MLP-Mixer: An all-MLP Architecture for Vision

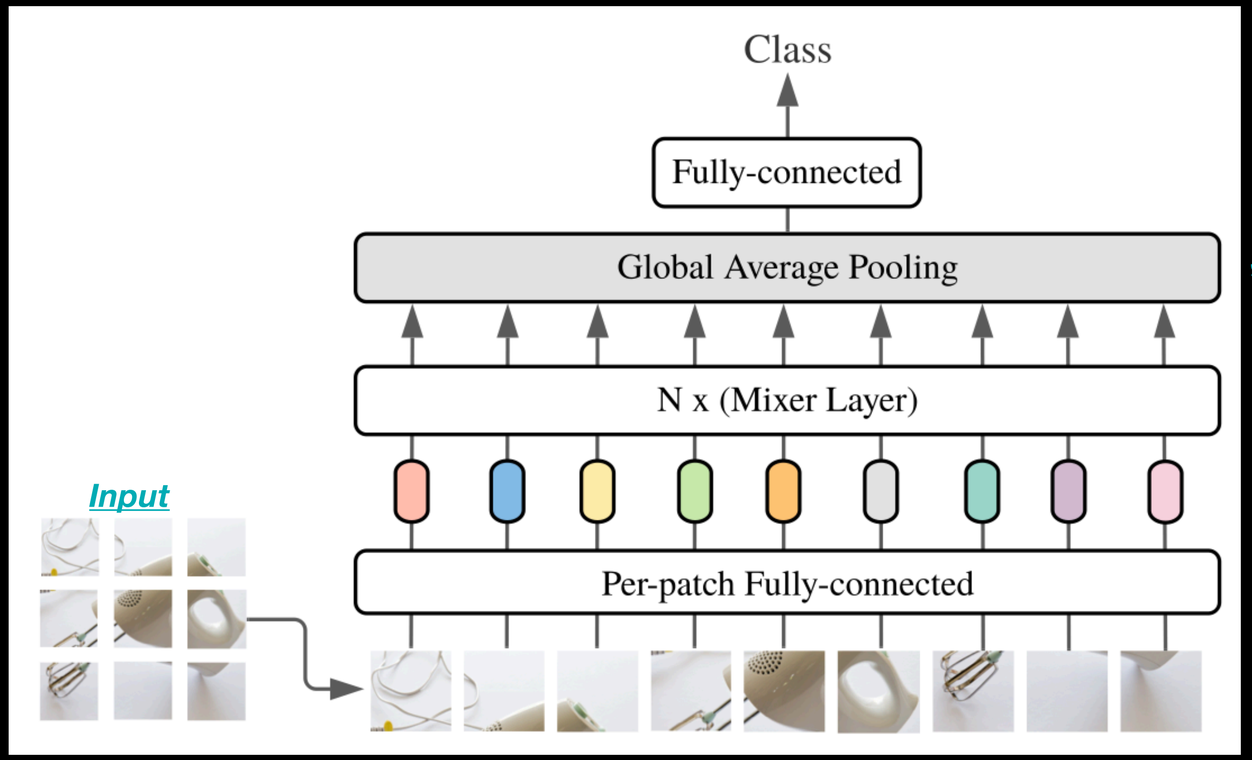
Google Research, Brain Team
CVPR 2021

0805 한지수

Motivation

- architecture entirely based on multi-layer perceptrons(MLPs)
  - MLP that are repeatedly applied across spatial locations/feature channels
  - relies only on basic matrix multiplication routines & changes to data layout & scalar nonlinearities
- On Image classification task
- Modern deep vision architectures mix features
  - given spatial location & per-location(channel-mixing) operation
    - 1x1 convolutions
    - self-attention layers
    - MLP-block
  - btw different spatial locations or both at once & cross-section(token-mixing) operation
    - NxN(larger kernels) convolutions and pooling
    - self-attention layer <- separate per-location operation and cross-location operation (by MLP layer)

Architecture



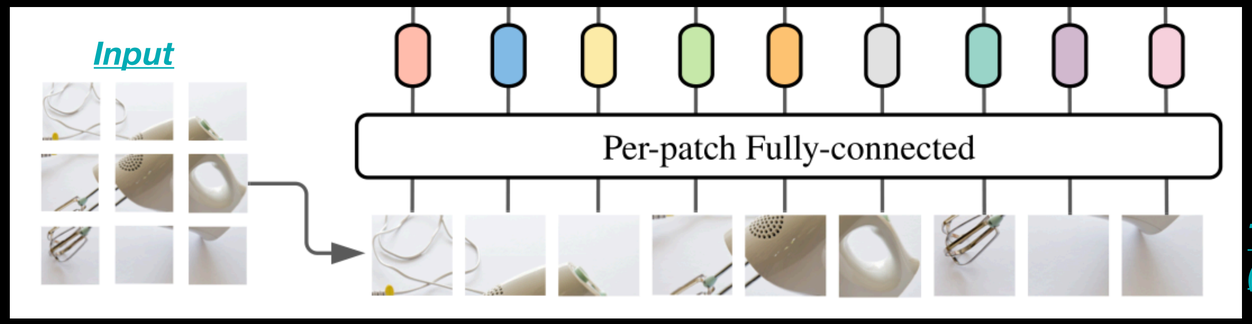
3. Classifier head(생략)

2. Mixer Layer ✨
(MLP 존재)

1. Per-patch linear embedding

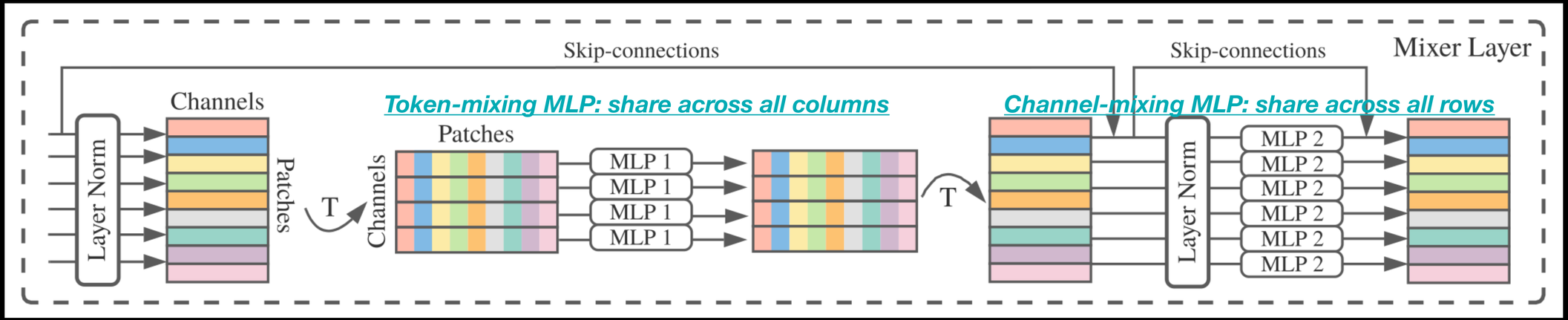
Tokens
(patches × channels) Transformer와 비슷

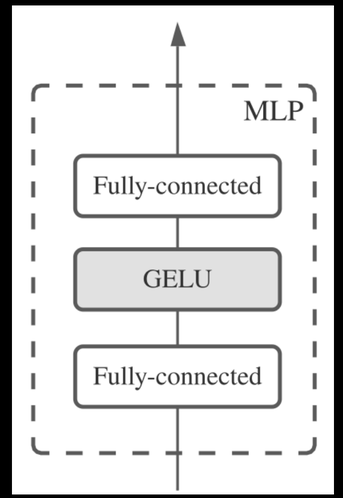
1. Per-patch linear embedding



- S non-overlapping image patches ($S = HW/P^2$), hidden dimension C
- Input table $X \in R^{S \times C}$

2. Mixer Layer

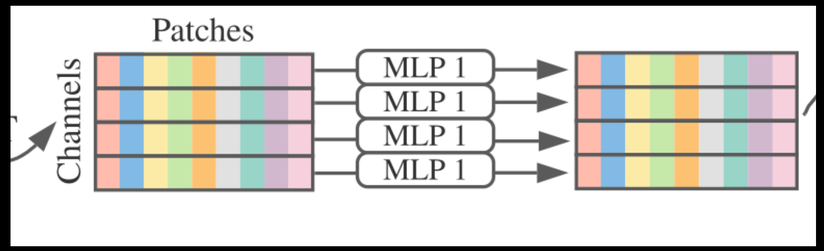




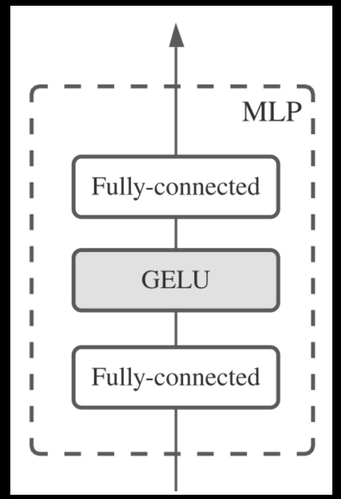
- MLP = [Fully Connected] - [GELU] - [Fully Connected]
- Token-mixing MLP
  - Allow communication btw different spatial locations(tokens)
  - Operate on each channel independently (take individual column of the table)
- Channel-mixing MLP
  - Allow communication btw different channels
  - Operate on each token independently (take individual row of the table)

2. Mixer Layer

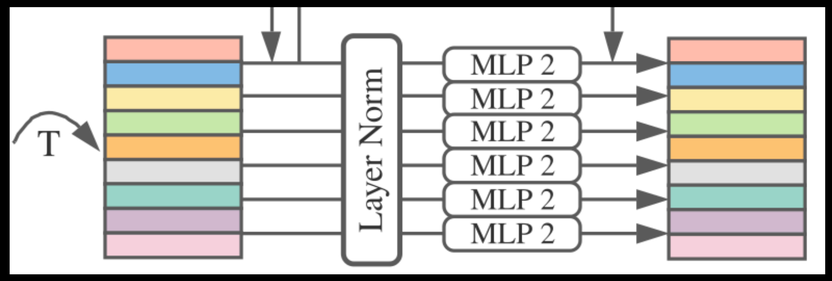
- Token-mixing MLP
  - Allow communication btw different spatial locations(tokens)
  - Operate on each channel independently (take individual column of the table)
  -



$$\mathbf{U}_{*,i} = \mathbf{X}_{*,i} + \mathbf{W}_2\,\sigma\big(\mathbf{W}_1\,\mathrm{LayerNorm}(\mathbf{X})_{*,i}\big), \quad \text{for } i = 1 \dots C,$$

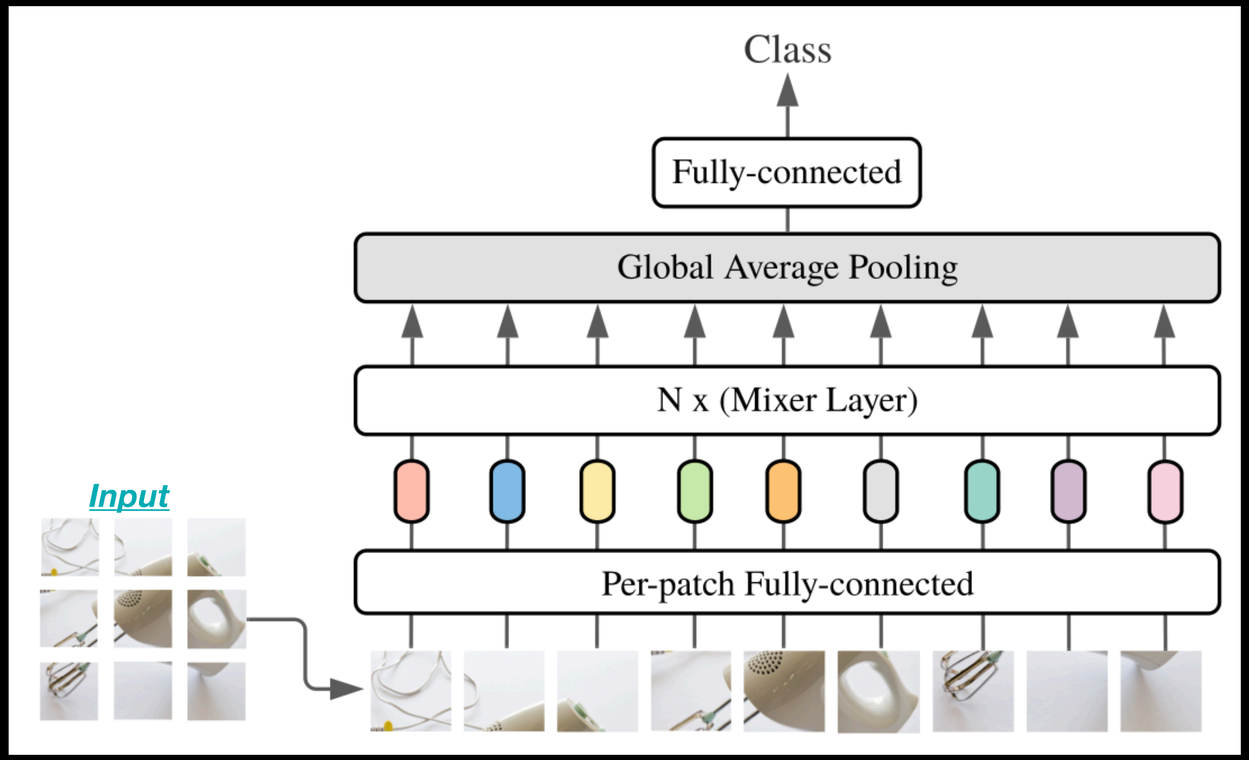


- Channel-mixing MLP
  - Allow communication btw different channels
  - Operate on each token independently (take individual row of the table)
  - Provide positional invariance (CNN과 비슷)



$$\mathbf{Y}_{j,*} = \mathbf{U}_{j,*} + \mathbf{W}_4\,\sigma\big(\mathbf{W}_3\,\mathrm{LayerNorm}(\mathbf{U})_{j,*}\big), \quad \text{for } j = 1 \dots S.$$

Pyramidal structure (Pyramidal ViT나 isotropic ResNet 등의 Typical Design)



3. Classifier head(생략)

2. Mixer Layer ✨
(MLP 존재)

1. Per-patch linear embedding

Tokens
(patches × channels)

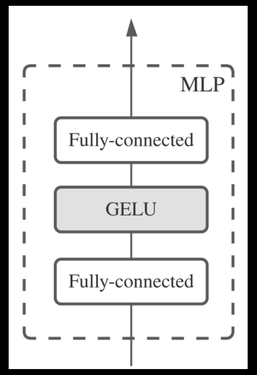
Code

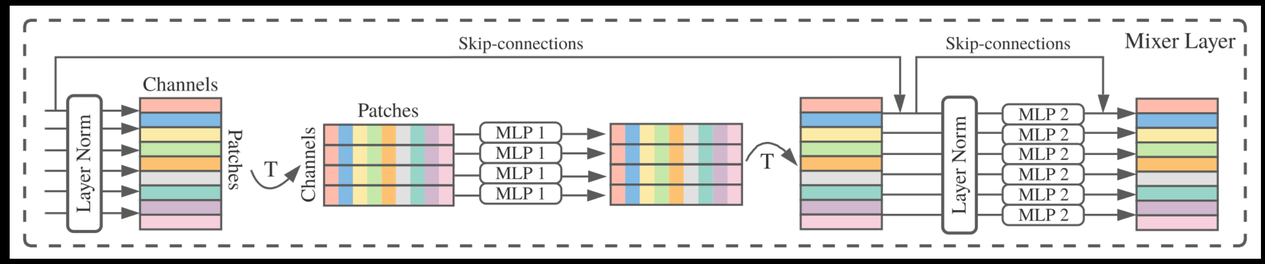
```python
import einops
import flax.linen as nn
import jax.numpy as jnp

class MlpBlock(nn.Module):
  mlp_dim: int
  @nn.compact
  def __call__(self, x):
    y = nn.Dense(self.mlp_dim)(x)
    y = nn.gelu(y)
    return nn.Dense(x.shape[-1])(y)

class MixerBlock(nn.Module):
  tokens_mlp_dim: int
  channels_mlp_dim: int
  @nn.compact
  def __call__(self, x):
    y = nn.LayerNorm()(x)
    y = jnp.swapaxes(y, 1, 2)
    y = MlpBlock(self.tokens_mlp_dim, name='token_mixing')(y)
    y = jnp.swapaxes(y, 1, 2)
    x = x+y
    y = nn.LayerNorm()(x)
    return x+MlpBlock(self.channels_mlp_dim, name='channel_mixing')(y)

class MlpMixer(nn.Module):
  num_classes: int
  num_blocks: int
  patch_size: int
  hidden_dim: int
  tokens_mlp_dim: int
  channels_mlp_dim: int
  @nn.compact
  def __call__(self, x):
    s = self.patch_size
    x = nn.Conv(self.hidden_dim, (s,s), strides=(s,s), name='stem')(x)
    x = einops.rearrange(x, 'n h w c -> n (h w) c')
    for _ in range(self.num_blocks):
      x = MixerBlock(self.tokens_mlp_dim, self.channels_mlp_dim)(x)
    x = nn.LayerNorm(name='pre_head_layer_norm')(x)
    x = jnp.mean(x, axis=1)
    return nn.Dense(self.num_classes, name='head',
                    kernel_init=nn.initializers.zeros)(x)
```

1. Per-patch linear embedding

2. Mixer Layer

3. Classifier head





Results

	ImNet top-1	ReaL top-1	Avg 5 top-1	VTAB-1k 19 tasks	Throughput img/sec/core	TPUv3 core-days
Pre-trained on ImageNet-21k (public)						
● HaloNet [51]	85.8	—	—	—	120	0.10k
● Mixer-L/16	84.15	87.86	93.91	74.95	105	0.41k
● ViT-L/16 [14]	85.30	88.62	94.39	72.72	32	0.18k
● BiT-R152x4 [22]	85.39	—	94.04	70.64	26	0.94k
Pre-trained on JFT-300M (proprietary)						
● NFNet-F4+ [7]	89.2	—	—	—	46	1.86k
● Mixer-H/14	87.94	90.18	95.71	75.33	40	1.01k
● BiT-R152x4 [22]	87.54	90.54	95.33	76.29	26	9.90k
● ViT-H/14 [14]	88.55	90.72	95.97	77.63	15	2.30k
Pre-trained on unlabelled or weakly labelled data (proprietary)						
● MPL [34]	90.0	91.12	—	—	—	20.48k
● ALIGN [21]	88.64	—	—	79.99	15	14.82k