Vision AI

2022 Arxiv Trends

2022-02

hutom

no.	Paper Title	Research group
1	Pushing the limits of self-supervised ResNets: Can we outperform supervised learning without labels on ImageNet?	DeepMind
2	A ConvNet for the 2020s	FAIR, UC Berkeley
3	OMNIVORE: A Single Model for Many Visual Modalities	Meta AI (Facebook)

no.	Paper Title	Correspondence	h-index
4	Vision Transformer for Small-Size Datasets	Byung Cheol Song	28
5	Patches Are All You Need?	Zico Kolter	50
6	Fast and Accurate Single-Image Depth Estimation on Mobile Devices, Mobile AI 2021 Challenge: Report	Fausto T. Benavides	-

Pushing the limits of self-supervised ResNets:
Can we outperform supervised learning without labels on ImageNet?

Nenad Tomasev [*1] Ioana Bica [*†23] Brian McWilliams [*1] Lars Buesing [1] Razvan Pascanu [1] Charles Blundell [1]
Jovana Mitrovic [*1]

https://arxiv.org/pdf/2201.05119.pdf

DeepMind

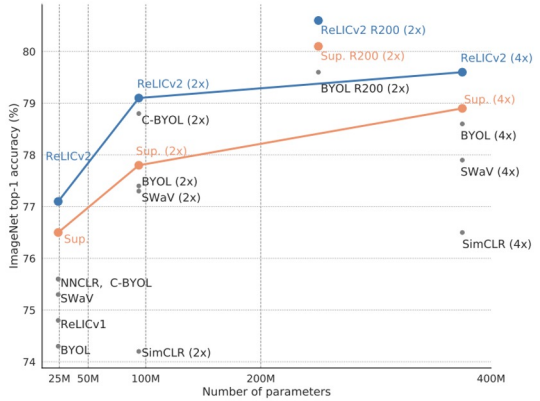


Figure 1. Top-1 linear evaluation accuracy on ImageNet using ResNet50 encoders with 1×, 2× and 4× width multipliers and a ResNet200 encoder with a 2× width multiplier.

ReLICv2

- **Self-supervised methods in representation learning with residual networks.**
- 이전까지의 residual network SSL 은 ImageNet classification benchmark, supervised learning 의 성능을 뛰어넘지 못함.
  - 제안하는 ReLICv2 로 그 성능을 뛰어넘음.

- Combine an explicit invariance loss with a contrastive objective over a varied set of appropriately constructed data view.
  - **invariance loss + contrastive loss 사용.**

- 성능
  - 77.1% top-1 classification accuracy on ImageNet using linear evaluation with a ResNet50.
  - **80.6% with larger ResNet models, outperforming previous SoTA self-supervised approaches.**
  - **First representation learning method to consistently outperform the supervised baseline.**
    - **supervised baseline 을 뛰어넘은 첫 번째 representation learning 기법.**

Representation Learning via Invariant Causal Mechanisms (ReLIC) framework

- Invariant prediction 원칙에 기반하여 representation 학습.
- invariance loss
  - 유사한 포인트와 유사하지 않은 포인트 간의 relationship invariance 계산.
  - Underlying data 의 geometry (기하학적 구조) 의 representations 을 학습.
  - 이러한 속성은 downstream tasks 을 더 잘 수행할 수 있게 representation 학습.

ReLIC (Mitrovic et al., 2021) introduces an *invariance loss* defined as the Kullback-Leibler divergence between the likelihood of the anchor point x_i and one of its positives x_i^+ both of which are computed as in equation 1

$$D_{\mathrm{KL}}(p(x_i)|p(x_i^+)) = \mathrm{sg}\left[\mathbb{E}_{p(x_i;x_i^+)} \log p(x_i; x_i^+)\right] - \mathbb{E}_{p(x_i;x_i^+)} \log p(x_i^+; x_i). \quad (2)$$

ReLICv2

1. Better strategies for selecting similar and dissimilar points.
2. contrastive loss 와 invariance objectives 통합.

ReLICv2

1. Better strategies for selecting similar and dissimilar points.
2. contrastive loss 와 invariance objectives 통합.
    a. minimize the combination of contrastive negative log likelihood and invariance loss.

적절한 positive and negative points 를 select 하는 방법이 ReLIC 과 차이남.

- **Standard SimCLR augmentations + two further augmentation strategies:**
    - **Multi-crop augmentations**
        - 4개의 larger crops, 2개의 smaller crops 으로 구성. 동일한 이미지에 대해 random crop
    - **Saliency-based background removal**
        - DeepUSPS의 fully unsupervised version 을 사용하여, 이미지의 fg 와 bg 분리. 이로써, 이미지 object의 위치 representation 학습 가능.
    - **Standard SimCLR augmentations**
        - Random horizontal flip and color distortion
            - random sequence of brightness, saturation, contrast and hue c

$$\log p(x_i; x_i^+) \qquad (1)$$
$$= \log \frac{e^{\phi_\tau(x_i; x_i^+)}}{e^{\phi_\tau(x_i; x_i^+)} + \sum_{x_i^- \in \mathcal{N}(x_i)} e^{\phi_\tau(x_i; x_i^-)}}$$

$$\ell_{\text{ReLICv2}}(x_i) = \sum_{x_i^+ \in \mathcal{P}(x_i)} -\alpha \log p(x_i; x_i^+) \qquad (3)$$
$$+ \beta D_{\text{KL}}(p(x_i)|p(x_i^+)),$$

$$D_{\text{KL}}(p(x_i)|p(x_i^+)) = \text{sg}\left[\mathbb{E}_{p(x_i; x_i^+)} \log p(x_i; x_i^+)\right] \qquad (2)$$
$$- \mathbb{E}_{p(x_i; x_i^+)} \log p(x_i^+; x_i).$$

Results

- Self-supervised learning 에서 ResNet 아키텍처 전반에 걸쳐 SoTA 성능.
- Supervised ResNet50 baseline 보다 성능이 뛰어난 첫 번째 self-supervised representation learning method.
  - 기존의 다른 실험 비교들은 동일한 네트워크 아키텍처를 사용하진 않았음.
  - Not a like-for-like comparison of network architectures.

without needing to change the network architecture. On top-1 classification accuracy on ImageNet RELICv2 achieves 77.1%, 78. with a ResNet50, while with a ResNet200 2× it achieves 80.6%.

1. Linear evaluation on ImageNet

Method	Top-1	Top-5
Supervised (Chen et al., 2020a)	76.5	93.7
SimCLR (Chen et al., 2020a)	69.3	89.0
MoCo v2 (Chen et al., 2020b)	71.1	-
InfoMin Aug. (Tian et al., 2020)	73.0	91.1
BYOL (Grill et al., 2020)	74.3	91.6
RELIC (Mitrovic et al., 2021)	74.8	92.2
SwAV (Caron et al., 2020)	75.3	-
NNCLR (Dwibedi et al., 2021)	75.6	92.4
C-BYOL (Lee et al., 2021)	75.6	92.7
RELICv2 (ours)	**77.1**	93.3

Table 1. Top-1 and top-5 accuracy (in %) under linear evaluation on the ImageNet test for a ResNet50 encoder set for different representation learning methods. For clarity of exposition we only compares against methods which were at one point state-of-art.

2. Semi-supervised training on ImageNet

Method	Top-1		Top-5	
	1%	10%	1%	10%
Supervised (Chen et al., 2020a)	25.4	56.4	48.4	80.4
SimCLR (Chen et al., 2020a)	48.3	65.6	75.5	87.8
BYOL (Grill et al., 2020)	53.2	68.8	78.4	89.00
SwAV (Caron et al., 2020)	53.9	70.2	78.5	89.9
NNCLR (Dwibedi et al., 2021)	56.4	69.8	80.7	89.3
C-BYOL (Lee et al., 2021)	**60.6**	70.5	**83.4**	90.0
RELICv2 (ours)	58.1	**72.4**	81.3	**91.2**

Table 2. Top-1 and top-5 accuracy (in %) after semi-supervised training with a fraction of ImageNet labels on a ResNet50 encoder for different representation learning methods.

Results

3. Transfer to other tasks
   - Generality of ReLICv2 representations
   - 학습된 features 들이 image domain 전반에 사용 가능한지.

Classification

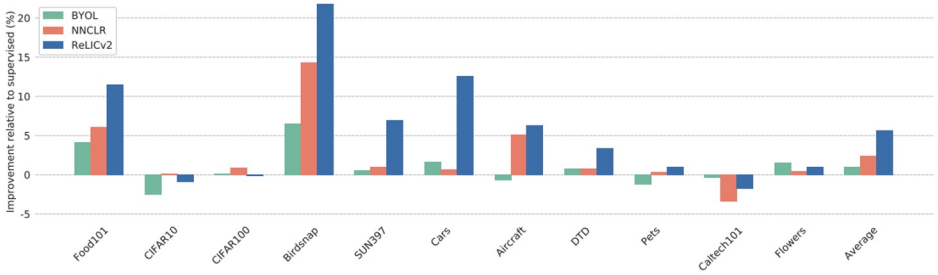


Figure 2. Transfer performance relative to the supervised baseline (a value of 0 indicates equal performance to supervised).

BYOL, NNCLR, ReLICv2 를 representations pre-trained 하여 transfer performance 를 비교
11개 task 중에 ReLICv2 가 7 task 에서 최고 성능을 보임.

Other vision tasks

Method	PASCAL	Cityscapes
BYOL (Grill et al., 2020)	75.7	74.6
DetCon (Hénaff et al., 2021)	77.3	**77.0**
ReLICv2 (ours)	**77.9**	75.2

On both PASCAL and Cityscapes, RELICv2 outperforms BYOL by a significant margin as can and remarkably, on PASCAL outperforms DetCon (Hénaff et al., 2021) which has been specifically trained for detection.

Results

4. Robustness and OOD generalization

Method	MF	T-0.7	Ti	IN-C
Supervised	65.1	73.9	78.4	40.9
SimCLR (Chen et al., 2020a)	53.2	61.7	68.0	31.1
BYOL (Grill et al., 2020)	62.2	71.6	77.0	42.8
RELIC (Mitrovic et al., 2021)	63.1	72.3	77.7	44.5
RELICv2 (ours)	**65.4**	**74.5**	**79.5**	**44.8**

(a) Datasets testing robustness.

Method	IN-R	IN-S	ObjectNet
Supervised	24.0	6.1	26.6
SimCLR (Chen et al., 2020a)	18.3	3.9	14.6
BYOL (Grill et al., 2020)	23.0	8.0	23.0
RELIC (Mitrovic et al., 2021)	23.8	9.1	23.8
RELICv2 (ours)	**23.9**	**9.9**	**25.9**

(b) Datasets testing OOD generalization.

Table 8. Top-1 Accuracy (in %) under linear evaluation on the ImageNetV2 and ImageNet-C datasets (robustness datasets) and ImageNet-R (IN-R), ImageNet-Sketch (IN-S), ObjectNet (out-of-distribution datasets) for different unsupervised representation learning methods. We evaluate on all three variants on ImageNetv2 – matched frequency (MF), Threshold 0.7 (T-0.7) and Top Images (TI). The results for ImageNet-C (IN-C) are averaged across the 15 different corruptions.

5. Large-scale transfer with JFT-300M

Method	Epochs	Top-1
BYOL (Grill et al., 2020)	1000	67.0
Divide and Contrast (Tian et al., 2021)	1000	67.9
RELICv2 (ours)	1000	**70.3**
BYOL (Grill et al., 2020)	3000	67.6
Divide and Contrast (Tian et al., 2021)	3000	69.8
RELICv2 (ours)	3000	**71.1**
BYOL (Grill et al., 2020)	5000	67.9
Divide and Contrast (Tian et al., 2021)	4500	70.7
RELICv2 (ours)	5000	**71.4**

Table 3. Top-1 accuracy (in %) on ImageNet when learning representations using the JFT-300M dataset. Each method is pre-trained on JFT-300M for an ImageNet-equivalent number of epochs and evaluted on the ImageNet validation set under a linear evaluation protocol.

vision transformers 비교

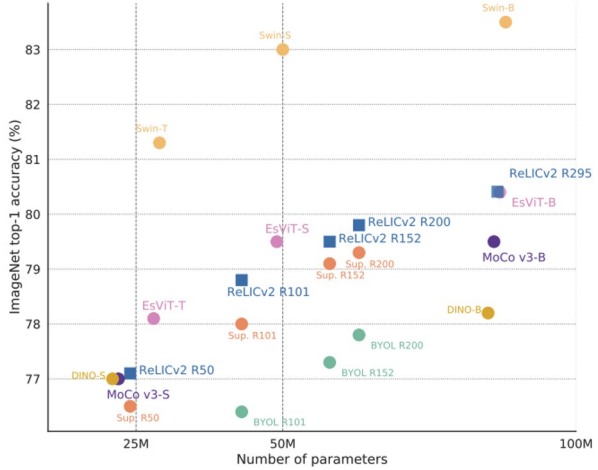


Figure 6. Comparison of ImageNet top-1 accuracy between RELICv2 and recent vision transformer-based architectures (Swin (Liu et al., 2021) represents a fully supervised transformer baseline).

ReLICv2 Pseudo-code

```
'''
f_o: online network: encoder + comparison_net
g_t: target network: encoder + comparison_net
gamma: target EMA coefficient
n_e: number of negatives
p_m: mask apply probability
'''
for x in batch: # load a batch of B samples
    # Apply saliency mask and remove background
    x_m = remove_background(x)
    for i in range(num_large_crops):
        # Select either original or background-removed
        # Image with probability p_m
        x = Bernoulli(p_m) ? x_m : x
        # Do large random crop and augment
        xl_i = aug(crop_l(x))

        ol_i = f_o(xl_i)
        tl_i = g_t(xl_i)

    for i in range(num_small_crops):
        # Do small random crop and augment
        xs_i = aug(crop_s(x))
        # Small crops only go through the online
     network
        os_i = f_o(xs_i)

    loss = 0
    # Compute loss between all pairs of large crops
    for i in range(num_large_crops):
        for j in range(num_large_crops):
            loss += loss_relicv2(ol_i, tl_j, n_e)

    # Compute loss between small crops and large crops
    for i in range(num_small_crops):
        for j in range(num_large_crops):
            loss += loss_relicv2(os_i, tl_j, n_e)
    scale = (num_large_crops + num_small_crops) *
            num_large_crops
    loss /= scale

    # Compute grads, update online and target networks
    loss.backward()
    update(f_o)
    g_t = gamma * g_t + (1 - gamma) * f_o
```

Listing 1. Pseudo-code for RELICv2.

1. 적절한 positive and negative points 를 select 하기 위한 전략
2. 최초로, positive and negative selection 모두에 대한 combine 함.
3. invariance loss 결합하여 사용.

제안한 접근법은 general 하여 특정 positive/negative selection 전략에 특화되어 있지 않음.

A ConvNet for the 2020s

Zhuang Liu[1,2*] Hanzi Mao[1] Chao-Yuan Wu[1] Christoph Feichtenhofer[1] Trevor Darrell[2] Saining Xie[1†]
[1]Facebook AI Research (FAIR) [2]UC Berkeley
Code: https://github.com/facebookresearch/ConvNeXt

https://arxiv.org/pdf/2201.03545v1.pdf

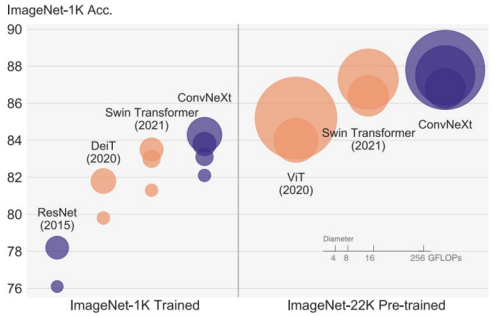
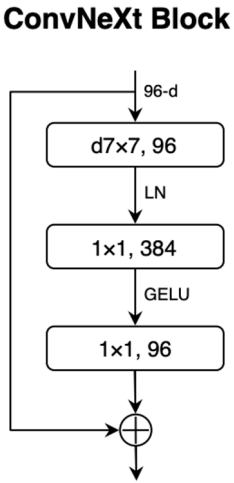


Figure 1. **ImageNet-1K classification** results for • ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take 224^2/384^2 images respectively. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.



<Abstract>

- the effectiveness of hybrid approaches is still largely credited to the intrinsic superiority of Transformers, rather than the inherent inductive biases of convolutions (ex. Swin Transformers)
- 따라서 본 논문에서는 pure ConvNet의 한계를 시험해보겠다! (너의 능력을 보여줘! 느낌)
- "modernize" a standard ResNet toward the design of a vision Transformer, and discover several key components that contribute to the performance difference along the way
- ConvNeXt: Constructed entirely from standard ConvNet modules
  - achieving 87.8% ImageNet top-1 accuracy
  - outperforming Swin Transformers on COCO detection and ADE20K segmentation
  - maintaining the simplicity and efficiency of standard ConvNets

https://arxiv.org/pdf/2201.03545v1.pdf

Modernizing a ConvNet

- ResNet-50 / Swin-T / FLOPs around 4.5×10^9 (주로 비교)
- ResNet-200 / Swin-B / FLOPs around 15.0 × 10^9

Network Modernization

1. Training Techniques: Applying similar training techniques used to train ViT and obtain much improved results compared to the original ResNet-50
   a. 최신의 ViT 테크닉을 resnet-50 에 적용
2. macro design
3. ResNeXt
   a. resnet -> resnext 의 테크닉 적용
4. inverted bottleneck
5. large kernel size
6. various layer-wise micro designs.

1. Training Techniques

- Training epochs: 300 epochs (기존 ResNets: 90 epochs. 더 길게 학습.)
- Optimizer: AdamW
- Data augmentation
  - Mixup
  - Cutmix
  - RandAugment
  - Random Erasing
- Regularization
  - Stochastic Depth
  - Label Smoothing
- This training recipe increased the performance of the ResNet-50 model from **76.1% to 78.8%(+2.7%)**

2. Macro Design : Changing stage compute ratio

- Swin-T's computation ratio of each stage: 1:1:3:1
- larger than Swin-T: 1:1:9:1
- ConvNeXt :
  - 기존 resnet 50 의 각 Stage 안의 block 수가 (3,4,6,4) 이었는데,
  - following the design 하여 the number of blocks in each stage: (3, 3, 9, 3) 으로 바꿈.
  - **The accuracy: 78.8% ⇒ 79.4% (+0.6%)**
  - ***we will use this stage compute ratio.***

C: number of channels
B: number of blocks

- ConvNeXt-T: $C = (96, 192, 384, 768), B = (3, 3, 9, 3)$
- ConvNeXt-S: $C = (96, 192, 384, 768), B = (3, 3, 27, 3)$
- ConvNeXt-B: $C = (128, 256, 512, 1024), B = (3, 3, 27, 3)$
- ConvNeXt-L: $C = (192, 384, 768, 1536), B = (3, 3, 27, 3)$
- ConvNeXt-XL: $C = (256, 512, 1024, 2048), B = (3, 3, 27, 3)$

2. Macro Design : Changing stem to "Patchify"

* Stem : 네트워크 처음 들어가는 부분. ⇒ patchify 형태로 바꾸겠다.

- The stem cell in standard ResNet: 7×7 convolution layer with stride 2, followed by a max pool (results in a 4×4 downsampling of the input images)
- VIT: "patchify" strategy is used as the stem cell, which corresponds to a large kernel size (e.g. kernel size = 14 or 16) and non-overlapping convolution.
- Swin Transformer: uses a similar "patchify" layer, but with a smaller patch size of 4 (architecture's multi-stage 때문에)
- **ConvNeXt: ResNet-style stem cell with a patchify layer implemented using a 4x4, stride 4 convolution layer**
  - ConvNeXt 에서 patchify 를 4*4, stride 4 convolution layer 사용.
- **The accuracy: 79.4% ⇒ 79.5%**
- ***We will use the "patchify stem" (4×4 non-overlapping convolution) in the network.***

3. ResNeXt-ify

- ResNeXt's guiding principle: "use more groups, expand width" : **ResNeXt 에서 group convolution 사용.**
  - 그룹 (cardinality) 을 많이 나누고, 대신에 width (channel) 를 늘려라
- **ConvNeXt: uses depthwise convolution (그룹 수=256),** a special case of grouped convolution where the number of groups equals the number of channels, ⇒ 연산량이 감소. (group convolution 을 극단적으로 사용)
  increases the network width to the same number of channels as Swin-T's (from 64 to 96)
- **The network performance to 80.5% with increased FLOPs (5.3G)**
  - 그룹을 나누면서 채널을 늘렸더니, FLOPs 가 약간 증가함.
- ***We will now employ the ResNeXt design.***

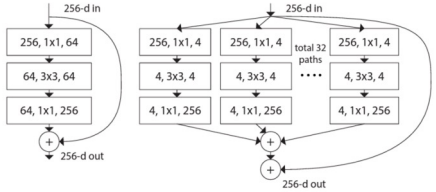


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

- ResNeXt의 핵심은 ResNet 구조의 Architecture에 Inception에서 사용하던 Cardinality 개념을 도입하여 Convolution 연산을 쪼개서 진행하고, 서로 다른 Weight를 구한뒤 합쳐주는 Split-Transform-Merge를 추가한 것이다.
- 이 때, 쪼개진 CNN이 몇개의 path를 가지는지를 결정하는 하이퍼파라미터가 바로 Cardinality이며, 각각의 path에서 가지는 채널을 depth라고 정의한다.
- 따라서 위 그림은, Conv2 Stage 기준 32개의 path와 4의 사이즈를 가지므로, Cardinality = 32, detph = 4의 ResNeXt-50 (32x4d)로 정의할 수 있다.

4. Inverted Bottleneck

-

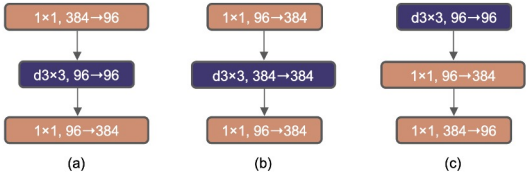
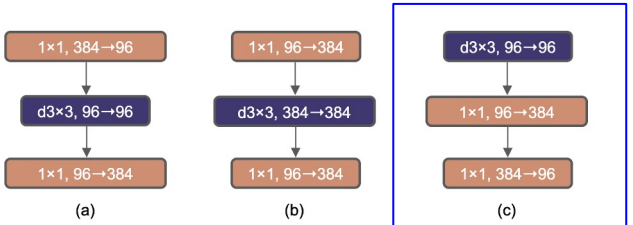


Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

- **results: reduces the whole network FLOPs to 4.6G, slightly improved performance, 80.5% ⇒ 80.6%**
- ***We will now use inverted bottlenecks***



실제 사용되는 inverted block

Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

5. Large Kernel Sizes : Moving up depthwise conv layer

- To explore large kernels, 한 가지 전제조건은 **depthwise conv layer의 위치를 위로 이동하는 것** (Transformers에서 확인가능)
- The MSA block 이 MLP layers 보다 앞선다.
  - MSA (multihead self-attention) 은 depthwise conv layer (3*3) 와 동일한 역할.
  - MSA 가 먼저 하고 MLP layer 를 사용하는 것처럼 depthwise conv layer 를 사용하는 것이 자연스럽다.
- **results: reduces the FLOPs to 4.1G, resulting in a temporary performance degradation to 79.9%.**

5. Large Kernel Sizes : Increasing the kernel size

- experimented with several kernel sizes, including 3, 5, 7, 9, and 11.
- The network's performance increases from 79.9% (3x3) to **80.6% (7x7),** while the network's FLOPs stay roughly the same
- **results: reduces the FLOPs to 4.1G, resulting in a temporary performance degradation to 79.9%.**
- ***We will use 7×7 depthwise conv in each block***

6. Micro Design : Replacing ReLU with GELU

- Activation functions을 Gaussian Error Linear Unit (GELU)사용 (ReLU대신) ⇒ can be thought of as a smoother variant of ReLU
  - Google's BERT, and OpenAI's GPT-2, and, most recently, ViTs
- **ReLU can be substituted with GELU in ConvNet, although the accuracy stays unchanged (80.6%).**

6. Micro Design : Fewer activation functions

- Transformer와 ResNet의 차이: Transformers have fewer activation functions.
- 1x1 conv을 포함한 각 convolutional layer에 activation function을 추가하는 것이 일반적이나,
- ConvNeXt는 Transformer block의 스타일을 복제하여 두 개의 1x1 layer 사이에 있는 layer을 제외한 나머지 블록에서 모든 GE
  을 제거.
- **Result: Increasing 0.7% (81.3%), practically matching the performance of Swin-T.**

6. Micro Design : Fewer normalization layers

- Transformer blocks: usually have fewer normalization layers
- ConvNeXt는 두 개의 BN(BatchNorm) layer을 제거하여 **1x1 layer 앞에 하나의 BN layer만 남김**
- **The accuracy: 81.3% ⇒ 81.4%, surpassing Swin-T's result.**

6. Micro Design : Substituting BN with LN

- **BN보다 Layer Normalization(LN)** has been used in Transformers (좋은 성능 나옴)
- 오리지널 ResNet에서 LN을 BN으로 직접 대체하면 성능이 좋지는 않으나, **ConvNeXt 모델은 LN을 사용한 training에 문제 없**
- **The accuracy: 81.4% ⇒ 81.5%**

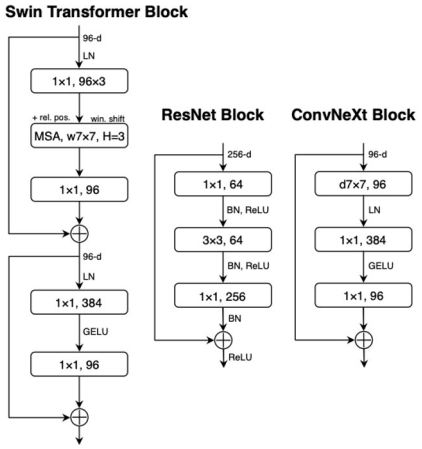
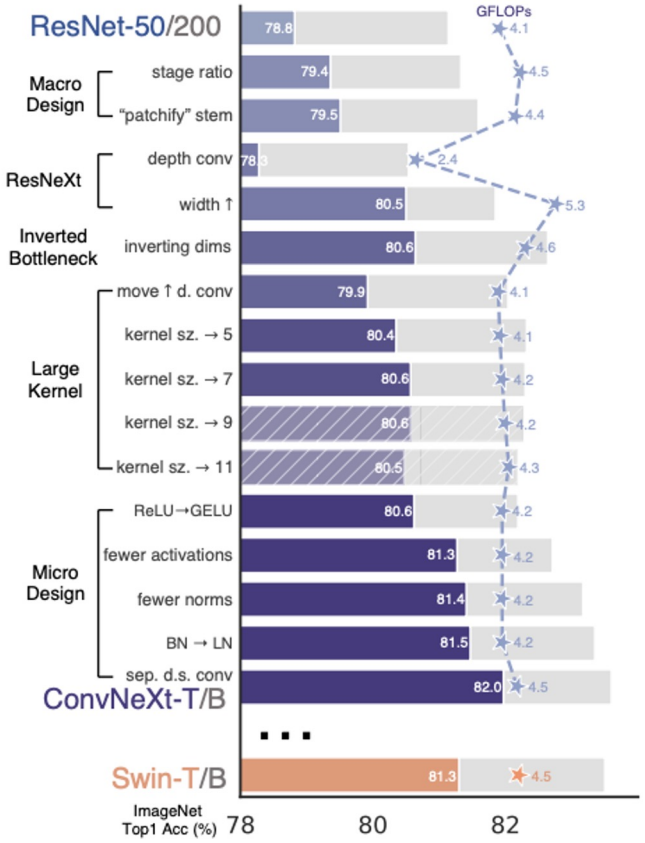


Figure 4. **Block designs** for a ResNet, a Swin Transformer, and a ConvNeXt. Swin Transformer's block is more sophisticated due to the presence of multiple specialized modules and two residual connections. For simplicity, we note the linear layers in Transformer MLP blocks also as "1×1 convs" since they are equivalent.

6. Micro Design :Separate downsampling layers

- In ResNet: is achieved by the residual block at the start of each stage, using 3x3 conv with stride 2 (1x1 conv with stride 2 at the shortcut connection)
- In Swin T: a separate downsampling layer is added between stages
  - **stage 사이에 따로 downsampling layer 존재.**
- ConvNeXt: uses 2x2 conv layers with stride 2 for spatial downsampling and **adding normalization layers** wherever spatial resolution is changed
- **The accuracy: 81.5% ⇒ 82% (Swin T 상당히 능가)**



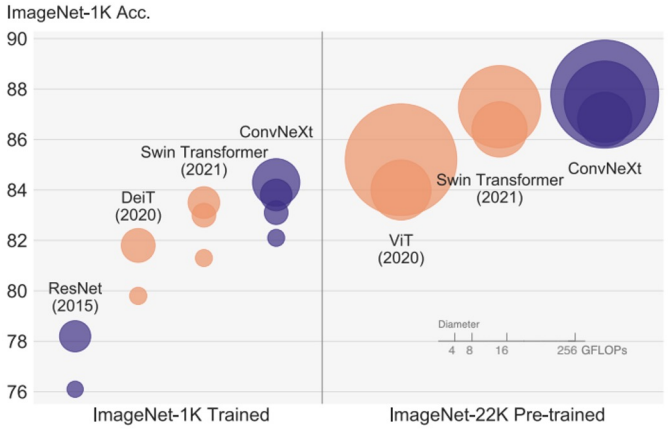


Figure 1. **ImageNet-1K classification** results for • ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take 224^2/384^2 images respectively. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

Results – ImageNet-1K

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
• RegNetY-4G [51]	224^2	21M	4.0G	1156.7	80.0
• RegNetY-8G [51]	224^2	39M	8.0G	591.6	81.7
• RegNetY-16G [51]	224^2	84M	16.0G	334.7	82.9
• EffNet-B3 [67]	300^2	12M	1.8G	732.1	81.6
• EffNet-B4 [67]	380^2	19M	4.2G	349.4	82.9
• EffNet-B5 [67]	456^2	30M	9.9G	169.1	83.6
• EffNet-B6 [67]	528^2	43M	19.0G	96.9	84.0
• EffNet-B7 [67]	600^2	66M	37.0G	55.1	84.3
○ DeiT-S [68]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [68]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
• ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
• ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
• ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
• ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
• ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
• ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**

Results – ImageNet-22K

ImageNet-22K pre-trained models					
• R-101x3 [36]	384^2	388M	204.6G	-	84.4
• R-152x4 [36]	480^2	937M	840.5G	-	85.4
○ ViT-B/16 [18]	384^2	87M	55.5G	93.1	84.0
○ ViT-L/16 [18]	384^2	305M	191.1G	28.5	85.2
○ Swin-B	224^2	88M	15.4G	286.6	85.2
• ConvNeXt-B	224^2	89M	15.4G	292.1	**85.8**
○ Swin-B	384^2	88M	47.0G	85.1	86.4
• ConvNeXt-B	384^2	89M	45.1G	95.7	**86.8**
○ Swin-L	224^2	197M	34.5G	145.0	86.3
• ConvNeXt-L	224^2	198M	34.4G	146.8	**86.6**
○ Swin-L	384^2	197M	103.9G	46.0	87.3
• ConvNeXt-L	384^2	198M	101.0G	50.4	**87.5**
• ConvNeXt-XL	224^2	350M	60.9G	89.3	**87.0**
• ConvNeXt-XL	384^2	350M	179.0G	30.2	**87.8**

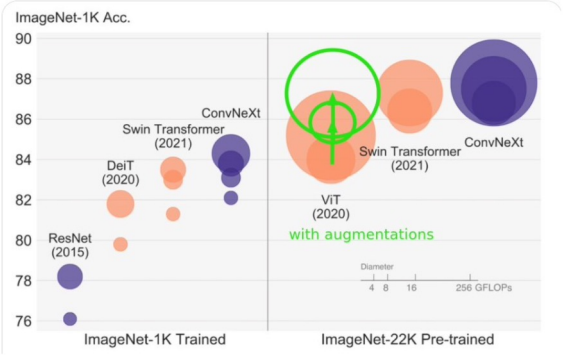
Table 1. **Classification accuracy on ImageNet-1K.** Similar to Transformers, ConvNeXt also shows promising scaling behavior with higher-capacity models and a larger (pre-training) dataset. Inference throughput is measured on a V100 GPU, following [42]. On an A100 GPU, ConvNeXt can have a much higher throughput than Swin Transformer. See Appendix E.

Lucas Beyer
@giffmana

The ConvNeXt paper is rightfully getting some attention: it's good work and has beautiful plots.

But, Fig1 needs a little correction IMO. They compare heavily aug/reg swin+convnext to plain ViT. We fixed this in arxiv.org/abs/2106.10270 which is what should always be compared to.

트윗 번역하기



Mingxing Tan
@tanmingxing

@yieldthought 님과 @ak92501 님에게 보내는 답글

Well, AutoML models are already ahead by a couple of months, but this paper choses to compare 2019 EffNetV1 instead of 2021 EffNetV2 . Here is a better comparison:

트윗 번역하기

Table 1. EfficientNetV2 (2021/4/1, ICML) vs ConvNeXt (2022/1/10).

		Accuracy	Params	FLOPs	Throughput
ImageNet-1k	EfficientNetV2-L	85.7%	120M	53B	163 fps
	ConvNeXt-L	85.5%	198M	101B	50 fps
ImageNet-21k	EfficientNetV2-L	86.8%	120M	53B	163 fps
	ConvNeXt-B	86.8%	89M	45B	96 fps

Isotropic ConvNeXt vs. ViT

model	#param.	FLOPs	throughput (image / s)	training mem. (GB)	IN-1K acc.
○ ViT-S	22M	4.6G	978.5	4.9	79.8
● ConvNeXt-S (*iso.*)	22M	4.3G	1038.7	4.2	79.7
○ ViT-B	87M	17.6G	302.1	9.1	81.8
● ConvNeXt-B (*iso.*)	87M	16.9G	320.1	7.7	82.0
○ ViT-L	304M	61.6G	93.1	22.5	82.6
● ConvNeXt-L (*iso.*)	306M	59.7G	94.4	20.4	82.6

Table 2. **Comparing isotropic ConvNeXt and ViT.** Training memory is measured on V100 GPUs with 32 per-GPU batch size.

Results – Object Detection and Segmentation on COCO, Semantic Segmentation on ADE20K

backbone	FLOPs	FPS	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
Mask-RCNN 3× schedule								
○ Swin-T	267G	23.1	46.0	68.1	50.3	41.6	65.1	44.9
● ConvNeXt-T	262G	25.6	**46.2**	67.9	50.8	**41.7**	65.0	44.9
Cascade Mask-RCNN 3× schedule								
● ResNet-50	739G	11.4	46.3	64.3	50.5	40.1	61.7	43.4
● X101-32	819G	9.2	48.1	66.5	52.4	41.6	63.9	45.2
● X101-64	972G	7.1	48.3	66.4	52.3	41.7	64.0	45.1
○ Swin-T	745G	12.2	50.4	69.2	54.7	43.7	66.6	47.3
● ConvNeXt-T	741G	13.5	**50.4**	69.1	54.8	**43.7**	66.5	47.3
○ Swin-S	838G	11.4	51.9	70.7	56.3	45.0	68.2	48.8
● ConvNeXt-S	827G	12.0	**51.9**	70.8	56.5	**45.0**	68.4	49.1
○ Swin-B	982G	10.7	51.9	70.5	56.4	45.0	68.1	48.9
● ConvNeXt-B	964G	11.4	**52.7**	71.3	57.2	**45.6**	68.9	49.5
○ Swin-B‡	982G	10.7	53.0	71.8	57.5	45.8	69.4	49.7
● ConvNeXt-B‡	964G	11.5	**54.0**	73.1	58.8	**46.9**	70.6	51.3
○ Swin-L‡	1382G	9.2	53.9	72.4	58.8	46.7	70.1	50.8
● ConvNeXt-L‡	1354G	10.0	**54.8**	73.8	59.8	**47.6**	71.3	51.7
● ConvNeXt-XL‡	1898G	8.6	**55.2**	74.2	59.9	**47.7**	71.6	52.2

Table 3. **COCO object detection and segmentation results** using Mask-RCNN and Cascade Mask-RCNN. ‡ indicates that the model is pre-trained on ImageNet-22K. ImageNet-1K pre-trained Swin results are from their Github repository [3]. AP numbers of the ResNet-50 and X101 models are from [42]. We measure FPS on an A100 GPU. FLOPs are calculated with image size (1280, 800).

backbone	input crop.	mIoU	#param.	FLOPs
ImageNet-1K pre-trained				
○ Swin-T	512^2	45.8	60M	945G
● ConvNeXt-T	512^2	**46.7**	60M	939G
○ Swin-S	512^2	49.5	81M	1038G
● ConvNeXt-S	512^2	**49.6**	82M	1027G
○ Swin-B	512^2	49.7	121M	1188G
● ConvNeXt-B	512^2	**49.9**	122M	1170G
ImageNet-22K pre-trained				
○ Swin-B‡	640^2	51.7	121M	1841G
● ConvNeXt-B‡	640^2	**53.1**	122M	1828G
○ Swin-L‡	640^2	53.5	234M	2468G
● ConvNeXt-L‡	640^2	**53.7**	235M	2458G
● ConvNeXt-XL‡	640^2	**54.0**	391M	3335G

Table 4. **ADE20K validation results** using UperNet [80]. ‡ indicates IN-22K pre-training. Swins' results are from its GitHub repository [2]. Following Swin, we report mIoU results with multi-scale testing. FLOPs are based on input sizes of (2048, 512) and (2560, 640) for IN-1K and IN-22K pre-trained models, respectively.

OMNIVORE: A Single Model for Many Visual Modalities

Rohit Girdhar* Mannat Singh* Nikhila Ravi* Laurens van der Maaten Armand Joulin Ishan Misra*
Meta AI
https://facebookresearch.github.io/omnivore

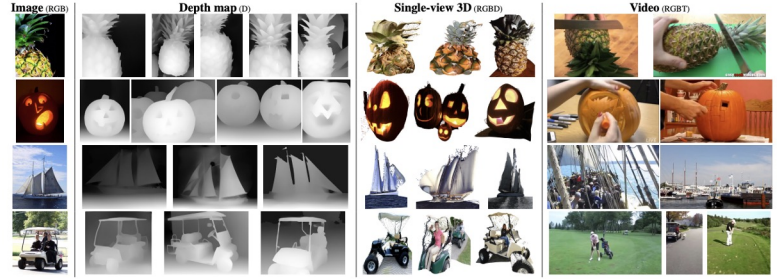


Figure 1. OMNIVORE is a single vision model for many different visual modalities. It learns to construct representations that are aligned across visual modalities, without requiring training data that specifies correspondences between those modalities. Using OMNIVORE's shared visual representation, we successfully identify nearest neighbors of **left:** an image (ImageNet-1K validation set) in vision datasets that contain **right:** depth maps (ImageNet-1K training set), single-view 3D images (ImageNet-1K training set), and videos (Kinetics-400 validation set).

https://arxiv.org/pdf/2201.08377v1.pdf

- Meta AI (https://github.com/facebookresearch/omnivore)
- three different visual modalites: Images, videos, and single-view 3D
  - flexibility of transformer-based architecture (각 Model does not use a custom architecture)
- off-the shelf standard dataset 사용
  - ImageNet 1k, Kinetics, SUN RGB-D
- 같은 사이즈의 modality specific model 보다 좋은성능

Contribution

1. being able to perform cross-modal generalization
2. able to save the research and engineering effort dedicated to optimizing models for a particular task
3. **this model is naturally multi-model that can leverage new visual sensors as they become available**

OMNIVORE Model

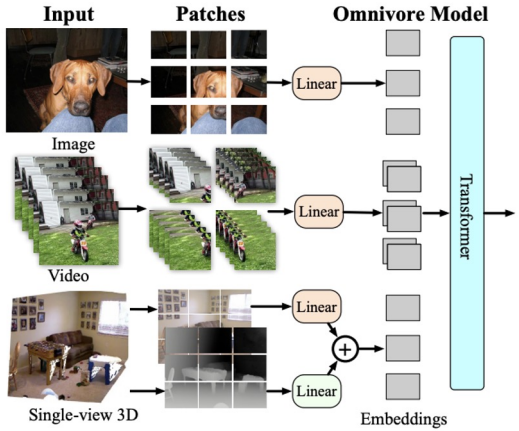
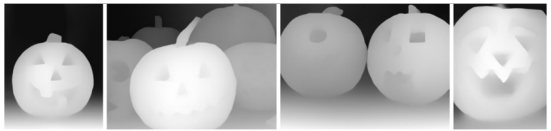


Figure 2. Multiple visual modalities in the OMNIVORE model. We convert image, video, and single-view 3D modalities into embeddings that are fed into a Transformer model. The images are converted into patches, videos into spatio-temporal tubes, and the single-view 3D images are converted into RGB patches and depth patches. The patches are projected into embeddings using linear layers. We use the same linear layer for (image or video) RGB patches and a separate one for depth patches.

Video - RGBT (Kinetics-400)

Image - RGB (ImageNet-1k)



Depthmap + Single-view 3D (RGBD)

Model Architecture

- visual modality 끼리는 최대한 parameter sharing 하도록 디자인
  - RGB ⇒ same layer to embedding
  - depth는 따로 embedding 해서 add it to RGB patch (Linear LN layer)
- Swin Transformer
- Self-attention

Expeirments (Eval)

1. transfer dataset

Dataset	Task	#cls	#train	#val
iNaturalist-2018 (iNat18) [36]	Fine-grained cls.	8142	437K	24K
Oxford-IIIT Pets (Pets) [67]	Fine-grained cls.	37	3.6K	3.6K
Places-365 (P365) [100]	Scene cls.	365	1.8M	36K
Something Something-v2 (SSv2) [31]	Action cls.	174	169K	25K
EPIC-Kitchens-100 (EK100) [20]	Action cls.	3806	67K	10K
NYU-v2 (NYU) [63]	Scene cls.	10	794	653
NYU-v2-seg (NYU-seg) [63]	Segmentation	40	794	653

Table 1. Transfer datasets used to evaluate OMNIVORE on image, video and single-view 3D modalities. The table reports the task, number of classes (#cls), number of training samples (#train), and number of validation samples (#val) for each dataset.

- off-the shelf standard dataset 사용 (pre-train)
  - ImageNet 1k, Kinetics, SUN RGB-D

2. modality-specific models와 비교했을 때 비슷하거나 성능 개선
   - same model architecture and number of parameter

Method	ImageNet-1K		Kinetics-400		SUN
	top-1	top-5	top-1	top-5	top-1
ImageSwin-T [49]	**81.2**	95.5	✗	✗	✗
VideoSwin-T [50]	✗	✗	78.8	93.6	✗
DepthSwin-T	✗	✗	✗	✗	**63.1**
OMNIVORE (Swin-T)	80.9	**95.5**	**78.9**	**93.8**	62.3
ImageSwin-S [49]	83.2	96.2	✗	✗	✗
VideoSwin-S [50]	✗	✗	80.6	94.5	✗
DepthSwin-S	✗	✗	✗	✗	**64.9**
OMNIVORE (Swin-S)	**83.4**	**96.6**	**82.2**	**95.4**	64.6
ImageSwin-B [49]	83.5	96.5	✗	✗	✗
VideoSwin-B [50]	✗	✗	80.6	94.6	✗
DepthSwin-B	✗	✗	✗	✗	64.8
OMNIVORE (Swin-B)	**84.0**	**96.8**	**83.3**	**95.8**	**65.4**

Table 2. OMNIVORE *vs.* modality-specific models that have the same model architecture and number of parameters. OMNIVORE is a single model trained from scratch jointly on the IN1K, K400 and SUN datasets whereas the modality-specific models are trained specifically for each dataset (modality). The ImageSwin model is trained from scratch while the VideoSwin and DepthSwin models are finetuned from the ImageSwin model. OMNIVORE performs at-par or outperforms modality-specific models.

Experiments (Eval)

3. downstream task fine-tuning 시 model 성능. better performance

Model	Method	P365		iNat18		Pets		SSv2		EK100		NYU	NYU-seg
		top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5	top-1	mIoU
Swin-T	Specific	57.9	87.3	**69.7**	87.6	93.7	99.6	62.2	88.7	41.8	62.8	72.5	47.9
	OMNIVORE	**58.2**	**87.4**	69.0	**87.7**	**94.2**	**99.7**	**64.4**	**89.7**	**42.7**	**63.1**	**77.0**	**49.7**
Swin-S	Specific	58.7	**88.1**	72.9	90.2	94.4	99.6	66.8	91.1	42.5	63.4	**76.7**	51.3
	OMNIVORE	**58.8**	88.0	**73.6**	**90.8**	**95.2**	**99.7**	**68.2**	**91.8**	**44.9**	**64.8**	**76.7**	**52.7**
Swin-B	Specific	58.9	**88.3**	73.2	90.9	94.2	99.7	65.8	90.6	42.8	64.0	76.4	51.1
	OMNIVORE	**59.2**	**88.3**	**74.4**	**91.1**	**95.1**	**99.8**	**68.3**	**92.1**	**47.4**	**67.7**	**78.8**	**54.0**

Table 3. Comparing OMNIVORE with modality-specific models after finetuning the models on seven downstream tasks. Results are presented for three different model sizes: T, S, and B. Our image specific model is pretrained on IN1K. The video specific and single-view 3D specific models are both initialized using inflation from the pretrained image-specific model and fine-tuned on K400 and SUN RGB-D respectively. OMNIVORE models are at par with or outperform modality-specific models on nearly all downstream tasks.

Experiments (Eval)

4. 3개의 data type (ImageNet1K, Kinetics, SUN) task와 비교하더라도

advanced model과 동등한 성능 or better performance

Method	ImageNet-1K top-1	ImageNet-1K top-5	Kinetics-400 top-1	Kinetics-400 top-5	SUN top-1
MViT-B-24 [24]	83.1	-	✗	✗	✗
ViT-L/16 [21]	85.3	-	✗	✗	✗
ImageSwin-B [49]	85.2	97.5	✗	✗	✗
ImageSwin-L [49]	86.3	97.9	✗	✗	✗
ViT-B-VTN [64]	✗	✗	79.8	94.2	✗
TimeSformer-L [8]	✗	✗	80.7	94.7	✗
ViViT-L/16x2 320 [5]	✗	✗	81.3	94.7	✗
MViT-B 64×3 [24]	✗	✗	81.2	95.1	✗
VideoSwin-B [50]	✗	✗	82.7	95.5	✗
VideoSwin-L [50]	✗	✗	83.1	95.9	✗
DF^{2}Net [48]	✗	✗	✗	✗	54.6
G-L-SOOR [77]	✗	✗	✗	✗	55.5
TRecgNet [22]	✗	✗	✗	✗	56.7
CNN-RNN [9]	✗	✗	✗	✗	60.7
Depth Swin-B	✗	✗	✗	✗	69.1
Depth Swin-L	✗	✗	✗	✗	68.7
OMNIVORE (Swin-B)	85.3	97.5	84.0	96.2	67.2
OMNIVORE (Swin-L)	86.0	97.7	84.1	96.3	67.1

Table 4. Comparing OMNIVORE with state-of-the-art models on the image, video, and single-view 3D classification datasets used to pre-train OMNIVORE. OMNIVORE performs on par with or better than state-of-the-art models on all three pre-training tasks, including modality-specific models of similar size.

5. Image classification fine tuning 실험에서의 sota model과 비교

Method	P365	iNat18	Pets
EfficientNet B6 [92]	58.5	79.1	95.4
EfficientNet B7 [92]	58.7	80.6	–
EfficientNet B8 [92]	58.6	81.3	–
DeiT-B [84] ↑	–	79.5	–
ViT-B/16 [21] ↑	58.2	79.8	–
ViT-L/16 [21] ↑	59.0	81.7	–
OMNIVORE (Swin-B)	59.3	76.3	95.5
OMNIVORE (Swin-B ↑)	59.6	82.6	95.9
OMNIVORE (Swin-L)	59.4	78.0	95.7
OMNIVORE (Swin-L ↑)	**59.9**	**84.1**	**96.1**

Table 5. Comparing OMNIVORE with state-of-the-art models in image classification finetuning experiments on three datasets. OMNIVORE representations generalize well to scene classification (P365) and fine-grained classification (iNat18, Pets). ↑ indicates finetuning on a higher resolution image (384 × 384px; see [85]).

정리

- **다양한 visual modalities 를 이용하여 모델 성능 향상 (Potential)**
- **하지만 몇가지 limitation 존재**
  1. OMNIVORE just perform on single-view 3D images; it doesn’t generalize to other 3D delegations (i,e voxels, point clouds..)
  2. Only perform on visual data (not co-occuring modalities like audio)
  3. using only classification and structured prediction tasks.

Fast and Accurate Single-Image Depth Estimation on Mobile Devices, Mobile AI 2021 Challenge: Report

Andrey Ignatov Grigory Malivenko David Plowman Samarth Shukla Radu Timofte
Ziyu Zhang Yicheng Wang Zilong Huang Guozhong Luo Gang Yu Bin Fu
Yiran Wang Xingyi Li Min Shi Ke Xian Zhiguo Cao Jin-Hua Du
Pei-Lin Wu Chao Ge Jiaoyang Yao Fangwen Tu Bo Li Jung Eun Yoo
Kwanggyoon Seo Jialei Xu Zhenyu Li Xianming Liu Junjun Jiang
Wei-Chi Chen Shayan Joya Huanhuan Fan Zhaobing Kang Ang Li
Tianpeng Feng Yang Liu Chuannan Sheng Jian Yin Fausto T. Benavides

Figure 1. The original RGB image and the corresponding depth map obtained with the ZED 3D camera.

- CVPR 2021 Challenges
- Depth estimation, 현재 제안된 솔루션들은 계산 비용이 많이 들어 모바일 장치에서의 추론 어려움
- In this paper, 이러한 문제점을 해결하기 위해서 여러 가지 방법들을 적용해보고 그 결과를 정리한 내용.
- 140팀중 10팀 완료
- 모든 참가자가 encoder-decoder 기반 architecture를 적용.
- 대부분의 참가자가 image classification backbone을 사용, architecture가 성능 개선을 위해 skip-connection 사용.
    - HIT-AIIA 팀만 EfficientNet-B1을 사용하고 나머지는 mobilenet을 사용
- knowledge distillation.
- Tencent GY-Lab(1위)의 경우 FastDepth 모델을 사용 ⇒ Raspberry-pi 4에서 10 FPS 수준의 속도달성.

Environments

- Raspberry pi 4 (Broadcom BMC2711, Cortex-A72, 1.5 GHz)
- Raspberry Pi OS (linux)
- TensorFlow Lite 2.5.0 Linux build

Dataset

- RGB-16bit-depth image pairs was collected using ZED stereo camera
  a. average depth estimation error of less than 0.2 [m]
  b. object located closer than 8 [m]
  c. around 8.3k VGA image pairs (640x480 pixels)

Scoring System

- Performance 측정
  - ⇒ Root Mean Squared Error (RMSE, absolute depth estimation accuracy)
  - ⇒ Relative Depth: Scale Invariant Root Mean Squared Error (si-RMSE, relative position of the object)
  - ⇒ Average "log_10" and Relative (REL) error

$$FinalScore = \frac{2^{-10 \cdot si-RMSE}}{C \cdot runtime}$$

Team	Author	Framework	Model Size, MB	si-RMSE↓	RMSE↓	LOG10↓	REL↓	Runtime, ms ↓	Final Score
Tencent GY-Lab	Parkzyzhang	PyTorch / TensorFlow	3.4	0.2836	3.56	0.1121	0.2690	**97**	**129.41**
SMART	KX_SMART	PyTorch / TensorFlow	15.0	0.2602	3.25	0.1043	0.2678	1197	14.51
Airia-Team1	dujinhua	TensorFlow	64.9	0.2408	3.00	0.0904	0.2389	1933	11.75
YTL	Jacob.Yao	PyTorch / TensorFlow	56.2	0.2902	3.91	0.1551	0.4700	1275	8.98
CFL2	jey	PyTorch / TensorFlow	9.6	0.2761	9.68	2.3393	0.9951	772	5.5
HIT-AIIA	zhyl	Keras / TensorFlow	56.0	**0.2332**	**2.72**	**0.0831**	**0.2189**	6146	4.11
weichi	weichi	TensorFlow	0.5	0.4659	7.56	0.4493	0.5992	582	1.72
MonoVision Palace	shayanj	TensorFlow	15.3	0.3543	4.16	0.1441	0.3862	3466	1.36
3dv oppo	fanhuanhuan	PyTorch / TensorFlow	187	0.2678	5.96	0.3300	0.5152	26494	0.59
MegaUe	faustChok	Keras / TensorFlow	118	0.3737	9.08	0.9605	0.8573	9392	0.38

Table 1. MAI 2021 Monocular Depth Estimation challenge results and final rankings. The runtime values were obtained on 640×480 px images on the Raspberry Pi 4 device. Team *Tencent GY-Lab* is the challenge winner, the best fidelity results are obtained by team *HIT-AIIA*.

- 400팀 중 10팀 완료
- 모든 참가자가 encoder-decoder 기반 architecture를 적용.
- 대부분의 참가자가 image classification backbone을 사용, architecture가 성능 개선을 위해 skip-connection 사용.
  - HIT-AIIA 팀만 EfficientNet-B1을 사용하고 나머지는 mobilenet을 사용
- knowledge distillation.
- Tencent GY-Lab(1위)의 경우 FastDepth 모델을 사용 ⇒ Raspberry-pi 4에서 10 FPS 수준의 속도발생.

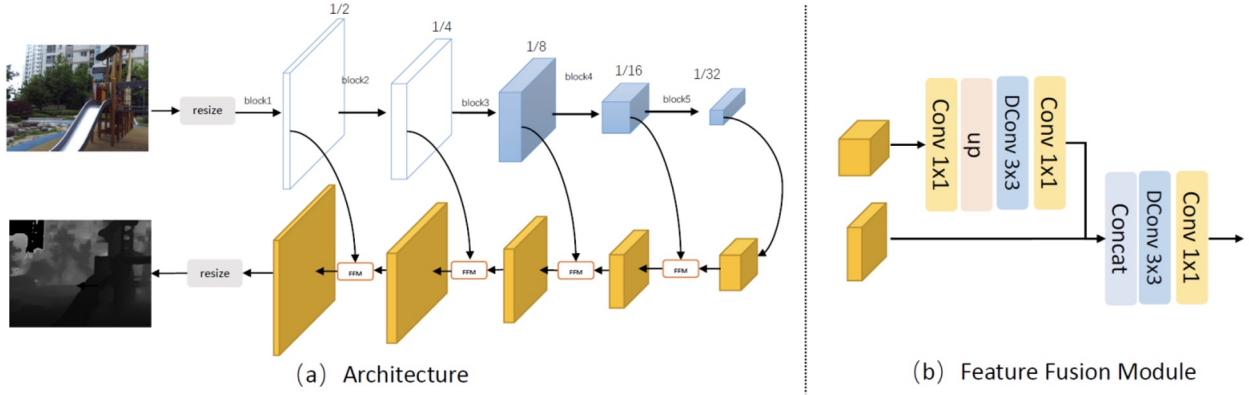


Figure 3. The model architecture and the structure of the Feature Fusion Module (FFM) proposed by team Tencent GY-Lab.

- MobileNet v3 기반 encoder를 적용한 U-Net like architecture
- 각 output block은 decoder feature를 concate 하는 Feature Fusion Module (FFM)을 적용하여 성능개선
- ViT-Large 기반 teacher network로 knowledge distillation 적용
- 500epoch
- PyTorch → ONNX → TFLite로 변환

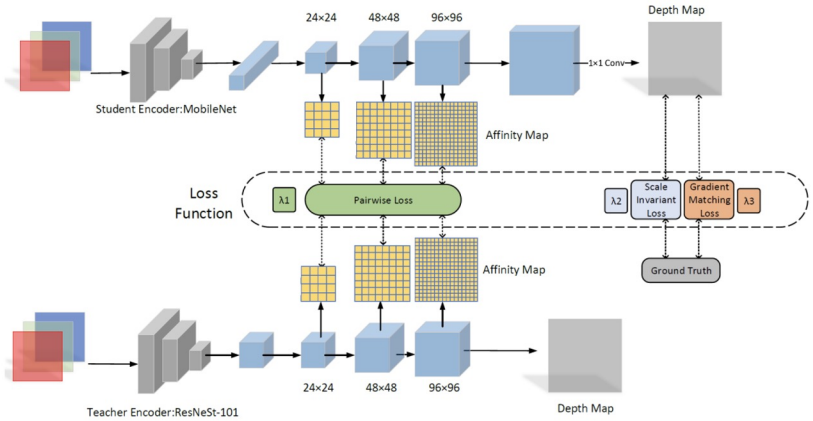


Figure 4. An overview of the knowledge distillation strategy used by team SMART.

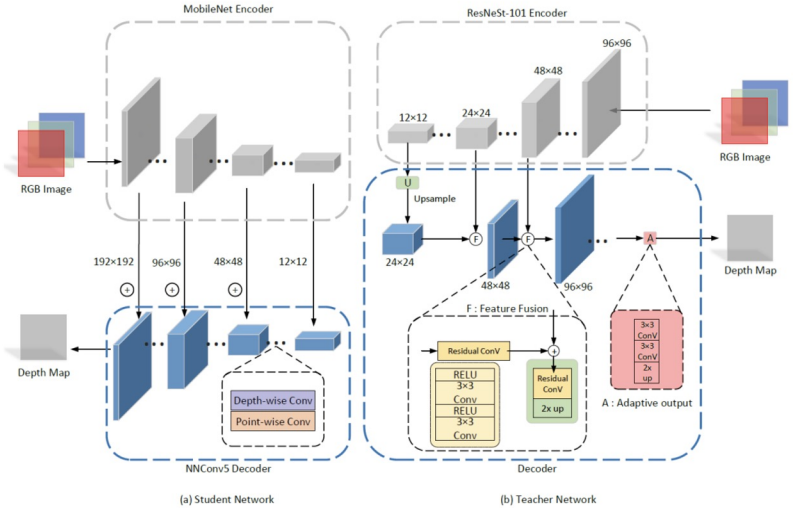


Figure 5. The architecture of the student and teacher models developed by team SMART.

- FastDepth architecture 사용 (MobileNet v1 backbone)
- ResNeSt-101 기반 Teacher Network를 먼저 학습시킨 후 knowledge distillation.
- 100epoch

https://arxiv.org/pdf/2112.13492.pdf

Vision Transformer for Small-Size Datasets

Seung Hoon Lee
Inha University
Incheon, South Korea
aanna0701@gmail.com

Seunghyun Lee
Inha University
Incheon, South Korea
lsh910703@gmail.com

Byung Cheol Song
Inha University
Incheon, South Korea
bcsong@inha.ac.kr

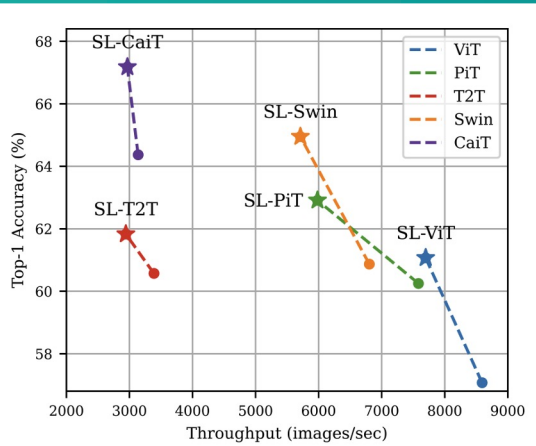


Figure 1. Effect of the proposed method on the overall performance when learning Tiny-ImageNet from scratch. Throughput refers to how many images can be processed per unit of time. The stars and dots indicate after and before the proposed method are applied, respectively.

This paper proposes Shifted Patch Tokenization (SPT) and Locality Self-Attention (LSA)

- 최근 ViT 는 CNN 성능을 능가하고 있지만,
  ViT 의 높은 성능은 JTF-300M 과 같은 large-size dataset 의 pre-training 의 영향이라고 여겨짐.
- 또한 large dataset 에 대한 의존성은 ViT의 low locality inductive bias 때문.
  - ViT inductive bias 설명 : https://robot-vision-develop-story.tistory.com/29

- 본 논문에서는 ViT의 locality inductive bias 문제를 해결하고, small-size dataset 에서 from scratch 학습을 사용함.
- 추가적으로, SPT와 LSA 는 다양한 ViT 에 적용 가능한 add-on 모듈.

- 성능
  - SPT와 LSA 가 적용된 ViT 의 성능이 Tiny-ImageNet 에서 평균적으로 2.96% 향상됨.
  - 특히, Swin Transformer 는 4.08% 향상됨.

This paper proposes Shifted Patch Tokenization (SPT) and Locality inductive bias

- 최근 ViT 는 CNN 성능을 능가하고 있지만,
  ViT 의 높은 성능은 JTF-300M 과 같은 large-size dataset 의 pre-training 의 영향이라고 여겨짐.
- 또한 large dataset 에 대한 의존성은 ViT의 low locality inductive bias 때문.
  - ViT inductive bias 설명 : https://robot-vision-develop-story.tistory.com/29

그럼 여기서 Inductive biase는 뭘까요? (하단에 참조한 자료들을 참고하였습니다.)

"위키피디아"와 박민호 님의 Blog에서는 이런 말을 합니다.
기계학습에서의 inductive bias는, 학습 모델이 **지금까지 만나보지 못했던 상황**에서 정확한 예측을 하기 위해 사용하는 **추가적인 가정**을 의미합니다.
(The **inductive bias** (also known as **learning bias**) of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.)

음 머신러닝이 생소하신 분들은 다소 어려우실 수 있습니다.
쉽게 설명드리자면 **추가적인 가정(우리가 풀려는 문제에 대한 정보를 모델에 적용)** 을 적용한다 생각하시면 됩니다.

CS229 5장에서 GDA(생성 모델)과 Logistic regression(판별 모델)에 대해서 비교하는 내용이 있습니다.(이진 분류 문제)

앤드류 응 교수님은 이 둘을 비교할 때 GDA는 Data가 Gaussian Distribution(Exponetial Family)를 따를 것이다. 라는 추가적인 Stronger Assumption이 맞다면 Weaker Assumption인 Logistic regression보다 모델이 문제를 더 잘 풀 것이다.

반면 Assumption이 틀린 경우(데이터가 푸아송 분포이거나)에는 Logistic보다 동작을 잘 못할 것이다.

그렇다면 Transformer는 왜 Inductive bias(가정)이 CNN보다 약한가?

이 내용은 Stack overflow에 있는 자료를 참고하였는데요
이런 Table이 있습니다.

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

Table 1: Various relational inductive biases in standard deep learning components. See also Section 2.

여기서 Conv 같은 경우 Locality에 대한 가정이 있는데요 우리가 Convolution을 처리하는 과정을 보면 Local(지역적인) 영역에서 Spatial한 정보를 잘 뽑아내는걸 알 수 있습니다. 이러한 모델 설계자체가 가정이들어간 거죠
"Vision Task는 지역적으로 정보를 얻을게 많다!"

RNN을 또 예로 들어볼까요? RNN 또한 추가적인 가정이 들어간 모델입니다. 시계열적 나열에서 가까운 애들한테 더 많은 영향을 주거나 받을 수 있게 되어있죠.

그래서 CNN이나 RNN은 한계를 지닙니다.
예를 들어 CNN은 Global한 영역에 대한 처리가 어렵기 때문에 Deformable 등의 연구로 Receptive field를 넓히려고 연구를 했죠, RNN 또한 긴 문장에 대해서 it과 같이 앞문장에 특정 주어를 지칭하는 문제에서 어려웠습니다.
(둘 다 지역적인 부분에 가정을 두다 보니 전체적인 큰 영역에서는 취약한 모습을 보임)

그렇지만 Transformer는 Positional Embedding이나 Self-Attention 메커니즘을 통해 모든 정보는 활용하지만 추가적인 가정(inductive bias)이 부족하다는 것입니다. 그래서 Robust하게 동작할 수 있지만 많은 양의 데이터가 필요하다는 것입니다.

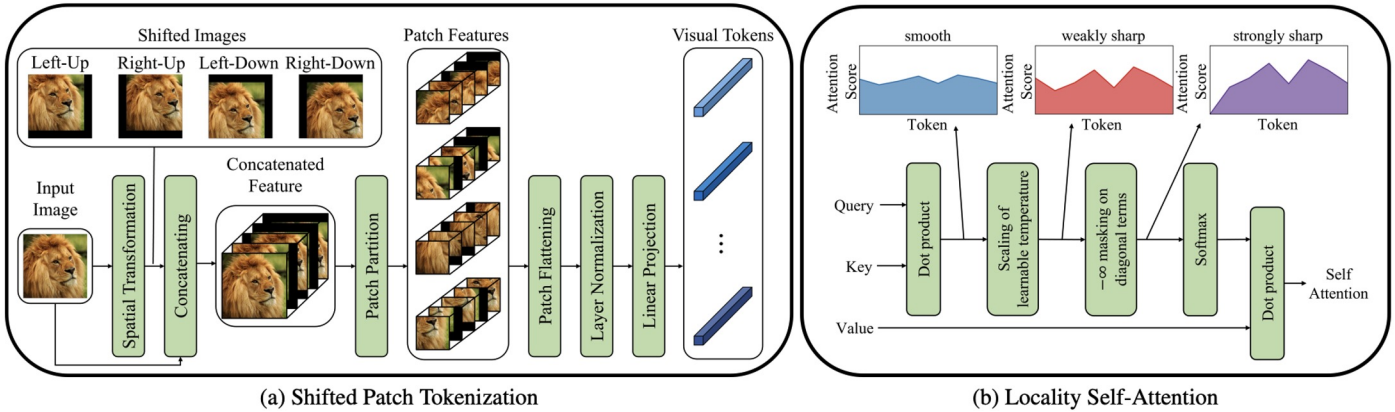


Figure 2. Architectures of the proposed SPT and LSA.

Architectures of Shifted Patch Tokenization (SPT)

- **input image 를 여러 방향으로 spatially shift 한 후, concatenates 한다.**
  - shifted features 를 input image 와 동일한 크기로 crop 한 후, concatenated.
- **Concatenated features 들은 non-overlapping patches 로 쪼개진다.**
- 그리고 visual token 으로 embedding 하기 위해, 3 단계가 순차적으로 진행 됨.
  - **patch flattening, layer normalization , linear projection**
- 결과적으로 SPT 는 더 많은 spatial information 을 visual token 으로 embed 시킬 수 있고, ViT 의 locality inductive bias 를 향상시킨다.

3.2.2 Patch Embedding Layer

This section describes how to use SPT as a patch embedding layer. We concatenate a class token to visual tokens and then add positional embedding. Here the class token is the token with representation information of the entire image, and the positional embedding gives positional information to the visual tokens. If a class token is not used, only positional embedding is added to the output of SPT. How to apply the SPT to the patch embedding layer is formulated as follows:

$$\mathrm{S}_{pe}(\mathbf{x}) = \begin{cases} [\mathbf{x}_{cls}; \mathrm{S}(\mathbf{x})] + \boldsymbol{E}_{pos} & \text{if } \mathbf{x}_{cls} \text{ exist} \\ \mathrm{S}(\mathbf{x}) + \boldsymbol{E}_{pos} & \text{otherwise} \end{cases} \tag{7}$$

where $\mathbf{x}_{cls} \in \mathbb{R}^{d_S}$ is a class token and $\boldsymbol{E}_{pos} \in \mathbb{R}^{(N+1)\times d_S}$ is the learnable positional embedding. Also, N is the number of embedded tokens in Eq. 6.

3.2.3 Pooling Layer

Tokenization is the process of embedding 3D-tensor features into 2D-matrix features. For example, it embeds $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ into $\mathbf{y} = \mathcal{T}(\mathbf{x}) \in \mathbb{R}^{N\times d}$. Since $N = HW/P^2$, the spatial size of the 3D feature is reduced by P^2 through the tokenization process. So, if tokenization is used as a pooling layer, the number of visual tokens can be reduced. Therefore, we propose to use SPT as a pooling layer as follows: First, class tokens and visual tokens are separated, and visual tokens in the form of 2D-matrix are reshaped into 3D-tensor with spatial structure, i.e., $\mathcal{R} : \mathbb{R}^{N\times d} \rightarrow \mathbb{R}^{(H/P)\times(W/P)\times d}$. Then, if the SPT of Eq. 6 is applied, new visual tokens with a reduced number of tokens are embedded. Finally, the linearly projected class token is connected with the embedded visual tokens. If there is no class token, only $\mathcal{R}$ is applied before the output of SPT. The whole process is formulated as Eq. 8:

$$\mathrm{S}_{pool}(\mathbf{y}) = \begin{cases} [\mathbf{x}_{cls}\boldsymbol{E}_{cls}; \mathrm{S}(\mathcal{R}(\mathbf{y}))] & \text{if } \mathbf{x}_{cls} \text{ exist} \\ \mathrm{S}(\mathcal{R}(\mathbf{y})) & \text{otherwise} \end{cases} \tag{8}$$

Architectures of Shifted Patch Tokenization (SPT)

- **input image 를 여러 방향으로 spatially shift 한 후, concatenates 한다.**
  - shifted features 를 input image 와 동일한 크기로 crop 한 후, concatenated.
- **Concatenated features 들은 non-overlapping patches 로 쪼개진다.**
- 그리고 visual token 으로 embedding 하기 위해, 3 단계가 순차적으로 진행 됨.
  - **patch flattening, layer normalization , linear projection**
- 결과적으로 SPT 는 더 많은 spatial information 을 visual token 으로 embed 시킬 수 있고, ViT 의 locality inductive bias 를 향상시킨다.

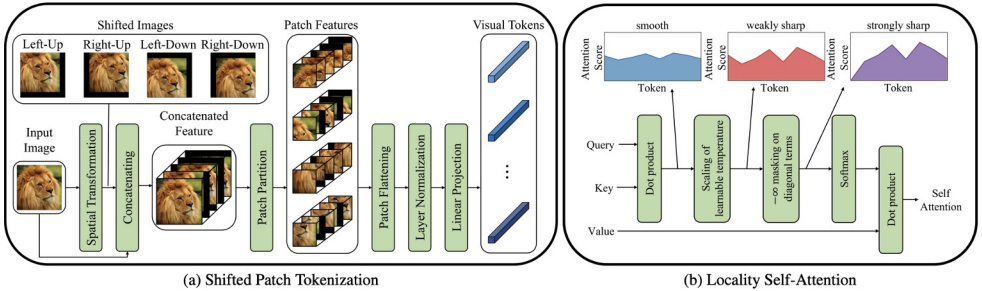


Figure 2. Architectures of the proposed SPT and LSA.

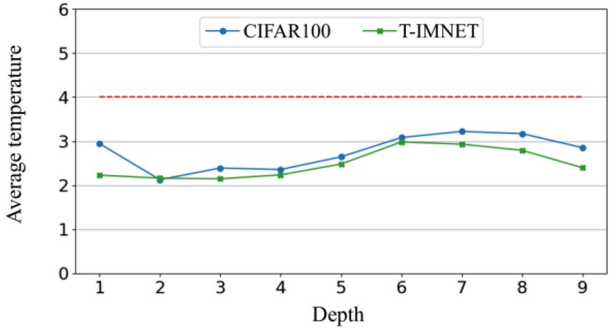


Figure 3. The learned temperature according to depth. Here, the red dashed line indicates the temperature of standard ViT.

Locality Self-Attention (LSA)

- Core idea 1) diagonal masking 2) learnable temperature scaling

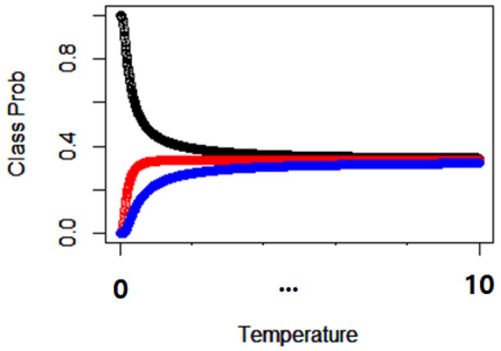
1) Diagonal (대각선의) Masking

- ViT가 own tokens 보다 other tokens 에 focus on 하여 attention 하도록 함.
- softmax 작업에서 기본적으로 self-token relations 을 배제함으로써, inter-token relations 에 더 큰 score 를 주는 역할.

1) Learnable Temperature Scaling

- 일반적으로 softmax function 는 output distribution 을 temperature scaling 을 통해서 smoothness 를 조절.
- **LSA는 softmax function 의 temperature parameters 를 학습함으로써, attention scores 의 분포를 sharpens 하게 함.**
- (figure) softmax temperature 가 학습 파라미터로 사용되었을 때의 평균 temperature (Depth 에 따름)

$$Softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \Rightarrow Softmax(z_i) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$



Locality Self-Attention (LSA)

2) Learnable Temperature Scaling

- ○ 일반적으로 low temperature 가 score distribution 을 sharpens 하게 함.
- ○ 따라서 학습된 파라미터들은 attention scores 를 sharpens 하게 함. (더 작아졌으니까)

- ○ 즉, LSA 는 attention score distribution 의 smoothing problem 을 해결.
- ○ (Figure 4)
  - ■ T : only learnable temperature scaling is applied to ViTs
  - ■ M : only diagonal masking is applied is applied to ViTs
  - ■ L : entire LSA is applied to ViT
  - ■ LSA module 사용했을 때, 일반적인 ViT 보다 0.5 크다.

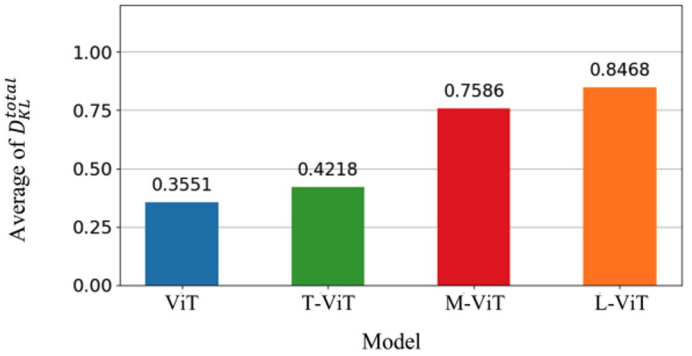


Figure 4. Kullback–Leibler Divergence (KLD) of attention score distributions. The average KLDs were measured on Tiny-ImageNet.

Experiment

- Quantitative result (정량적 평가)
  - (Table 2) Proposed method was applied to ViTs. SL : boah SPT and LSA were applied
  - (Table 3) Training a mid-size dataset ImageNet from scratch
    - In ViT, SPT was applied only to patch embeddings,
      and in PiT and Swin, SPT was applied to both patch embedding and pooling layers.

Table 2. Top-1 accuracy comparison of different models on small-size datasets.

MODEL	THROUGHPUT (images/sec)	FLOPs (M)	PARAMS (M)	CIFAR10	CIFAR100	SVHN	T-ImageNet
ResNet 56	4295	506.2	0.9	95.70	76.36	97.73	58.77
ResNet 110	2143	1020.0	1.7	**96.37**	79.86	97.85	62.96
EfficientNet B0	4078	123.9	3.7	94.66	76.04	97.22	66.79
ViT	8593	189.8	2.8	93.58	73.81	97.82	57.07
SL-ViT	7697	199.2	2.9	94.53	76.92	97.79	61.07
T2T	3388	643.0	6.7	95.30	77.00	97.90	60.57
SL-T2T	2943	671.4	7.1	95.57	77.36	97.91	61.83
CaiT	3138	613.8	9.1	94.91	76.89	98.13	64.37
SL-CaiT	2967	623.3	9.2	95.81	**80.32**	**98.28**	**67.18**
PiT	7583	279.2	7.1	94.24	74.99	97.83	60.25
SL-PiT w/o S_{pool}	6632	280.4	7.1	94.96	77.08	97.94	60.31
SL-PiT w/ S_{pool}	5981	322.9	8.7	95.88	79.00	97.93	62.91
Swin	6804	242.3	7.1	94.46	76.87	97.72	60.87
SL-Swin w/o S_{pool}	6384	247.0	7.1	95.30	78.13	97.88	62.70
SL-Swin w/ S_{pool}	5711	284.9	10.2	95.93	79.99	97.92	64.95

Table 3. Top-1 accuracy (%) of the proposed method on ImageNet dataset.

MODEL	TOP-1 ACCURACY (%)
ViT	69.95
SL-ViT	71.55 (+1.60)
PiT	75.58
SL-PiT	77.02 (+1.44)
Swin	79.95
SL-Swin	81.01 (+1.06)

Experiment

- Qualitative result (정성적 평가)
  - (Figure 5) Attention scores of the final class token when SPR and LSA were applied to various ViTs.

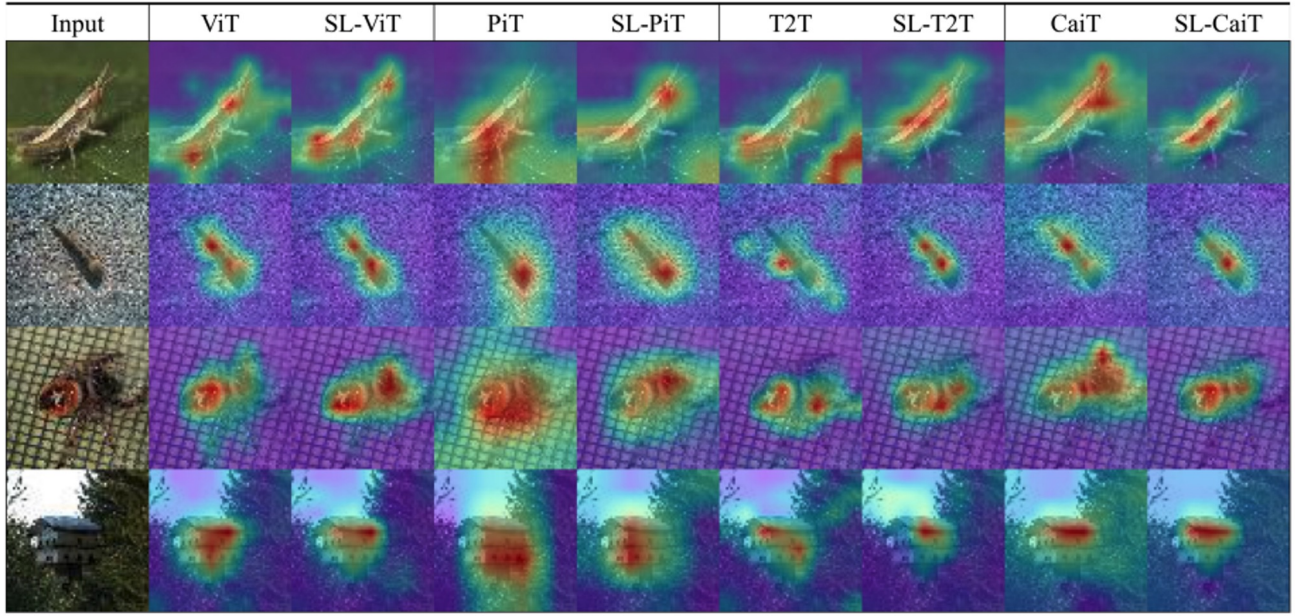
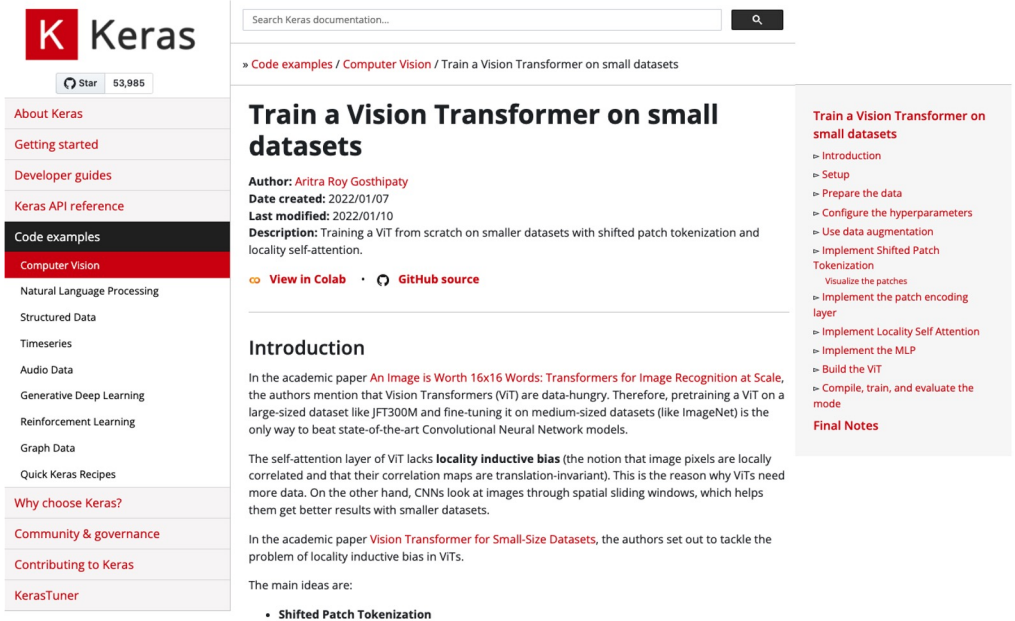
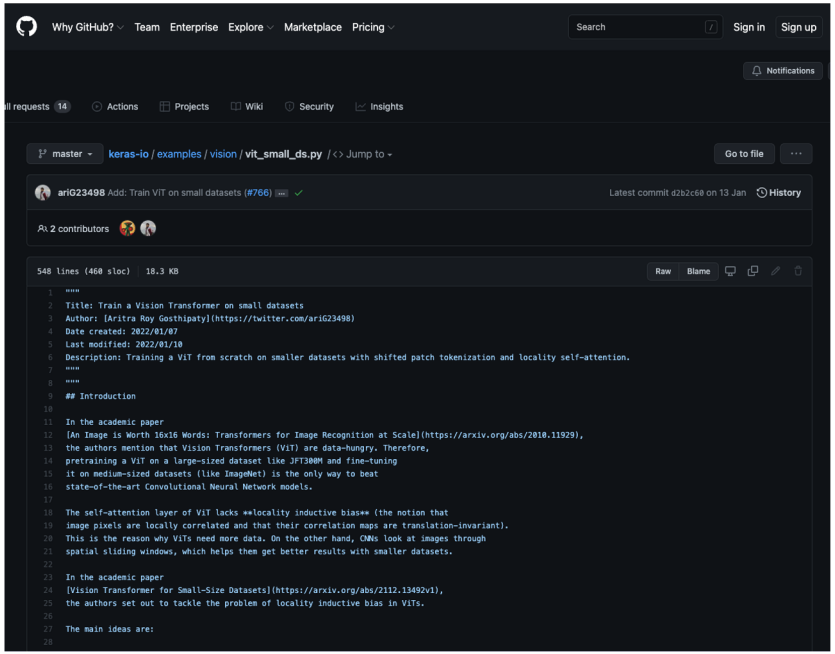


Figure 5. Visualization of attention scores of final class tokens.

This paper proposes Shifted Patch Tokenization (SPT) and Locality Self-Attention (LSA)

- https://keras.io/examples/vision/vit_small_ds/





https://arxiv.org/pdf/2201.09792v1.pdf

~~Convolutions Attention MLPs~~
Patches Are All You Need?

Asher Trockman, J. Zico Kolter[1]
Carnegie Mellon University and [1]Bosch Center for AI

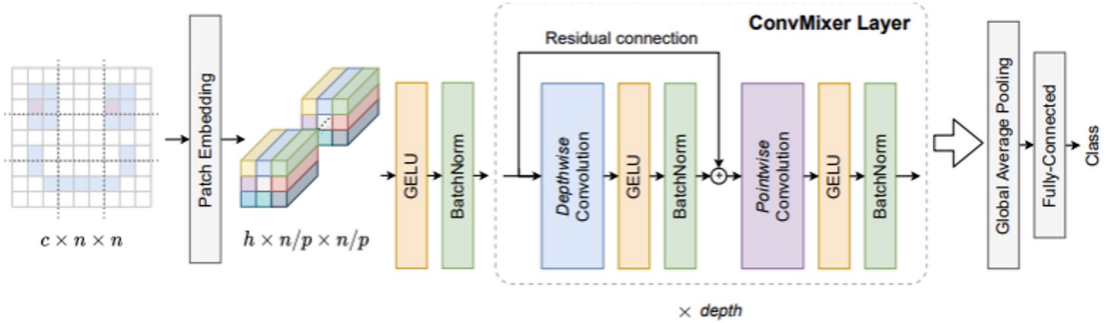


Figure 2: ConvMixer uses "tensor layout" patch embeddings to preserve locality, and then applies d copies of a simple fully-convolutional block consisting of *large-kernel* depthwise convolution followed by pointwise convolution, before finishing with global pooling and a simple linear classifier.

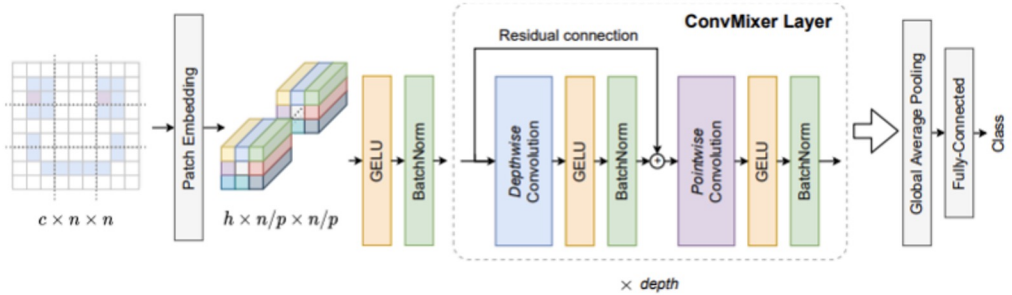
〈Abstract〉

- Transformers의 self-attention 레이어의 quadratic runtime으로 인해 ViT는 더 큰 image sizes에 적용하기 위해 image의 작은 영역을 single input features으로 그룹화하는 patch embedding을 사용 ⇒ **ViT의 성능은 본질적으로 more powerful Transformer architecture 때문? 아니면 부분적으로 패치를 input representation으로 사용하기 때문?**
- 본 논문에서는 부분적으로 패치를 input representation으로 사용하기 때문이다 라고 생각하고 이것에 대한 몇 가지 증거를 제시.
- MLP-Mixer과 비슷한 ConvMixer를 제안
  - 직접적으로 patch에 작용해 모든 레이어에 걸쳐 equal-resolution-and-size representation를 유지
  - channel-wise mixing과 spatial mixing을 구분 (Depthwise Convolution, Pointwise Convolution 나눔)
  - standard convolution으로만 작동
  - 단순함에도 불구하고 ConvMixer가 ViT, MLP-Mixer 및 유사한 parameter 수 및 dataset sixe에 대해 성능을 능가할 뿐만 아니라 ResNet과 같은 기존 비전 model보다 성능이 우수
- github: https://github.com/locuslab/convmixer

https://arxiv.org/pdf/2201.09792v1.pdf

- **Isotropic architecture**
  - ViT는 네트워크에 걸쳐 equal size와 shape를 사용하는, isotropic architecture의 제시. 이후 MLP-Mixer, ResMLP등이 나왔지만,
  - ConvMixer에의 조사가 제시했듯 이러한 작업들이 self-attention, MLP와 같은 새로운 operation의 효과와 patch embedding의 사용과 그로 인한 isotropic architecture의 효과를 융합해서 생각해버렸을거라 예상 (저자들이 예상하기에)
  - self-attention, MLP등이 중요하다기보다 patch embedding과 isotropic architecture이 중요한데 그 점을 무시하지 않았나 지적
- **Patches aren't all you need.**
  - 여러 paper에서 ViT의 performance를 standard patch embedding을 다른 것으로 바꾸면서 향상
  - 예를 들어 인접한 patch embedding을 combine하는 연구가 존재
  - 다만 이러한 연구는 locality inductive bias의 효과와 patch embedding의 효과를 함부로 합쳐 버렸을 가능성이 있다고 저자들이 판단
  - 그래서 patches의 쓰임새에 집중했다.
- **CNNs meet ViTs**
  - 지금까지 CNN을 ViT에 합치려는 노력은 계속되어 왔다.
  - self-attention이 convolution을 emulate할 수 있다는 연구도 있었고, attention이 convolution 비슷하게 작동하도록 initialized되거나 regularized 될 수 있다는 연구도 있었고, 단순히 convolution operation을 transformer에 더하려는 연구도 있었음
  - 그리고 pyramid-shaped convolutional network를 본떠 transformer에 downsampling을 포함한 것까지 있었으나
  - 다들 나름의 성과가 있었으나 본 연구와는 방향이 다르다.

https://arxiv.org/pdf/2201.09792v1.pdf




```
def ConvMixer(h, depth, kernel_size=9, patch_size=7, n_classes=1000):
    Seq, ActBn = nn.Sequential, lambda x: Seq(x, nn.GELU(), nn.BatchNorm2d(h))
    Residual = type('Residual', (Seq,), {'forward': lambda self, x: self[0](x) + x})
    return Seq(ActBn(nn.Conv2d(3, h, patch_size, stride=patch_size)),
        *[Seq(Residual(ActBn(nn.Conv2d(h, h, kernel_size, groups=h, padding="same"))),
          ActBn(nn.Conv2d(h, h, 1))) for i in range(depth)],
         nn.AdaptiveAvgPool2d((1,1)), nn.Flatten(), nn.Linear(h, n_classes))
```

Figure 3: Implementation of ConvMixer in PyTorch; see Appendix D for more implementations.

1. Patches로 나눔 ⇒ 이 때 3 channel → h channel, stride & kernel & size patch_size인 convolution 연산을 사용
2. depth번만큼, Depthwise Convolution+Pointwise Convolution을 반복
    a. Depthwise Convolution은 spatial location의 mix
    b. Pointwise Convolution은 channel location의 mix를 하기 위해 시행
    c. 중간중간 Activation+Batch Norm 넣어줌
3. Global Average Pooling후, (1, 1, h) 벡터를 linear transform해 classification 값을 얻음

tures more performant on smaller data sets. Importantly, this is despite the fact that *we did not design our experiments to maximize accuracy nor speed, in contrast to the models we compared against. Our results suggest that,* at least to some extent, the *patch representation itself* may be a critical component to the "superior" performance of newer architectures like Vision Transformers. While these

https://arxiv.org/pdf/2201.09792v1.pdf

- patch size: *p*, embedding dimension: *h (비교군에서 term)* ⇒ convolution with *cin* input channels: *h*, output channels, kernel size: *p*, and stride: *p (ours term)*

$$z_0 = \text{BN}\left(\sigma\{\texttt{Conv}_{c_{\text{in}} \to h}(X, \texttt{stride}{=}p, \texttt{kernel\_size}{=}p)\}\right) \quad (1)$$

- The ConvMixer block: consists of depthwise convolution (i.e., grouped convolution with groups equal to the number of channels, h) followed by pointwise (i.e., kernel size 1 × 1) convolution.

⇒ ConvMixer는 depthwise convolution을 위해 **unusually large kernel sizes를 사용하는 것이 제일 좋은 결과** (used convolutions with an unusually large kernel size to mix distant spatial locations)

Each of the convolutions is followed by an activation and post-activation BatchNorm

$$z'_l = \text{BN}\left(\sigma\{\texttt{ConvDepthwise}(z_{l-1})\}\right) + z_{l-1} \quad (2)$$

$$z_{l+1} = \text{BN}\left(\sigma\{\texttt{ConvPointwise}(z'_l)\}\right) \quad (3)$$

After many applications of this block, we perform global pooling to get a feature vector of size h, which we pass to a softmax classifier.

https://arxiv.org/pdf/2201.09792v1.pdf

Current "Most Interesting" **ConvMixer** Configurations *vs.* Other Simple Models							
Network	Patch Size	Kernel Size	# Params ($\times 10^6$)	Throughput (img/sec)	Act. Fn.	# Epochs	ImNet top-1 (%)
ConvMixer-1536/20	7	9	51.6	134	G	150	81.37
ConvMixer-768/32	7	7	21.1	206	R	300	80.16
ResNet-152	–	3	60.2	828	R	150	79.64
DeiT-B	16	–	86	792	G	300	81.8
ResMLP-B24/8	8	–	129	181	G	400	81.0

Table 1: Models trained and evaluated on 224 × 224 ImageNet-1k only. See more in Appendix A.

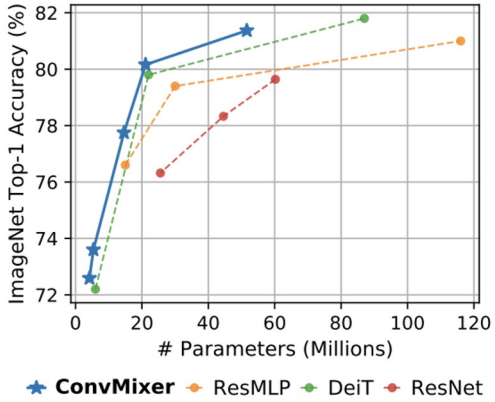


Figure 1: Accuracy *vs.* parameters, trained and evaluated on ImageNet-1k.

- timm framework를 이용해 RandAugment, mixup, CutMix, random erasing, gradient norm clipping을 default timm augmentation에 더해 사용
- **그런데 hyperparameter tuning을 안 함..!**
- **kernel size가 클 때 잘 되었고, smaller patches일 때(patch로 나누지 않고 넣을 때와 비슷해질 때) 더 결과가 잘 나온다고 한다**
- larger patches에서 결과가 잘 나오려면 deeper해야 한다고 생각한다고 한다. (저자의 뇌피셜)
- parameter 수를 비슷하게 해서 훈련시켰을 때, DeiT와 ResMLP는 ConvMixer와 다르게 hyperparameter tuning이 되어 있음에도(특히 ResNet은 tuning하는데 지금까지 엄청난 자원이 소모되었음에도), epoch도 2배임에도 ConvMixer을 0.2%밖에 outperform하지 못했다고 자랑질
- CIFAR-10 Experiments: 0.7M의 적은 parameter로 CIFAR-10에서 96% 넘는 정확도를 얻었다. **이는 Convolutional inductive bias의 data efficiency를 증명**

https://arxiv.org/pdf/2201.09792v1.pdf

Comparison with other simple models trained on **ImageNet-1k only** with input size 224.

Network	Patch Size	Kernel Size	# Params ($\times 10^6$)	Throughput (img/sec)	Act. Fn.	# Epochs	ImNet top-1 (%)
ConvMixer-1536/20★	7	9	51.6	134	G	150	82.20
ConvMixer-1536/20 ●	7	9	51.6	134	G	150	81.37
ConvMixer-1536/20★	7	3	49.4	246	G	150	81.60
ConvMixer-1536/20	7	3	49.4	246	G	150	80.43
ConvMixer-1536/20	14	9	52.3	538	G	150	78.92
ConvMixer-1536/24★	14	9	62.3	447	G	150	80.21
ConvMixer-768/32 ●	7	7	21.1	206	R	300	80.16
ConvMixer-1024/16	7	9	19.4	244	G	100	79.45
ConvMixer-1024/12	7	8	14.6	358	G	90	77.75
ConvMixer-512/16	7	8	5.4	599	G	90	73.76
ConvMixer-512/12 ●	7	8	4.2	798	G	90	72.59
ConvMixer-768/32	14	3	20.2	1235	R	300	74.93
ConvMixer-1024/20 ●	14	9	24.4	750	G	150	76.94
ResNet-152 ●	–	3	60.2	828	R	150	79.64
ResNet-101 ●	–	3	44.6	1187	R	150	78.33
ResNet-50	–	3	25.6	1739	R	150	76.32
DeiT-B†	7	–	86.7	83	G	–	–
DeiT-S†	7	–	22.1	174	G	–	–
DeiT-Ti†	7	–	5.7	336	G	–	–
DeiT-B ●	16	–	86	792	G	300	81.8
DeiT-S ●	16	–	22	1610	G	300	79.8
DeiT-Ti ●	16	–	5.7	2603	G	300	72.2
ResMLP-S12/8 ●	8	–	22.1	872	G	400	79.1
ResMLP-B24/8 ●	8	–	129	181	G	400	81.0
ResMLP-B24	16	–	116	1597	G	400	81.0
Swin-S ●	4	–	50	576	G	300	83.0
Swin-T ●	4	–	29	878	G	300	81.3
ViT-B/16 ●	16	–	86	789	G	300	77.9
Mixer-B/16 ●	16	–	59	1025	G	300	76.44
Isotropic MobileNetv3 ●	8	3	20	–	R	–	80.6
Isotropic MobileNetv3 ●	16	3	20	–	R	–	77.6

Table 2: Throughputs measured on an RTX8000 GPU using batch size 64 and fp16. ConvMixers and ResNets trained ourselves. Other statistics: DeiT (Touvron et al., 2020), ResMLP (Touvron et al., 2021a), Swin (Liu et al., 2021b), ViT (Dosovitskiy et al., 2020), MLP-Mixer (Tolstikhin et al., 2021), Isotropic MobileNets (Sandler et al., 2019). We think models with matching colored dots (●) are informative to compare with each other. †Throughput tested, but not trained. Activations: **R**eLU, **G**ELU. ★Using new, better regularization hyperparameters based on Wightman et al. (2021)'s A1 procedure.

Ablation of ConvMixer-256/8 on CIFAR-10

Ablation	CIFAR-10 Acc. (%)
Baseline	95.88
– Residual in Eq. 2	95.57
+ Residual in Eq. 3	94.78
BatchNorm → LayerNorm	94.44
GELU → ReLU	95.51
– Mixup and CutMix	95.92
– Random Erasing	95.24
– RandAug	92.86
– Random Scaling	86.24
– Gradient Norm Clipping	86.33

Table 3: Small ablation study of training a ConvMixer-256/8 on CIFAR-10.

End of the Document