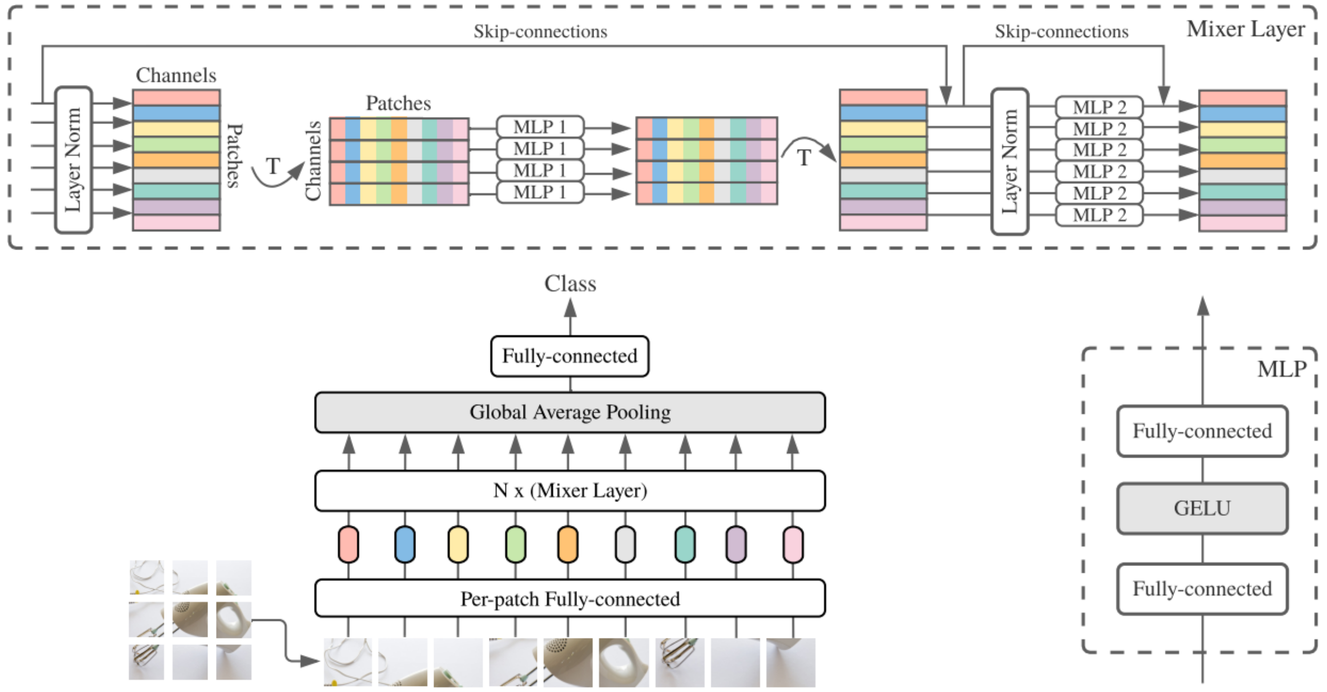
FPGA Accelerator for MLP-Mixer Model



Stefan Abi-Karam

SHARC Lab @ Georgia Tech

Deep Learning for Vision

- Classification
- Detection (Classification + Localization)
- Segmentation
- Style Transfer
- Colorization
- Reconstruction
- Super-Resolution
- Synthesis
- Forecasting
- Compression
- Rendering

We have many different architectures for all the vison tasks

Each architecture has a specific component responsible for learning spatial features

1. Traditional Feature Extraction
2. Convolutions
3. Transformers / Attention

So what's next? Can we do better?

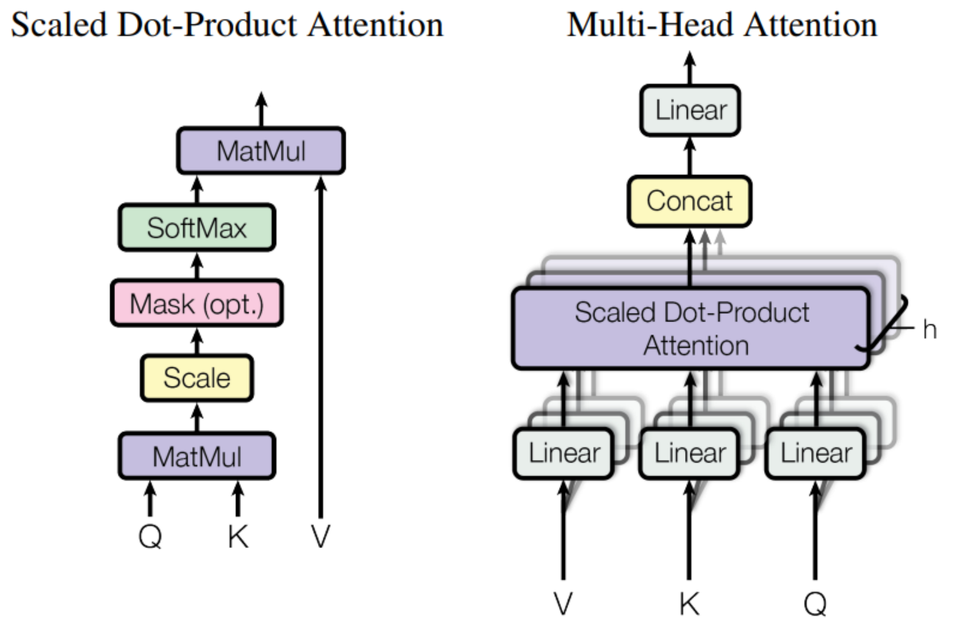
Deep Learning for Vision: Key Motivation

CNNs need more layers (more weights) to get better

Vision transformers can do the same with fewer weights but require a lot of time to train and a lot of data to get better

For CNNs, you need to have hardware to do convolutions fast

For ViTs, you need to have hardware to do multi-head attention fast, which means you need to do Matrix-Matrix multiply fast

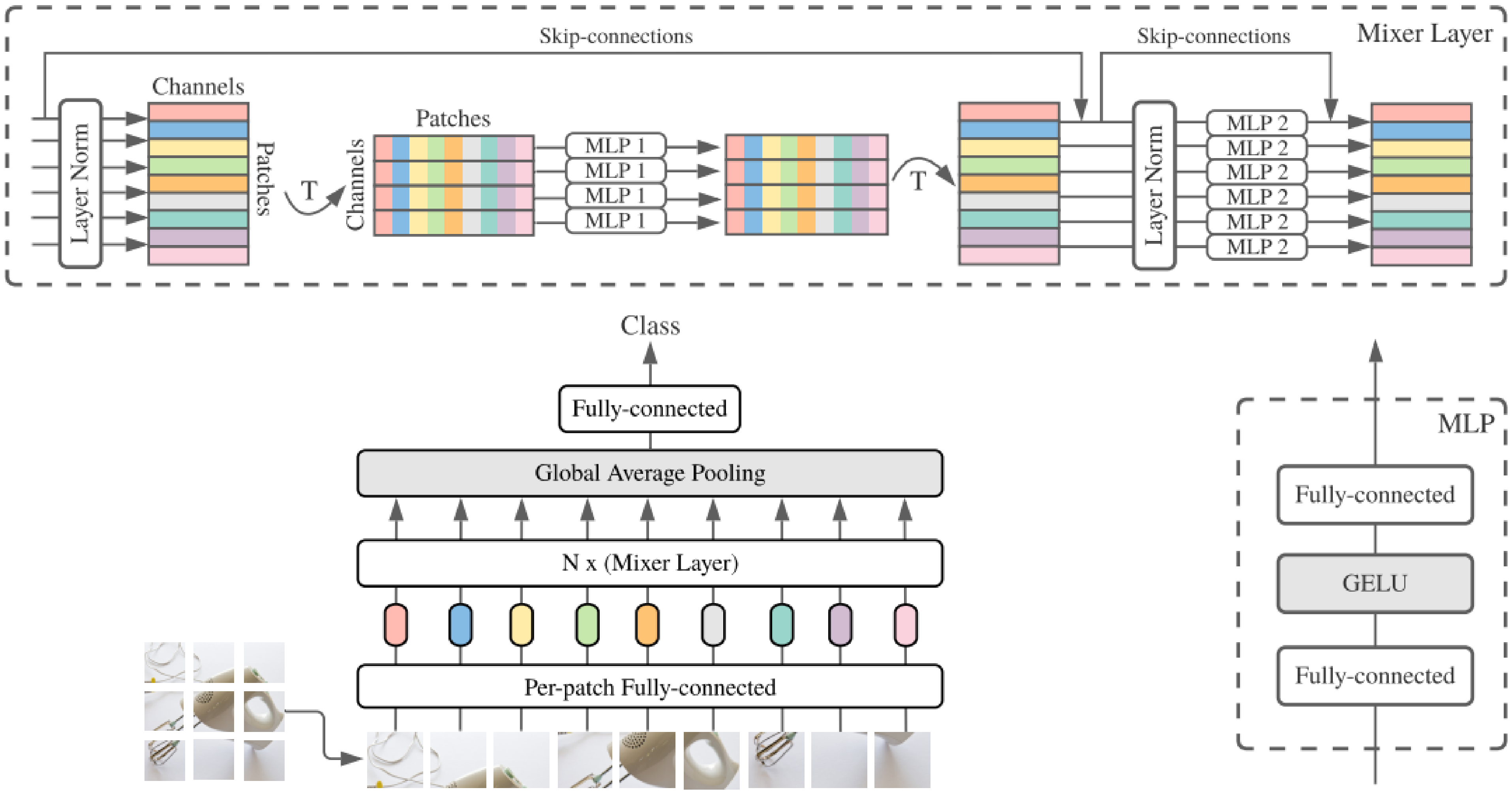


Deep Learning for Vision: Key Motivation

Other Opinions for Motivations:

- Sometimes its good to look at different / orthogonal approaches for learning
- Without this we would never discover new architectures and deep learning insights
- Sometimes its just fun to explore new ideas

MLP-Mixer


Skip-connections
Skip-connections
Mixer Layer
Channels
Layer Norm
Patches
T
Patches
Channels
MLP 1
MLP 1
MLP 1
MLP 1
T
Layer Norm
MLP 2
MLP 2
MLP 2
MLP 2
MLP 2
MLP 2
Class
Fully-connected
Global Average Pooling
N x (Mixer Layer)
Per-patch Fully-connected
MLP
Fully-connected
GELU
Fully-connected

MLP-Mixer: Patches and Patch Embeddings

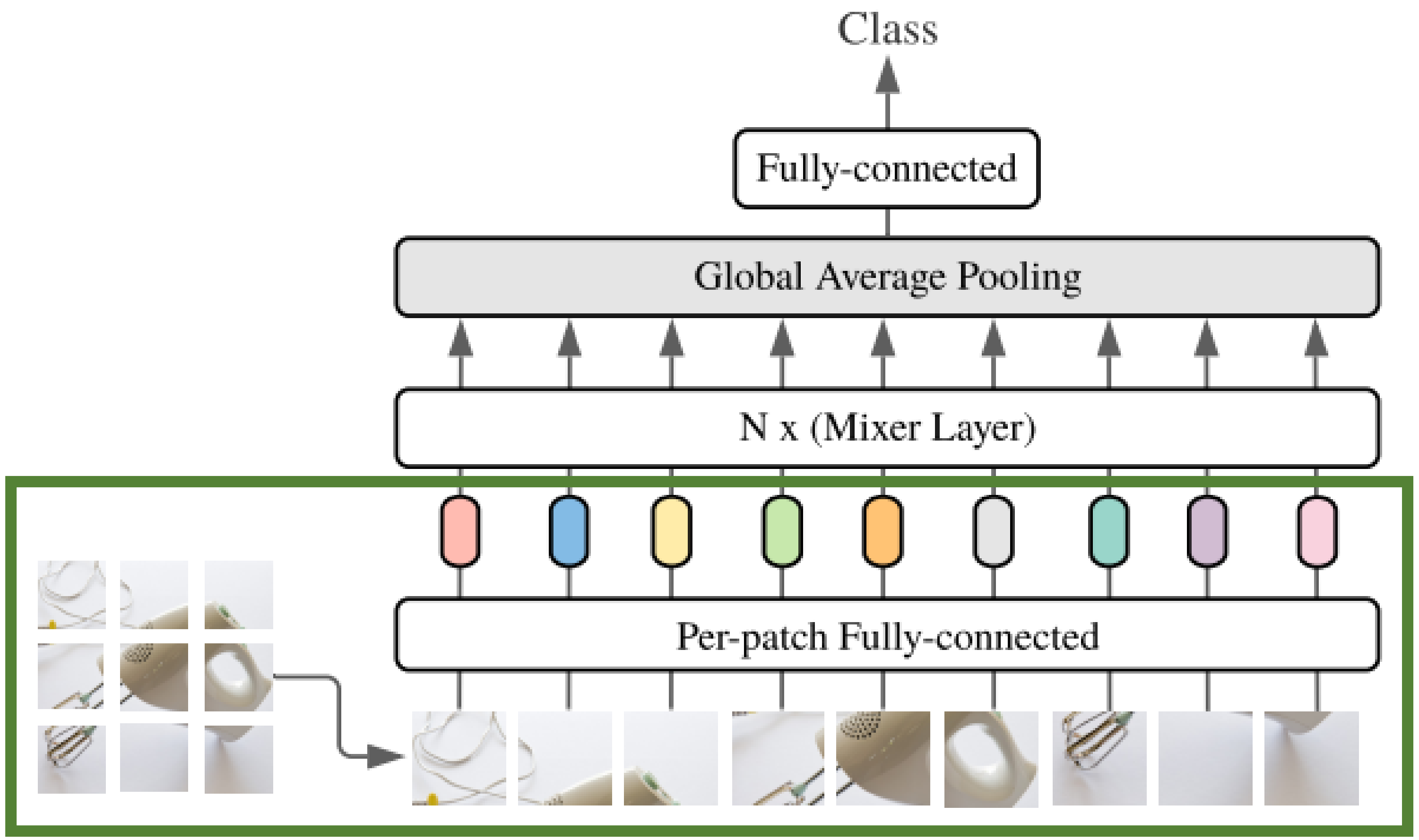
1. Split image into patches
2. Flatten each patch
3. Apply MLP embedded each patch into a vector representation

If image is multi-channel, just flatten each patch with the channels in the patch

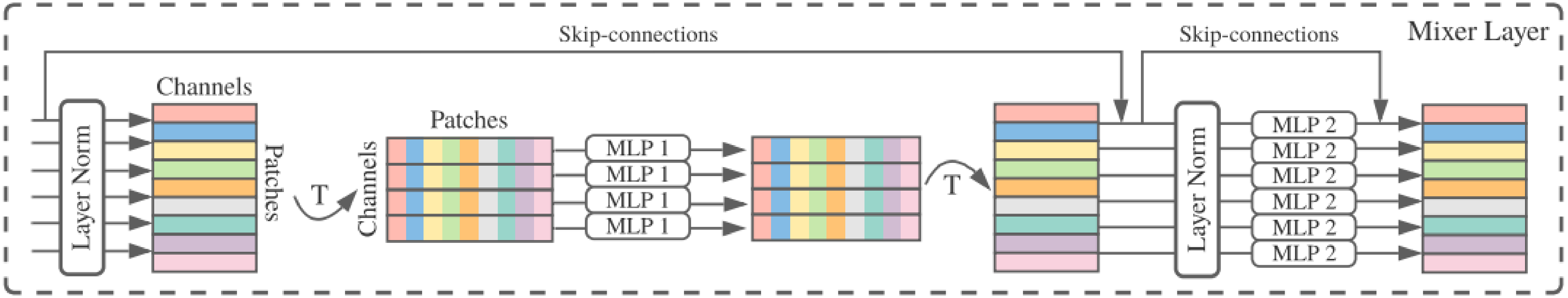
Ex. 16x16x1 -> flatten -> 256 -> MLP -> 128
Ex. 12x12x3 -> flatten -> 432 -> MLP -> 128

Uses the same MLP applied to all patches

If you are paying attention, this is a conv2d with the size=stride=patch_size and the #_out_channels=embedding_size, with a flatten in the resulting image across the height+width dimensions



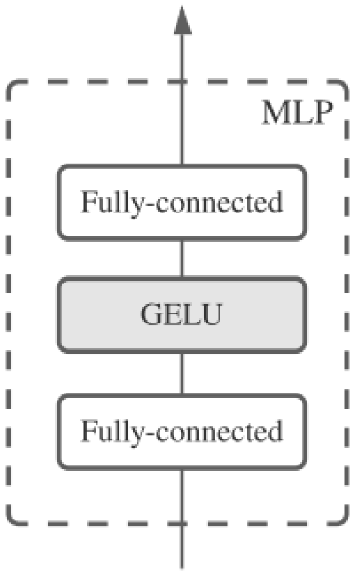
MLP-Mixer: Mixer Layers



1. Apply MLP to mix features across patches embeddings
   a. Transpose grouped embeddings
   b. Single vector now has same channel from all patches
   c. Apply shared MLP to learn relationships across all patches
   d. Transpose results to get back to original per patch embeddings

1. Apply MLP to mix features within patches
   a. Apply shared MLP to learn relationships within a single patches

Add skip connections and layer norm to help with learning

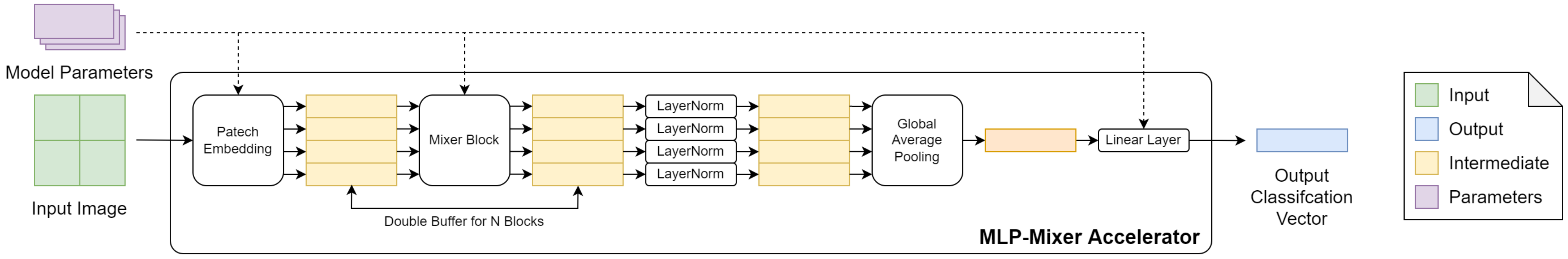


To be honest, I don't know why they use GELU activation

They probably tired a bunch of stuff, and this worked best

GELU is annoying to compute exactly, there are approximations for it

MLP-Mixer Accelerator

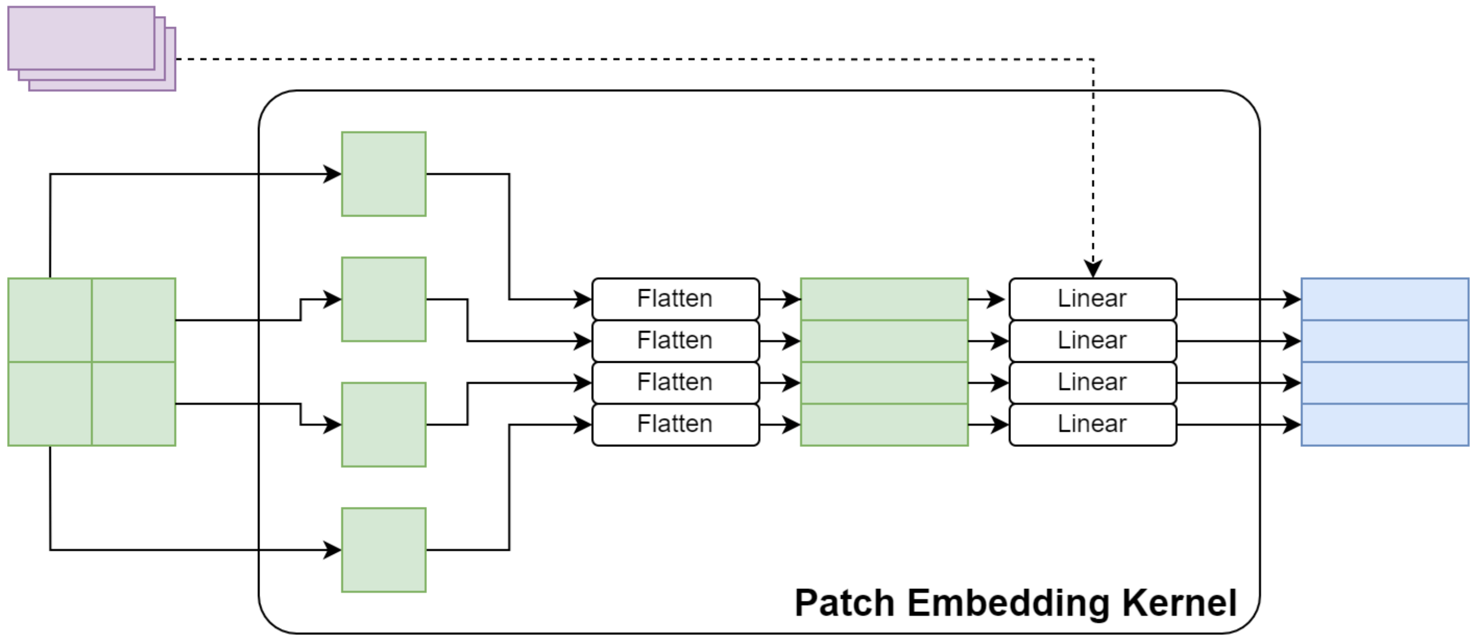


1. Load parameters if flag is passed in
2. Load input image
3. Compute patch embeddings
4. For N blocks
    1. Compute mixing block
5. Compute global average polling
6. Compute linear classification head

Note the use of the double buffer for the mixer block embeddings, this allows for 2 memories instead of N memories needed for mixer block computation

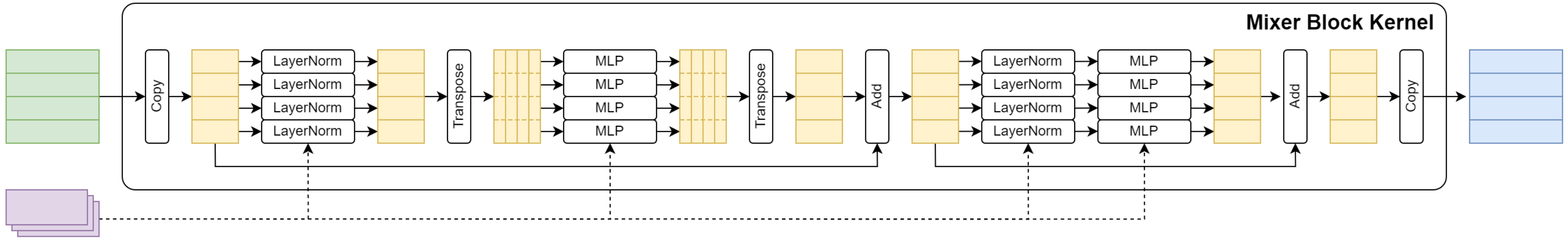
Design heavy leans on template functions for tweaking parallelization factors

MLP-Mixer Accelerator: Patch Embedding



The patch embedding kernel flattens patches from the input buffer using some indexing arithmetic and applies a linear transformation to embed each patch into the patch embedding array

MLP-Mixer Accelerator: Mixer Block



The whole mixer block kernel region has a dataflow pragma applied to optimize data movement such as with the skip connections.

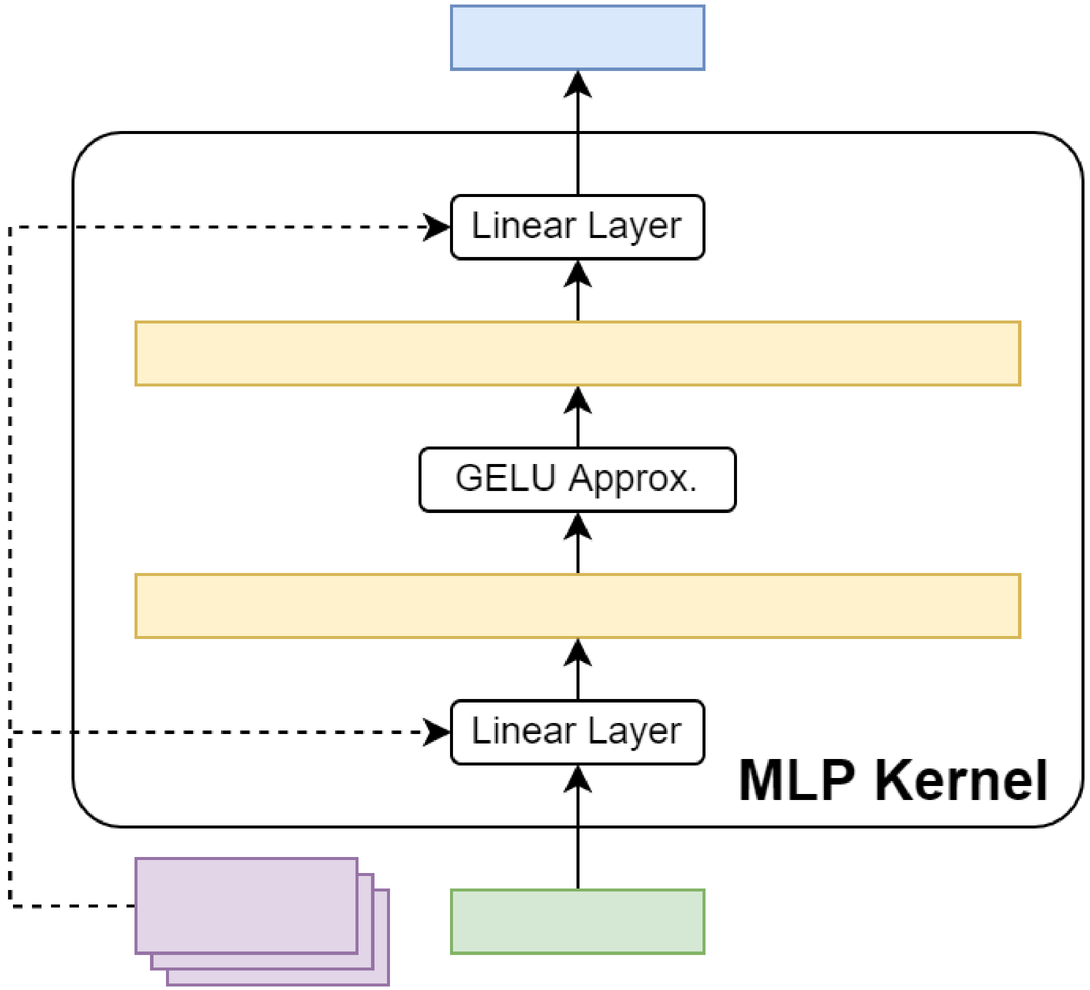
Implemented as a template functions to allow for different sizes of embeddings as well as different parallel factors for the layer norm and MLP kernels

MLP-Mixer Accelerator: MLP

The MLP kernel follows a 2-layer implementation from the paper

The hidden dimension expands the input into a bigger size and contacts back down to the input size

Linear layers are a block parallel implementation with two parallelization factors for the input and output block partition size



MLP-Mixer Accelerator: GELU

GELU has two approximations

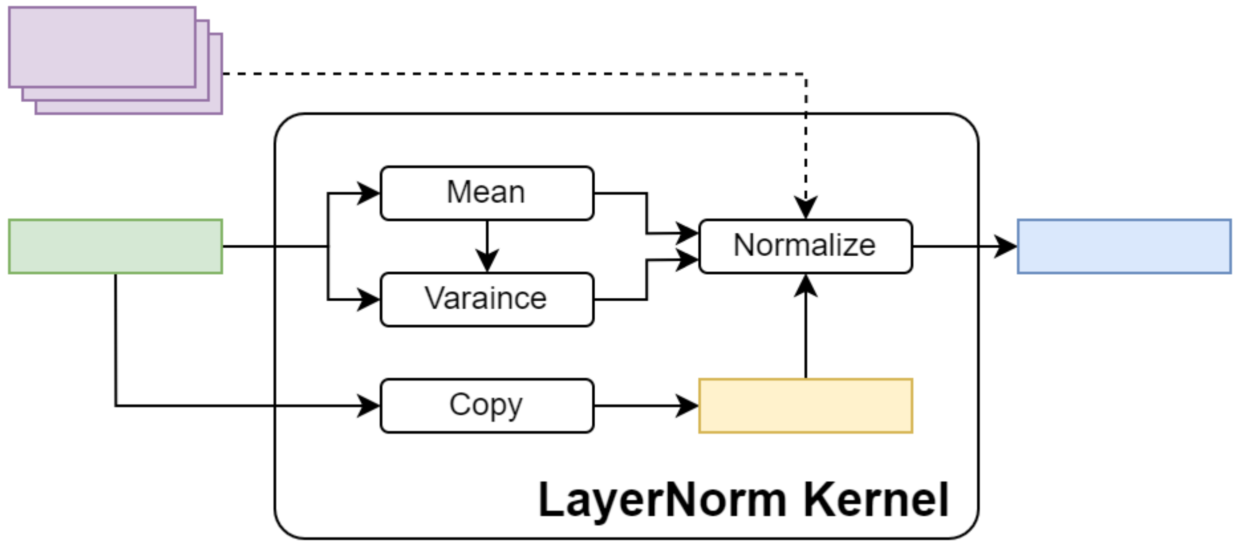
- Original: $\mathrm{GELU}(x) = x\frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$
- Approx. 1: $\mathrm{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right)$
- Approx. 2: $\mathrm{GELU}(x) = x\sigma(1.702x)$

I implemented the 2nd approximation using the sigmoid function in hardware since it requires the least number of DSPs with minimal accuracy loss

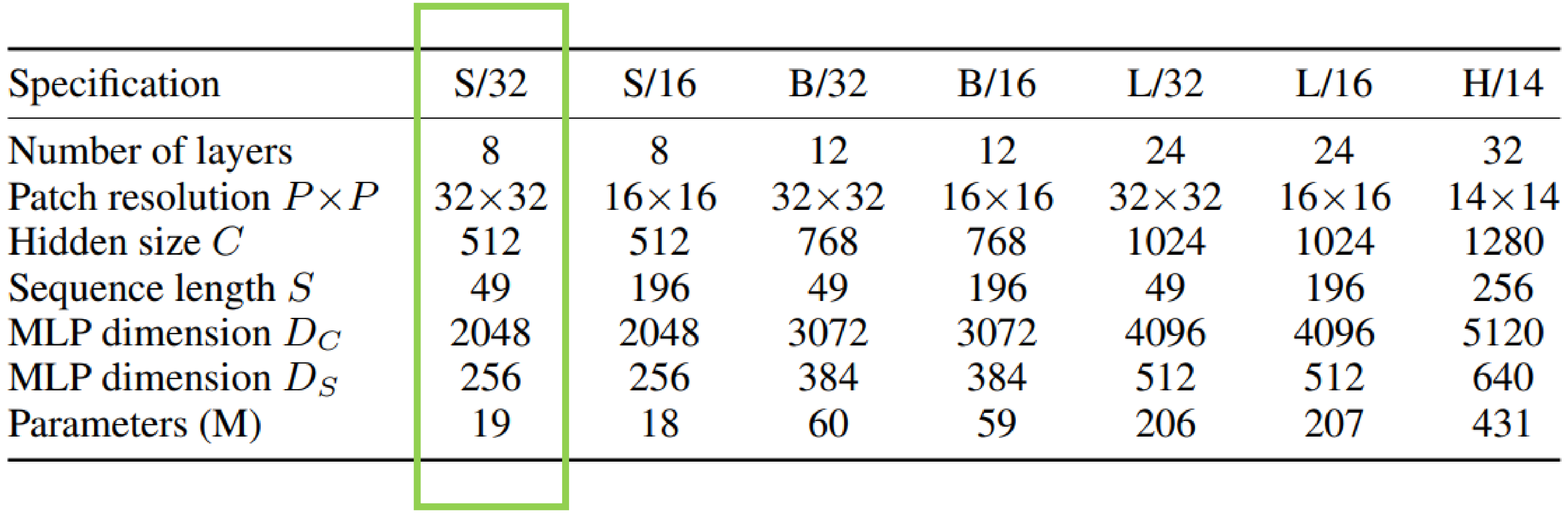
MLP-Mixer Accelerator: Layer Normalization

The layer normalization kernel follows a 3-pass calculation, one for the mean, one for the variance, and one for the normalization

Implemented as block parallel with a single parallelization factor for the vector block partition



MLP-Mixer Accelerator: Implementation



Specification	S/32	S/16	B/32	B/16	L/32	L/16	H/14
Number of layers	8	8	12	12	24	24	32
Patch resolution $P \times P$	32×32	16×16	32×32	16×16	32×32	16×16	14×14
Hidden size C	512	512	768	768	1024	1024	1280
Sequence length S	49	196	49	196	49	196	256
MLP dimension D_C	2048	2048	3072	3072	4096	4096	5120
MLP dimension D_S	256	256	384	384	512	512	640
Parameters (M)	19	18	60	59	206	207	431

All the model configuration parameters can be defined in the model header file

For this project I ran synthesis runs for the S/32 model

Uses Vitis HLS + Fixed Point / ap_fixed<32,16> + C++ template functions

MLP-Mixer Accelerator: Implementation

Parallelization Factors

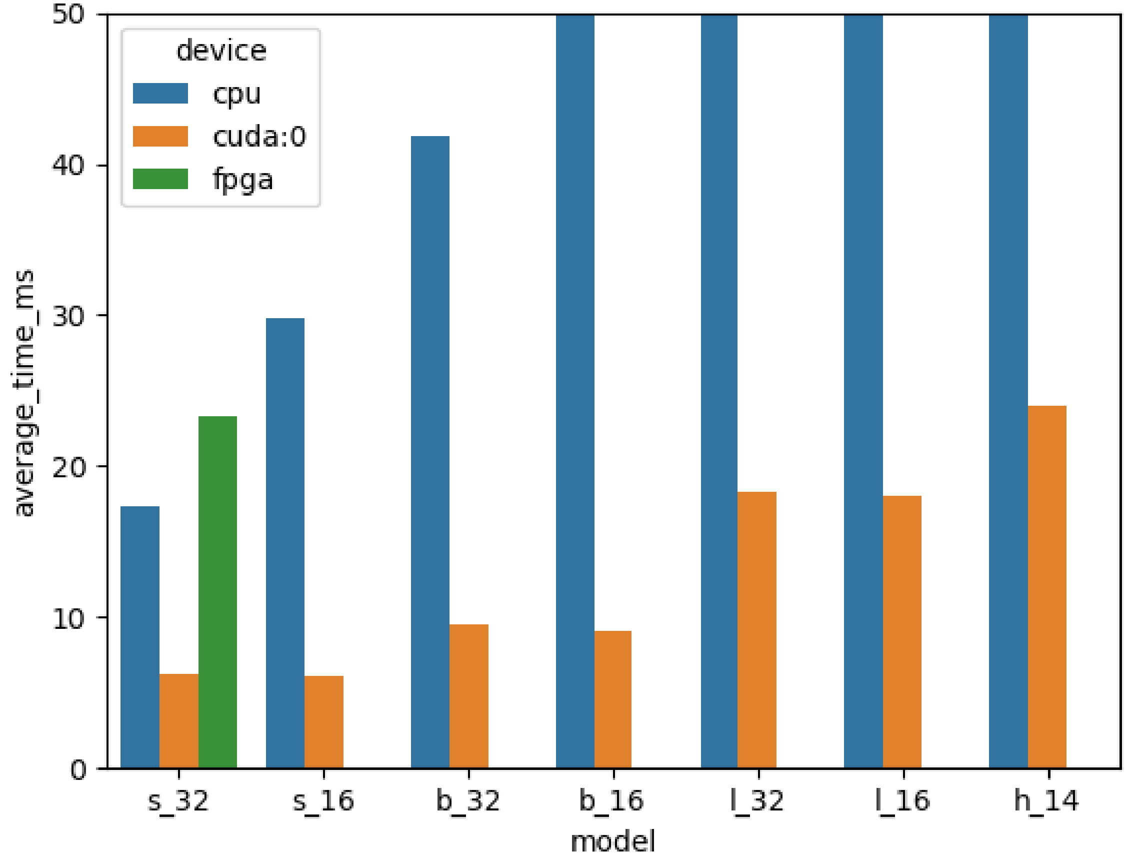
- Patch Embedding Linear
  - BLOCK_SIZE_IN: 8
  - BLOCK_SIZE_OUT: 8
- Pre-Head Layer Norm
  - BLOCK_SIZE: 8
- Classification Head Linear
  - BLOCK_SIZE_IN: 8
  - BLOCK_SIZE_OUT: 10
- Mixer Block Layer Norm 1
  - BLOCK_SIZE: 32
- Mixer Block MLP 1
  - BLOCK_SIZE_IN: 7
  - BLOCK_SIZE_OUT: 32
- Mixer Block Layer Norm 2
  - BLOCK_SIZE: 32
- Mixer Block MLP 2
  - BLOCK_SIZE_IN: 32
  - BLOCK_SIZE_OUT: 32

MLP-Mixer Accelerator: Results

Reported latency from synthesis for inference on a 224x224x3 image takes 23.09 ms.

This is 43.3 FPS which is more than enough for automounts driving at highways speeds

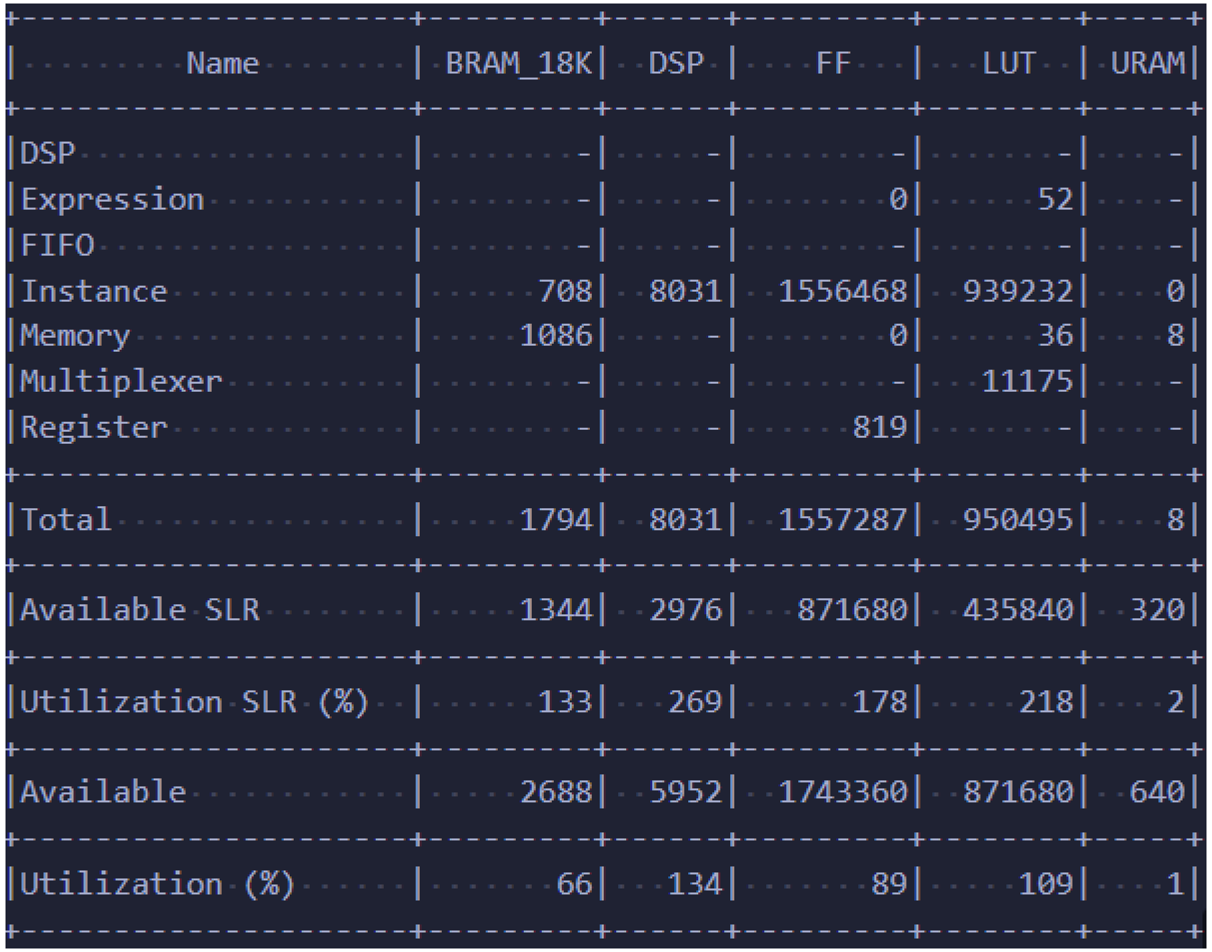
This can certainly be optimized further by optimizing data movement of copying buffers as well as looking into streaming based computations



MLP-Mixer Accelerator: Results

However, the current resource usage exceeds the amount of BRAM memory that can fit in one SLR as well as the number of DSPs and LUTS in one device

More optimizations can be made here to use the URAM as well as less intermediate BRAM memory buffers for processing data



```
+---------------------+---------+------+---------+--------+-----+
|         Name        | BRAM_18K|  DSP |    FF   |   LUT  | URAM|
+---------------------+---------+------+---------+--------+-----+
|DSP                  |        -|     -|        -|       -|    -|
|Expression           |        -|     -|        0|      52|    -|
|FIFO                 |        -|     -|        -|       -|    -|
|Instance             |      708|  8031|  1556468|  939232|    0|
|Memory               |     1086|     -|        0|      36|    8|
|Multiplexer          |        -|     -|        -|   11175|    -|
|Register             |        -|     -|      819|       -|    -|
+---------------------+---------+------+---------+--------+-----+
|Total                |     1794|  8031|  1557287|  950495|    8|
+---------------------+---------+------+---------+--------+-----+
|Available SLR        |     1344|  2976|   871680|  435840|  320|
+---------------------+---------+------+---------+--------+-----+
|Utilization SLR (%)  |      133|   269|      178|     218|    2|
+---------------------+---------+------+---------+--------+-----+
|Available            |     2688|  5952|  1743360|  871680|  640|
+---------------------+---------+------+---------+--------+-----+
|Utilization (%)      |       66|   134|       89|     109|    1|
+---------------------+---------+------+---------+--------+-----+
```

Conclusion

Initial porotype has been demonstrated with initial code framework setup for architecture design

More work needs to be done in order to make more optimizations beyond trivial parallelization as well as more debugging of timing violations to better optimize dataflow

Code: https://github.com/stefanpie/mlp-mixer-acc