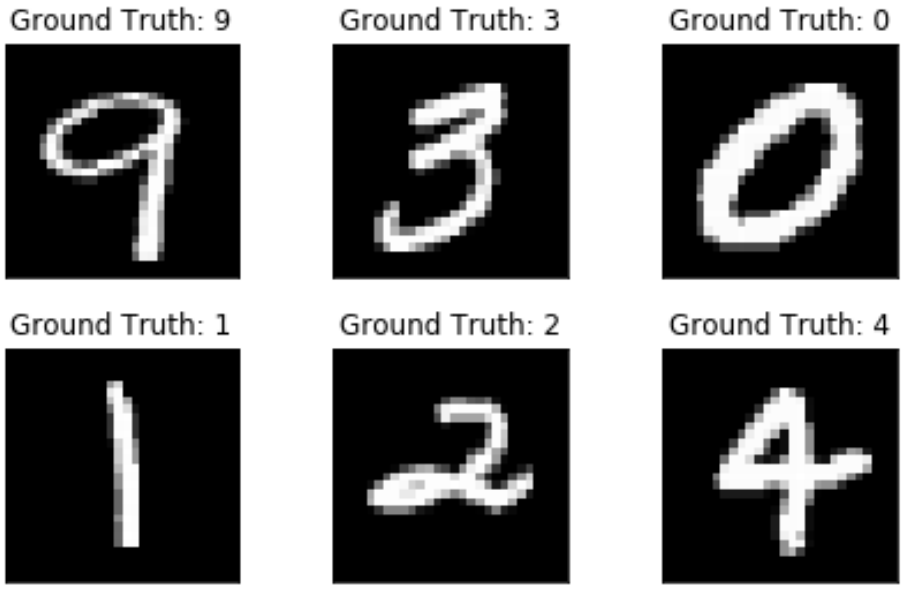
Data Augmentation

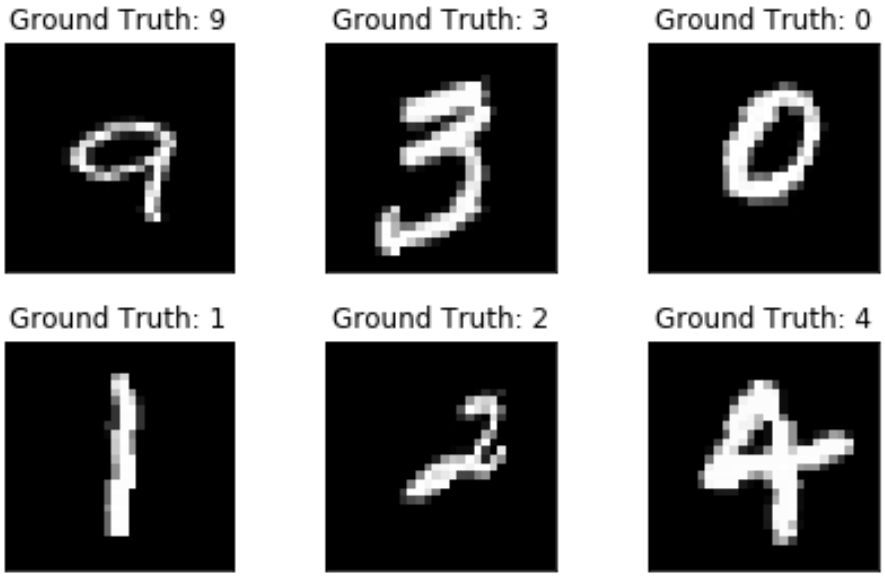
MNIST images after loading and normalization.



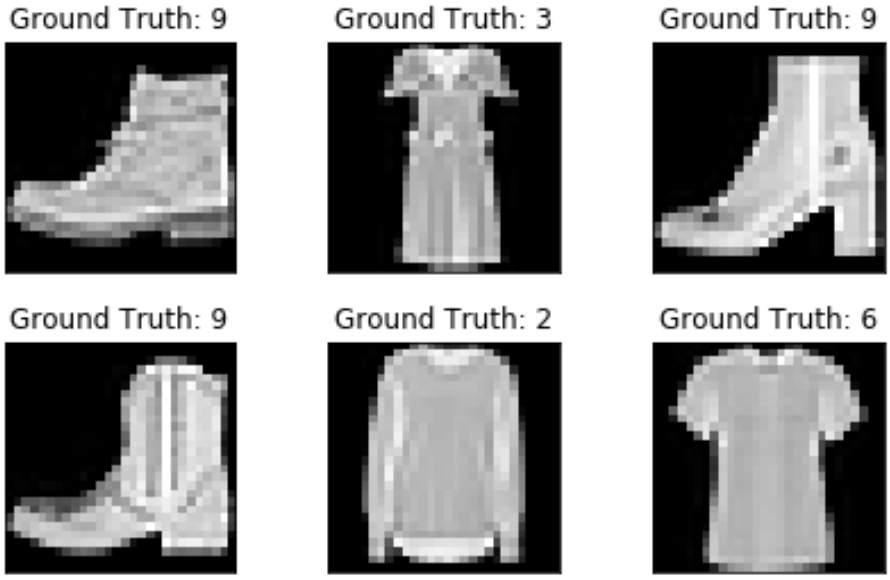
MNIST images after applying:

torchvision.transforms.RandomPerspective(),

torchvision.transforms.RandomRotation(10, fill=(0,)),



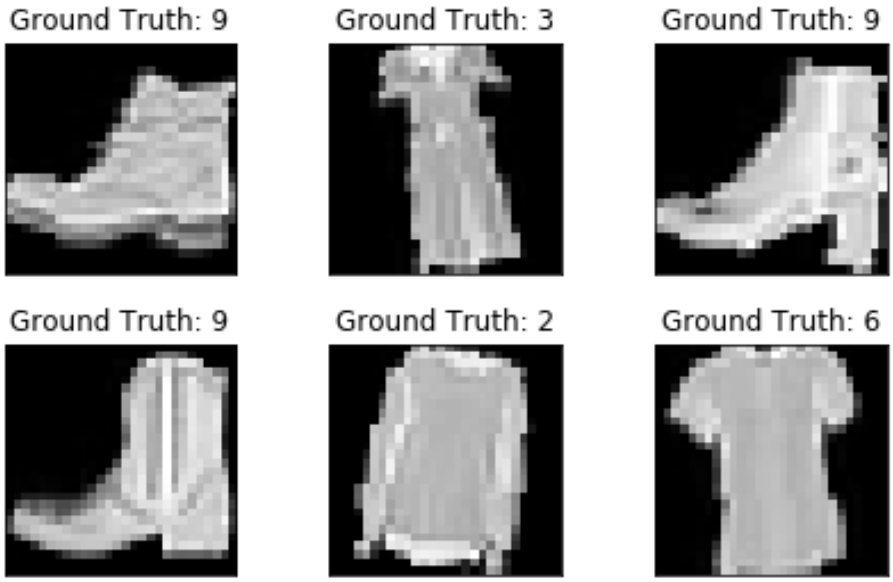
Fashion-MNIST images after loading and normalization.



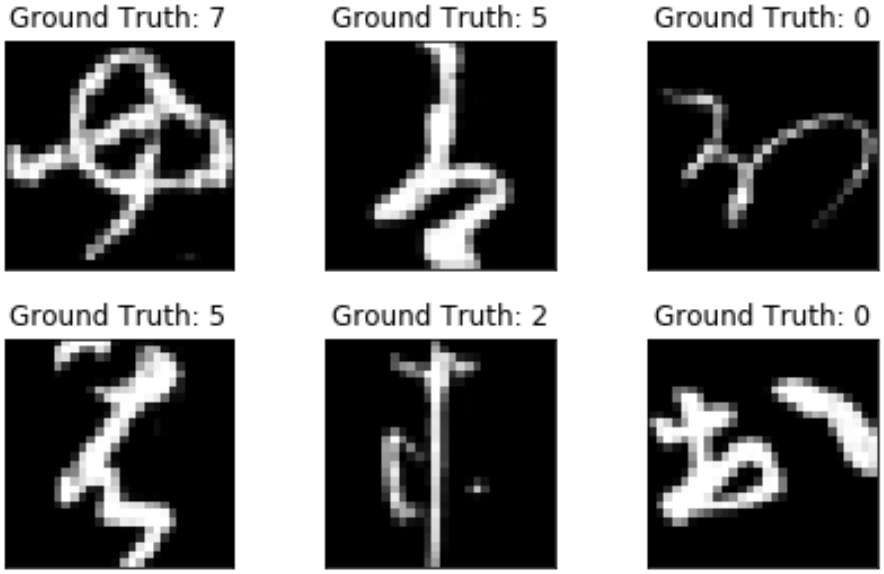
Fashion-MNIST images after applying:

torchvision.transforms.RandomRotation(10, fill=(0,)),

torchvision.transforms.RandomResizedCrop(28, scale=(0.95,1)),



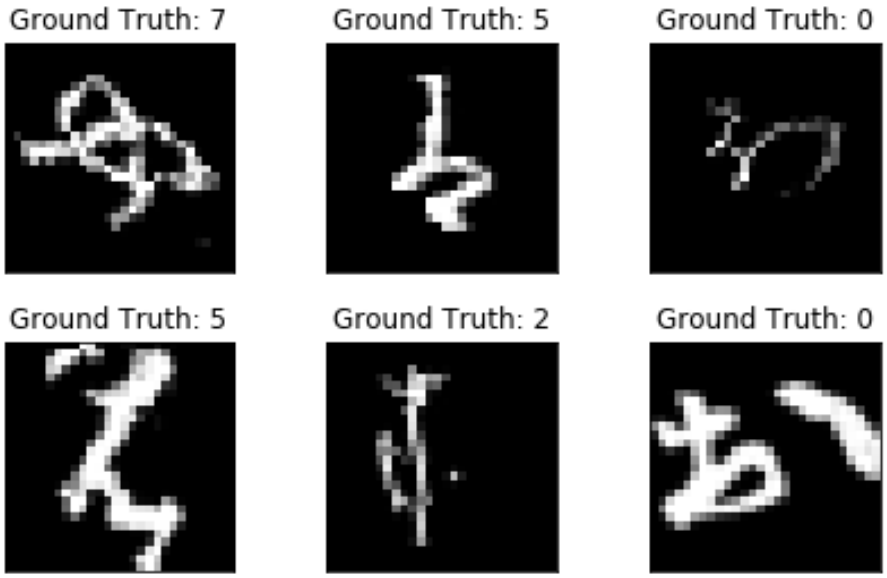
KMNIST images after loading and normalization.



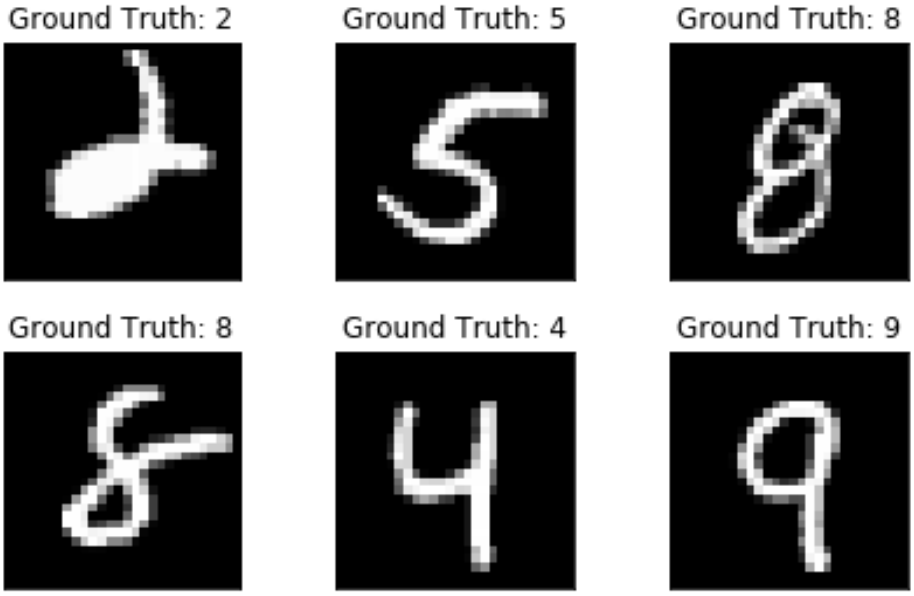
KMNIST images after applying:

torchvision.transforms.RandomPerspective(),

torchvision.transforms.RandomRotation(10, fill=(0,)),



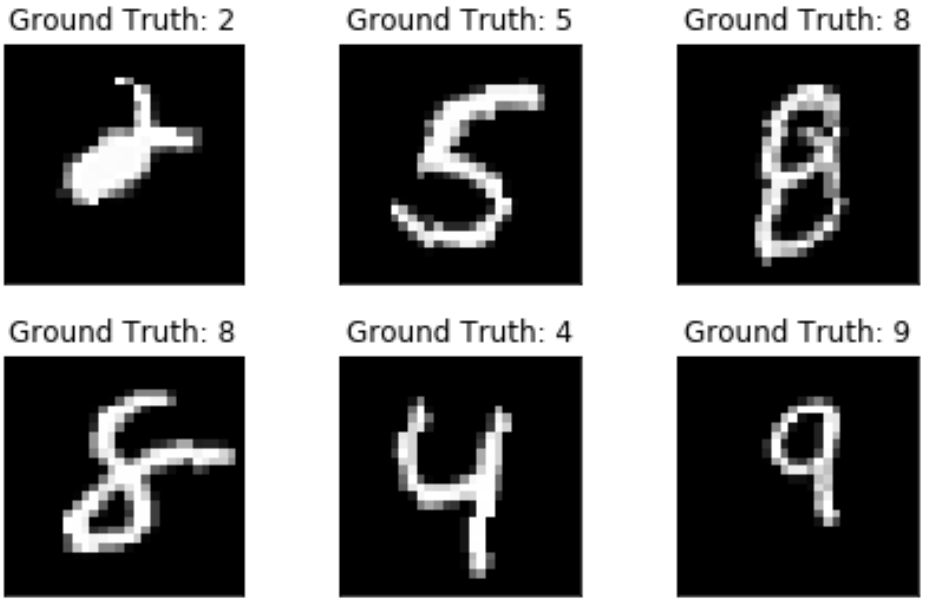
QMNIST images after loading and normalization.



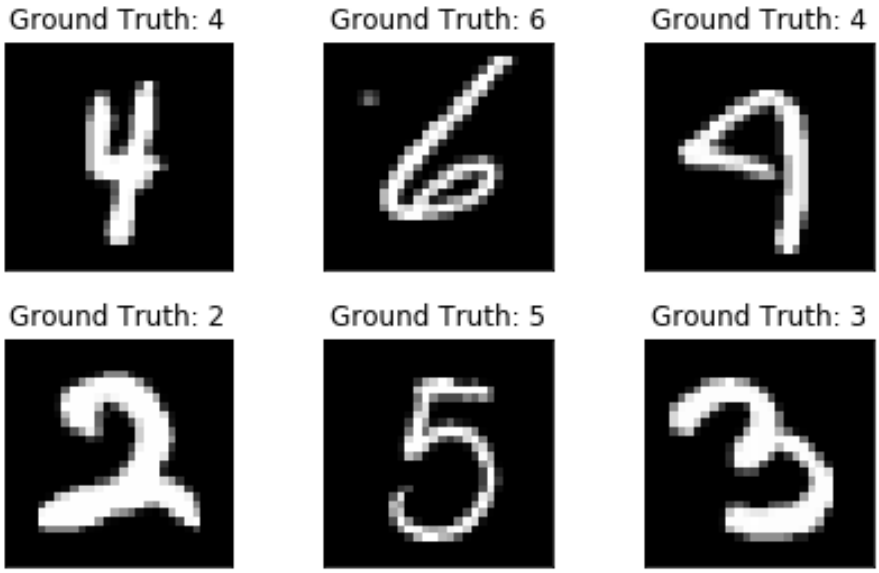
QMNIST images after applying:

torchvision.transforms.RandomPerspective(),

torchvision.transforms.RandomRotation(10, fill=(0,)),



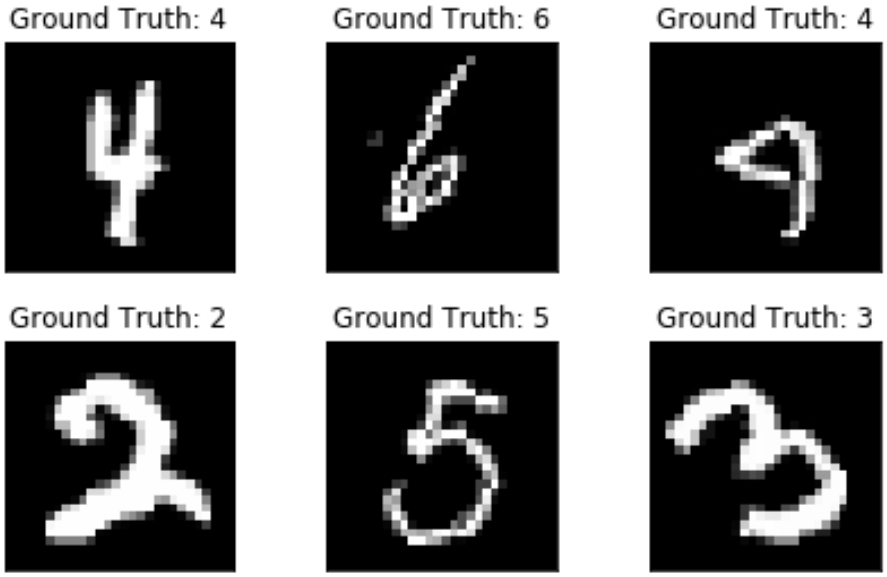
EMNIST (Digits) images after loading and normalization.



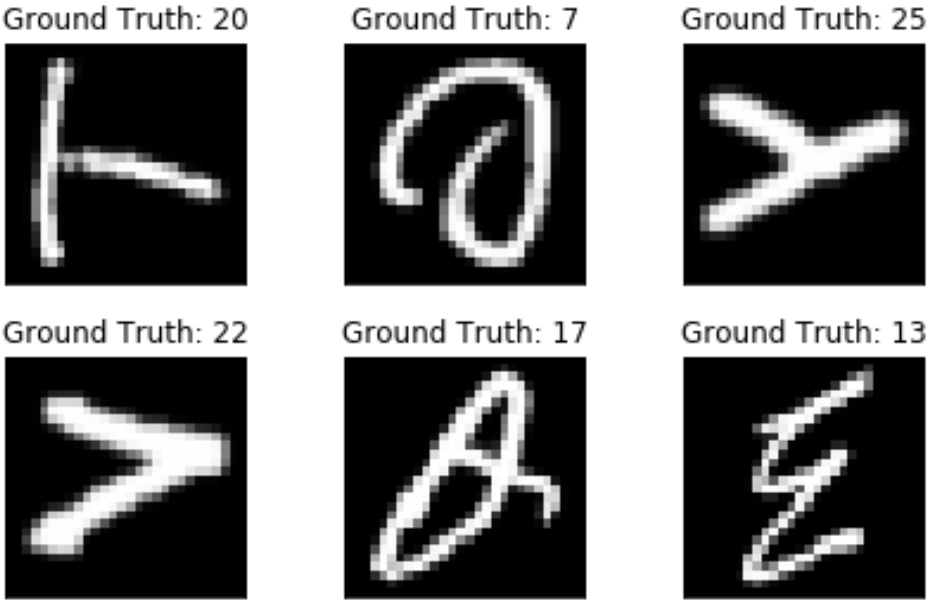
EMNIST (Digits) images after applying:

torchvision.transforms.RandomPerspective(),

torchvision.transforms.RandomRotation(10, fill=(0,)),



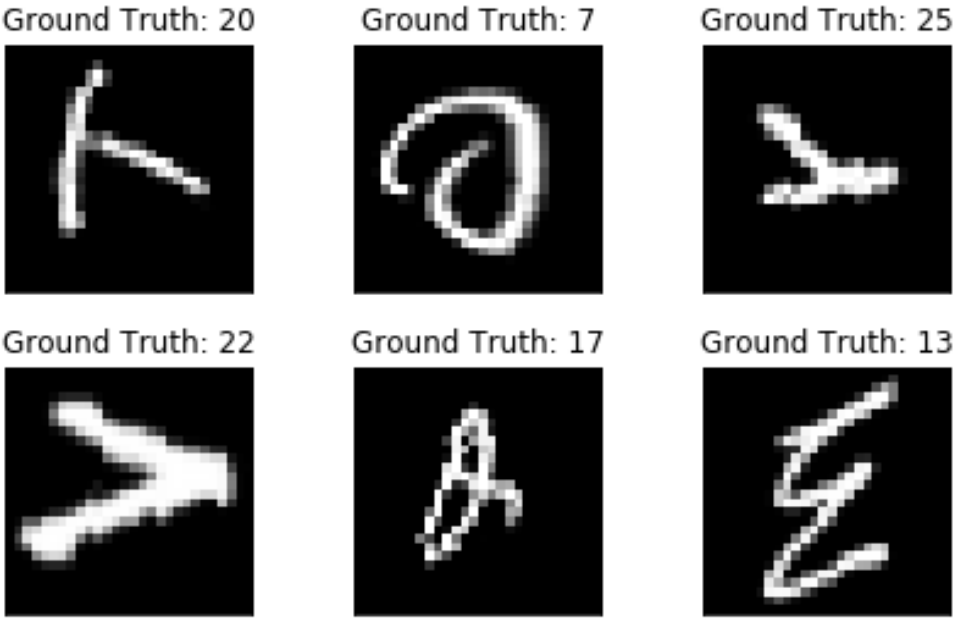
EMNIST (Letters) images after loading and normalization.



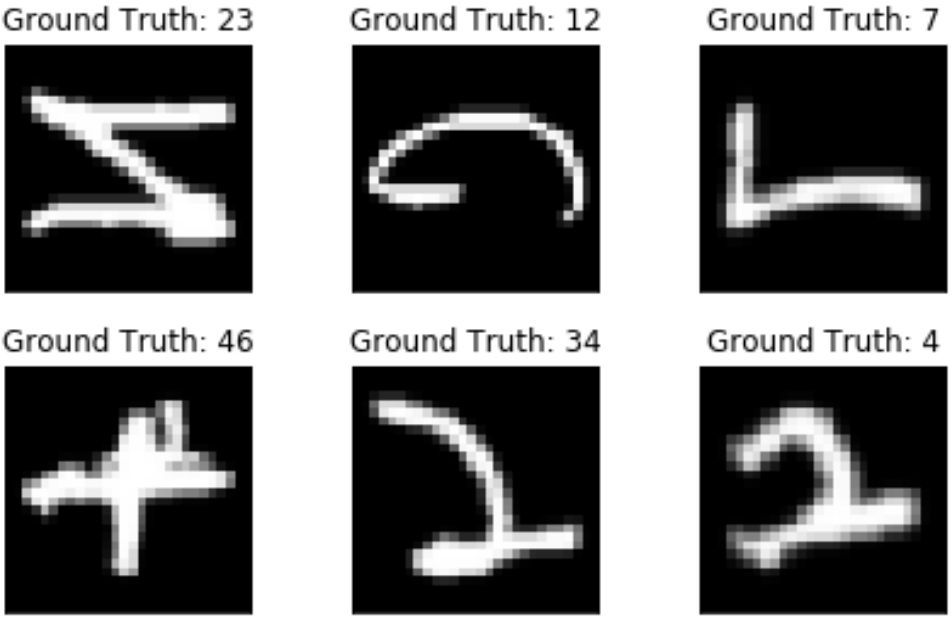
EMNIST (Letters) images after applying:

torchvision.transforms.RandomPerspective(),

torchvision.transforms.RandomRotation(10, fill=(0,)),



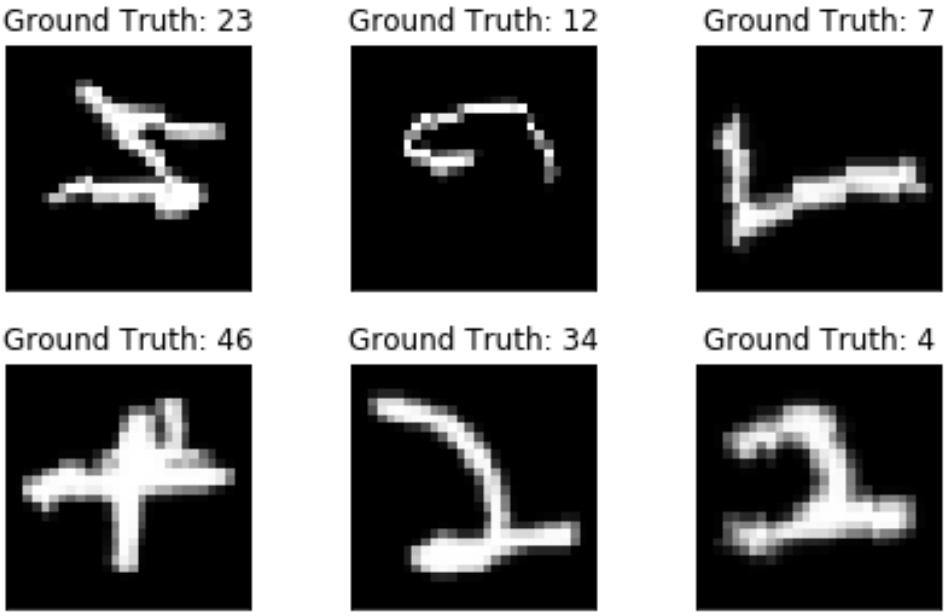
EMNIST (Balanced) images after loading and normalization.



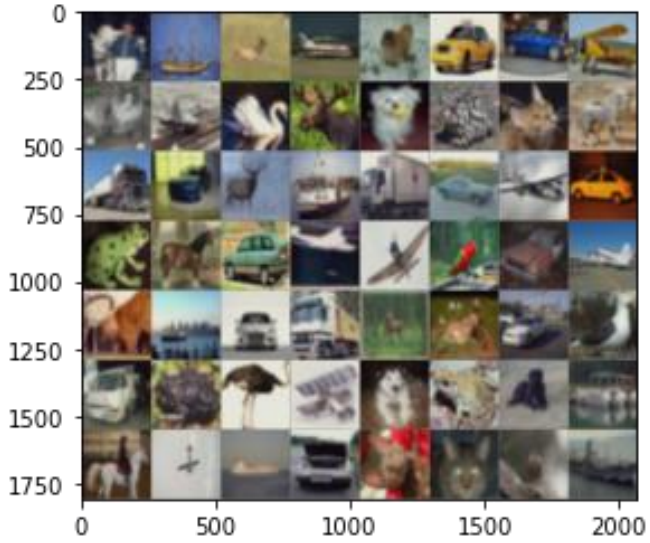
EMNIST (Balanced) images after applying:

torchvision.transforms.RandomPerspective(),

torchvision.transforms.RandomRotation(10, fill=(0,)),



CIFAR-10 images after loading, resize and normalization.



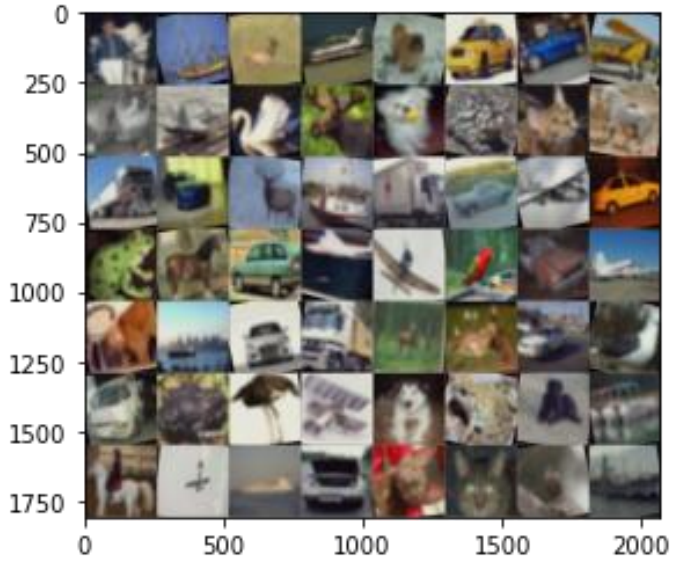
CIFAR-10 images after applying:

transforms.Resize((272,272)),

transforms.RandomRotation(15,),

transforms.RandomCrop(256),

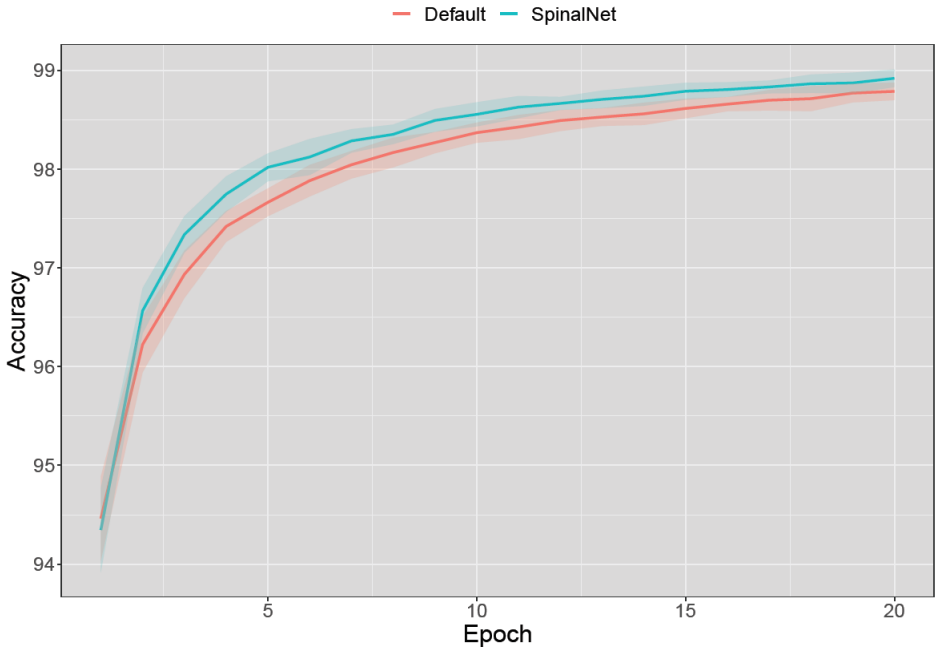
transforms.RandomHorizontalFlip(),

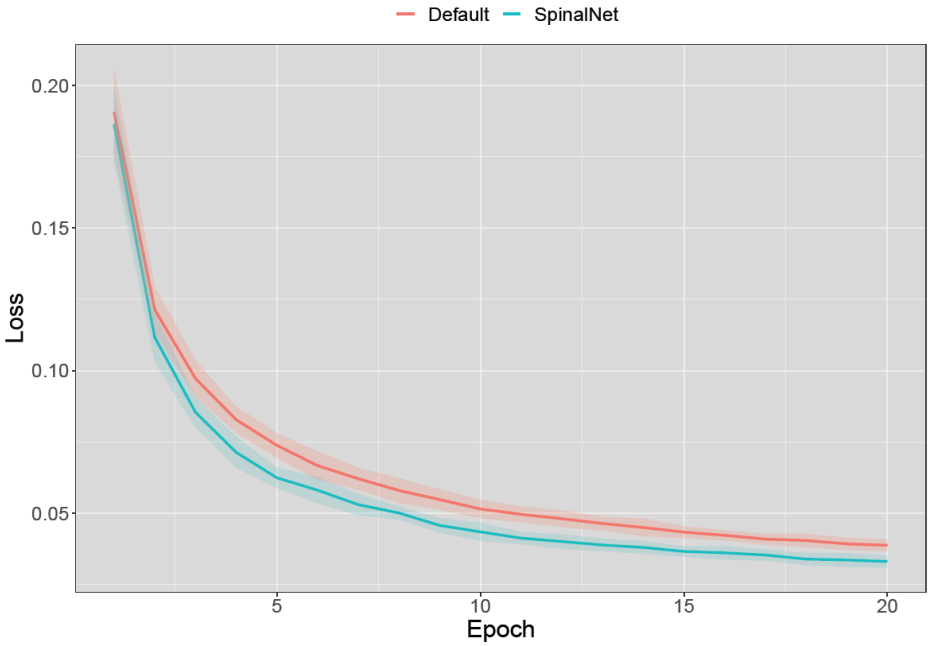


Similar augmentations are performed in other datasets. All augmentation details are available in the github code.

Training Convergence

The accuracy and loss over epochs are recorded for the MNIST dataset to observe training. CNN with a Spinal fully-connected layer achieves higher accuracy in the lower epoch, on average. Curves are showing mean with a standard deviation shadow.





Sensitivity

The sensitivity in neural network is represented by partial derivatives [A]. In a deep NN, the sensitivity of one input to output changes based on the value of other inputs. Sensitivity of one neurons output to the input of next layer can be represented as follows [A]:

$$\frac{\partial z_k^l}{\partial y_i^{l-1}} = w_{ik}^l$$

Sensitivity of all neurons in the current layer to all inputs in the next layer can be represented by the following equation [A]:

$$\frac{\partial \mathbf{z}_{[1\times n^l]}^l}{\partial \mathbf{y}_{[1\times n^{l-1}]}^{l-1}} = \begin{bmatrix} \frac{\partial z_1^l}{\partial y_1^{l-1}} & \frac{\partial z_2^l}{\partial y_1^{l-1}} & \cdots & \frac{\partial z_{n^l}^l}{\partial y_1^{l-1}} \\ \frac{\partial z_1^l}{\partial y_2^{l-1}} & \frac{\partial z_2^l}{\partial y_2^{l-1}} & \cdots & \frac{\partial z_{n^l}^l}{\partial y_2^{l-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_1^l}{\partial y_{n^{l-1}}^{l-1}} & \frac{\partial z_2^l}{\partial y_{n^{l-1}}^{l-1}} & \cdots & \frac{\partial z_{n^l}^l}{\partial y_{n^{l-1}}^{l-1}} \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{21}^l & \cdots & w_{n^l 1}^l \\ w_{12}^l & w_{22}^l & \cdots & w_{n^l 2}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n^{l-1}}^l & w_{2n^{l-1}}^l & \cdots & w_{n^l n^{l-1}}^l \end{bmatrix} = \mathbf{W}_{[n^{l-1}\times n^l]}^{*l}$$

Although weights are constant in a trained NN, the activation functions are not purely linear. Such as, for the ReLU activation function, output becomes input for non-negative values of input and output becomes zero for negative values of inputs. The ReLU has different derivatives for different input range. The authors of the paper also presented a table for different activation function. The table is as follows [A]:

Name	Function	Derivative
sigmoid	$f(z) = \frac{1}{1+\exp(-z)}$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right)$
tanh	$f(z) = \tanh z$	$\frac{\partial f}{\partial z}(z) = 1 - \tanh z$
linear	$f(z) = z$	$f'(z) = 1$
ReLU	$f(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$
PReLU	$f(z,a) = \begin{cases} a \cdot z & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z,a) = \begin{cases} a & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$
ELU	$f(z,a) = \begin{cases} a \cdot (e^z - 1) & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z,a) = \begin{cases} a \cdot (e^z - 1) + a & \text{when } z \leqslant 0 \\ a & \text{when } z > 0 \end{cases}$
step	$f(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z,a) = \begin{cases} NaN & \text{when } z = 0 \\ 0 & \text{when } z \neq 0 \end{cases}$
arctan	$f(z) = \arctan z$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^z}$
softplus	$f(z) = \ln(1 + e^z)$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^{-z}}$
softmax	$f_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k (e^{z_k})}$	$\frac{\partial f_i}{\partial z_j}(\mathbf{z}) = \begin{cases} f_i(\mathbf{z}) \cdot (1 - f_j(\mathbf{z})) & \text{when } i = j \\ -f_j(\mathbf{z}) \cdot f_i(\mathbf{z}) & \text{when } i \neq j \end{cases}$

The sensitivity from the output of one layer to the output of next layer will be the weight matrix multiplied by the input-dependent derivatives. To observe sensitivity we, need to keep calculating from the input to the output, layer by layer.

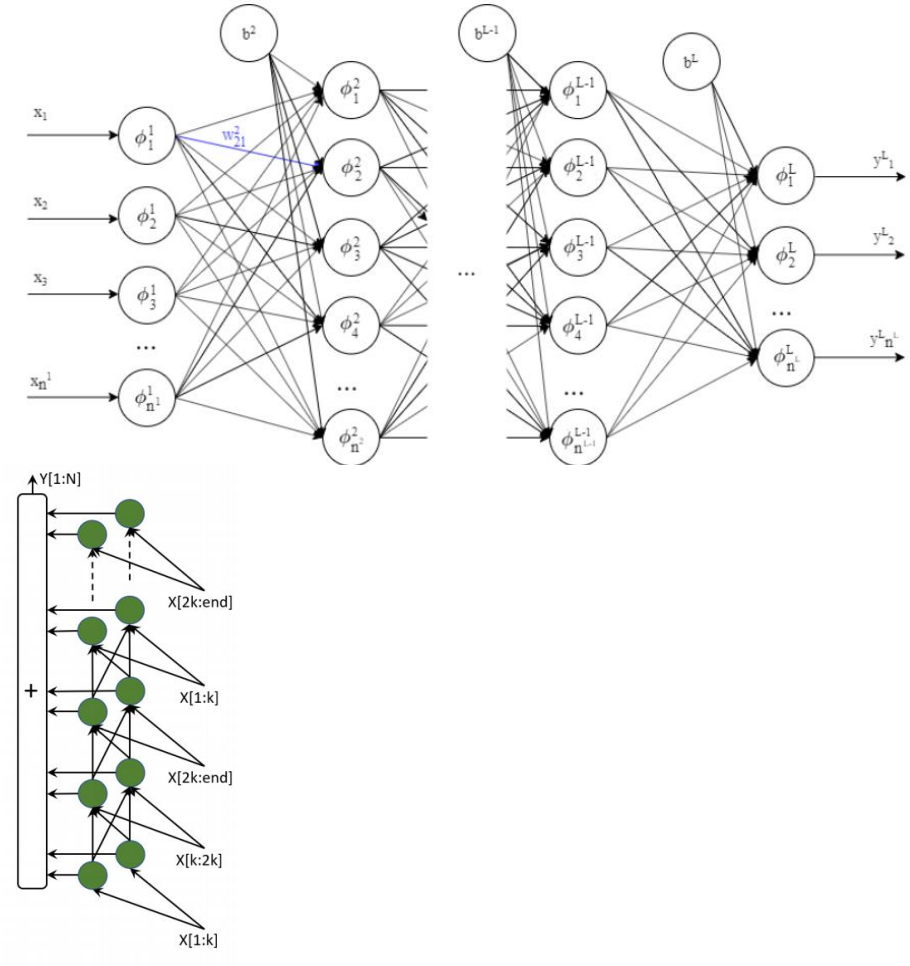


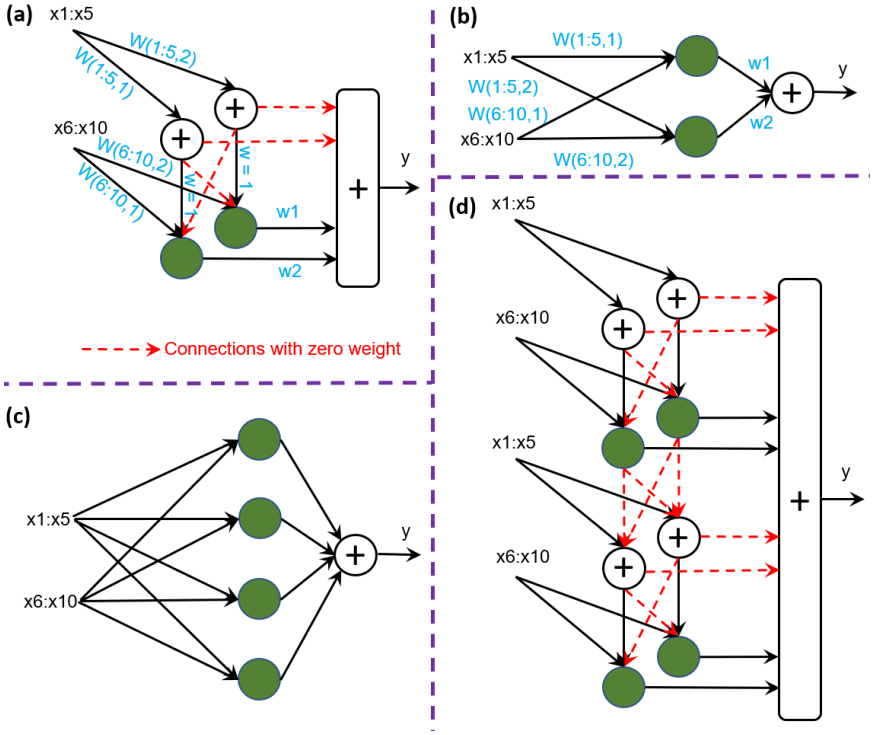
Figure: Traditional DNN [A] and SpinalNet.

The structure of the SpinalNet is slightly different. The first hidden layer is similar to the first hidden layer of traditional DNN. The same matrix multiplication and conditional derivative is applicable between the input and the output of first hidden layer. From the second layer, we need to perform summation of two different matrix multiplication. One matrix is the relation between the current layer and the previous layer. Another matrix is the relation between the current layer and a part of input. Adding the result of these matrix multiplications, we need to consider conditional derivatives in each layer. The output is also summation of outputs of different layers. There is no activation function in the output layer. To find the overall sensitivity of one input or one parameter to output, we need to compute all weights and derivatives of activations situated on the path.

[A] Pizarroso, J., Portela, J. and Muñoz, A., 2020. NeuralSens: Sensitivity Analysis of Neural Networks. *arXiv preprint arXiv:2002.11423*.

Universal Approximation

We have prepared a visual proof in the paper. Here, we are proving that representation for several type of activation functions. The visual proof, we presented in the paper is as follows:



Here, we are showing a mathematical proof with the help of existing theories that how one activation function of first hidden layer may act like an 'purelin' function. The following figure shows the structure of an activation function:

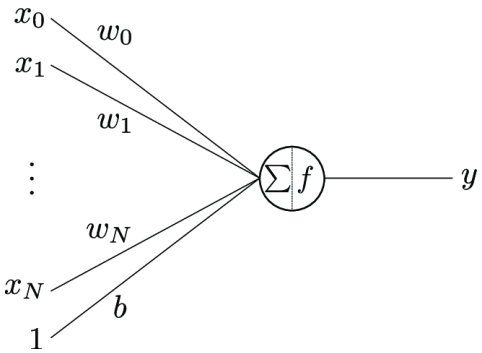


Fig. A neuron, the basic building block of NN. At first, weighted summation of inputs and a bias(Threshold) addition is taken to obtain input of the functional block. An activation function is applied to the input to obtain the output.

Source of the figure
(https://www.researchgate.net/publication/327172877_Structural_Priors_in_Deep_Neural_Networks)

Proof for the Sigmoid Activation:

The sigmoid activation function is defined by the following equation:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1}.$$

The following figure presents the sigmoid function and its derivative.

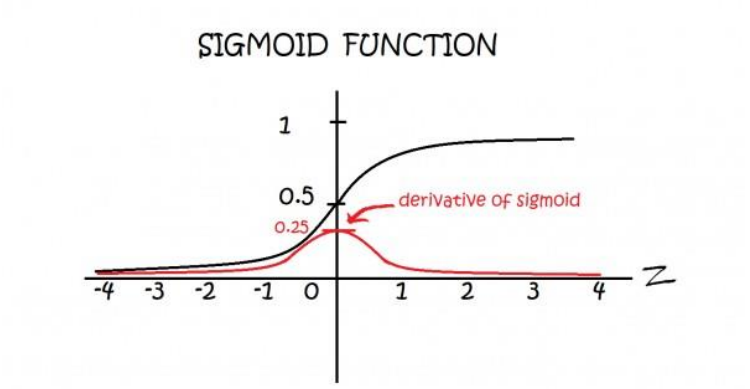


Fig. The Sigmoid function and its derivative. (Collected from https://www.goeduhub.com/10450/what-is-vanishing-gradient-problem?show=10451)

The Sigmoid function has an approximately linear region with non-zero slope when the input is close to zero. Therefore, if incoming weights W(1:5,1) and W(1:5,2) are very small and biases are properly adjusted, we can get summation weighted summation of x1:x5 in the first layer of the NN, shown in (a). For the sigmoid activation function, the output of the first layer will be very close to 0.5, as the linear region exists near 0 (±1e-3) and the output in that region is close to 0.5(±2.5e-4).

Therefore, we need high weight between the first layer and the second layer. Also, we need high bias in the second layer to mitigate the effect of 0.5. Assigning zero weights is also possible. Therefore, 2-layer NN can be approximated to the NN in sub-plot (a). The NN in sub-plot (a) can be equivalent to the NN in subplot (b).

Proof for the ReLU/ Leaky ReLU Activations:

The following figure presents ReLU/ Leaky ReLU activation.

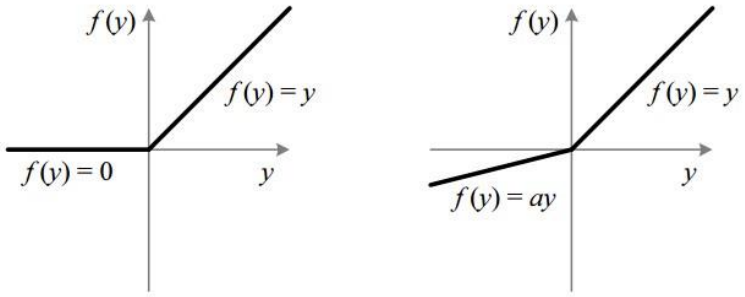


Fig. ReLU/ Leaky ReLU functions have purely linear region with non-zero slope for non-negative inputs. (The figure is collected from: https://www.quora.com/What-is-leaky-ReLU)

To operate the first neuron in positive region, we need to set the bias of the first neuron to a high value. That value should be higher than the highest possible negative value of (x1:x5) × W(1:5,1). The output of the first neuron will also contain the bias value. That bias value needs to be subtracted with the bias value of the identical neuron of the second layer. Therefore, the neurons at the first layer can act as 'purelin' function for ReLU/ Leaky ReLU activation functions.

For other activations:

A small linear or approximately linear region with non-zero slope can be obtained for many other activation functions. Among the function mentioned in the following table step and SoftMax functions have no approximate linear region with non-zero slope. Therefore, the universal approximation is possible for all cost function except step and SoftMax. Step and SoftMax are not used in the first hidden layer. These two functions are usually used in the output layer in classification problems.

Name	Function	Derivative
sigmoid	$f(z) = \frac{1}{1+\exp(-z)}$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right)$
tanh	$f(z) = \tanh z$	$\frac{\partial f}{\partial z}(z) = 1 - \tanh z$
linear	$f(z) = z$	$f'(z) = 1$
ReLU	$f(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$
PReLU	$f(z,a) = \begin{cases} a \cdot z & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z,a) = \begin{cases} a & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$
ELU	$f(z,a) = \begin{cases} a \cdot (e^z - 1) & \text{when } z \leqslant 0 \\ z & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z,a) = \begin{cases} a \cdot (e^z - 1) + a & \text{when } z \leqslant 0 \\ a & \text{when } z > 0 \end{cases}$
step	$f(z) = \begin{cases} 0 & \text{when } z \leqslant 0 \\ 1 & \text{when } z > 0 \end{cases}$	$\frac{\partial f}{\partial z}(z,a) = \begin{cases} NaN & \text{when } z = 0 \\ 0 & \text{when } z \neq 0 \end{cases}$
arctan	$f(z) = \arctan z$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^z}$
softplus	$f(z) = \ln(1 + e^z)$	$\frac{\partial f}{\partial z}(z) = \frac{1}{1+e^{-z}}$
softmax	$f_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k (e^{z_k})}$	$\frac{\partial f_i}{\partial z_j}(\mathbf{z}) = \begin{cases} f_i(\mathbf{z}) \cdot (1 - f_j(\mathbf{z})) & \text{when } i = j \\ -f_j(\mathbf{z}) \cdot f_i(\mathbf{z}) & \text{when } i \neq j \end{cases}$

We can conclude that with the utilization of incoming-outgoing weights and the bias of the next layer, the first hidden layer can act as adder, except for Step and SoftMax activation functions.

Therefore, the SpinalNet can be approximated as a single hidden-layer NN. The number of activation neuron in the first hidden layer depends on the depth of the SpinalNet.

The universal approximability of single hidden-layer NN with several activation functions are already proved in several literature. Such as, the work of Hornik et. al.

[A] Hornik, K., 1993. Some new results on neural network approximation. *Neural networks*, *6*(8), pp.1069-1072.

Bar-chart of accuracy: