method may be biased to replacing the training of numbers with letters. Whereas the proposed non-random method may be more biased to extending the training to letters, while still retaining more of what was learnt from numbers. It follows that there are more subjects for further research.

C. Further research

Synthetic noise replacement: A further extension to this work may look at synthetic noise like injection methods into the dataset consistent with the Schwinn approach [26], but not to be dataset coupled, perhaps as an augmentation to the epoch scheme, and it was noted that dataset order is also still an effect on performance.

Model architectures and dataset: Also in further work, a focus on different datasets, particularly those datasets with multi-channel images and in other applications. There may also be more dense layer configurations depending on the layers prior, that will increase performance in neural network generalisations. It may be that the proposed method is a more general approach by not being coupled to the dataset directly through data sampling, as used in other related works. However, it is indirectly coupled, as the model architectures are coupled to the datasets and applications. As such the proposed method is connected to hyper-parameters. The proposed non-random method having more striped forms through a filter may make the filter size less sensitive to hyper-parameters. As with the random speckled form the initialisation filter patterns become more specific as the filter is enlarged. Further research is required to set the resultant filter patterns with variations in the hyper-parameters used in different model architectures, datasets and applications to perfect the algorithms proposed, such that they can be a general case, as a direct alternative to the existing random forms.

Increasing and optimising filter diversity: Another subject of research is increasing the number of filters on offer, and making them tolerant to the hyper-parameters in more architectures. Such that the proposed method is adapted to an architecture generally and is not constraining to it, providing the best filters for any model architecture. That may also include examining the success of current proposed filters, such that a subsequent proposed method can be optimised towards higher performing filters.

References

1. Hubel, D.H.; Wiesel, T.N., Receptive fields and functional architecture of monkey striate cortex, *The Journal of Physiology*, **1958**, vol. 195, no. 1, pp. 215-243.

2. Hubel, D.H.; Wiesel, T.N. Shape and arrangement of columns in cat's striate context, *The Journal of Physiology*, **1963,** vol. 165, pp. 559-568.
3. LeCun, Y.; Kavukcuoglu, K.; Farabet C., Convolutional networks and applications in vision, *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS). **2010***, pp. 253-256.
4. Masci, J.; Meier, U.; Cireşan, D.; Schmidhuber. J., Stacked convolutional auto-encoders for hierarchical feature extraction, *Honkela T., Duch W., Girolami M., Kaski S. (eds) Artificial Neural Networks and Machine Learning – ICANN 2011. ICANN 2011. Lecture Notes in Computer Science*, **2011**, vol. 6791. Springer, Berlin, Heidelberg.
5. Krizhevsky, A.; Sutskever, I.; Hinton, G. ImageNet classification with deep convolutional neural networks, *NIPS'12: Proc. of the 25th International Conference on Neural Information Processing Systems.*, **2012,** vol. 1, pp 1097-1105.
6. Srivastava, S.; Bisht, A; Narayan, N. Safety and security in smart cities using artificial intelligence - a review, *Proc. of 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, **2017**, pp 130-133.
7. Knight, J., Safety critical systems: challenges and directions, *Proc. of International Conference on Software Engineering*. **2002,** pp. 547 - 550.
8. Serban, A., Designing safety critical software systems to manage inherent uncertainty, *Proc of 2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. **2019**, pp 246-249.
9. Carpenter, P., Verification of requirements for safety-critical software, *Proc. of the 1999 annual ACM SIGAda international conference on Ada*. **2019**. Volume: XIX, pp. 23-29.
10. Rudd-Orthner, R.; Mihaylova, L., Non-random weight initialisation in deep learning networks for repeatable determinism, *Proc. of IEEE International Conference on Dependable Systems, Services and Technologies (DESSERT)*, **2019**, pp. 223-230.
11. Rudd-Orthner, R; Mihaylova, L., Repeatable determinism using non-random weight initialisations in smart city applications of deep learning. *Journal of Reliable Intelligent Environments*, **2020**, vol. 6, pp. 31-49.
12. Blumenfeld, Y.; Gilboa D.; Soudry, D., Beyond signal propagation: is feature diversity necessary in deep neural network initialization?, *Proc. of the 37th International Conference on Machine Learning*, **2020**, vol. 119, pp. 960-969.
13. Ding, W.; Sun, Y.; Ren, L.; Ju, H.; Feng Z.; Li, M., Multiple lesions detection of fundus images based on convolution neural network algorithm with improved SFLA, *Special Section on Deep Learning Algorithms for Internet of Medical Things*, **2020**, vol. 8 pp 97618-97631.
14. Wang, Y.; Rong, Y.; Pan, H.; Liu, K.; Hu, Y.; Wu, F.; Peng, W.; Xue X.; Chen, J., PCA based kernel initialization for convolutional neural networks, *Tan Y., Shi Y., Tuba M. (eds) Data Mining and Big Data. DMBD 2020. Communications in Computer and Information Science*, **2020,** vol. 1234. Springer, Singapore.
15. Ferreira, M.F.; Camacho R.; Teixeira, L.F., Autoencoders as weight initialization of deep classification networks for cancer versus cancer studies, *BMC medical informatics and decision making*, **2019**, vol. 20. sup. 5:141
16. Arat, M.M., Weight Initialization Schemes - Xavier (Glorot) and He, *Mustafa Murat ARAT*, **2019**, Available: https://mmuratarat.github.io/2019-02-25/xavier-glorot-he-weight-init.
17. LeCun, Y.; Cortes C.; Burges, C., MNIST handwritten digit database, *Yann.lecun.com*. Available at: http://yann.lecun.com/exdb/mnist/ [Accessed on 28/05/2021].
18. Torres, J., Convolutional neural networks for beginners using Keras & TensorFlow 2, Medium, 2020, Available at: https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25
19. Kassem, MNIST: simple CNN keras (accuracy : 0.99)=>top 1%, *Kaggle.com*, **2019**, Available at: https://www.kaggle.com/elcaiseri/mnist-simple-cnn-keras-accuracy-0-99-top-1.
20. Kakaraparthi, V., Xavier and He normal (he-et-al) initialization, *Medium*, **2018**, Available at: https://medium.com/@prateekvishnu/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528.
21. Hewlett-Packard, HP-UX Floating-point guide HP 9000 computers Ed 4, *Citeseerx.ist.psu.edu*, **1997**, p. 38.. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.9291&rep=rep1&type=pdf.
22. Rudd-Orthner R.; Mihaylova, L., Numerical discrimination of the generalisation model from learnt weights in neural networks, *In Proc. of the International Conf. on Computing, Electronics & Communications Engineering (iCCECE), IEEE*, **2019**.
23. Goodfellow, I.; McDaniel P.; Papernot, N., Making machine learning robust against adversarial inputs, *Communications of the ACM*, **2018,** vol. 61, no. 7, pp. 56-66.
24. Molnar, C., 6.2 Adversarial examples | interpretable machine learning. *Christophm.github.io*. **2021**, Available at: https://christophm.github.io/interpretable-ml-book/adversarial.html.
25. Theiler, S.. Implementing Adversarial Attacks and defenses in keras & tensorflow 2.0, *Medium*, **2019**, Available at: https://medium.com/analytics-vidhya/implementing-adversarial-attacks-and-defenses-in-keras-tensorflow-2-0-cab6120c5715.
26. Schwinn, L.; Raab, R.; Eskofier, B., Towards rapid and robust adversarial training with one-step attacks *ArXiv preprints arXiv:2002.10097*, **2020**.