Dimension Mixer Model: Group Mixing of Input Dimensions for Efficient Function Approximation

Suman Sapkota*
NAAMII
Kathmandu, Nepal
suman.sapkota@naamii.org.np

Binod Bhattarai
University of Aberdeen
Aberdeen, UK
binod.bhattarai@abdn.ac.uk

Abstract

The recent success of multiple vision architectures like CNN, ViT and MLP-Mixer motivates us to find the similarities and differences between these architectures. We find that these architectures with MLPs could be interpreted using a general concept of dimension mixing, which we call the Dimension Mixer Model. Dimension Mixer Model requires function approximators like Neural Networks to communicate between all input dimensions. The literature on Coupling-Flows, ShuffleNet, Megatron-LM etc. supports that partial signal mixing is sufficient for function approximation. We find that non-linear, learnable, multi-layered and *group-wise sparse mixing* of input dimensions is sufficient for approximating any function over the inputs. Using butterfly structure mixing inspired by Fast Fourier Transform (FFT), which we call butterfly mixer, we develop a non-linear mixer called MLP Butterfly Mixer and Grid-Bilinear Butterfly Mixer models. We also propose sparse Butterfly Attention for Transformers architecture. Experiments show these butterfly structured dimension mixer models are highly scalable and efficient. We also develop a new type of vision MLP-Mixer model called Patch-Only MLP-Mixer based on the dimension mixing theory which produces competitive accuracy compared to the original MLP-Mixer across datasets. We also touch on the benefits of the Dimension Mixer Model for distributed architectures. We show these resultant architectures as an application of the Dimension Mixer Model.

1 Introduction

The recent success of various Neural Network Architectures such as CNN(32; 31; 45; 24), Transformers(55; 15), and MLP-Mixer(53) can be credited to the structured processing of input signals and the function or parameter sharing used in these architectures. These architectures can perform similar tasks like image classification while working differently from each other. Although these architectures share common methods such as processing in patches and sharing processing functions across patches, they also mix signals at different locations in different ways depending on the architecture. This abundance of processing the same input in different ways for achieving the same task allows us to analyse and find common mechanisms among these architectures, their benefits and drawbacks.

Furthermore, partial signal processing is observed in many deep architectures. Architectures such as Coupling-Flows(14; 20) suggest that partial mixing of inputs is sufficient for dense unstructured input signals as well as provides the benefit of invertibility. ShuffleNet(59; 39) shows that efficiency can be increased with decreased parameters if we use partial channel mixing along with the shuffling of channels. Megatron-LM(44) uses sparse processing by sharding blocks over two clusters. These

*https://tsumansapkota.github.io/

Preprint. Under review.

architectures motivate us to find a general parallelizable and efficient signal processing method by sparsely processing input signals and combining the hidden states in subsequent layers.

Neural Network Architectures are generally used for processing structured data, which can be highly parallelisable and also parameter efficient due to function sharing. However, for unstructured input signals, general dense networks do not scale well with the dimension of input signals. Furthermore, general sparse matrix multiplication algorithm is slower than dense multiplication in GPUs. The low-rank matrix reduces complexity by multiplying two smaller matrices and is applicable in various architectures(28). A structured sparse matrix such as butterfly matrices have been used previously in Fast Fourier Transform (FFT) (11) and on Neural Networks for efficient matrix multiplication (42; 5; 13) by approximating a dense matrix.

Considering the motivation for finding the general mechanism of different vision architectures, sequence models and unstructured architectures as well as the efficiency of accelerated hardware like GPUs, we propose a general sparse signal processing model called as Dimension Mixer Model. Dimension Mixer Model is a signal processing model that uses group-wise mixing of input signals, processed in layer-wise fashion such that every input signal communicates with each other, (i.e. for $N \rightarrow M$ transformation, all M output dimensions can have non-zero derivative with all N inputs dimensions.) Dimension Mixer Model consists of parallel Dimension Select and Process Units. This mechanism is repeated in layers such that input dimensions mix completely. This is depicted by Figure 1. Although Dimension Mixer Models can have heterogeneous dimension select methods and processing units, structured mixing is preferable for utilizing GPUs.

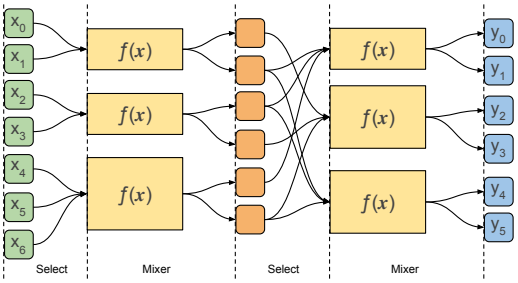


Figure 1: An example of a general semi-structured Dimension Mixer Model. The model is split into multiple layers of (i) **Select** and (ii) **Mixer** functions. The Select method selects input dimensions for each of the Mixer units and the Mixer unit processes the signal and is learned via Back Propagation. The example shows that each mixer can select dimensions arbitrarily and the mixing function can have different capacities. The mixing is performed such that there is a path from any input to any output dimension for efficient mixing.

Inspired by the widely used sparse signal mixing method of Fast Fourier Transform (FFT), which uses butterfly structure and block-wise processing, we apply butterfly structure to multiple architectures such as linear transform, MLPs and Transformers. Since such mixing takes the benefit of efficiency and scalability from FFT, such signal mixing has $N \log_a(N)$ complexity, where N represents the number of dimensions and a represents mixing block size(or Radix-a). Although butterfly structures have already been used in Linear Transformation replacing dense Matrix Multiplication with previous works, we extend this structure to the non-linear domain and structured mixing of different dimensions of the input tensor. We extend the butterfly structure to MLP by using MLP as a mixing function. Furthermore, we use Butterfly Attention for efficient attention among tokens. Experiments show that sparse BlockMLPs can produce a better result than dense ones on the CIFAR-10 dataset as compared to dense counterparts. Furthermore, we find that sparse Butterfly Attention is sufficient for CIFAR-10 and CIFAR-100 datasets, produces faster results and scales well for large sequence lengths as well.

MLP-Mixer algorithm uses patch and channel mixing strategies to mix signals between inputs. We use a Patch of different sizes to mix the input signal which we call Patch-Only MLP-Mixer. Our method lies between the original MLP-Mixer and CNN, which helps to unify the working mechanism of both architectures. Furthermore, our method produces similar accuracy to the original MLP-Mixer when experimented on the CIFAR datasets.

We believe that semi-structured processing using the Dimension Mixer Model by using a sparse and unstructured selection of dimensions and processing of selected dimensions can be used to distribute neural networks across devices. It allows model parallelism not only in a layer-wise manner but also breaks a single layer into parallelizable layers of different sizes.

2 Background and Related Works

2.1 Structured Processing in Deep Learning

The success of deep learning can be partly credited to the structured processing units used by various deep architectures to process structured data. Structured data can be processed by the same processing unit and are highly parallelizable benefiting from accelerated computing like GPUs and TPUs. If we consider only Vision Architectures that perform image classification, we can analyse the working mechanism of these architectures. CNN(32) was developed quite early for processing image signals. It was inspired by the visual system of the brain (18). CNN generally have sliding window filters which apply a linear transformation to patches of the image. If we look from the perspective of signal mixing, the CNN architecture generally works by increasing the receptive field of filters as the number of layers increases. This allows the later layers to attend to a larger region of input image although indirectly through previous kernels. This has been the only architecture for vision till Vision Transformers (ViT) were introduced (15). ViT generally work with non-overlapping patches and the signals of the patches are mixed immediately by the attention layer without having to wait for later layers to comprehend the input signals as a whole and the signals are processed per patch/token by the MLP layer. However, Vision Transformers in general lack a sliding window, thus lacking the shift equivariance inductive bias of CNNs. Furthermore, the Attention Layer was replaced by using Channel Mixing Layer in MLP-Mixer (53) and shows that it also performs well.

Variations in Transformers/ViT Since transformers perform poorly against large sequence lengths, images cannot be divided into large numbers of patches. This problem is tackled by methods like Swin Transformer (38) RegionViT(8) by using convolutional priors and hierarchical attention. Most of the variation of efficient Attention Mechanism (51) focuses on creating sparse attention pattern such that complexity is reduced (54; 23; 6; 7; 2; 10; 12; 50; 56) while some focus on making the MLP block sparse (17).

Variations in MLP-Mixer The variation in architecture is also abundant in MLP-Mixer literature(36; 49; 9; 57; 21; 60; 58; 26). Different architectures aim for different ways of processing the input signals. Some use gating-based mixing (35) while others use shift in channels (34) for better mixing of signals. The majority of methods involve mixing the patch and channel in different ways. This abundance in the method to mix/process the image signal motivates us to design a Patch-Only MLP-Mixer which only mixes patches-wise for signal processing similar to CNN.

2.2 Partial Signal Processing

Partial Processing of Input Dimensions is observed in different architectures in deep learning and for varying reasons. ShuffleNet (59) uses group convolution over the channel dimensions which allows for efficient convolution due to the reduced number of channels. It also uses shuffling/mixing of the channels for the even distribution of output signals to the next layer of convolution. Furthermore, one of the earliest successful architectures AlexNet(31) uses parallel grouped convolution for accelerating on two GPUs and combining the hidden states in some layers to process them jointly. We can also see the parallel processing in Megatron-LM(44) which splits the input tokens into two blocks, processed independently and again combined at the end of the Attention and MLP blocks. Such split, process and combine method allows for processing a large number of tokens in a parallel manner so the model scales well.

Coupling Flows (14; 30; 29; 25) and Reversible ResNet (20), use split and process mechanism that allows for invertibility. Previous work (52) shows that they can represent any function based on partial mixing of signals. Furthermore, such partial processing makes computing the Jacobian efficient as well. The key takeaway with these architectures is that partial signal processing is sufficient for function approximation and provides benefits of efficiency and scalability. These architectures serve as the main motivation for the partial signal processing mechanism of the Dimension Mixer Model.

2.3 Sparse Linear and Non-Linear Models

The main problem with matrix multiplication is that it does not scale well with an increase in dimensions. The complexity of matrix multiplication for a vector input is known to be N^2 where N is the dimension of the input. This problem has been tackled to some extent by using Fast Matrix Multiplications (16; 47) as well as Low-Rank Matrix Decomposition. The low-rank transformation

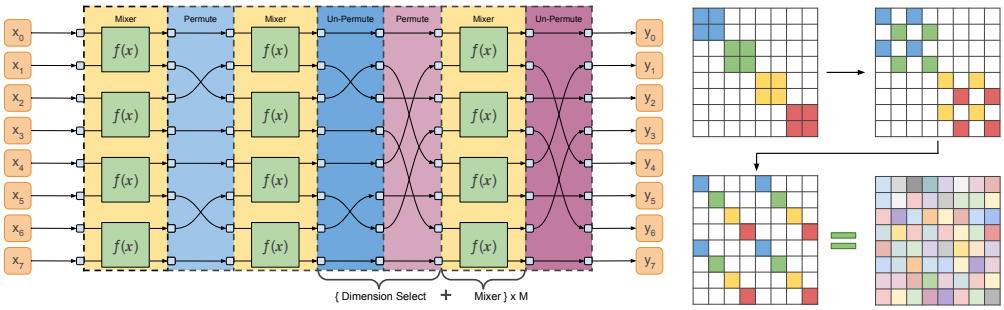


Figure 2: An example of FFT style Butterfly Non-Linear Mixer. This example shows the mixing of an 8-dimensional input signal using Radix-2 Butterfly. The first layer selects the dimension as it is. However, later layers use Permute to bring different dimensions in a block and later perform un-permute to place the dimension in their original location. For Radix-4 Butterfly a mixer block takes 4 dimensions as input and permutes accordingly.

has been widely used in efficient CNN architectures (41; 37). Models like EffecientNet-v2(48) and MobileNets (27) use depth-wise and point-wise convolution as a low-rank factorization of a standard convolution.

We can get a highly sparse matrix using pruning-based techniques (22). However, these methods accelerate neural networks on CPUs and mobile devices but fail to accelerate significantly on GPUs. Furthermore, block sparse matrices (19) has been used widely to accelerate matrix multiplication. Another alternative is to use butterfly matrices (42) which is inspired by the Fast Fourier Transform (FFT). These matrices scale well with dimensions as well. Previous works (42; 5; 13) have successfully used linear butterfly transformation as a replacement for dense transformation and have produced highly efficient architectures.

3 Dimension Mixer Models

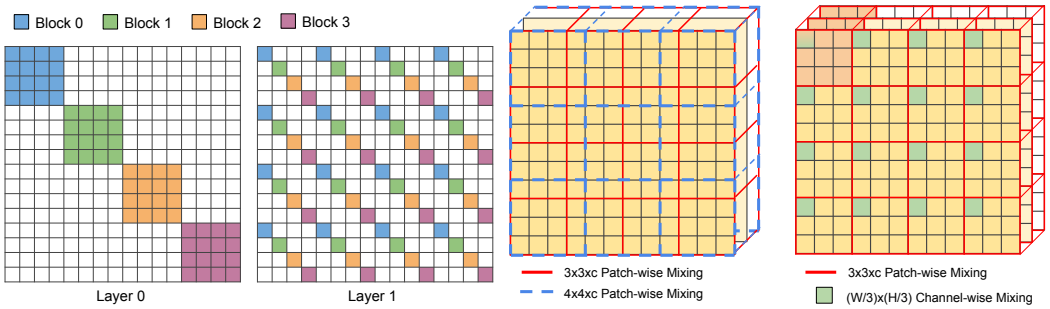
Dimension Mixer Model is a simple and general structure observed among most deep learning architectures. The key observation is that of structured signal processing and performing mixing of all input signals efficiently. The efficiency is gained due to partial (or group-wise) mixing of input signals. For structured data, the mixing is itself structured similarly to the data and mixing functions like MLPs or filters can be shared.

Geometric Deep Learning (4; 3) tackles the generality of these architectures by generalizing to graphs as an underlying data structure, however, we generalize the signal mixing/processing aspects of different architecture. Dimension Mixer Model allows us to develop structured processing for unstructured data as well. Figure 1 shows a simple example of a general Dimension Mixer Model using two major operations (i) Select a group of dimensions and (ii) Process or Mix the selection. This operation is carried out in parallel as well as sequentially for efficient mixing of signals. The signal mixing can be evaluated with Graph Theory for the effective flow of signals.

The structure of Dimension Mixer Model be arbitrary in terms of input grouping/selection and mixing/processing function, as long as the input signals are efficiently mixed. The butterfly structure of the FFT is highly efficient, especially for GPUs. Hence, we focus on using the Butterfly Structure for its immediate utility to accelerate current Neural Network Architectures.

3.1 Non-Linear Butterfly Mixer

Although butterfly structure has been used widely in Linear Models (42; 13; 5), our theory of Dimension Mixer Model requires non-linear mixing for effective mixing of input signals. We propose a non-linear butterfly structure model as shown in Figure 2. The linear butterfly transformation is a special case with linear $f(x)$. Furthermore, we may use any learnable function as a mixing function. This allows our model to use a function such as MLP but other methods such as learnable bi-linear interpolation for N-Dimensions, soft decision trees or even Radial Basis Function (RBF). To this end, we propose two Non-Linear Mixers called MLP Butterfly Mixer and 2D-Grid Bilinear Butterfly



(a) Butterfly Attention (b) MLP-Mixer Comparison

Figure 3: **(a)** An example of Butterfly Attention pattern on sequence length of 16 with butterfly structure of Radix-4. Using Radix-$\sqrt{N}$ creates two sparse attention matrix for complete mixing of signals. **(b)** *(left)*Patch Only MLP mixer (ours) compared to *(right)* Patch-Wise and Channel-Wise Mixing for 12x12 image size. The Channel-wise Mixing is replaced by Different size Patch-wise Mixing by our method.

Mixer. The mixing algorithm with permutation for a given radix of butterfly mixer is shown by Algorithm 1.

MLP Butterfly Mixer MLP Butterfly Mixer uses MLP as a mixing function. We implement a column of MLPs in parallel which helps utilize the acceleration of GPUs. This simply breaks a whole MLP into blocks of MLPs in a butterfly structure for efficient mixing. Thus produced architecture is more parameter efficient as well as allows having arbitrary MLP design (in terms of depth, width and activation) as a mixing function. Our method is more flexible than using Butterfly Linear as a replacement for a dense layer. If we take the Radix-N butterfly for N-dimensional input, it becomes a single mixing using the dense method. Hence, MLP butterfly Mixer Generalizes to standard MLP. However Radix-M; (M<N) butterfly structure shards MLP into multiple smaller MLPs which is more parameter efficient, as well as highly parallelizable.

2D Grid Bilinear Butterfly Mixer Taking the advantage of Mixing using the Radix-2 Butterfly Structure, we use the 2D grid for approximating the mixing function. We use an evenly spaced grid and learn the Y values. We can simply use the input to index the neighbourhood points and apply bi-linear interpolation. This model also allows us to use 2D grid-based activation function on MLPs. We implement this structure on the PyTorch framework, however, due to inefficiency, we implement the *cuda kernel* which is highly efficient and scales well with grid size. This method consumes high memory if the input dimension is high or if the grid size is large. Hence, this algorithm may better benefit from a CPU rather than expensive memory of GPUs. Our implementation has more areas to improve upon, however, the current implementation shows that it can work successfully.

3.2 Butterfly Attention Mixer

We also apply the efficient mixing of butterfly structures in the Attention mechanism of Transformers architecture (different from Pixelated Butterfly (5)). It is widely known that the complexity of

Algorithm 1: Permutation of Non-Linear Butterfly Mixers - PyTorch like pseudocode

```
# x : Input with shape [ BATCH SIZE, DIMENSION].
# function_layers : layers of block mixing function.
# fn: block mixing function; is parallel non-linear mixer operating per block.
# block_dim: radix or input size of a mixing function.
y = x
for i, fn in enumerate(function_layers):
    y = y.view(-1, block_dim, block_dim**i).transpose(2, 1).view(batch_size, -1)
    y = fn(y)
    y = y.view(-1, block_dim**i, block_dim).transpose(2, 1).view(batch_size, -1)
# y : Output with same shape as Input x
```

N	Linear Butterfly					Low Rank Factorization	
	2	$\sqrt{N}/2$	$\sqrt{N}$	Parallel	Sequential	Same Param	N/4
16	1.71x10^{-1} 128 0.27	- -	1.36x10^{-1} 128 0.12	1.24x10^{-7} 512 1.15	7.8x10^{-4} 512 1.13	1.23x10^{-1} 128 0.06	3.33x10^{-2} 256 0.048
64	0.260479 768 0.3966	0.240193 768 0.1764	0.212060 1024 0.1177	0.021295 4608 2.4778	0.038296 4608 2.4494	0.208368 1024 0.0557	0.035452 4096 0.0491
256	0.304516 4096 0.5198	0.278291 6144 0.1765	0.264758 8192 0.1194	0.136453 32768 5.2029	0.141043 32768 4.2688	0.263254 8192 0.0675	0.035383 65536 0.0604
1024	0.323299 20.5K 0.6258	0.303273 49.2K 0.1792	0.296384 65.5K 0.3635	0.238191 204.8K 41.17	0.238211 204.8K 23.14	0.296054 65.5K 0.0624	0.035430 1.048M 0.0603
4096	0.33 98.3K 20.55	0.3174 393.2K 0.1837	0.3139 524.3K 4.9312	- -	- -	0.3139 524.3K 0.0691	0.0354 16.77M 0.3235

Table 1: Sparse approximations for dense matrix. The data contain MSE / Parameters / Time Taken (in milliseconds) for different Matrix Size (NxN). Linear Butterfly architecture uses various radix values. Similarly, Low-Rank Decomposition uses various rank values. The Same Param column contains low-rank factorization with the same parameter as $\sqrt{N}$ Linear Butterfly.

Attention is N^2 with sequence length N. Furthermore, there is a large number of cases where the sequence length can blow up significantly such as on large images when using small patch size, on long documents and on audio/video signals. To solve this issue, we apply partial block-wise attention using the butterfly structure , which reduces the complexity of Attention to just N, however it would require $\log_a(N)$ layers of Attention for complete mixing of tokens where a is Radix. For experiments, we use Radix-$\sqrt{N}$ for mixing which creates complete mixing within two attention blocks *(see Figure 3a)*. Despite having partial attention, experiments show that it performs better than full attention when used on CIFAR-10 and CIFAR-100 datasets. Furthermore, the experiments show that the butterfly attention mechanism is faster for training and inference as well as memory efficient.

3.3 Patch Only MLP-Mixer for Vision

In the MLP-Mixer architecture, the mixing between two or more patches is done by token mixing (or channel-wise mixing) MLP. However, in CNN architecture, the only use of patch-wise mixing is sufficient. This is due to the sliding window used in convolution which helps to mix the signals of different regions along with an increase in the receptive field with an increase in depth. We search for a mechanism that allows for mixing using patch-only but without the sliding window.

We propose Patch-only MLP Mixer (Figure 3b. The architecture consists of Image of size I and Kernels/Patches of size K_1, K_2, K_3, $\cdots$ such that $I = K_1 \times K_2 \times K_3 \cdots$ such that K_i and K_j are not factors of each other for different i and j. If patches are factors of each other then the mixing occurs in different partitions without complete mixing. We choose only 2 factors for patch size for the simplicity of experiments. Patch Only MLP-Mixer lies in between MLP-Mixer and Convolution as it uses MLP for processing patches but only uses patches for overall input mixing.

3.4 Distributed Dimension Mixer Model

If we are to create distributed deep architectures, we might need sparse processing of different sizes. Our semi-structured Dimension Mixer Model is useful in cases where we need varying block size and varying input size processing. Furthermore, training such models takes a large number of iterations and is lagged by network bandwidth. However, methods involving target propagation (33; 40) and local optimization (1) might help reduce the effects. Furthermore, there might be the problem with drop in compute nodes or blocks during the training and inference. To alleviate this we need dynamically adjustable, noisy (43) and robust neural networks for adapting to the fluctuations in computation devices. We may simply apply robust training such as dropout (46) per mixer nodes.

4 Experiments

Approximation capacity of Linear Butterfly Transforms We compare the dense approximation capacity of sparse linear transforms, i.e. low-rank factorization and Butterfly Linear Mixer. The experiments use these algorithms for approximating some random uniform (-1, 1) dense matrix. For comparison, we test on different-sized square matrices and using different block sizes (Radix-M) butterfly linear mixers with different ranks of Low-Rank Factorization.

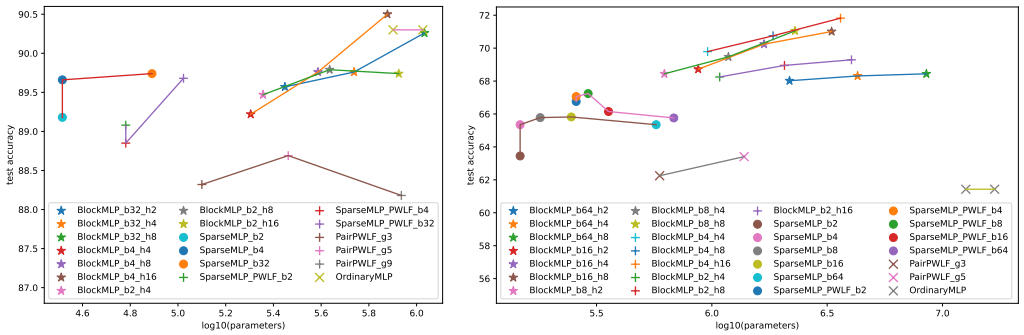


Figure 4: Parameters vs Accuracy plot for Different Sparse Models (left) FMNIST (right) CIFAR-10. BlockMLP represents MLP Butterfly Mixer with b{Block Size} h{Hidden Expansion}. SparseMLP represents Linear Butterfly Mixer-based MLP with b{Block Size}. SparseMLP_PWLF represents Linear Butterfly Mixer-based MLP but with 2D grid activation function with grid-size 5. Similarly, PairPWLF represents 2D-Grid Bi-linear Butterfly Mixer with g{Grid Size}. OrdinaryMLP represents dense MLP. Since input expansion does not benefit the dense matrix, the reduced parameter OrdinaryMLP represents the effective parameter size.

Table 1 shows that butterfly linear produces a sparse approximation of dense layer with similar performance to low-rank factorization while also being full rank. The mean squared error of the same parameter of both Butterfly Linear and Low-Rank Factorization is similar. Furthermore, we also test the approximation by the sum of the parallel block linear model as well as the sequential block linear model. We find that further mixing using parallel or sequential mixing improves the approximation capacity. We also observe that butterfly linear models are slower for low-rank models as the number of layers increases and are executed by different GPU kernels. However, low-rank matrix multiplication is carried out by two kernels only and has faster execution. This might be solved by using fused kernels or by hardware implementations and is not the limitation of the transformation itself.

Non Linear Butterfly Mixers at Action We compare the use of Non-Linear Butterfly Mixer Models as a simple replacement of 2 Layer MLP for FMNIST and CIFAR-10 datasets. The experiment configuration for MLP Butterfly Mixer consists of various hidden expansion and butterfly radix sizes. We test for butterfly linear mixer using different radix sizes and test the performance of 2D Grid Bi-linear Butterfly Mixer. We also test the performance of the Linear Butterfly Mixer with the 2D grid as an activation function. Since the butterfly structure works on input dimensions power of 2, we expand the remaining dimensions to the closest value by randomly repeating some dimensions, i.e 784$\rightarrow$1024 for FMNIST and 3072$\rightarrow$4096 for CIFAR-10.

Experiments show that MLP Butterfly Mixer mixes the function better compared to the Dense model as it mixes signal in a non-linear fashion in multiple layers of blocks. Furthermore, MLP Butterfly Mixer can produce a spectrum of models in terms of block size, hidden units as well as the number of layers in individual blocks of MLP. The Butterfly Linear Model is also highly sparse and mixes signals efficiently.

For the experiments, we train a model with some mixing backbone and linear classification. Mixing backbone can be Linear Butterfly based MLP, MLP Butterfly, 2D-Grid Bilinear Butterfly or dense MLP as shown in the Figure 4. We set the learning rate of 0.001 with Cosine Decay. We train for 200 epochs for cifar-10 and 50 epochs for FMNIST. Furthermore, due to the instability of training 2D grid bilinear butterfly mixer, we experiment with a 0.00003 learning rate.

Butterfly Attention Mixer We test the capacity of Butterfly Attention as compared to dense Attention proposed in the original paper. We experiment on CIFAR-10 and CIFAR-100 datasets. We experiment on different patch sizes and a different number of layers without using Positional Embeddings. In Table 2 we show the accuracy, training time, memory usage and parameters of the vision transformers. The time taken and the memory usage is given by average values of 50 steps of training for a batch size of 32. The experiments are designed to test the resource consumption and accuracy of both sparse and dense attention models. Experiments show that butterfly attention

Dataset	CIFAR-10								CIFAR-100			
Patch Size	4				2		1		4		2	
Layers	4		8		4		4		4		4	
B-Attention	F	T	F	T	F	T	F	T	F	T	F	T
Accuracy (Rand Token)	80.84	83.33 81.63	81.13	83.67 81.57	77.21	78.95 76.60	64.80	72.82 70.88	53.20	56.70 53.63	49.64	52.86 48.30
Time	11.66	11.52	20.81	21.67	12.66	11.55	125.54	25.50	10.48	11.43	12.91	10.87
Memory(MiB)	457	441	537	505	911	519	8629	1181	478	462	976	560
Params(M)	0.618		1.148		0.299		0.79		1.355		1.773	

Table 2: An experiment comparing Butterfly Attention with Dense Attention. The B-Attention column represents if the Butterfly Attention is used. The Sequence Length for patch size 4, 2 and 1 are 64, 256 and 1024 respectively. Rand Token represents accuracy when tokens are randomized.

Architecture	Layers	CIFAR-10	CIFAR-100
MLP Mixer (c1)	7	83.81 (0.897M)	57.17 (1.95M)
	10	83.03 (1.23M)	56.34 (2.28M)
MLP Mixer (c2)	7	84.16 (0.88M)	58.38 (1.77M)
	10	84.20 (1.22M)	57.81 (2.1M)
Patch Only	7	84.66 (0.81M)	55.55 (1.14M)
	10	85.49 (1.14M)	56.29 (1.47M)

Table 3: Vision MLP Mixers Comparision. The results are structured as: Accuracy (Parameters)

scales better with longer sequence lengths, and the accuracy is comparable to dense attention. Our method produces a faster, less memory consuming and better-performing model as compared to dense attention. The experiments are carried out for 300 epochs and 128 batch size with cosine decay of learning rate. For patch 1 training, we use 64 batch size to reduce memory consumption. We use hidden channels of 128 dimensions for patch 4, and 64 dimensions for patch 2 and patch 1 models. We use a total of 8 Attention heads on all the models.

Vision MLP-Mixers We compare our Patch Only MLP mixer with the original MLP-Mixer architecture with a similar number of layers and a similar number of parameters. The experiments show that our method produces comparative or even better results than the original MLP mixer on CIFAR-10 and CIFAR-100 datasets (32x32 images). For MLP Mixer config-1 *(c1)* we scale the input image to 36x36 size, expand channels by a factor of 3 and apply MLP-Mixer on a patch size of 5. Furthermore, for config-2 *(c2)* we only expand channels by a factor of 3.20 and use a patch size of 4. On Patch Only MLP Mixer, we expand the image to 35 = 5x7 size and mix over the patch of sizes 5 and 7 with no channel expansion. MLP-Mixer produces large hidden image size and our method produces smaller hidden image size as well as has a smaller classifier layer. We compare two methods with a similar number of parameters in the mixing block. The results in Table 3 show that our method performs competitively or even better than the original method.

5 Conclusion

In this paper, we introduce a general theory of efficient signal processing model called Dimension Mixer Model which we apply for sparsifying multiple architectures such as MultiLayered Perceptron(MLP) and Attention Layers of Transformer Architecture. Furthermore, we introduced Patch Only MLP mixer as intermediate architecture between the original MLP-Mixer and Convolutional Neural Network. We propose to use the sparse processing structure of the Dimension Mixer Model for distributed neural network architectures. All the proposed models are the application of the Dimension Mixer Model which we find is insightful for analysing the signal processing on Deep Learning architectures and also for designing newer models.

Limitations and Future Work A major limitation of our work is in our experiments. It is limited to a small number of small-scale image datasets which are sufficient for conveying the idea, however, experiments on larger-scale datasets would make the results strong. We aim to apply a non-linear butterfly structure for convolutional neural networks as well as test distributed dimension mixer models in real-world settings. We also take finetuning pre-trained transformers using butterfly attention as our future work.

References

[1] Ehsan Amid, Rohan Anil, and Manfred Warmuth. Locoprop: Enhancing backprop via local loss optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 9626–9642. PMLR, 2022.

[2] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[3] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[4] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[5] Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*, 2021.

[6] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426, 2021.

[7] Bin Chen, Ran Wang, Di Ming, and Xin Feng. Vit-p: Rethinking data-efficient vision transformers from locality. *arXiv preprint arXiv:2203.02358*, 2022.

[8] Chun-Fu Chen, Rameswar Panda, and Quanfu Fan. Regionvit: Regional-to-local attention for vision transformers. *arXiv preprint arXiv:2106.02689*, 2021.

[9] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv preprint arXiv:2107.10224*, 2021.

[10] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[11] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[12] Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*, 2019.

[13] Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR, 2022.

[14] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[16] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[17] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[18] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[19] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

[20] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

[21] Jianyuan Guo, Yehui Tang, Kai Han, Xinghao Chen, Han Wu, Chao Xu, Chang Xu, and Yunhe Wang. Hire-mlp: Vision mlp via hierarchical rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 826–836, 2022.

[22] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[23] Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. Neighborhood attention transformer. *arXiv preprint arXiv:2204.07143*, 2022.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pages 2722–2730. PMLR, 2019.

[26] Qibin Hou, Zihang Jiang, Li Yuan, Ming-Ming Cheng, Shuicheng Yan, and Jiashi Feng. Vision permutator: A permutable mlp-like architecture for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[27] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
[28] Yerlan Idelbayev and Miguel A Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020.
[29] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.
[30] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.
[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
[32] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.
[33] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.
[34] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. As-mlp: An axial shifted mlp architecture for vision. *arXiv preprint arXiv:2107.08391*, 2021.
[35] Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34:9204–9215, 2021.
[36] Ruiyang Liu, Yinghui Li, Linmi Tao, Dun Liang, and Hai-Tao Zheng. Are we ready for a new paradigm shift? a survey on visual deep mlp. *Patterns*, 3(7):100520, 2022.
[37] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022.
[38] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.
[39] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.
[40] Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F Grewe. A theoretical framework for target propagation. *Advances in Neural Information Processing Systems*, 33:20024–20036, 2020.
[41] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.
[42] Anish Prabhu, Ali Farhadi, Mohammad Rastegari, et al. Butterfly transform: An efficient fft based neural architecture design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12024–12033, 2020.
[43] Suman Sapkota and Binod Bhattarai. Noisy heuristics nas: A network morphism based neural architecture search using heuristics. *arXiv preprint arXiv:2207.04467*, 2022.
[44] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.
[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
[47] Volker Strassen et al. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.
[48] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021.
[49] Yehui Tang, Kai Han, Jianyuan Guo, Chang Xu, Yanxi Li, Chao Xu, and Yunhe Wang. An image patch is a wave: Phase-aware vision mlp. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10935–10944, 2022.
[50] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning*, pages 10183–10192. PMLR, 2021.
[51] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, 2020.
[52] Takeshi Teshima, Isao Ishikawa, Koichi Tojo, Kenta Oono, Masahiro Ikeda, and Masashi Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. *Advances in Neural Information Processing Systems*, 33:3362–3373, 2020.
[53] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture

for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021.

[54] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. *arXiv preprint arXiv:2204.01697*, 2022.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[56] Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. Cluster-former: Clustering-based sparse transformer for long-range dependency encoding. *arXiv preprint arXiv:2009.06097*, 2020.

[57] Ziyu Wang, Wenhao Jiang, Yiming M Zhu, Li Yuan, Yibing Song, and Wei Liu. Dynamixer: a vision mlp architecture with dynamic mixing. In *International Conference on Machine Learning*, pages 22691–22701. PMLR, 2022.

[58] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S2-mlpv2: Improved spatial-shift mlp architecture for vision. *arXiv preprint arXiv:2108.01072*, 2021.

[59] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[60] Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Mixing and shifting: Exploiting global and local dependencies in vision mlps. *arXiv preprint arXiv:2202.06510*, 2022.