# DIFFERENTIABLE LEARNING BY MEANS OF NEURAL NETWORK PRUNING

A DRAFT

**Prathyush S Parvatharaju** [1]
Department of Data Science
Worcester Polytechnic University
Worcester, MA 01609
psparvatharaju@wpi.edu

**Shreesha Narasimha Murthy** [2]
Department of Data Science
Worcester Polytechnic University
Worcester, MA 01609
snarasimhamurthy@wpi.edu

May 16, 2020

**ABSTRACT**

The idea of linear flow i.e, each node in a layer is connected to a certain weight $\hat{\mathbf{W}}_{\mathbf{ij}}$ to every other node in the following layer, for the deep neural network is limiting in the sense of the way we, humans think. It is a constraint for DNN as they process data and emulate relationships in higher dimensions. By replacing the linear flow with a gradient-based decision process combined with skip connections give the ability for the network to develop much deeper constructs from minimal data. We propose a novel idea of extending the capabilities of previously used short-circuits as differentiable functions, essentially solving "What to feed?". To tackle the problem, "When to stop?" we propose an algorithm for early stopping criterion based on Information Transfer derived from differentiable short-circuits.

In our experiments, we show that the increase in weighted representation capability of the network results in achieving better accuracy in fewer iterations compared to the standard architecture. We also demonstrate that with 10% of the previous parameters, the architecture achieves similar results and builds better representations with minimal data compared to a standard architecture. In the end, we demonstrate differentiable short-circuit's ability to visualize the relevant features for a given layer as it learns to drop common features among different classes and this reduces class ambiguity and thereby increasing the confidence score of a class.

***K*eywords** Differentiable Learning · Pruning · Neural Architecture Search

## 1 Introduction

The ability to recognize patterns and develop strong relations among them has made neural networks supersede state of the art machine learning techniques. The depth of representations is of central importance for many visual recognition tasks. Right from Alex Net [1] up until ResNet [2] the flow of data followed a linear fashion i.e the data flow from one layer to the next layer and so on. ResNet brought in innovation by creating *"Short Circuits"* - there exists a skip connection where, as usual, data is fed to the next layer and in certain circumstances, it is fed to the layer after its immediate next. This paved way to train deep-networks as it increased the representational capability of the network by overcoming issues such as vanishing / exploding gradients [3, 4]. In our work we set out a goal to explore the *"representational capability"* and its effects on the performance of the network. This led to the question - "What to feed?".

ResNet brings in short circuits between residual components and the number of skip connections is restricted. We propose *"Global Short Circuit"*(GSC), a neural architecture to investigate the behavior of short circuits by revoking the restriction and providing skip connections from one layer to every other layer. Thereby enhancing the representational capability of the network and therefore yields better accuracy in fewer iterations compared to the standard architecture (See section 2.1). However, this modification results in a large network, due to which we observer an exponential increase in the trainable parameters, rendering the network unscalable. As a solution, we propose an enhanced version of

GSC i.e, *" Differentiable Short Circuit"*(DSC) - a novel neural architecture to help network scale by pruning unwanted connections over time.

To tackle the issue of *"When to stop"*?, we propose *Information Transfer Rate*. It is a measure of the amount of flow of data from one layer to another. Keeping track of the variance of *information transfer rate* over iterations, based on a certain threshold, we formulate a stopping criterion.

## 2 Differentiable Learning

Neural networks are differentiable by design. The standard neural architecture (Section2.1) is static and does not change over time. However, learning changes over time. To adapt the neural architecture to the dynamics of learning, there have been efforts towards an automated way of finding the best architecture for the given data i.e, Neural Architecture Search (NAS) [5]. The recent advent of DARTS [6] a differentiable NAS, suggests that a learning-based adaptive architecture performs better compared to a static architecture. To understand the nature of selection criteria, we have to learn the connectivity amongst individual neurons. For instance, given $N$ layers, to identify the optimal data paths, we can use brute-force and try out $n(n-1)$ combinations of connections. However, as the depth of the network increases, the search space increases exponentially, and training each configuration becomes a tedious and time-consuming process. Gradient descent [7] - a first-order iterative search space optimization algorithm that finds a local minimum of a differentiable function can be used to find the optimal skip connections. Initially, we describe a standard feed-forward neural network and later propose, Global short circuit (GSC) and Differentiable Short Circuit (DSC) neural architectures.

### 2.1 Feed Forward Neural Network

Multilayer perceptron trained using back-propagation [8], a supervised learning algorithm is capable of approximating solutions for complex problems. Acting as a universal approximator, the architecture has a linear flow; i.e, a layer takes the output of the immediate previous layer alone as its input. Also known as a densely connected layer, it provides learning features from all the combinations of the features of the previous layer. With a linear increase in the number of connections (based on the nodes in the layer) the architecture is scalable and is time-efficient. $\hat{\mathbf{Z}}$ in Equation 1 is the result of an affine transformation and is capable of learning an offset ($\hat{\mathbf{b}}$) and a rate of correlation ($\hat{\mathbf{W}}$) that fits the given input.

$$\hat{\mathbf{Z}} = Input \cdot \hat{\mathbf{W}} + \hat{\mathbf{b}} \tag{1}$$

This type of transformation can only learn linear relationships. In order to learn any non-linearity, a squashing non-linear transformation - sigmoid $(\sigma)$ [9] is applied to the output of each node. Termed as an activation function, it results in a probability as it restricts the range of the output between 0 and 1.

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad x \in \mathbb{R}, 0 < \sigma(x) < 1 \tag{2}$$

Figure 1 represents a standard feed-forward architecture and equation 11 is the mathematical formulation of the network. Each node in one layer is connected to a certain weight $\hat{\mathbf{W}}_{\mathbf{ij}}$ to every node in the following layer.

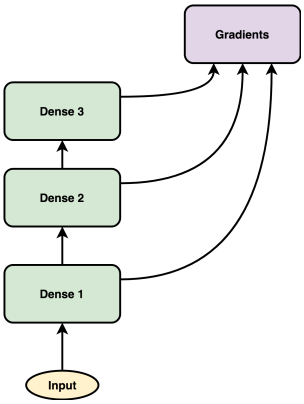



Figure 1: Feed Forward Network

$$\begin{aligned} Dense_1 &= \sigma(Input \cdot \hat{\mathbf{W}}_1 + \hat{\mathbf{b}}_1) \\ Dense_2 &= \sigma(Dense_1 \cdot \hat{\mathbf{W}}_2 + \hat{\mathbf{b}}_2) \\ Dense_3 &= \sigma(Dense_2 \cdot \hat{\mathbf{W}}_3 + \hat{\mathbf{b}}_3) \end{aligned} \tag{3}$$

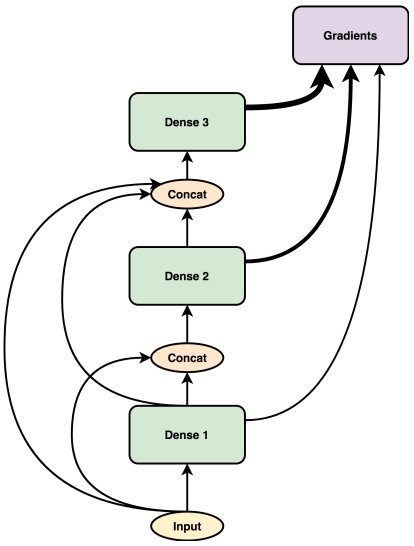
For the previously mentioned standard network, the number of trainable parameters for a given layer is a function of the layer's weight, bias, and input. For a given network, the same is represented as,

$$
\begin{aligned}
Params = \sum_{n=1}^{N} (\underbrace{l_{n-1}}_{\text{input}} * \underbrace{\quad l_n)}_{\text{weights}} \quad + \underbrace{l_n}_{\text{bias}} \\
\text{where } N &= \text{Number of layers} \\
l_n &= \text{Number of nodes in the layer}
\end{aligned}
\tag{4}
$$

For a 3 layer network with node configurations [100,50,10] and 784 input features, using equation 4 we arrive at 83,550 trainable parameters

### 2.2 Global Short Circuit (GSC)

This work proposes *"Global Short Circuit"* neural architecture to explore the impact of skip connections on the representational capabilities of the network. There exists a connection from every layer to every succeeding layer in the network. Some of the possible ways of merging layers are explored in DenseNet[10] and ResNet[2]. In ResNet weighted feature maps are merged using $max$ or $sum$. DenseNet merges features using a concatenation operation. We define a merge operation similar to DenseNet's $concat$ . However, this type of merging has an issue when it comes to Convolutional layers, where there is a requirement to merge feature maps of different shapes, unlike feed-forward layers. As a solution, we use downsampling/padding operators to get identical shapes for the merge operation. Figure 2 shows the standard architecture defined in section 2.1 having connections from one layer to the subsequent layers, merged using concat operation. The difference in the connections between the layers is relative to the number of gradient operations required for the respective layer. Equation 5 provides mathematical representation of the architecture.




$$
\begin{aligned}
Dense_1 &= \sigma(Input \cdot \hat{\mathbf{W}}_1 + \hat{\mathbf{b}}_1) \\
Dense_2 &= \sigma([Input \oplus Dense_1] \cdot \hat{\mathbf{W}}_2 + \hat{\mathbf{b}}_2) \\
Dense_3 &= \sigma([Input \oplus Dense_1 \oplus Dense_2] \cdot \hat{\mathbf{W}}_3 + \hat{\mathbf{b}}_3)
\end{aligned}
\tag{5}
$$

Figure 2: Global Short Circuit

Experiments show that GSC achieves better accuracy compared to standard architecture on various datasets (See section 3). This is due to increased representational capacity of the network. A layer *n* in the network now has access to every *n-1* previous outputs. This change paves way to create much deeper constructs and complex relationships among the layers. To advocate that our proposed method - GSC creates much deeper constructs than the standard architecture, the class confidences of the predictions are measured. Confidence score is unique for each class and the use of *Softmax* function restricts the score between 0 and 1. At the end of each epoch, mean of class confidence scores are calculated across a large unseen test set. This measure speaks for the trust in the prediction. Among the variable factors of the measure, the rate of increase and the actual value are vital for the analysis. MNIST [11], Fashion-MNIST [12], CIFAR-10 datasets [13] are chosen to account for the variability in the pattern complexity. The training samples are restricted to 10% of the whole, while still maintaining a large test sample to test the described hypothesis. Figure 3 shows the performance when the node level configurations across standard and GSC are the same. GSC has 30% more trainable parameters due to merge operation. To rule out the difference in the trainable parameters as a reason

for achieving high confidence scores, we create 10p architectures. Here, the node level configurations across the models varies but the number of trainable parameters are kept constant across all architectures. The performance of 10p architectures are shown in Figure 4.


Figure 3: 10p Data, 100p Model


Figure 4: 10p Data, 10p Model

In figure 3 the difference in the confidence scores of MNIST among standard and GSC is negligible. This is due to the lack of complex patterns across classes in the dataset. There is a marginal difference in the score in Fashion MNIST and a distinct difference in CIFAR-10. CIFAR-10 comprises of 10 classes of real-world complex object patterns and GSC neural architecture enables the model to create much deeper constructs compared to the standard architecture. We notice a similar behavior in the confidence measure across 10p architectures where the number of trainable parameters across the networks is kept constant.

However, GSC suffers from an exponential increase in connections resulting in delayed training time. A rise in the number of stacked layers is noticed with the increase in the complexity of the network in terms of depth and is not a scalable solution. Compared to the standard architecture, GSC has a considerable expansion in the number of training parameters. This is due to the effect of merge operations of previous $n-1$ layers for every $n$ layer.

$$Params = \sum_{n=1}^{N} \left[ \underbrace{(\sum_{i=0}^{n} l_i}_{\text{merged inputs}} * \underbrace{l_n)}_{\text{weights}} + \underbrace{l_n}_{\text{bias}} \right] \tag{6}$$

where $N$ = Number of layers

$l_n$ = Number of nodes in the layer

For the aforementioned 3 layer network, an exponential increase of 30% i.e, 122,750 parameters has to be trained.

### 2.3 Differentiable Short Circuit (DSC

Finally, we propose a Differentiable Short Circuit (DSC) neural architecture to counter the effect of the exponential increase in training parameters due to merge operation. It is done so by expressing the connection between layers as a weighted sigmoid gate. For a connection, we introduce connection weight $\hat{\mathbf{C}}$, a trainable parameter whose suitable values are found using $\frac{\partial loss}{\partial \hat{\mathbf{C}}}$. A change in the value of the connection weight affects the loss. Based on this, we use the gradient descent algorithm to obtain optimum values for connection weight. The direction and magnitude of the weight affect the performance of the network, by introducing undesired changes in the architecture.

To check these effects, a transformation function that is capable of restricting the direction and magnitude of the weight in a specified range is used. Neural arithmetic logic unit [14] propose a layer whose transformation matrix $\hat{\mathbf{W}}$ consists just of -1's, 0's and 1's. To prevent the layer from changing the scale of the representation of the numbers when mapping the input to the output, a squashing non-linearity ($\sigma$) is applied on just the weights, $\hat{\mathbf{W}}$. Hence, the sigmoid function, defined in equation 2, is applied on $\hat{\mathbf{C}}$. Along with forcing the range of output to be in a specific interval, it also helps in preserving the direction of the input.

With the use of weighted sigmoid gates, gives the network a chance to distinguish between essential and redundant data pathways. Based on a certain threshold, we propose a pruning algorithm which over time, learns to delete redundant connections. Each value in the weight of a layer is optimized based on the conditional dependence among others. Hence it is essential to start with all possible connections so that the network can develop stronger constructs before the prune operation. Pruning introduces sparsity in the network, decreasing training time exponentially. The proposed solution has the advantages of GSC without adding a major overhead to the training time. Figure 5 shows the use of

connection weights, applied to the output of concat operation, essentially keeping the connection in check before it is fed to the next layer. The difference in the representation of connections between various blocks and the gradient operation shows the number of gradient computations required for the layer or the operation. Equation 7 provides mathematical representation of the network.

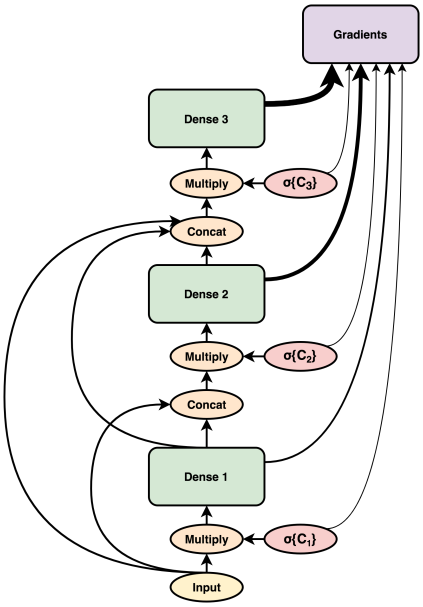



Figure 5: Differentiable Short Circuit

$$
\begin{aligned}
Dense_1 &= \sigma((\sigma(\hat{\mathbf{C}}_1) \cdot Input) \cdot \hat{\mathbf{W}}_1 + \hat{\mathbf{b}}_1) \\
Dense_2 &= \sigma((\sigma(\hat{\mathbf{C}}_2) \cdot [Input \oplus Dense_1]) \cdot \hat{\mathbf{W}}_2 + \hat{\mathbf{b}}_2) \\
Dense_3 &= \sigma((\sigma(\hat{\mathbf{C}}_3) \cdot [Input \oplus Dense_1 \oplus Dense_2]) \cdot \hat{\mathbf{W}}_3 + \hat{\mathbf{b}}_3)
\end{aligned} \tag{7}
$$

For DSC, we notice a slight increase in trainable parameters compared to GSC's equation 6 due to the use *weighted gates*. There exists a trainable gate for every connection. The number of parameters for connections is defined by $\sum_{i=0}^{n} l_i$, where $l_i$ represents the number of incoming connections for a layer. As a result, for the network structure defined in the standard architecture, the number of trainable parameters are 124,418, a rise of 1.3% over GSC and 32% over the standard architecture. However, due to the use of pruning, the number of parameters decreases significantly over time, reducing training overhead.

$$
Params = \sum_{n=1}^{m} \left[ \underbrace{(\sum_{i=0}^{n} l_i}_{\text{merged inputs}} \; * \; \underbrace{l_n)}_{\text{weights}} \; + \; \underbrace{l_n}_{\text{bias}} \; + \; \underbrace{\sum_{i=0}^{n} l_i}_{\text{connection weights}} \right] \tag{8}
$$

where $N$ = Number of layers
$l_n$ = Number of nodes in the layer

The benefit of having trainable gates are of two folds,

1. By keeping track of the variance of $\sigma(\hat{\mathbf{C}})$ over time has proven to be useful in designing a stopping criterion, elucidated in section 2.3.1. The connection weights associated with the input layer to every other layer can be expressed as a saliency score for a pixel in the input. Using these, we visualize the network's behavior in selecting the required features and discarding the rest. (Details in Section 3.3)

2. Since the output of the gates is probabilistic, we use a binary threshold function to transform continuous values into a binary mask. The mask is then used to prune unwanted skip connections. Section 2.3.2 describes the pruning methodology in detail.

### 2.3.1 Information Transfer Rate

We propose *Information transfer rate* (itr) as a percentage measure of the data flow from one layer to the next. The weighted gated connections having sigmoid activation translates to a scaling operation. Each output is scaled based on its significance before the same is fed to the next layer. The scaling transformation is also applied to skip connections as shown in figure 5. Equation 9 describes the rate of information transfer from one layer to the next.

$$itr_{layer} = \frac{\sum_{c=1}^{N} \hat{\mathbf{W}}_c}{|\hat{\mathbf{W}}_c|},$$

$$\begin{aligned} \text{where } N &= \text{Number of nodes,} \\ \hat{\mathbf{W}}_c &= \text{Connection weight} \\ |\hat{\mathbf{W}}_c| &= \text{Length of connection weight} \end{aligned} \tag{9}$$

To calculate the information transfer of the network, based on the number of layers and their respective information transfer rates, we propose Equation 10. It is a conditional mean of $itr_{layers}$. There are no *recurrent* associations and connections are always *unidirectional*. Figure 6 represents $itr_{layer}$ in a matrix format. The first column represents *"from"* and the first row shows *"to"* relations. Ex: From $Dense\ 1$ to $Dense\ 3$ the gated data flow is measured at 26%.

| Contents | Dense 1 | Dense 2 | Dense 3 |
|---|---|---|---|
| Input | 100 % | 50 % | 67 % |
| Dense 1 | -- | 32 % | 26 % |
| Dense 2 | -- | -- | 35 % |

Figure 6: Sample Information Transfer

$$itr_{network} = \frac{\sum_{i=0}^{n-1} \sum_{j=1}^{n} \begin{cases} itr_{ij}, & \text{if } i < j \\ 0, & \text{otherwise} \end{cases}}{\frac{n(n+1)}{2}}$$

$$\text{where } n = \text{Number of layers} \tag{10}$$

Figure 7 shows that over time, the number of features picked by the network saturates. On close observation, the saturation point of the $itr_{network}$ metric is similar to the saturation point of the accuracy function. MNIST and Fashion MNIST datasets are relatively easy datasets compared to CIFAR-10 and the saturation point of the metric can be used as an early-stopping criterion. In CIFAR-10, the metric changes over time and so does the accuracy and hence it does not meet the early exit criterion.

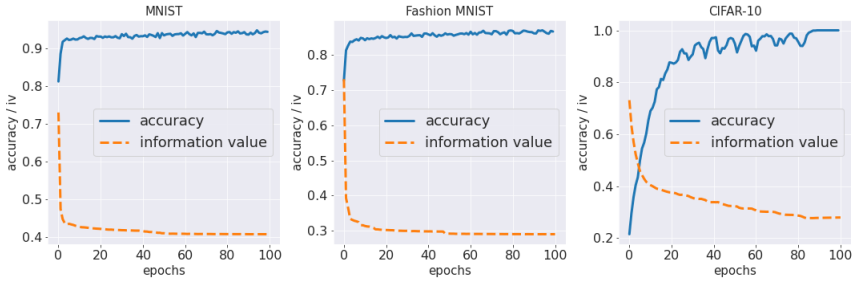



Figure 7: Information Transfer Rate vs Accuracy

### 2.3.2 Pruning

To address the issue of an exponential increase in trainable parameters due to the merge operation in GSC, differentiable short circuit's weighted connection gates are used to selectively prune redundant nodes by removing connections. If the current input is not important to the task, we define a binary function $f_{binary}$ whose output, when 0, the current input information shall not be passed on to the next layer. The input to the binary function ranges between 0 and 1. The probability of obtaining 0 or 1 for the binary function can be computed by,

$$\begin{aligned} P(f_{binary(x)} = 1) &= x \\ P(f_{binary(x)} = 0) &= 1 - P(f_{binary(x)} = 1) \end{aligned} \tag{11}$$

However, the binary connection gate function is not differentiable and breaks backpropagation. There are generally two solutions. The first is to REINFORCE algorithms [15] and considering its applications, it is computationally expensive and the reward is difficult to design to approximate the gradients. In recent times, few gradient estimators are proposed and among them, the Straight through gradient estimator [16] is found to be useful due to its reduced computational complexity. The function approximates the step function by the identity when computing gradients during the backward pass:

$$\frac{\partial f_{binaray}(x)}{\partial x} = 1 \tag{12}$$

With the implementation of a straight-through estimator, the model parameters can now be trained to optimize the use of connection weights $\hat{\mathbf{C}}$ with standard backpropagation without defining any additional supervision or reward signal. This assures that the computation complexity does not increase.

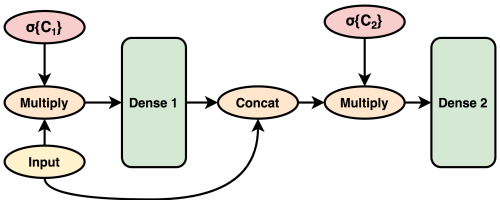



Figure 8: 2-Layer DSC architecture

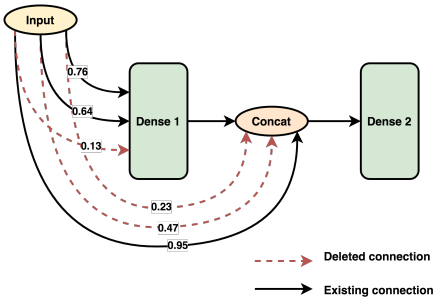



Figure 9: Pruned connections

Figure 8 shows a 2-layered DSC network with all the necessary operations (concat, multiply . . .) including skip connection from input to the dense layer. To the right, Figure 9 shows connections from input to dense layers. The floating-point number represents the value of $\sigma(\hat{\mathbf{C}}_\mathbf{i})$. Based on the binary gate, a threshold of 0.5 is used, and the ones having values less than 0.5 have a different representation to mark the connection to be pruned.

**Algorithm 1** Dependency based pruning

**Result:** Prune the network based on binary mask
**for** *layer in network layers;* **do**
  | *prune nodes*$^{(i)}$ ← *compute nodes to prune(layer)*
**end**
i ← *Number of layers*
**while** *i* < *0* **do**
  | *layer* ← *prune(layer*$^{(i)}$*, prune nodes*$^{(i)}$*)*
**end**
**procedure** PRUNE($layer, nodes$)
  $layer.weight \leftarrow layer.weight[:, nodes]$
  $layer.bias \leftarrow layer.bias[:, nodes]$
  $layer.connections \leftarrow layer.connections[nodes]$
  **return** $layer$
**end procedure**

In the first for loop we iterate the network layer wise, build dependency graph, collect the nodes falling under prune criteria and store it in $prune\ nodes$. We then iterate backwards to change layer weights, biases and connection weights based on $prune\ nodes$.

The $prune$ function takes a layer and the corresponding prune indices and calculates new layer dimensions. It also clones the layer weights, layer bias, connection weights and extracts appropriate indices to match the layer dimensions of the new layer. Finally the new set of parameters are assigned to layer created in the previous step.

## 3 Experiments

For the experiments, we prefer to use MNIST, Fashion-MNIST and CIFAR-10 datasets. These datasets are chosen based on the complexity in the class patterns. We run hyperparmeter tuning [1] across CV-5 for each of our experiments. This ensures that each network is tuned respectively to get the best performance. We define two sets of data and network architectures,

[1]Combinations of Batch size[32,64,256], learning rate[1e-1, 1e-2, 1e-3], learning rate decay[False, 1] and regularization coefficients[1, 1e-1, 1e-2, 1e-3] are tried against 5 splits of training, validation and test datasets. Tables 1 and 2 describes the splits across datasets in detail.

- 100p Data (Table 1) - Model is trained on all of the available training data and is tested against the standard set containing 10,000 samples. This is a standard set to determine the optimum performance of the networks.

- 10p Data (Table 2) - Model is trained on 10% of the available training data and is tested against a large set i.e, 10,000 samples. We use this experiment to test our hypothesis - GSC and DSC neural architectures enable models to develop deep constructs compared to the standard architecture.

- 100p Params (Table 3) - This set defines a standard neural architecture for the model design, where the number layers and the number of nodes in the layer are kept constant across architectures. However, we see a 30% rise in trainable parameters for GSC and DSC architectures compared to the standard architecture. This is due to the merge operation on skip connections.

- 10p Params (Table 4) - Here, 10% of the original DSC's architecture is used. Based on the resulting trainable parameters, the other networks are designed to match this. Though the architectures are different, the number of trainable parameters are kept constant across the networks in this set. It is designed as a counter to the argument - The difference in the performance of the networks is solely dependent on the number of trainable parameters.

| Dataset | Train | Valid | Test |
|---|---|---|---|
| MNIST | 50000 | 10000 | 10000 |
| Fashion MNIST | 50000 | 10000 | 10000 |
| CIFAR | 40000 | 10000 | 10000 |

Table 1: 100p Data sample split

| Dataset | Train | Valid | Test |
|---|---|---|---|
| MNIST | 5000 | 1000 | 10000 |
| Fashion MNIST | 5000 | 1000 | 10000 |
| CIFAR | 4000 | 1000 | 10000 |

Table 2: 10p Data sample split

| Model | MNIST | Fashion MNIST | CIFAR-10 |
|---|---|---|---|
| FFN | 83550 | 83550 | 162810 |
| GSC | 122750 | 122750 | 224250 |
| DSC | 124418 | 124418 | 226448 |

Table 3: 100p Model Parameters

| Model | MNIST | Fashion MNIST | CIFAR-10 |
|---|---|---|---|
| FFN | 32425 | 32425 | 31215 |
| GSC | 15800 | 15800 | 25775 |
| DSC | 17378 | 17378 | 27838 |

Table 4: 10p Model Parameters

### 3.1 100p Data - Large Training Set, Large Test Set

Results show that DSC achieves the best performance across different datasets compared to standard and GSC architectures. The difference in performance is noticeably higher in CIFAR-10 due to its increased pattern complexity. Our proposed architecture achieves better accuracy in fewer iterations ( 5000 iterations less) compared to standard architecture. When the 10p model is used, the accuracy does drop but is relatively better than the standard network.

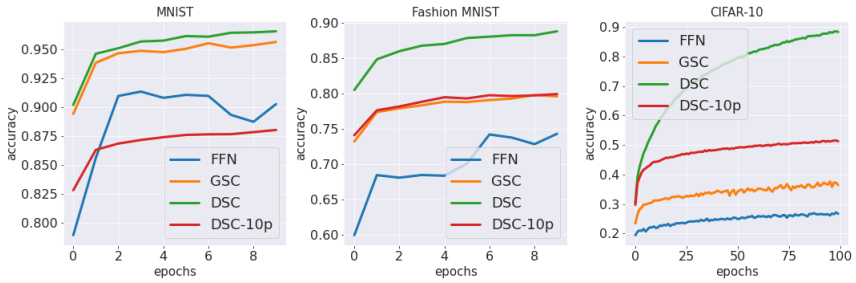



Figure 10: 100p Train comparison

| Model | Parameter | Dataset | Train | | Test |
|---|---|---|---|---|---|
| | | | Accuracy @ Steps | Loss @ Steps | Accuracy |
| Linear FFN | 83550 | MNIST | 0.90 @ 7000 | 0.02 @ 7000 | 0.91 |
| | | FashionMNIST | 0.74 @ 7000 | 0.05 @ 7000 | 0.74 |
| | | CIFAR-10 | 0.27 @ 7000 | 1.87 @ 7000 | 0.20 |
| GSC | 122750 | MNIST | 0.93 @ 1400 | 0.0113 @1400 | 0.93 |
| | | FashionMNIST | 0.72 @ 700 | 0.0481 @ 700 | 0.79 |
| | | CIFAR-10 | 0.28 @ 2100 | 2.53 @ 7000 | 0.29 |
| **DSC** | **124418** | **MNIST** | **0.95 @ 1400** | **0.01 @ 1400** | **0.96** |
| | | **FashionMNIST** | **0.74 @ 700** | **0.05 @ 700** | **0.79** |
| | | **CIFAR-10** | **0.3 @ 700** | **2.1 @ 700** | **0.40** |
| **DSC-10p** | **17378** | **MNIST** | **0.89 @ 2800** | **0.02 @ 2100** | **0.89** |
| | | **FashionMNIST** | **0.71 @ 700** | **0.07 @ 700** | **0.74** |
| | | **CIFAR-10** | **0.29 @ 700** | **2.2 @ 700** | **0.38** |

Table 5: 100% Data Results

### 3.2 10p Data - Minimal Training Set, Large Test Set

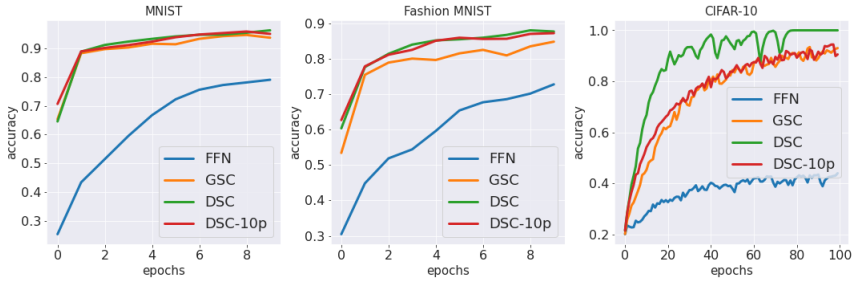



Figure 11: 10p Train comparison

| Model | Parameter | Dataset | Train | | Test |
|---|---|---|---|---|---|
| | | | Accuracy @ Steps | Loss @ Steps | Accuracy |
| Linear FFN | 83550 | MNIST | 0.79 @ 7000 | 0.70 @ 7000 | 0.74 |
| | | FashionMNIST | 0.72 @ 7000 | 0.70 @ 7000 | 0.7 |
| | | CIFAR-10 | 0.44 @ 7000 | 1.45 @ 7000 | 0.22 |
| GSC | 122750 | MNIST | 0.79 @ 2100 | 0.69 @ 2100 | 0.85 |
| | | FashionMNIST | 0.72 @ 2100 | 0.70 @ 2100 | 0.76 |
| | | CIFAR-10 | 0.44 @ 700 | 2.46 @ 700 | 0.28 |
| **DSC** | **124418** | **MNIST** | **0.88 @ 1400** | **2.13 @ 1400** | **0.89** |
| | | **FashionMNIST** | **0.77 @ 1400** | **1.91 @ 1400** | **0.79** |
| | | **CIFAR-10** | **0.64 @ 700** | **1.05 @ 700** | **0.33** |
| **DSC-10p** | **17378** | **MNIST** | **0.75 @ 700** | **1.56 @ 700** | **0.87** |
| | | **FashionMNIST** | **0.77 @ 1400** | **1.51 @ 1400** | **0.77** |
| | | **CIFAR-10** | **0.43 @ 700** | **1.6 @ 700** | **0.26** |

Table 6: 10% Data Results

This set tests the generalization of the networks. Figure 11 shows the performance of DSC against standard architecture across different datasets. As the number of training samples is limited to 10% of the original data and a large test set it adopted, we monitor whether the network has an issue of overfitting. This is not the case with our proposed architecture. Table 6's Test accuracy of DSC establishes that our proposed architecture has reduced overfit compared to others. There exists a noticeable drop in the performance of standard architecture in test metrics when compared to 100p data. The same is not observed in the DSC, as the 10p data model follows 100p data more closely.

### 3.3 Understanding connection weights ($\hat{C}$)

To comprehend the nature of gated connections, we create thresholded heatmaps of the weights. This is possible since connection weight is a function of $\sigma$ and skip connections from input to every other layer is considered. Figure 12 and 13 is a visual representation of connection weights from input to the other three layers defined in section 2.3 for MNIST and Fashion MNIST respectively. The borders of the input do not contain any valuable information and the network learns to discard these connections over time. The model learns to drop common features among different classes and this reduces class ambiguity and thereby increasing the confidence score of a class. The mean utilization of inputs represents the commonly used pixels across different classes and layers with high importance.

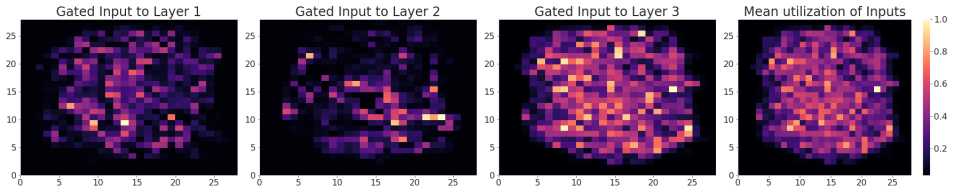



Figure 12: MNIST Dataset

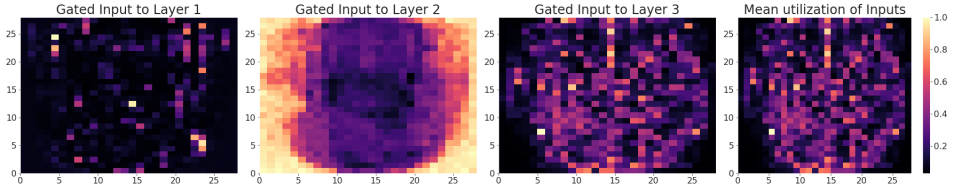



Figure 13: Fashion MNIST Dataset

## 4 Conclusion

Our work establishes the usefulness of global short circuits in improving the performance of the network by enabling the model to develop much deeper constructs compared to the standard architecture. We elucidate issues with global short circuit and propose differentiable short circuit as a solution. Differentiable short circuit builds upon GSC with the goal of scalability. DSC architecture with the help of connection weights brings in innovations such as Information Transfer Rate, Pruning, Layer intake visualization among others. The architecture achieves the top accuracy in fewer iterations compared to standard architecture and GSC across all datasets.

From our experiments, we can conclude that DSC reduces model overfit, achieves high training accuracy while still maintaining a relatively high test accuracy. As a counter to the argument - increase in the trainable parameters is solely responsible for the increase in performance, we design and test a relatively smaller network, where the number of trainable parameters is kept constant across architecture and the results show that DSC performs better compared to other architectures. The added benefit is that our proposed method is capable of learning at a much faster pace ($learningrate > 0.1$) and with the optimal use of learning rate scheduler, the method achieves local minima in fewer iterations compared to other models. With the help of pruning, we introduce the factor of sparsity which is aimed at reducing the number of trainable parameters while still maintaining a fairly high test accuracy. This can be highly beneficial on mobile and embedded devices where hardware resource is scarce.

## 5 Future Work

DSC on feed-forward neural networks stands as a testimony proving the proposed new architecture holds water. We see the surge in performance when skip connections along with differentiable units are introduced.

- We plan to implement a Differentiable Convolutional Neural Network, where a layer $N$ is fed filters from every layer before it i.e. ($N-1$ to 1).
- Also, many interesting questions about how a Deep Neural Network captures representations can be answered from the process of Differentiable Network Pruning. To name a few,
  - One can see if DSC has modelled any direct relationship between input layer and output layer(as DSC maintains connection weights between input and output. If the connection weights are close to 1, then all of input is being used by the output layer).
  - Also, it is interesting to analyze which is the layer on which the output depends the most(i.e. output layer feeds off of every previous layer's output).
- Analyze the parameters to be cut down by pruning and dynamically control the pruning percentage based on required accuracy levels or training times etc.

## References


[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[3] Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 2:157–66, 1994.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[5] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2016.

[6] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ArXiv*, abs/1806.09055, 2019.

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. 1986.

[9] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *IWANN*, 1995.

[10] Chia-Yu Li and Ngoc Thang Vu. Densely connected convolutional networks for speech recognition. In *ITG Symposium on Speech Communication*, 2018.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 1998.

[12] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[14] Andrew Trask, Felix Hill, Scott E. Reed, Jack W. Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *NeurIPS*, 2018.

[15] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[16] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.