Vision AI

2021 arXiv Trends

2021-11

hutom

no.	Paper Title	Correspondence	h-index
1	Understanding and Improving Robustness of Vision Transformers through Patch-based Negative Augmentation	Xuezhi Wang	13
2	NeRV: Neural Representations for Videos	Abhinav Shrivastava	20
3	Learning in High Dimension Always Amounts to Extrapolation	Yann LeCun	129
4	TorchXRayVision: A library of chest X-ray datasets and models	Hadrien Bertrand	6

no.	Paper Title	research group
5	Palette: Image-to-Image Diffusion Models	Google Research
6	Masked Autoencoders Are Scalable Vision Learners	Facebook AI Research (FAIR)
7	Gradients are Not All You Need	Google Research, Brain Team
8	Data Augmentation Can Improve Robustness	DeepMind
9	A Survey on Vision Transformer	

Understanding and Improving Robustness of Vision Transformers through Patch-based Negative Augmentation

Yao Qin Chiyuan Zhang Ting Chen
Balaji Lakshminarayanan Alex Beutel Xuezhi Wang
Google Research

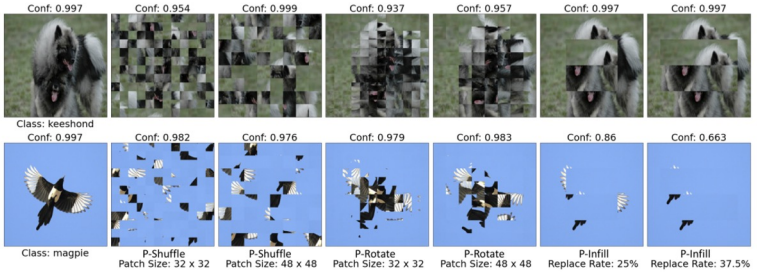


Figure 1: Patch-based transformations largely destroy images to be unrecognizable to humans whereas ViT recognizes them as the original class (e.g., keeshond or magpie) with high confidence. Visualization of patch-based transformations. On the top of each image, we display the predicted confidence score of ViT-B/16 pretrained on ImageNet-21k and finetuned on ImageNet-1k.

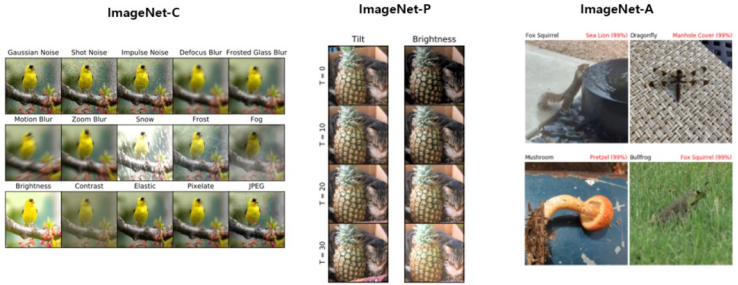
- **Research Group: Google**

- **ViT = Patch 기반 Model**
  - ViT가 패치 기반 Augmentation에 매우 둔감을 발견.
  - ViT가 사실 Unrecognition by humans image에 대해서도 인식을 잘 함.
  - 이러한 특징으로 Vision에서 높은 accuray달성, but break down under distribution shifts.

- **can training the model to rely less on these features improve ViT robustness and out-of-distribution performance?**
  (의존이 덜 하도록 모델을 교육하면 견고한 ViT를 만드는데 도움이 되는가?)
  - **Patch-based Negative Augmentation을 사용할 경우 ViT의 Robustness 증가 (ImageNet 기반 robustness benchmark)**
  - **Positive augmentation(traditional)을 상호보완하고 성능이 더 향상**

Understanding and Improving Robustness of Vision Transformers through Patch-based Negative Augmentation

- **PRELIMINARIES**
  - **Vision Transformer**
  - **Model Variants** // ViT-B/16 == 'Base, 16x16' // ImageNet-21k or 1k (pretrain) -> ImageNet-1k(finetune),
  - **Robustness Bechmarks** // ImageNet-A, ImageNet-C(widely used), ImageNet-R



- *[leftmargin=20pt]*
- ***Patch-based Shuffle (P-Shuffle)**: we randomly shuffle the input image patches to change their positions[1].*
- ***Patch-based Rotate (P-Rotate)**: we randomly select a rotation degree from the set $\Omega = \{0°, 90°, 180°, 270°\}$ and rotate each image patch independently.*
- ***Patch-based Infill (P-Infill)**: we replace the image patches in the center region of an image with the patches on the image boundary[2].*

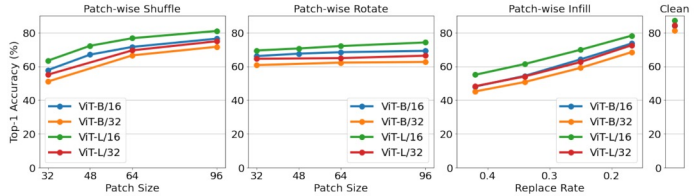


Figure 2: ViTs can rely on features surviving patch-based transformations to maintain a high accuracy, even after images have been heavily transformed to be largely unrecognizable. Top-1 accuracy of ViT models when tested on patch-based transformed images using the semantic class of the corresponding clean image as ground-truth. The test accuracy on ImageNet-1k validation set is shown on the right. All ViT models are pre-trained on ImageNet-21k and fine-tuned on ImageNet-1k.

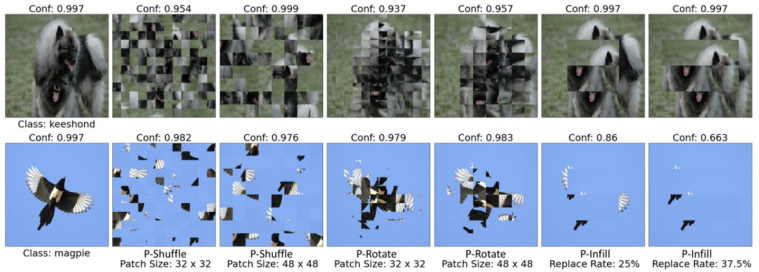


Figure 1: Patch-based transformations largely destroy images to be unrecognizable to humans whereas ViT recognizes them as the original class (e.g., keeshond or magpie) with high confidence. Visualization of patch-based transformations. On the top of each image, we display the predicted confidence score of ViT-B/16 pretrained on ImageNet-21k and finetuned on ImageNet-1k.

- **One Key Contributions are Follws:**

1. **Understanding Non-Robust Features in ViT**

   - ViT는 사람이 알아보지 못할 정도의 non-robust features(patch-based transformation) 에서도 살아남을 수 있다.

2. **Modeling**

   - semantic-preserving(“positive”) augmentations과 상호보완적인 non-robust feature를 줄때, 본 논문에서 제안한 negatively augmented view 방식이 효과적이다.

3. **Improved Robustness of ViT**

   - Negative Augmented views 가 ViT’s robustness and complementary to “positive” augmentation.

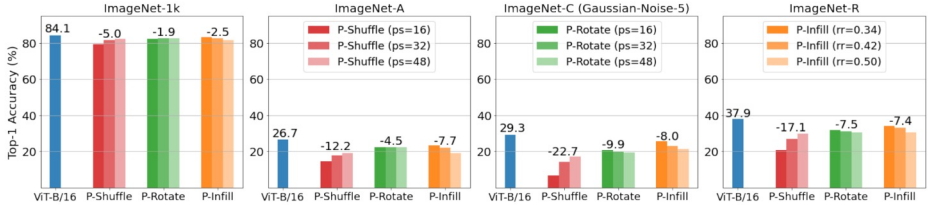


Figure 3: Features preserved in patch-based transformations are useful but non-robust as training ViT on them impedes robustness. Top-1 Accuracy (%) on ImageNet-1k validation set and ImageNet robustness datasets: ImageNet-A, ImageNet-C, ImageNet-R. The baseline model is ViT-B/16 in (Dosovitskiy et al., 2021) trained on original images. Other models are trained on patch-based transformed images, e.g., “P-Shuffle” stands for a ViT-B/16 model trained on patch-based shuffled images. Numbers above the bars are either accuracy (e.g., ViT-B/16) or the *max* accuracy difference between each model family and the baseline ViT-B/16. The patch size in P-Shuffle and P-Rotate and replacement ratio in P-Infill is denoted by “ps” and “rr” respectively.

- 가장 왼쪽표를 보면 해당 사람이 알아보지 못하는 이러한 patch-based trnsformation에서도 잘 살아 남음

- **Improving Robustness of Vision Tranformer (loss design)**

$$\mathcal{L}_{ce}(\mathbb{B};\boldsymbol{\theta}) + \lambda \cdot \mathcal{L}_{neg}(\mathbb{B},\tilde{\mathbb{B}};\boldsymbol{\theta}),$$

  - lambda coefficient 를 이용해 중요한 clean training data과 negative views data 사이의 balancin

1. **Label Space: uniform loss**

$$\mathcal{L}_{neg}(\tilde{\mathbb{B}};\boldsymbol{\theta}) = -\frac{1}{|\tilde{\mathbb{B}}|}\sum\nolimits_{(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}})\in\tilde{\mathbb{B}}} \tilde{\boldsymbol{y}} \log \mathrm{softmax}(f(\tilde{\boldsymbol{x}};\boldsymbol{\theta})),$$

where $\tilde{\boldsymbol{y}}$ denotes the uniform distribution: $\tilde{y}_k = \frac{1}{K}$ where K is the total number of classes.

1. **Logit space: l2 loss**

$$\mathcal{L}_{neg}(\mathbb{B},\tilde{\mathbb{B}};\boldsymbol{\theta}) = -\frac{1}{|\tilde{\mathbb{B}}|}\sum\nolimits_{\boldsymbol{x}\in\mathbb{B},\tilde{\boldsymbol{x}}\in\tilde{\mathbb{B}}} \|\mathrm{softmax}(f(\boldsymbol{x};\boldsymbol{\theta})) - \mathrm{softmax}(f(\tilde{\boldsymbol{x}};\boldsymbol{\theta}))\|_2.$$

1. **Representation space: contrastive loss ()**

$$\mathcal{L}_{neg}(\mathbb{B},\tilde{\mathbb{B}};\boldsymbol{\theta}) = -\frac{1}{|\mathbb{B}|}\sum_{\boldsymbol{x}_i\in\mathbb{B}} \frac{1}{|\mathbb{P}_i|}\sum_{\boldsymbol{x}_j\in\mathbb{P}_i} \log \frac{\exp(\mathrm{sim}(\boldsymbol{x}_i,\boldsymbol{x}_j)/\tau)}{\sum_{\boldsymbol{x}_k\in\mathbb{Q}_i}\exp(\mathrm{sim}(\boldsymbol{x}_i,\boldsymbol{x}_k)/\tau)},$$

minibatch $\mathbb{B}$ sharing the same class as $\boldsymbol{x}_i$. The anchor $\boldsymbol{x}_i$ is excluded from its positive set $\mathbb{P}_i$. Next, we can generate the negative set composed of two types of negative examples: 1) all the examples in the minibatch $\mathbb{B}$ with a different class as $\boldsymbol{x}_i$, 2) the patch-based negatively transformed images $\tilde{\boldsymbol{x}} \in \tilde{\mathbb{B}}$. For each anchor $\boldsymbol{x}_i$, we can in total have $2|\mathbb{B}| - |\mathbb{P}_i| - 1$ negative pairs, where $|\mathbb{B}|$ is the batch size and $|\mathbb{P}_i|$ is the cardinality of the

- **Experiments (per loss design)**

Table 1: Top-1 accuracies for ViT-B/16 pre-trained and fine-tuned on ImageNet-1k with or without the proposed negative augmentation.

Model	ImageNet-1k	ImageNet-A	ImageNet-C	ImageNet-R
ViT-B/16 (Dosovitskiy et al., 2021)	77.6	6.7	50.8	20.3
+ P-Rotate / Uniform	78.2 (+0.6)	7.0 (+0.3)	52.4 (+1.6)	21.4 (+1.1)
+ P-Rotate / L2	77.8 (+0.2)	6.7 (+0.0)	51.6 (+0.8)	21.0 (+0.7)
+ P-Rotate / Contrastive	78.9 (+1.3)	8.6 (+1.9)	54.1 (+3.3)	23.6 (+3.3)

- **(v.s wise augmentation)**

Table 2: Top-1 accuracies for ViT-B/16 pre-trained and fine-tuned on ImageNet-1k using Rand-Augment (Cubuk et al., 2020) or AugMix (Hendrycks et al., 2020). The proposed negative augmentation is added on top of either positive augmentation. See Table 8 in Appendix for a **full** table with three losses for each patch-based transformation. Patch-based negative augmentation is complementary to "positive" data augmentation.

Model	ImageNet-1k	ImageNet-A	ImageNet-C	ImageNet-R
Rand-Augment (Cubuk et al., 2020)	79.1	7.2	55.2	23.0
+ P-Rotate / L2	79.1 (+0.0)	7.9 (+0.7)	56.7 (+1.5)	23.8 (+0.8)
+ P-Infill / Uniform	79.2 (+0.1)	7.8 (+0.6)	56.4 (+1.2)	24.0 (+1.0)
+ P-Rotate / Contrastive	79.9 (+0.8)	9.4 (+2.2)	58.4 (+3.2)	25.4 (+2.4)
+ P-Infill / Contrastive	79.9 (+0.8)	9.3 (+2.1)	57.9 (+2.7)	25.0 (+2.0)
AugMix (Hendrycks et al., 2020)	78.8	7.7	57.8	24.9
+ P-Rotate / L2	79.0 (+0.2)	8.3 (+0.6)	58.8 (+1.0)	26.0 (+1.1)
+ P-Infill / Uniform	79.3 (+0.5)	8.3 (+0.6)	58.4 (+0.6)	25.7 (+0.8)
+ P-Rotate / Contrastive	79.6 (+0.8)	9.8 (+2.1)	60.0 (+2.2)	27.5 (+2.6)
+ P-Infill / Contrastive	79.6 (+0.8)	9.9 (+2.2)	60.3 (+2.5)	27.3 (+2.4)

- **Robustness improvements even under larger pre-training datasets**

Table 3: Top-1 accuracies of ViT-B/16 pretrained on ImageNet-21k and finetuned on ImageNet-1k. Patch-based negative augmentation is helpful even with large-scale pretraining.

Model	ImageNet-1k	ImageNet-A	ImageNet-C	ImageNet-R
ViT-B/16 (Dosovitskiy et al., 2021)	84.1	26.7	65.2	37.9
Rand-Augment (Cubuk et al., 2020)	84.4	28.7	67.2	38.7
+ P-Shuffle / Uniform	**84.5** (+0.1)	29.9 (+1.2)	67.7 (+0.5)	38.9 (+0.2)
+ P-Shuffle / L2	**84.5** (+0.1)	29.7 (+1.0)	68.0 (+0.8)	**39.6** (+0.9)
+ P-Shuffle / Contrastive	84.3 (-0.1)	**30.8** (+2.1)	**68.1** (+0.9)	38.6 (-0.1)

- **Negative view augmentation을 fine-tune/pre-train/both stage에서 적용했을 때, both가 best**

Table 4: Effect of patch-based negative augmentation in pre-training and fine-tuning stages. Top-1 accuracies of ViT-B/16 pretrained and fine-tuned on ImageNet-1k. Under 'Stage' we denote which training stage patch-based negative augmentation is used.

Model	Stage	ImageNet-1k	ImageNet-A	ImageNet-C	ImageNet-R
AugMix (Hendrycks et al., 2020)	-	78.8	7.7	57.8	24.9
+ P-Shuffle / Contrastive	Fine-tune	79.2 (+0.4)	8.3 (+0.6)	58.4 (+0.6)	25.9 (+1.0)
+ P-Shuffle / Contrastive	Pre-train	79.4 (+0.6)	8.7 (+1.0)	59.5 (+1.7)	26.7 (+1.8)
+ P-Shuffle / Contrastive	Both	**79.6** (+0.8)	**9.0** (+1.3)	**60.1** (+2.3)	**27.3** (+2.4)

Understanding and Improving Robustness of Vision Transformers through Patch-based Negative Augmentation

- **Appendix**

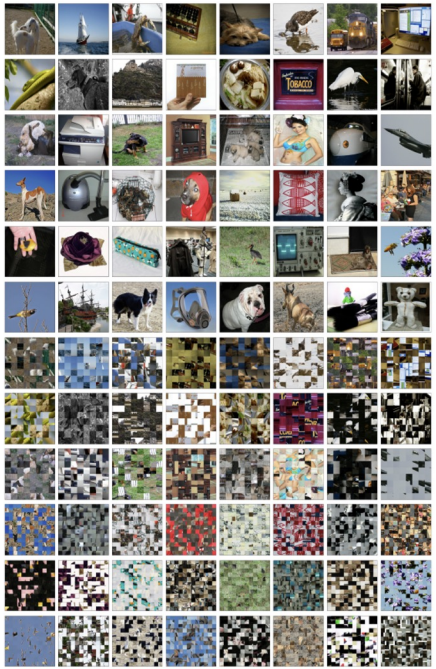


Figure 5: Examples of original images (on the top) and their corresponding patch-based shuffle (at the bottom) with either patch size 32 or 48 without cherry-picking.

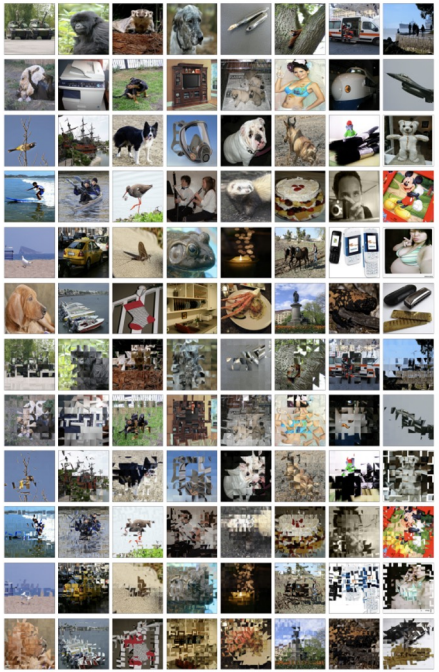


Figure 6: Examples of original images (on the top) and their corresponding patch-based rotation (at the bottom) with either patch size 32 or 48 without cherry-picking.

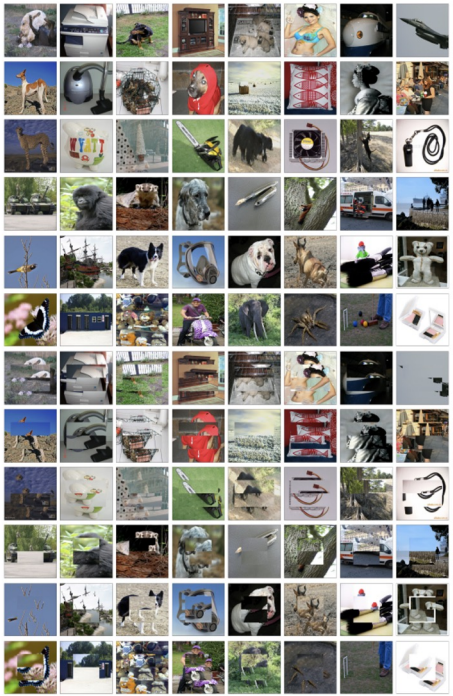


Figure 7: Examples of original images (on the top) and their corresponding patch-based infill (at the bottom) with either replace rate 0.25 or 0.375 without cherry-picking.

https://arxiv.org/pdf/2110.13903v1.pdf

NeRV: Neural Representations for Videos

Hao Chen[1], Bo He[1], Hanyu Wang[1], Yixuan Ren[1], Ser-Nam Lim[2], Abhinav Shrivastava[1]
[1]University of Maryland, College Park, [2]Facebook AI
{chenh, bohe, hywang66, yxren, abhinav}@umd.edu, sernamlim@fb.com

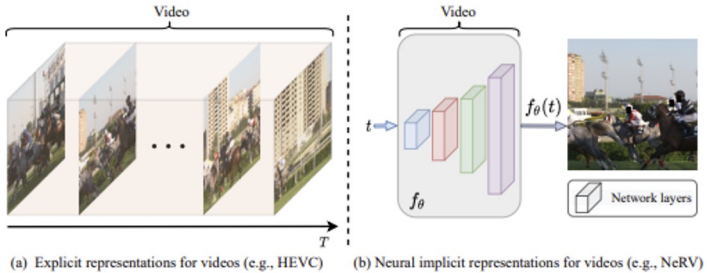


Figure 1: **(a)** Conventional video representation as **frame sequences**. **(b)** NeRV, representing video as **neural networks**, which consists of multiple convolutional layers, taking the normalized frame index as the input and output the corresponding RGB frame.

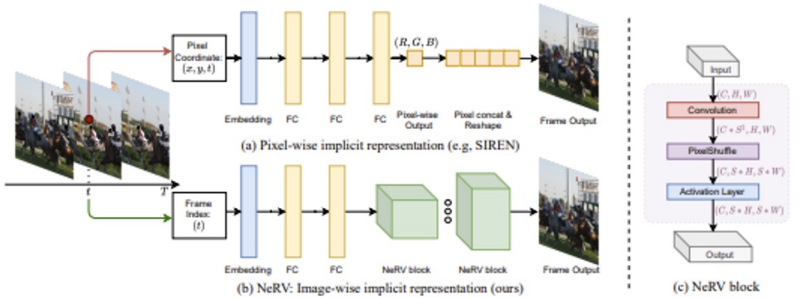


Figure 2: **(a) Pixel-wise** implicit representation taking pixel coordinates as input and use a simple MLP to output pixel RGB value **(b) NeRV: Image-wise** implicit representation taking frame index as input and use a MLP + ConvNets to output the whole image. **(c) NeRV block** architecture, upscale the feature map by S here.

Table 1: Comparison of different video representations. Although explicit representations outperform implicit ones in encoding speed and compression ratio now, NeRV shows great advantage in decoding speed. And NeRV outperforms pixel-wise implicit representations in all metrics.

	Explicit (frame-based)		Implicit (unified)	
	Hand-crafted (*e.g.*, HEVC [2])	Learning-based (*e.g.*, DVC [3])	Pixel-wise (*e.g.*, NeRF [4]	Image-wise (Ours)
Encoding speed	**Fast**	Medium	Very slow	Slow
Decoding speed	Medium	Slow	Very slow	**Fast**
Compression ratio	Medium	**High**	Low	Medium

<Abstract>

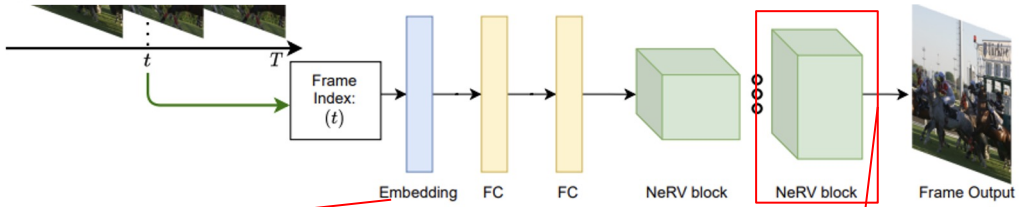
- Neural representation for videos (NeRV) which encodes videos in neural networks.
- taking frame index as input, outputs the corresponding RGB image.
- Video encoding in NeRV is simply fitting a neural network to video frames and decoding process is a simple feedforward operation
- pixel wise implicit representation에 비해 좋은 효율성을 보여 encoding 속도를 25배에서 70배, decoding 속도를 38배에서 132배 향상시키면서 더 나은 비디오 화질을 달성
- Github: https://github.com/haochen-rye/NeRV.git

https://arxiv.org/pdf/2110.13903v1.pdf

3.1 NeRV Architecture

In NeRV, each video $V = \{v_t\}_{t=1}^{T} \in \mathbb{R}^{T\times H\times W\times 3}$ is represented by a function $f_\theta : \mathbb{R} \to \mathbb{R}^{H\times W\times 3}$, where the input is a frame index t and the output is the corresponding RGB image $v_t \in \mathbb{R}^{H\times W\times 3}$. The encoding function is parameterized with a deep neural network θ, $v_t = f_\theta(t)$. Therefore, video encoding is done by fitting a neural network f_θ to a given video, such that it can map each input timestamp to the corresponding RGB frame.

The encoding function is parameterized with a deep neural network θ, $v_t = f_\theta(t)$. Therefore, video encoding is done by fitting a neural network f_θ to a given video, such that it can map each input timestamp to the corresponding RGB frame.



(b) NeRV: Image-wise implicit representation (ours)

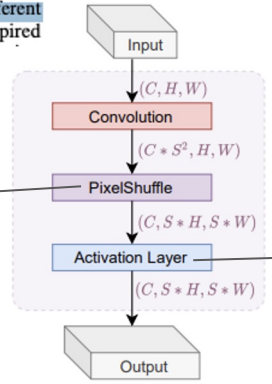
The detailed architecture can be found in the supplementary material.

input embedding fun

$$\Gamma(t) = \left(\sin\left(b^0\pi t\right), \cos\left(b^0\pi t\right), \ldots, \sin\left(b^{l-1}\pi t\right), \cos\left(b^{l-1}\pi t\right)\right)$$

그렇다고 뒤에 뭔 더 설명이 있나? 그것도 아님

large. Therefore, we stack multiple NeRV blocks following the MLP layers so that pixels at different locations can share convolutional kernels, leading to an efficient and effective network. Inspired



for upscaling method

to enhance the expressibilty

(c) NeRV block

Loss Objective. For NeRV, we adopt combination of L1 and SSIM loss as our loss function

$$L = \frac{1}{T}\sum_{t=1}^{T} \alpha \left\| f_\theta(t) - v_t \right\|_1 + (1-\alpha)(1 - \text{SSIM}(f_\theta(t), v_t))$$

- **Model Compression**

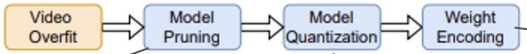


Figure 3: NeRV-based video compression pipeline.

Entropy Encoding. Finally, we use entropy encoding to further compress the model size. By taking advantage of character frequency, entropy encoding can represent the data with a more efficient codec. Specifically, we employ Huffman Coding [50] after model quantization. Since Huffman Coding is lossless, it is guaranteed that a decent compression can be achieved without any impact on the reconstruction quality. Empirically, this further reduces the model size by around 10%.

$$\theta_i = \begin{cases} \theta_i, & \text{if } \theta_i \geq \theta_q \\ 0, & \text{otherwise,} \end{cases}$$

Model Quantization. After model pruning, we apply model quantization to all network parameters. Note that different from many recent works [31, 47–49] that utilize quantization during training, NeRV is only quantized post-hoc (after the training process). Given a parameter tensor μ

$$\mu_i = \text{round}\left(\frac{\mu_i - \mu_{\min}}{2^{\text{bit}}}\right) * \text{scale} + \mu_{\min}, \quad \text{scale} = \frac{\mu_{\max} - \mu_{\min}}{2^{\text{bit}}} \tag{4}$$

https://arxiv.org/pdf/2110.13903v1.pdf

- **Experiments**

Table 2: Compare with pixel-wise implicit representations. Training speed means time/epoch, while encoding time is the total training time.

Methods	Parameters	Training Speed ↑	Encoding Time ↓	PSNR ↑	Decoding FPS ↑
SIREN [5]	3.2M	1×	2.5×	31.39	1.4
NeRF [4]	3.2M	1×	2.5×	33.31	1.4
NeRV-S (ours)	3.2M	**25×**	**1×**	**34.21**	**54.5**
SIREN [5]	6.4M	1×	5×	31.37	0.8
NeRF [4]	6.4M	1×	5×	35.17	0.8
NeRV-M (ours)	6.3M	**50×**	**1×**	**38.14**	**53.8**
SIREN [5]	12.7M	1×	7×	25.06	0.4
NeRF [4]	12.7M	1×	7×	37.94	0.4
NeRV-L (ours)	12.5M	**70×**	**1×**	**41.29**	**52.9**

Table 3: PSNR *vs.* epochs. Since video encoding of NeRV is an over-fit process, the reconstructed video quality keeps increasing with more training epochs. NeRV-S/M/L mean models with different sizes.

Epoch	NeRV-S	NeRV-M	NeRV-L
300	32.21	36.05	39.75
600	33.56	37.47	40.84
1.2k	34.21	38.14	41.29
1.8k	34.33	38.32	41.68
2.4k	**34.86**	**38.7**	**41.99**

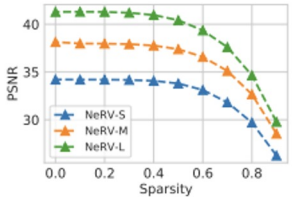


Figure 4: Model **pruning**. Sparsity is the ratio of parameters pruned.

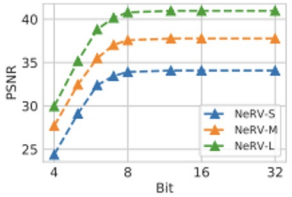


Figure 5: Model **quantization**. Bit is the bit length used to represent parameter value.

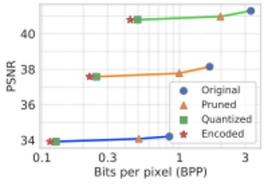


Figure 6: Compression **pipeline** to show how much each step contribute to compression ratio.

최대 신호 대 잡음비(Peak Signal-to-noise ratio, **PSNR**)는 신호가 가질 수 있는 최대 전력에 대한 잡음의 전력을 나타낸 것이다. 주로 영상 또는 동영상 손실 압축에서 화질 손실 정보를 평가할때 사용된다.

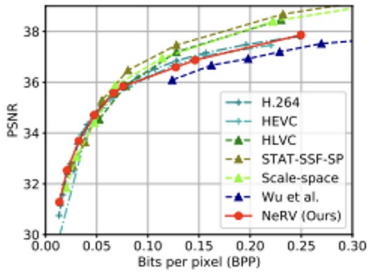


Figure 7: PSNR *vs.* BPP on UVG dataset.

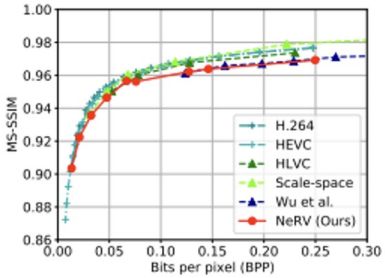


Figure 8: MS-SSIM *vs.* BPP on UVG dataset.

(a) Ground Truth Frame (b) NeRV Decoded Frame (0.077 BPP) (c) HEVC Decoded Frame (0.075 BPP)

Figure 9: Video compression visualization. At similar BPP, NeRV reconstructs videos with better details.

https://arxiv.org/pdf/2110.13903v1.pdf

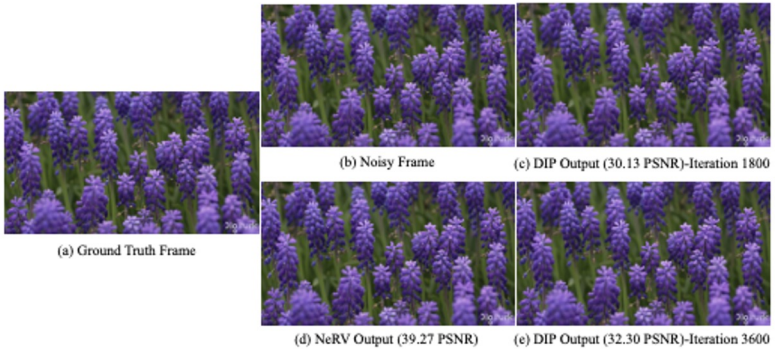
- **Experiments**

Table 4: **Decoding speed** with BPP 0.2 for 1080p videos

Methods	FPS ↑
Habibian et al. [14]	$10^{-3.7}$
Wu et al. [24]	10^{-3}
Rippel et al. [57]	1
DVC [3]	1.8
Liu et al [58]	3
H.264 [8]	9.2
NeRV (FP32)	5.6
NeRV (FP16)	**12.5**

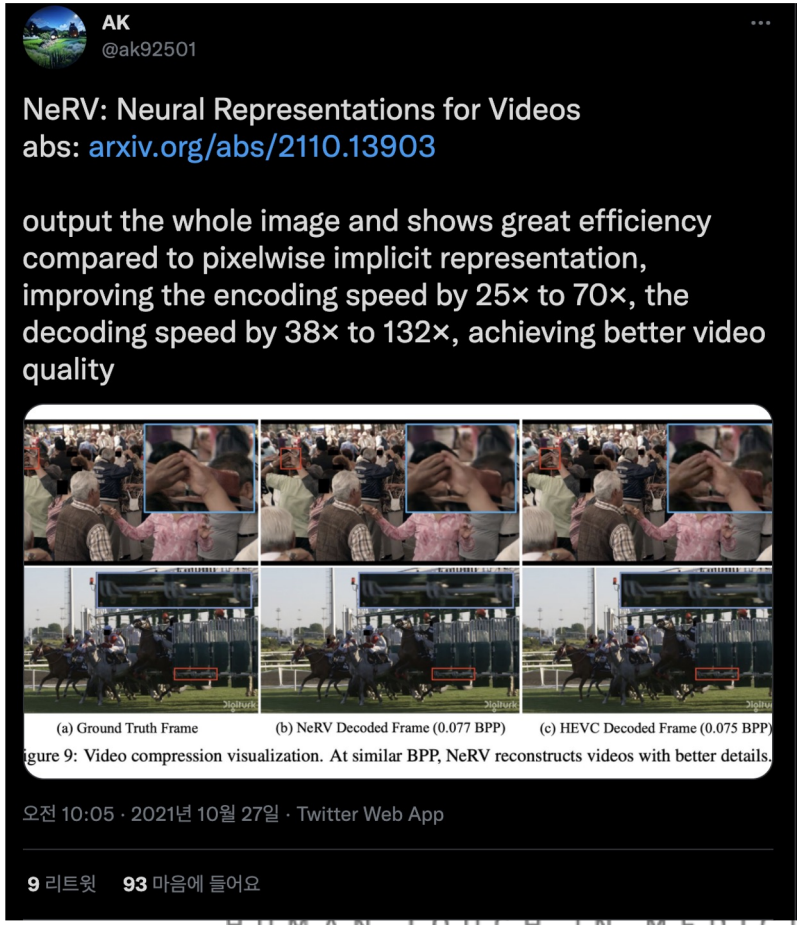
Table 5: PSNR results for **video denoising**. "baseline" refers to the noisy frames before any denoising

noise	white ↑	black ↑	salt & pepper ↑	random ↑	Average ↑
Baseline	27.85	28.29	27.95	30.95	28.74
Gaussian	30.27	30.14	30.23	30.99	30.41
Uniform	29.11	29.06	29.10	29.63	29.22
Median	**33.89**	33.84	33.87	33.89	33.87
Minimum	20.55	16.60	18.09	18.20	18.36
Maximum	16.16	20.26	17.69	17.83	17.99
NeRV	33.31	**34.20**	**34.17**	**34.80**	**34.12**



(a) Ground Truth Frame (b) Noisy Frame (c) DIP Output (30.13 PSNR)-Iteration 1800 (d) NeRV Output (39.27 PSNR) (e) DIP Output (32.30 PSNR)-Iteration 3600

Figure 10: Denoising visualization. (c) and (e) are denoising output for DIP [59]. Data generalization of NeRV leads to robust and better denoising performance since all frames share the same representation, while DIP model overfits one model to one image only.



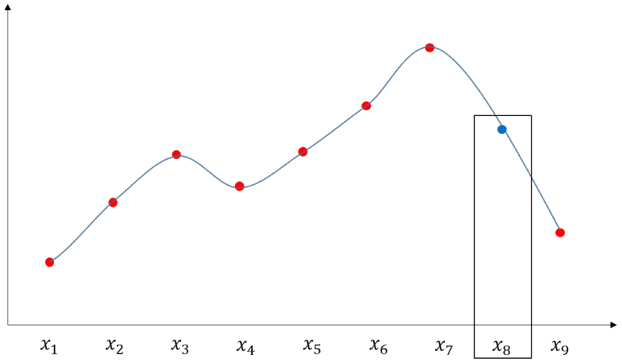
Learning in High Dimension Always Amounts to Extrapolation

Randall Balestriero[1], Jérôme Pesenti[1], and Yann LeCun[1,2]

[1]Facebook AI Research, [2]NYU

{rbalestriero,pesenti,yann}@fb.com

https://arxiv.org/pdf/2110.09485v2.pdf



Facebook + Yann LeCun

- **Interpolation(내삽), Extrapolation(외삽) ?**
  - Interpolation occurs for a sample x whenever this sample falls inside or on the boundary of the given dataset's convex hull.
    - convex hull: 여러 개의 점이 주어졌을 때, 모든 점들을 포함하는 최소 크기의 볼록 다각형.
  - Extrapolation occurs when x falls outside of that convex hull.

- **Fundamental (mis)conception**
  1) SoTA 알고리즘은 training data 를 올바르게 interpolate 하기 때문에 성능이 좋은 것이다. (== interpolation 올바르게 잘해서 sota가 된 것이다.)
  2) Interpolation 은 모든 tasks 와 datasets 에서 발생한다, 많은 intuitions 과 theory 가 이러한 가정에 의존하고 있음. (== 우리는 모든 데이터셋에서 interplolation이 잘 될 꺼다라고 생각하는데, 이건 틀린 것
     왜? 우리 실험에서 high dimension에서는 extrapolation한 특성발견)

- 본 논문에서 위의 두 가지 (mis)conception 에 대해 논하고, 실제로 high-dimensional (>100) dataset 은 interpolation 이 거의 발생하지 않음을 보임.
- 즉, 고차원 데이터셋을 생각하면, 한 데이터가 어떤 다른 데이터셋의 convex hull 안에 들어갈 확률이 매우 작다.

내삽과 빅데이터

이제껏 논한 머신러닝의 예측은 내삽 (**interpolation**)이라 보는 것이 더 정확하다. 한 웹사이트에서 발췌한 내삽의 정의는 아래와 같다. 내삽 (예측)을 아주 명확하게 정리한 좋은 문장이라고 생각한다!

- 변수 x의 함수 $f(x)$의 형이 미지이고 일정한 간격을 둔 둘 이상의 변수의 값 x_i $(i=1,\ldots,n)$에 대한 함수값 $f(x_i)$가 알려져 있을 때 그 사이 임의의 x에 대한 함수값을 추정하는 조작.

요약하면 머신러닝은 미래 x 값에 대한 y값을 예측하는 것이 아니라, 기존 데이터 범위 내 x 값에 대한 y 값을 내삽하는 것이라고 정의하는 것이 더 정확하다. 그리고 더 넓은 범위의 x 값에 대한 내삽을 위해서는 빅데이터가 필요하다.

Learning in High Dimension Always Amounts to Extrapolation

Theorem 1 (Bárány and Füredi (1988))**.** Given a d-dimensional dataset $\boldsymbol{X} \triangleq \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ with i.i.d. samples uniformly drawn from an hyperball, the probability that a new sample $\boldsymbol{x}$ is in interpolation regime (recall Def. 1) has the following asymptotic behavior

$$\lim_{d\to\infty} p(\underbrace{\boldsymbol{x} \in \text{Hull}(\boldsymbol{X})}_{\text{interpolation}}) = \begin{cases} 1 & \iff N > d^{-1}2^{d/2} \\ 0 & \iff N < d^{-1}2^{d/2} \end{cases}$$

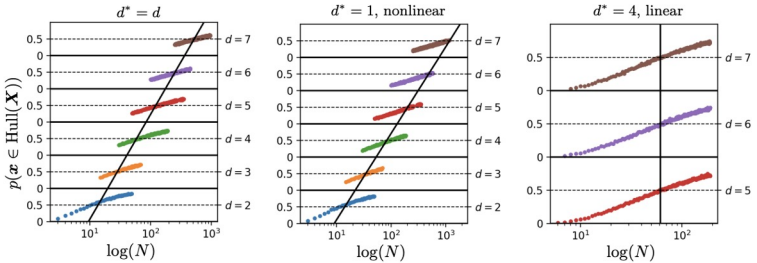


Figure 1: Depiction of the evolution of the probability that a new sample is in interpolation regime (y-axis, $p(\boldsymbol{x} \in \text{Hull}(\boldsymbol{X}))$) given increasing dataset size (x-axis, N) seen in logarithmic scale, and for various ambient space dimensions (d) based on Monte-Carlo estimates on 500,000 trials. On the **left**, the data is sampled from a Gaussian density $\boldsymbol{x}_i \sim \mathcal{N}(0, I_d)$ while in the **middle**, the data is sampled from a nonlinear continuous manifold with intrinsic dimension of 1 (see Fig. 2 for details on the manifold data) and on the **right**, the data is sampled from a Gaussian density that lives in an affine subspace of constant dimension 4 (while the ambient dimension increases). It is clear from those figures that **in order to maintain a constant probability to be in interpolation regime, the training set size has to increase exponentially with d^* regardless of the underlying intrinsic manifold dimension** where d^* is the dimension of the lowest dimensional affine subspace including the entire data manifold i.e. the convex hull dimension.

interpolation regime 에 존재할 일정 수준의 probability 를 유지하기 위해서,
Data manifold intrinsic dimension 과는 관계없이,
Training set 크기가 기하급수적으로 커진다.

Interpolation almost surely never occurs in high-dimensional spaces (>100), regardless of the underlying intrinsic dimension of the data manifold.

The Role of the Intrinsic, Ambient and Convex Hull Dimensions

- d : Ambient dimension : the dimension of the space in which the data lives (데이터가 존재하는 공간의 차원)
- d* : Data manifold intrinsic dimension : the number of variables needed in a **minimal representation of the data** (Bennett, 1965) (데이터의 minimal representation 에 필요한 variable 의 수)

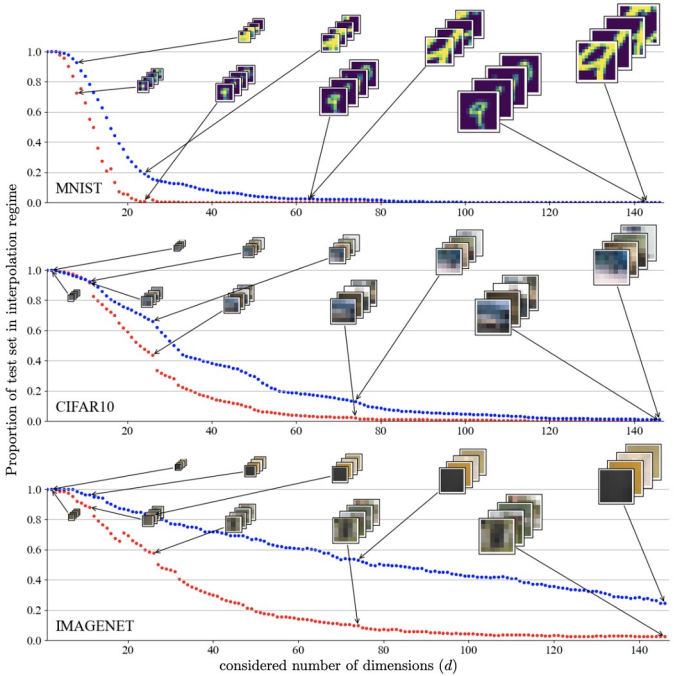


Figure 3: Depiction of the proportion of the test set that is in interpolation of the training set for MNIST (**top**), CIFAR (**middle**) and Imagenet (**bottom**) as a function of the number of selected dimensions. We propose two settings (**blue**) selecting increasingly large central patches (some cases consist of irregular patches for intermediate dimension values) and (**red**) smoothing-subsampling the original images (some cases consist of irregular images for intermediate dimension values). Note that the blue line is always decreasing with d, and that $d = 147$ (right of the x-axis) represents 19% of MNIST total number of dimensions, 5% for CIFAR and less than 1% for Imagenet. As can be seen throughout those settings **the proportion of the test set that is in interpolation regime decreases exponentially fast with respect to the number of dimensions** ultimately becoming negligible well prior reaching the full data dimensionality. The different slopes of those curves can be explained by the smallest dimensional affine space containing each type of data of (see Tab. 1).

Real Datasets and Embeddings are no Exception

- The probability that new samples are in interpolation or extrapolation regimes in more specific scenarios.
- Dataset
  - MNIST
  - CIFAR10
  - IMAGENET
- Test set 이 interpolation regime 내부에 있는 proportion 은 dimension 이 증가할수록, 기하급수적으로 작아진다.
- (Blue) Increasingly large central patches.
- (Red) Smoothing-subsampling the original images.

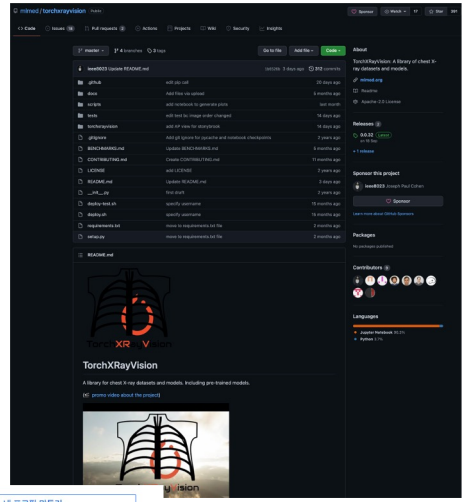
TorchXRayVision: A library of chest X-ray datasets and models

https://arxiv.org/pdf/2111.00595v1.pdf

TorchXRayVision

TorchXRayVision:
A library of chest X-ray datasets and models

Joseph Paul Cohen[S,M,Q,A] Joseph D. Viviano[Q] Paul Bertin[M,Q]
Paul Morrison[Q,F] Parsa Torabian[W] Matteo Guarrera[B,E,P] Matthew P Lungren[S,A]
Akshay Chaudhari[S,A] Rupert Brooks[N] Mohammad Hashir[M,Q] Hadrien Bertrand[Q]



Joseph Paul Cohen

Postdoctoral Fellow, AIMI (Stanford University) + Mila (Quebec AI Institute)
stanford.edu의 이메일 확인됨 - 홈페이지

Medical Imaging Genomics Computer Vision ML in Healthcare Representation Learning

제목	인용	연도
COVID-19 Image Data Collection: Prospective Predictions are the Future	971	2020

내 프로필 만들기

인용

	전체	2016년 이후
서지정보	2231	2158
h-index	19	19
i10-index	25	25

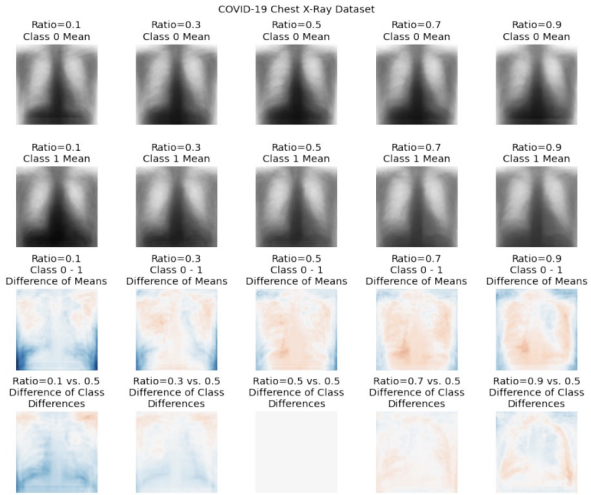


Figure 4: An example of the mean class difference drawn from the COVID-19 dataset at different covariate ratios. Here, the first COVID-19 dataset consisted of only AP images, whereas the second dataset consisted of only PA images. The third row shows, for each ratio, the difference in the class means, demonstrating the effect of sampling images from the two views on the perceived class difference. The fourth row shows the difference between each ratio's difference image, and the difference image with a ratio of 0.5 (balanced sampling from all views).

<Abstract>

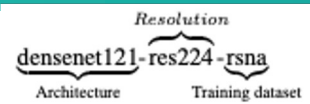
- TorchXRayVision is an open source software library for working with chest X-ray datasets and deep learning models. (Pytorch based, modeled after the torchvision library)
- It provides a common interface and common pre-processing chain for a wide set of publicly available chest X-ray datasets.
- pip로 install 가능 / import torchxrayvision as xrv.
- Github: https://github.com/mlmed/torchxrayvision
- 소개 동영상: https://www.youtube.com/watch?v=Rl7xz0uULGQ
- Joseph Paul Cohen 박사의 자랑페이지: https://josephpcohen.com/w/

TorchXRayVision: A library of chest X-ray datasets and models

https://arxiv.org/pdf/2111.00595v1.pdf

Core classifier

- Models are specified using the “weights" parameter which have the general form of



- More details about each set of weights is available at the head of the models.py file.
- each image was randomly rotated up to 45 degrees, translated up to 15% and scaled larger of smaller up to 10%.

```
# Generic function to load any core model
model = xrv.models.get_model(weights="densenet121-res224-all")

# DenseNet 224x224 model trained on multiple datasets
model = xrv.models.DenseNet(weights="densenet121-res224-all")

# DenseNet trained on just the RSNA Pneumonia dataset
model = xrv.models.DenseNet(weights="densenet121-res224-rsna")

# ResNet 512x512 model trained on multiple datasets
model = xrv.models.ResNet(weights="resnet50-res512-all")
```

Baseline Classifiers

- **The models adhere to the same interface as the core models and can be easily swapped out.**

```
# DenseNet121 from JF Healthcare for the CheXpert competition
model = xrv.baseline_models.jfhealthcare.DenseNet()

# Official Stanford CheXpert model
model = xrv.baseline_models.chexpert.DenseNet()
```

TorchXRayVision: A library of chest X-ray datasets and models

- **Visualization**
  pre-trained models can also be used as features extractors for semi-supervised training or transfer learning task

```
feats = model.features(img)
```

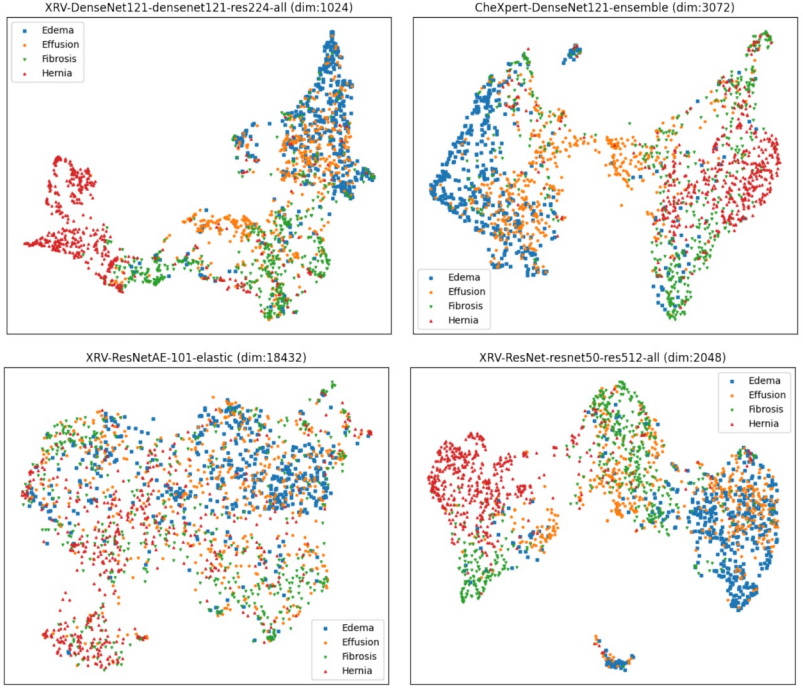


Figure 1: UMAP visualizations of the representations from different models. 2048 images, each containing only one of the 4 pathologies listed, are included in the UMAP.

- **Dataset**

Table 1: Details of datasets that are included in this library. Number of images shows total images / usable frontal images. Useable frontal means images that are readable, have all necessary metadata, and are in AP, PA, AP Supine, or AP Erect view.

Name	# Images (Total/Frontal)			Citation	Geographic Region
National Library of Medicine Tuberculosis	800	/	800	Jaeger et al. [2014]	USA+China
OpenI (National Library of Medicine)	7,470	/	4,014	Demner-Fushman et al. [2016]	USA
ChestX-ray8 (NIH)	112,120	/	112,120	Wang et al. [2017]	Northeast USA
RSNA Pneumonia Challenge	26,684	/	26,684	Shih et al. [2019]	Northeast USA
CheXpert (Stanford University)	223,414	/	191,010	Irvin et al. [2019]	Western USA
Google Labelling of NIH data	4,376	/	4,376	Majkowska et al. [2019]	Northeast USA
MIMIC-CXR (MIT)	377,095	/	243,324	Johnson et al. [2019]	Northeast USA
PadChest (University of Alicante)	158,626	/	108,722	Bustos et al. [2020]	Spain
SIIM-ACR Pneumothorax Challenge	12,954	/	12,954	Filice et al. [2020]	Northeast USA
COVID-19 Image Data Collection (CIDC)	866	/	698	Cohen et al. [2020c]	Earth
StonyBrook COVID-19 RALO Severity	2,373	/	2,373	Cohen et al. [2021b]	Northeast USA
Object-CXR (JF Healthcare)	9,000	/	9,000	-	China
VinBrain VinDr-CXR	15,000	/	15,000	Nguyen et al. [2020]	Vietnam

```
dataset = xrv.datasets.VinBrain_Dataset(imgpath="./train",
                                        csvpath="./train.csv")
```

EGO 4D

[Download Ego 4D] Please help to download Ego 4D 외부 받은편지함 ×

지현규 <hyeongyuc96@hutom.co.kr> 11월 1일 (월) 오후 3:44 (11일 전)
info에게

Dear Ego4D,

I am a researcher from VisionAI team at hutom(https://hutom.io) in the Republic of Korea.
Our company is working on surgical intelligence to help surgeons make smarter decisions for better patient care with computer vision.

I read your paper, "Ego4D: Around the World in 3,000 Hours of Egocentric Video" published in Arxiv Sanity and found that this dataset will be very helpful for our research.

However, the dataset seems not available right now on the website(https://ego4d-data.org/#download). I wonder if there is an application procedure to access the dataset.

Best, Hyeon-Gyu

Info for Ego4D Project Mailbox 11월 2일 (화) 오전 1:30 (10일 전)
나에게

영어 > 한국어 메일 번역 영어 번역 안함 ×

Hello,

Thank you for the message. Our data will be available to researchers able to agree to our Data Use Agreement at the end of the November. Please check back at our website for more details at the end of this month. Let us know if you have any questions in the meantime.

All the best,
Ego4D

Palette: Image-to-Image Diffusion Models

Palette: Image-to-Image Diffusion Models

Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet* and Mohammad Norouzi *
{sahariac, williamchan, davidfleet, mnorouzi}@google.com
Google Research

https://arxiv.org/pdf/2111.05826.pdf

Variational Diffusion Models

Diederik P. Kingma*, Tim Salimans*, Ben Poole, Jonathan Ho
Google Research

Abstract

https://arxiv.org/pdf/2107.00630.pdf

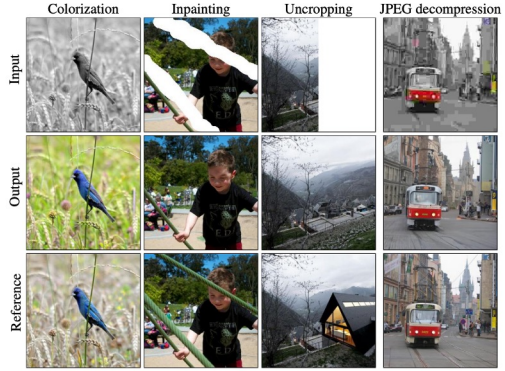


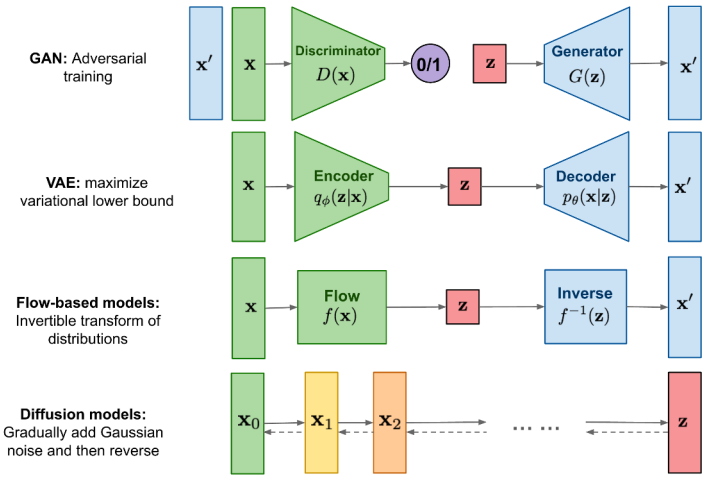
Figure 1: Illustration of Palette's performance on four image-to-image translation tasks.

- Google Research
- Palette
  - **A simple and general framework for image-to-image translation using conditional diffusion models.**
  - On four challenging image-to-image translation tasks **(colorization, inpainting, uncropping, and JPEG decompression)**
  - SoTA
    - Strong GAN, regression baselines 성능 능가
    - without task-specific hyper-parameter tuning, architecture customization, auxiliary loss

https://iterative-refinement.github.io/palette/

  - Contributions

1. We study and confirm the versatility and general applicability of diffusion models to image-to-image translation. Palette, our implementation of image-to-image diffusion models, is able to achieve SoTA performance on four tasks, colorization, inpainting, uncropping, and JPEG decompression, with no task-specific tuning or customization.
2. We recognize the impact of using L_1 *vs.* L_2 in the denoising objective on sample diversity, and show the importance of self-attention through empirical studies of the neural network architecture.
3. We propose a unified evaluation protocol across image translation tasks based on the ImageNet dataset, and report several sample quality scores for several baselines including Palette, a surprisingly strong regression baseline, and prior work where possible.
4. We show that a multi-task Palette model performs as well or better than task-specific models.



Diffusion models

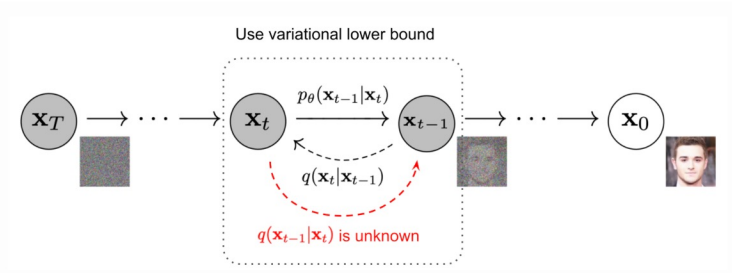


Fig. 2. The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. (Image source: Ho et al. 2020 with a few additional annotations)

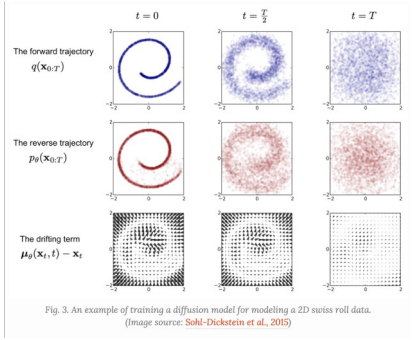


Fig. 3. An example of training a diffusion model for modeling a 2D swiss roll data. (Image source: Sohl-Dickstein et al., 2015)

- **Diffusion models (blog link)**
  - GAN, VAE and Flow-based models 과 다른 새로운 타입의 Generative model.
  - 모호하고 복잡한 데이터 분포 (arbitrarily complex data distribution) 을 더 잘 학습하고, high-quality images 생성이 뛰어남. (SoTA GAN)
  - **Input data (X0) 에 slowly add random noise 한 후, noise (z) 에서부터 construct desired data samples 을 얻을 수 있도록 diffusion process reverse 학습.**

- **Conditional diffusion models**
  - Diffusion models: 반복적인 Denoising process 과정을 통해서 standard Gaussian distribution 샘플을 empirical data distribution 샘플로 변환.
  - Input signal 에 조건부로 (conditional) denoising process 추가.

- **Palette: conditional diffusion models 형태.**
  - form p(y|x)
    - x: a grayscale image
    - y: a reference color image
  - image super-resolution 적용.
  - Loss function

$$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}\mathbb{E}_{\gamma}\left\|f_\theta(\boldsymbol{x},\underbrace{\sqrt{\gamma}\,\boldsymbol{y}+\sqrt{1-\gamma}\,\boldsymbol{\epsilon}}_{\widetilde{\boldsymbol{y}}},\gamma)-\boldsymbol{\epsilon}\right\|_p^p,$$

  - Architecture
    - a standard U-Net architecture, with several architecture modifications inspired by recent work.
    - based on the 256 * 256 class-conditional U-Net model
      - 차이점 1) class-conditioning 없음
      - 차이점 2) concatenation 을 통한 source image 의 추가적인 conditioning

- **Evaluating Image-to-image Translation Models**
  - 기존 연구
    - colorization: FID scores and human evaluation
    - inpainting, uncropping : qualitative evaluation
    - JPEG : PSNR and SSIM
  - 새로운 평가 지표 제안
    - inpainting, uncropping, JPEG decompression on ImageNet
    - det to its scale, diversity, and public availability

- **Experiments**

We study the general applicability of *Palette* to a suite of four distinct and challenging image-to-image translation tasks:

1. **Colorization** transforms an input grayscale image to a plausible color image.
2. **Inpainting** fills in user-specified masked regions of an image with realistic content.
3. **Uncropping** extends an input image along one or more directions to enlarge the image.
4. **JPEG decompression** corrects for JPEG compression artifacts to recover plausible details.

- **Experiments (SoTA)**

Figure 3: Illustration of colorization methods on ImageNet validation images. Baselines: †[Guadarrama et al., 2017], ‡[Kumar et al., 2021], and our own strong regression baseline. Figure C.2 shows more samples.

Model	FID-5K ↓	IS ↑	CA ↑	PD ↓	Fool rate (3 sec) ↑
Prior Work					
cGAN [Isola et al., 2017b]	24.41	-	-	-	-
PixColor [Guadarrama et al., 2017]	24.32	-	-	-	29.90%
Coltran [Kumar et al., 2021]	19.37	-	-	-	36.55%
This paper					
Regression	17.89	169.8	68.2%	60.0	39.45%
Palette	**15.78**	**200.8**	**72.5%**	**46.2**	**47.80%**
Reference	14.68	229.6	75.6%	0.0	-

Table 1: Colorization quantitative scores and fool rates on ImageNet val set. Appendix C.1 has more results.

Figure 5: Image uncropping results on Places2 validation images. Baselines: Boundless† [Teterwak et al., 2019] and InfinityGAN†† [Lin et al., 2021] trained on a scenery subset of Places2. Figure C.7 shows more samples.

Model	ImageNet				Places2		
	FID ↓	IS ↑	CA ↑	PD ↓	FID ↓	PD ↓	Fool rate ↑
Boundless [Teterwak et al., 2019]	18.7	104.1	58.8%	127.9	11.8	129.3	20.7%
Palette (Ours)	**5.8**	**138.1**	**63.4%**	**85.9**	**3.53**	**103.3**	**39.9%**
Reference	2.7	250.1	76.0%	0.0	2.1	0.0	-

Table 3: Quantitative scores and human raters' Fool rates on uncropping.

Figure 4: Comparison of inpainting methods on object removal. Baselines: ‡Photoshop's *Content-aware Fill* built on PatchMatch [Barnes et al., 2009], †[Yu et al., 2019], and ††[Yi et al., 2020]. Figure C.4 has more samples.

Mask Type	Model	ImageNet				Places2	
		FID ↓	IS ↑	CA ↑	PD ↓	FID ↓	PD ↓
20-30% free form	DeepFillv2 [Yu et al., 2019]	9.4	174.6	68.8%	64.7	13.5	63.0
	HiFill [Yi et al., 2020]	12.4	157.0	65.7%	86.2	15.7	92.8
	Palette (Ours)	**5.2**	**205.5**	**72.3%**	**27.6**	**11.7**	**35.0**
128×128 center mask	DeepFillv2 [Yu et al., 2019]	18.0	135.3	64.3%	117.2	15.3	96.3
	HiFill [Yi et al., 2020]	20.1	126.8	62.3%	129.7	16.9	115.4
	Palette (Ours)	**6.6**	**173.9**	**69.3%**	**59.5**	**11.9**	**57.3**
	Reference	5.1	231.6	74.6%	0.0	11.4	0.0

Table 2: Quantitative evaluation for free-form and center inpainting on ImageNet and Places2 validation images.

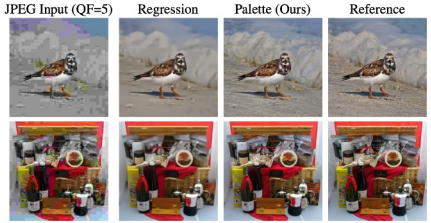


Figure 6: Visual comparison for JPEG decompression on ImageNet validation images.

QF	Model	FID-5K ↓	IS ↑	CA ↑	PD ↓
5	Regression	29.0	73.9	52.8%	155.4
	Palette (Ours)	**8.3**	**133.6**	**64.2%**	**95.5**
10	Regression	18.0	117.2	63.5%	102.2
	Palette (Ours)	**5.4**	**180.5**	**70.7%**	**58.3**
20	Regression	11.5	158.7	69.7%	65.4
	Palette (Ours)	**4.3**	**208.7**	**73.5%**	**37.1**
	Ground Truth	2.7	250.1	75.9%	0.0

Table 4: Quantitative evaluation for JPEG decompression for various Quality Factors (QF).

https://arxiv.org/pdf/2111.06377v1.pdf

Masked Autoencoders Are Scalable Vision Learners

Kaiming He*,† Xinlei Chen* Saining Xie Yanghao Li Piotr Dollár Ross Girshick

*equal technical contribution †project lead

Facebook AI Research (FAIR)

https://arxiv.org/pdf/2111.06377.pdf

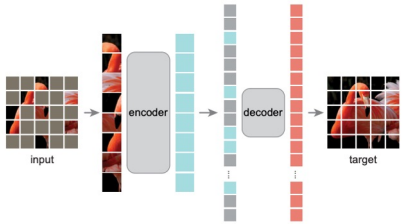


Figure 1. **Our MAE architecture**. During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images to produce representations for recognition tasks.

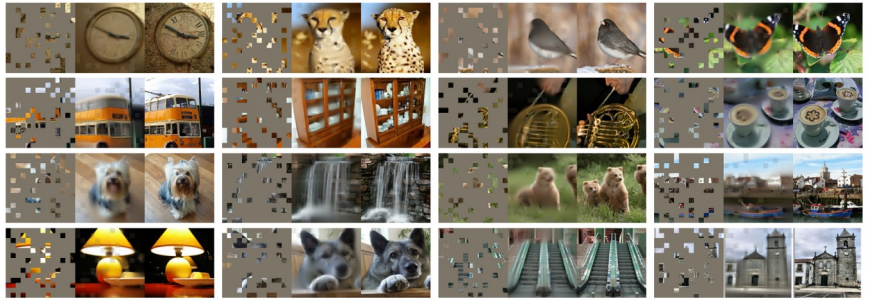


Figure 2. Example results on ImageNet *validation* images. For each triplet, we show the masked image (left), our MAE reconstruction† (middle), and the ground-truth (right). The masking ratio is 80%, leaving only 39 out of 196 patches. More examples are in the appendix. †*As no loss is computed on visible patches, the model output on visible patches is qualitatively worse. One can simply overlay the output with the visible patches to improve visual quality. We intentionally opt not to do this, so we can more comprehensively demonstrate the method's behavior.*

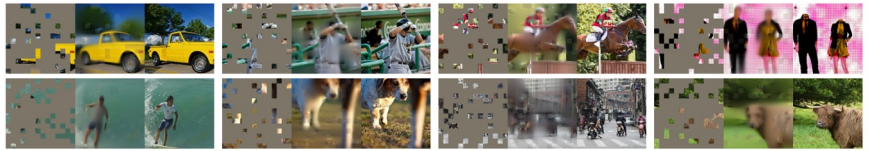


Figure 3. Example results on COCO validation images, using an MAE trained on ImageNet (the same model weights as in Figure 2). Observe the reconstructions on the two right-most examples, which, although different from the ground truth, are semantically plausible.

- **FAIR, Ross Girshick**
- **Masked Autoencoders (MAE) 제안**
  - **Autoencoder 활용하는 방법, masked 되지 않은 조각들만 보고 나머지를 reconstructs 할 수 있도록 학습.**
    - Encoder : latent representation 학습
    - Decoder : original signal 로 reconstructs
  - **Classical autoencoders 와의 차이점**
    - **asymmetric design (encoder 와 decoder 가 asymmetric 하게 생김)**
    - **encoder 는 masking token 을 사용하지 않고,**
    - **decoder 에 갈 때, masking token 을 집어넣고, decoder 에서만 masked token 사용**

https://arxiv.org/pdf/2111.06377v1.pdf

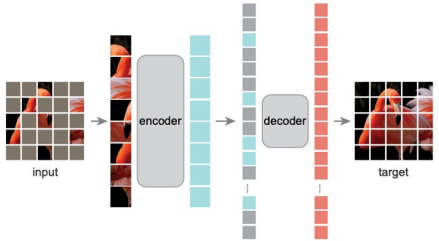


Figure 1. **Our MAE architecture**. During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images to produce representations for recognition tasks.

MAE Architecture

1) Input image 를 overlap 되지 않게, patch 로 나눔. (굉장히 높은 비중으로 masking, default 75%)
2) Masking 되지 않은 patch 만을 사용하여 encoder 에 입력하여 latent space 로 encoding. (Encoder : subset of visible patches)
3) Decoder 로 갈 때, mask patch 입력 (회색부분), Decoder 를 통과시켜 mask patch 복원. (Decoder: the full set of encoded patches and mask tokens)
4) 이렇게 autoencoder 형태로 학습을 완료 한 후, decoder 부분은 잘라내고, encoder 만 남겨서, downstream task 추가 학습 (decoder 는 pre-trained 할 때만 사용된다)

MAE Details

- Masking
  - Following ViT, image 를 overlapping 되지 않게 patch. 그리고 일부 랜덤 patch 를 masked.
  - uniform distribution 에서 sampling (전반적으로 고르게 sample 될 수 있도록)
  - 굉장히 많은 비중이 masked 되기 때문에 단순히 extrapolation 으로 복원되기가 어렵다. (즉, 좋은 representation 을 학습하는 결과)
- Encoder
  - MAE encoder is a ViT. (masked 된 patch 는 빼고 encoding)
  - input 에 들어갈 때, ViT 와 동일하게 patch를 linear projection 하여 positional embeddings 추가.
  - 25% 정도만 가지고 학습하기 때문에 3배 정도 빠르게 학습할 수 있고, 더 큰 모델로 scale up (확장) 할 수 있다.
- Decoder
  - Encoder 에서 만든 patch + masked token
  - positional embedding 사용
  - decoder 는 pre-trained 에만 사용되고, 실제 downstream task 에서는 사용되지 않기 때문에 decoder 아키텍처를 encoder 와 완전히 independent 하게 가져갈 수 있다.

https://arxiv.org/pdf/2111.06377v1.pdf

MAE Details

- Reconstruction target
  - original image 복원하는 게 목적
  - loss function
    - MSE (mean squared error) between 복원된 pixel 에 대해서만. (즉, masked 되지 않은 pixel 에 대해서는 loss function 적용하지 않음)
    - The loss function computes the mean squared error (MSE) between the reconstructed and original images in the pixel space. Computing the loss only on masked patches, similar to BERT.

- Simple implementation
  - patch random shuffle 하여 75% 선택
  - decoding 에서는 다시 unshuffle 해서 원래 masking 된 자리에 mask token 을 넣는 방식으로 한다.

https://arxiv.org/pdf/2111.06377v1.pdf

Experiments

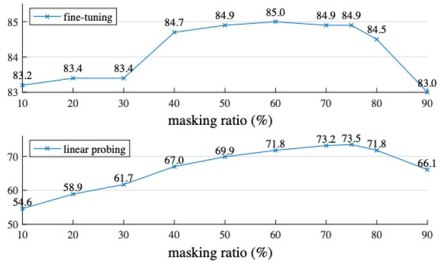


Figure 5. **Masking ratio**. A high masking ratio (75%) works well for both fine-tuning (top) and linear probing (bottom). The y-axes are ImageNet-1K validation accuracy (%) in all plots in this paper.

Masking ratio

blocks	ft	lin
1	84.8	65.5
2	**84.9**	70.0
4	**84.9**	71.9
8	**84.9**	**73.5**
12	84.4	73.3

(a) **Decoder depth**. A deep decoder can improve linear probing accuracy.

dim	ft	lin
128	**84.9**	69.1
256	84.8	71.3
512	**84.9**	**73.5**
768	84.4	73.1
1024	84.3	73.1

(b) **Decoder width**. The decoder can be narrower than the encoder (1024-d).

Decoder design (Encoder 와 독립적으로 디자인)

- *충분히 deep 한 decoder 가 필요하다.*
- *fine tuning 을 하면 큰 차이는 없다.*

case	ft	lin	FLOPs
encoder w/ [M]	84.2	59.6	3.3×
encoder w/o [M]	**84.9**	**73.5**	**1**×

(c) **Mask token**. An encoder without mask tokens is more accurate and faster (Table 2).

Encoder 에서 mask token 을 사용했을 때?

encoder	dec. depth	ft acc	hours	speedup
ViT-L, w/ [M]	8	84.2	42.4	-
ViT-L	8	84.9	15.4	2.8×
ViT-L	1	84.8	11.6	**3.7**×
ViT-H, w/ [M]	8	-	119.6†	-
ViT-H	8	85.8	34.5	3.5×
ViT-H	1	85.9	29.3	**4.1**×

Table 2. **Wall-clock time** of our MAE training (800 epochs), benchmarked in 128 TPU-v3 cores with TensorFlow. The speedup is relative to the entry whose encoder has mask tokens (gray). The decoder width is 512, and the mask ratio is 75%. †: This entry is estimated by training ten epochs.

Training speed up

case	ft	lin
pixel (w/o norm)	84.9	73.5
pixel (w/ norm)	**85.4**	**73.9**
PCA	84.6	72.3
dVAE token	85.3	71.6

(d) **Reconstruction target**. Pixels as reconstruction targets are effective.

Reconstruction Target

https://arxiv.org/pdf/2111.06377v1.pdf

Experiments & results

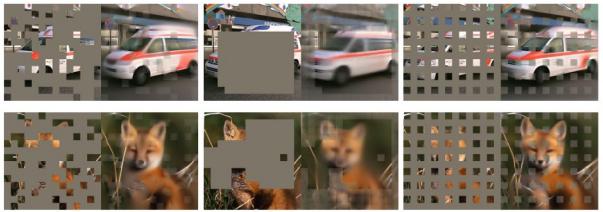


Figure 6. **Mask sampling strategies** determine the pretext task difficulty, influencing reconstruction quality and representations (Table 1f). Here each output is from an MAE trained with the specified masking strategy. Left: random sampling (our default). Middle: block-wise sampling [2] that removes large random blocks. Right: grid-wise sampling that keeps one of every four patches. Images are from the validation set.

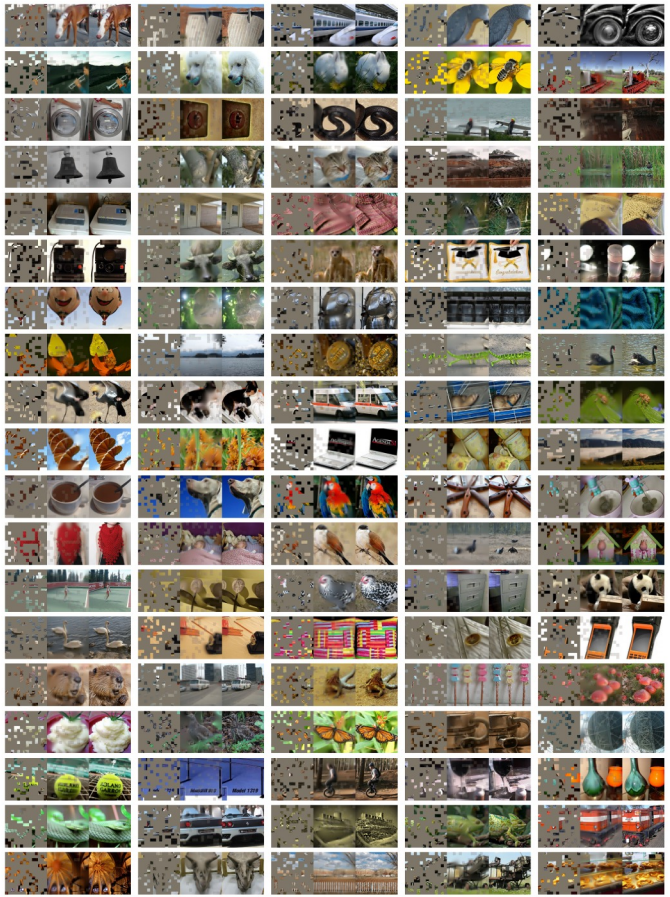


Figure 10. **Uncurated random samples** on ImageNet *validation* images. For each triplet, we show the masked image (left), our MAE reconstruction (middle), and the ground-truth (right). The masking ratio is 75%.

Gradients are Not All You Need

Luke Metz* **C. Daniel Freeman*** **Samuel S. Schoenholz**
Google Research, Brain Team
{lmetz, cdfreeman, schsam}@google.com

Tal Kachman
Radboud University
Donders Institute for Brain, Cognition and Behaviour
tal.kachman@donders.ru.nl

https://arxiv.org/pdf/2111.05803.pdf

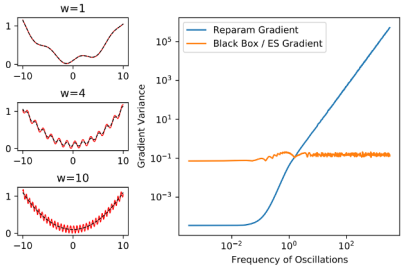


Figure 1: Sometimes, black box gradient estimates can result in lower variance gradient estimates. On the left, we plot $l(x) = 0.1\sin(xw/(\pi)) + (x/10)^2 + 0.1$ for different values of w in red, as well as the loss smoothed by convolving with a 0.3 std Gaussian. On the figure to the right we show the max gradient variance computed over all $x \in [-10, 10]$. When the frequency of oscillations grows higher, the reparameterization gradient variance also grows while the back box gradient remains constant.

파란색 : Reparam Gradient
주황색 : Black Box / ES Gradient

x축 : Gradient Variance
항상 한 개의 Gradient 가 일정한 Gradient Variance 성능을 보이는 게 아니라,
각 상황에 따라 다른 Gradient Variance 성능을 가지는 Gradient 가 존재한다!

- deep learning techniques는 task에 맞는 신경망 구축 후 미분 가능하게 만들고 파이프라인의 적절한 위치에 배치하여 “end to end”로 훈련하기만 하면 됨.
- 이를위해 다양한 Automatic differentiation(i.e, pytorch auto_grad) 로 간단한 handling이 가능하지만, **제한된 계산 및 메모리 오버헤드가 함께 종종 발생.**
  - ⇒ 수학적으로 의미하는 기울기는 공식적으로는 “정확”할 수 있지만 알고리즘 적으로는 유용하지 않을 수 있음.
- In this work, we discuss one potential issue that arises when working with **iterative differentiable systems (⇒ 이거는 Chaos 한 성격을 가짐),**
  즉, 계속 반복적으로 미분을 계산하는 시스템 (iterative differentiable systems) 안에서 어떤 이슈가 있는지 보고싶다.
- **다양한 미분상황 속 (여러 tasks)** 에서 어떤 potential issue가 있는지 discuss.

- In this work, we discuss one potential issue that arises when working with **iterative differentiable systems (⇒ 이거는 Chaos 한 성격을 가짐),**

  즉, 계속 반복적으로 미분을 계산하는 시스템 (iterative differentiable systems) 안에서 어떤 이슈가 있는지 보고싶다.
- **다양한 미분상황 속 (여러 tasks)** 에서 어떤 potential issue가 있는지 discuss.
  - recurrent neural networks
  - numerical physics simulation
  - training learned optimizers.

Rigid Body Physics

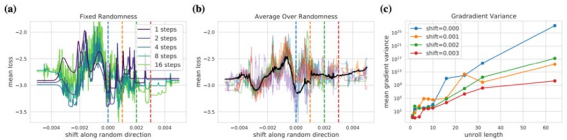


Figure 2: Loss surface and gradient variance for a stochastic policy on a a robotics control task – the Ant environment from Brax. **(a)**: We show a 1D projection of the loss surface along a random direction. All randomness in this plot is fixed. Color denotes different lengths of unroll when computing the loss. For small numbers of iterations the loss is smooth. For higher numbers of steps the underlying loss becomes highly curved. **(b)**: Instead of fixing randomness as done in the left plot, we average over multiple random samples for the 8 step unroll (average is in black, samples are in colors). We find that averaging greatly smooths the underlying loss surface. **(c)**: We look at gradient variance of gradients computed over multiple random samples from the stochastic policy. We show three different parameter values (shifts corresponding to the x-axis in the first two plots and are denoted with the same color vertical dashed lines). Despite having a seemingly smooth loss surface, the gradient variance explodes in exponential growth.

Meta-learning

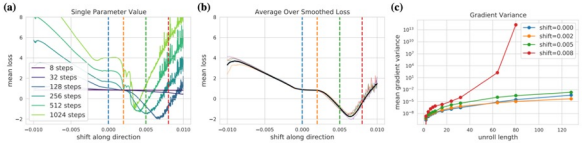


Figure 3: Loss surface and gradient variance calculations for meta-learning an optimizer. **(a)**: We show a 1D projection of the meta-loss surfaces (loss with respect to learned optimizer parameters) for different length unrolls – in this case, different numbers of application of the learned optimizer. For small numbers of steps, we find a smooth loss surfaces, but for higher numbers of steps we see a mix of smooth, and high curvature regions. **(b)**: We show an average of the meta-loss over Gaussian perturbed learned optimizer weights. The average is shown in black, and the losses averaged over are shown in color. We find this averaged loss is smooth and appears well behaved. **(c)**: We plot gradient variance over the different perturbations of the learned optimizer weights. These perturbations are shifts corresponding to the x-axis in the first two figures and are marked there with colored dashed vertical lines. For some settings of the learned optimizer weights (corresponding to the x-axis of the first 2 figures) we find well behaved gradient variance. For others, e.g. red, we find exponential growth in variance

Molecular Dynamics

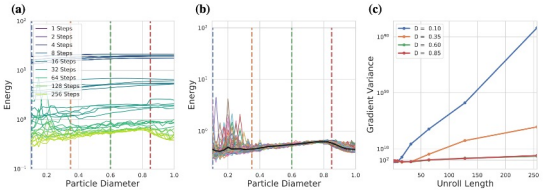


Figure 4: Energy for packings of bi-disperse disks varying the diameter of the small disk, D, and the number of optimization steps. **(a)**: The energy of the system as a function of D for different numbers of optimization steps. We see that the energy decreases with more steps of optimization. **(b)**: The energy for the maximum number of optimization steps considered (256). Each individual curve is the energy for one random configuration and the black line indicates the energy averaged over many random seeds. **(c)**: The variance of the gradient estimate for different values of D as a function of the number of steps of optimization.

Table 1: Different kinds of machine learning techniques often resemble differentiating through some iterative system. f is the iteratively applied function, s is an input, θ stands for parameters and l is the optimization objective

Domain	f	s	θ	l
Neural Network Training	A layer transformation of a neural network	The inputs to that layer	The weight matrix and bias vector for that layer	cross entropy mean squared error, l2 regularization, etc.
Reinforcement Learning	The step function of an environment	The state data of the environment and agent	The parameters of a policy	The reward function for the environment
Learned Optimization	The application of an optimizer	The parameters in a network being optimized	The tunable parameters for the optimizer, e.g., learning rate	The performance of the network being optimized on a task after some number of steps of optimization

- 현재 input을 다음 input으로 transformation 하는 A
- largest eigenvalue is less than one, trajectories will tend to vanish.(?)
- 아마 왼쪽은 Reinforcement Learning 의 각 state?
- 가운데는 transition function? 오른쪽은 NN sum of loss function? (l은 loss function, N은 number of iteration)

$$s_{k+1} = A_k s_k \qquad s_{t+1} = f(s_t, \theta) \qquad l(\theta) = \sum_{t=0}^{N} l_t(s_t, \theta).$$

- Loss sum이므로 사실 모든 iteration (t)에서 발생시키는 loss를 줄여야 함
- 각 step 에서 발생시키는 gradient loss ⇒ Generalize func

$$\frac{\mathrm{d}l_0}{\mathrm{d}\theta} = \frac{\partial l_0}{\partial s_0}\frac{\partial s_0}{\partial \theta} + \frac{\partial l_0}{\partial \theta}$$
$$\frac{\mathrm{d}l_1}{\mathrm{d}\theta} = \frac{\partial l_1}{\partial s_1}\frac{\partial s_1}{\partial s_0}\frac{\partial s_0}{\partial \theta} + \frac{\partial l_1}{\partial s_1}\frac{\partial s_1}{\partial \theta} + \frac{\partial l_1}{\partial \theta}$$
$$\frac{\mathrm{d}l_2}{\mathrm{d}\theta} = \frac{\partial l_2}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial s_0}\frac{\partial s_0}{\partial \theta} + \frac{\partial l_2}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial \theta} + \frac{\partial l_2}{\partial s_2}\frac{\partial s_2}{\partial \theta} + \frac{\partial l_2}{\partial \theta}$$

$$\frac{\mathrm{d}l_t}{\mathrm{d}\theta} = \frac{\partial l_t}{\partial \theta} + \sum_{k=1}^{t} \frac{\partial l_t}{\partial s_t}\left(\prod_{i=k}^{t} \frac{\partial s_i}{\partial s_{i-1}}\right)\frac{\partial s_k}{\partial \theta}$$

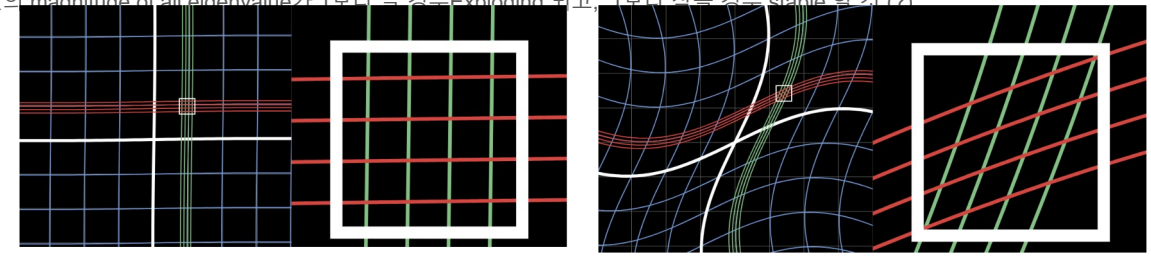
total loss:

$$\frac{\mathrm{d}l}{\mathrm{d}\theta} = \frac{1}{N}\sum_{t=0}^{N}\left[\frac{\partial l_t}{\partial \theta} + \sum_{k=1}^{t} \frac{\partial l_t}{\partial s_t}\left(\prod_{i=k}^{t} \frac{\partial s_i}{\partial s_{i-1}}\right)\frac{\partial s_k}{\partial \theta}\right]$$

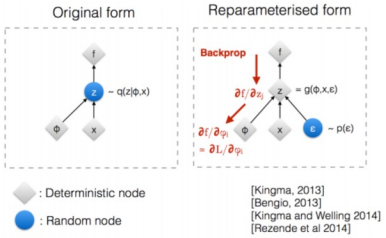
- jacorbian of the dynamical system (f) and precisely the iterated structure dicussed in the beginning of this section.
- 자코비안 행렬? '비선형 변환'을 선형 변환'으로 근사 시킨 것 (비선형 변환을 국소적으로 관찰하면 변환 후에도 격자들 직선 형태에 가까움).
- 이에 gradient of loss function이 jacobian 행렬의 스펙트럼에 밀접하게 의존하다는 사실.
- N grows, 해당 연산이 쌓일수록 Vanishing Gradient or Exploding Gradient될 것.
- 해당 값의 magnitude of all eigenvalue가 1보다 클 경우Exploding 되고, 1보다 작을 경우 stable 할 것 (?)

total loss:

$$\frac{\mathrm{d}l}{\mathrm{d}\theta} = \frac{1}{N}\sum_{t=0}^{N}\left[\frac{\partial l_t}{\partial \theta} + \sum_{k=1}^{t}\frac{\partial l_t}{\partial s_t}\left(\prod_{i=k}^{t}\frac{\partial s_i}{\partial s_{i-1}}\right)\frac{\partial s_k}{\partial \theta}\right]$$



- 위와 같이 loss는 mini batch 내의 data different나 randomness에 따라 항상 다른 값을 가질 것. ⇒ Train variational 존재.
- 상황에 따라 항상 달라지는 Train Variational을 해결하고자 The Reparameterization Trick
  - Encoder - Decoder 구조의 latent vector를 생각해보자, 학습하여 representation하는 latent vector를 생성하는 Encoder
  - 반면 Variational Encoder는 Latent vector의 각 원소에 (mean, var)로 이루어진 gaussian distribuction을 사용하여 Latent vector를 그려냄.



VAE idea? Stochastic mode를 stocastic한 부분과 deterministic 한 부분으로 분해시켜서 deterministic 한 부분으로 backpropagation을 흐르게 하자

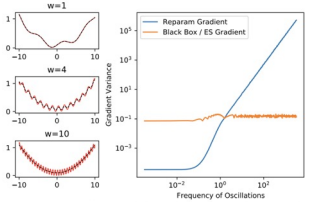


Figure 1: Sometimes, black box gradient estimates can result in lower variance gradient estimates. On the left, we plot $l(x) = 0.1\sin(xw/(\pi)) + (x/10)^2 + 0.1$ for different values of w in red, as well as the loss smoothed by convolving with a 0.3 std Gaussian. On the figure to the right we show the max gradient variance computed over all $x \in [-10, 10]$. When the frequency of oscillations grows higher, the reparameterization gradient variance also grows while the back box gradient remains constant.

- black box gradient estimates can results in lower variance gradient estimates.
- 왼쪽 그림은, sine wave function 을 w의 변화에 따라 그린 그래프 (gausian 0.3 convolve)
- w가 증가하면 더 쭈글쭈글한 그래프모형 ⇒ w=1 일때, reparameteization trick works good
  ⇒ blue things is the true objective you want to estimate the gradient from
  ⇒ reparameterization trick gradient is red
- when the freqnecy of oscillations grow higer,
  reparameteriztion gradient variance also grows higer
  반면, black box gradient ramains constant

- **In this paper we dive into chaos as a potential issue when computing gradients through dynamical systems.**
- Finally, many systems of interest could have extremely flat, or hard to explore loss landscapes making gradient based learning extremely tricky.

- **Part of the reason gradient descent works in neural networks is due to over parameterization [Kawaguchi, 2016] and known weights prior/initialization, which is often not possible in simulated systems have.**
    - Gradient descent 가 잘 작동하는 이유는, Gradient descent 가 실제로 좋은 영향을 미치고 있어서가 아니라, over parameterization, weights initialization 이 잘 되어있기 때문일 수도 있겠다.
- **In summary, gradients are not all you need. Just because you can take a gradient doesn't mean you always should.**

Data Augmentation Can Improve Robustness

Sylvestre-Alvise Rebuffi*, Sven Gowal*, Dan Calian,
Florian Stimberg, Olivia Wiles and **Timothy Mann**
DeepMind, London
{sylvestre,sgowal}@deepmind.com

https://arxiv.org/pdf/2111.05328v1.pdf

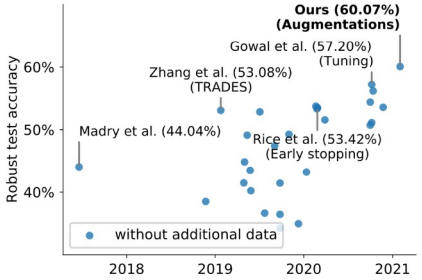


Figure 1: Robust accuracy of various models submitted to RobustBench [11] against AUTOATTACK [10] on CIFAR-10 with ℓ_∞ perturbations of size $8/255$ displayed in publication order. Our method builds on Gowal et al. [20] (shown above with 57.20%) and explores how augmented data can be used to improve robust accuracy by +2.87% without using any additional external data.

Robust accuracy 확인

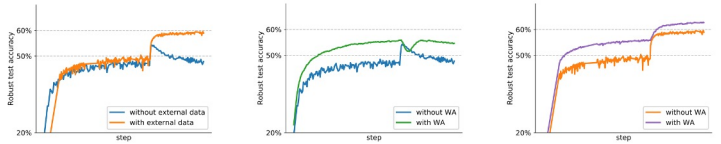
- *against AUTOATTACK on CIFAR-10*
- *Out method builds on Gowal et al. and explores how augmented data can be used to improve robust accuracy by +2.87% without using any additional external data.*

- **DeepMind, Adversarial training (Robust accuracy)**
- **NeurIPS 2021**

- Adversarial training 은 대부분 robust overfitting 에 의해서 야기됨.
  - robust overfitting : robust test accuracy 가 training 동안에 감소하는 현상.
- **본 연구에서는, 일반적인 data augmentation 기법을 사용함으로써, robust overfitting 을 감소시키고자 함**.
  - **model weight averaging 와 결합했을 때, data augmentation 이 robust accuracy 를 boost 하는데 도움이 됨.**
  - 기존 SoTA 성능과 비교했을 때, robust accuracy 가 +2.93%, +2.16% 증가함을 보임. (CIFAR-10 dataset)

- 다른 아키텍처를 사용하거나 다른 데이터 셋 (CIFAR-100, SVHNm TINYIMAGENET) 을 사용했을 때도 이러한 성능 향상 확인.

- **Preliminaries and hypothesis**
  - **Adversarial training**
  - **Robust overfitting**
    - train set 에서의 robust accuracy 는 증가하지만, test set 의 robust accuracy 는 감소하는 현상 (이 때, train/test set 의 clean accuracy 는 모두 잘 증가함)
  - **Model weight averaging** (Averaging Weights Leads to Wider Optima and Better Generalization)
    - optimization & regularization
    - Optimization 을 진행할 때, 일정 주기마다 weight 를 average 하여 w를 업데이트 시키는 방법
    - 여러 주기마다 weight 를 업데이트시키는 것은 local solutions 를 ensemble 하는 것과 같음. 따라서 어느 한쪽에 쏠리는 것을 막아 regularize 를 수행하며, general 한 solution 을 찾게 됨.
    - WA 가 robust overfitting 현상을 완화시킴.
    - Figure 2(b) 는 training 단계에서 WA 를 사용했을 때, robust accuracy 가 얼만큼 영향을 받는지.
  - **Hypothesis**
    - **Model weight averaging works better when robust overfitting is reduced.**



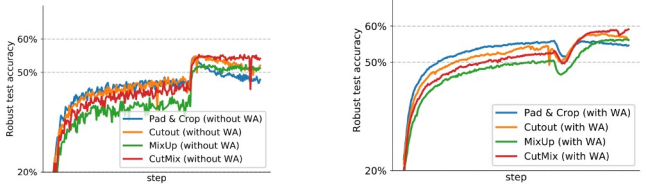
(a) Adversarial training with and without additional data from 80M-TI (without WA)

(b) Effect of WA without external data

(c) Effect of WA with external data

Figure 2: We compare the robust accuracy against $\epsilon_\infty = 8/255$ on CIFAR-10 of an adversarially trained Wide ResNet (WRN)-28-10. Panel (a) shows the impact of using additional external data from 80M-TI [50] and illustrates *robust overfitting*. Panel (b) shows the benefit of *model weight averaging* (WA) despite robust overfitting. Panel (c) shows that WA remains effective and useful even when robust overfitting disappears. The graphs show the evolution of the robust accuracy as training progresses (against PGD^{40}). The jump two-thirds through training is due to a drop in learning rate.

- **Data augmentations can be used to verify this hypothesis**



(a) Without WA (b) With WA

Figure 3: Accuracy against $\epsilon_\infty = 8/255$ on CIFAR-10 with and without using model weight averaging (WA) for different data augmentation schemes. The model is a WRN-28-10 and both panels show the evolution of the robust accuracy as training progresses (against PGD^{40}). The jump in robust accuracy two-thirds through training is due to a drop in learning rate.

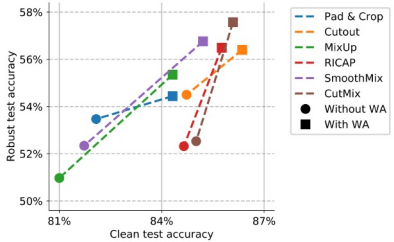


Figure 4: Clean (without adversarial attacks) accuracy and robust accuracy (against AA+MT) for a WRN-28-10 trained against $\epsilon_\infty = 8/255$ on CIFAR-10 for different data augmentation techniques. The lines from circles to squares represent the performance change obtained when using WA.

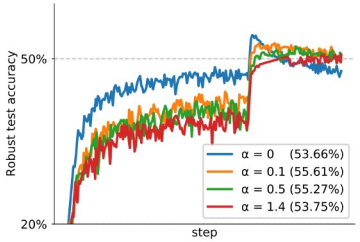


Figure 5: The graph shows the robust test accuracy against PGD^{40} with $\epsilon_\infty = 8/255$ on CIFAR-10 without using WA as we vary the mixing rate α of *MixUp*. We report in the legend the robust accuracy (against AA+MT) after applying weight averaging to the corresponding runs.

- *Figure 3) Data augmentation & WA 적용 여부에 따른 robust accuracy 측정*
- *Figure 4) Data augmentation 이 clean accuracy, robust accuracy 에 미치는 영향 & WA 적용 여부*
- *Figure 5) without using WA & mixing rate 별 robust accuracy*

A Survey on Vision Transformer	3	University of Sydney	1. Dacheng Tao 2. Yunhe Wang	1. 136 2. 26

A Survey on Vision Transformer

Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, Zhaohui Yang, Yiman Zhang, and Dacheng Tao *Fellow, IEEE*

Abstract—Transformer, first applied to the field of natural language processing, is a type of deep neural network mainly based on the self-attention mechanism. Thanks to its strong representation capabilities, researchers are looking at ways to apply transformer to computer vision tasks. In a variety of visual benchmarks, transformer-based models perform similar to or better than other types of networks such as convolutional and recurrent networks. Given its high performance and less need for vision-specific inductive bias, transformer is receiving more and more attention from the computer vision community. In this paper, we review these vision transformer models by categorizing them in different tasks and analyzing their advantages and disadvantages. The main categories we explore include the backbone network, high/mid-level vision, low-level vision, and video processing. We also include efficient transformer methods for pushing transformer into real device-based applications. Furthermore, we also take a brief look at the self-attention mechanism in computer vision, as it is the base component in transformer. Toward the end of this paper, we discuss the challenges and provide several further research directions for vision transformers.

Index Terms—Transformer, Self-attention, Computer Vision, High-level vision, Low-level vision, Video.

https://arxiv.org/pdf/2012.12556.pdf

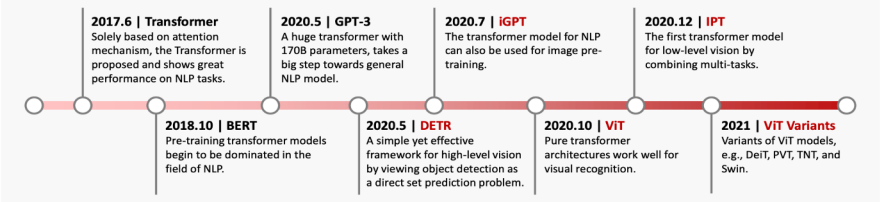


Fig. 1: Key milestones in the development of transformer. The vision transformer models are marked in red.

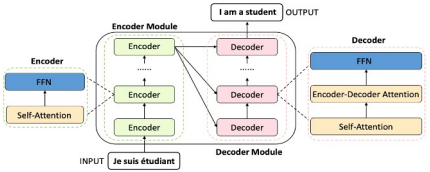


Fig. 2: Pipeline of vanilla transformer.

- **Vision Transformer**
  - 최근 많은 ViT 연구가 소개되고 있음.
  - 본 논문에서는, 여러 tasks 로 카테고리하여 ViT model 을 review 하고, 장 단점 분석.
  - Main categories
    - backbone network
    - high/mid-level vision,
    - low-level vision,
    - video processing
    - efficient transformer methods for pushing transformer into real device-based applications.

TABLE 1: Representative works of vision transformers.

Category	Sub-category	Method	Highlights	Publication
Backbone	Supervised pretraining	ViT [55]	Image patches, standard transformer	ICLR 2021
		DeiT [219]	Data-efficient training, ViT model	ICML 2021
		Swin [17]	Shifted window, window-based self-attention	ICCV 2021
	Self-supervised pretraining	iGPT [29]	Pixel prediction self-supervised learning, GPT model	ICML 2020
		MoCo v3 [32]	Contrastive self-supervised learning, ViT	ICCV 2021
High/Mid-level vision	Object detection	DETR [19]	Set-based prediction, bipartite matching, transformer	ECCV 2020
		Deformable DETR [291]	DETR, deformable attention module	ICLR 2021
		ACT [284]	Adaptive clustering transformer	arXiv 2020
		UP-DETR [49]	Unsupervised pre-training, random query patch detection	CVPR 2021
		TSP [210]	New bipartite matching, encoder-only transformer	arXiv 2020
	Segmentation	Max-DeepLab [228]	PQ-style bipartite matching, dual-path transformer	CVPR 2021
		VisTR [235]	Instance sequence matching and segmentation	CVPR 2021
		SETR [285]	sequence-to-sequence prediction, standard transformer	CVPR 2021
	Pose Estimation	Hand-Transformer [102]	Non-autoregressive transformer, 3D point set	ECCV 2020
		HOT-Net [103]	Structured-reference extractor	MM 2020
		METRO [138]	Progressive dimensionality reduction	CVPR 2021
Low-level vision	Image generation	Image Transformer [171]	Pixel generation using transformer	ICML 2018
		Taming transformer [58]	VQ-GAN, auto-regressive transformer	CVPR 2021
		TransGAN [111]	GAN using pure transformer architecture	arXiv 2021
	Image enhancement	IPT [27]	Multi-task, ImageNet pre-training, transformer model	CVPR 2021
		TTSR [251]	Texture transformer, RefSR	CVPR 2020
Video processing	Video inpainting	STTN [268]	Spatial-temporal adversarial loss	ECCV 2020
	Video captioning	Masked Transformer [288]	Masking network, event proposal	CVPR 2018
Multimodality	Classification	CLIP [180]	NLP supervision for images, zero-shot transfer	arXiv 2021
	Image generation	DALL-E [185]	Zero-shot text-to image generation	ICML 2021
		Cogview [51]	VQ-VAE, Chinese input	arXiv 2021
	Multi-task	UniT [100]	Different NLP & CV tasks, shared model parameters	arXiv 2021
Efficient transformer	Decomposition	ASH [159]	Number of heads, importance estimation	NeurIPS 2019
	Distillation	TinyBert [113]	Various losses for different modules	EMNLP Findings 2020
	Quantization	FullyQT [176]	Fully quantized transformer	EMNLP Findings 2020
	Architecture design	ConvBert [112]	Local dependence, dynamic convolution	NeurIPS 2020

4 Vision Transformer

In this section, we review the applications of transformer-based models in computer vision, including image classification, high/mid-level vision, low-level vision and video processing. We also briefly summarize the applications of the self-attention mechanism and model compression methods for efficient transformer.

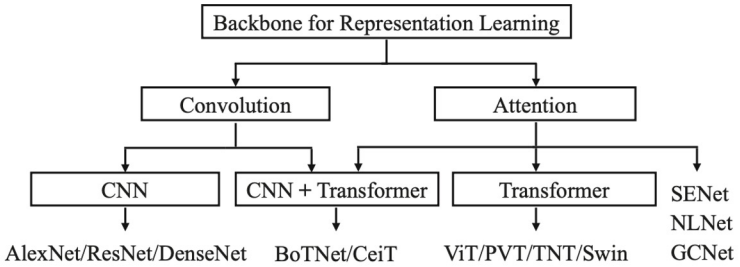


Fig. 5: A taxonomy of backbone using convolution and attention for representation learning.

TABLE 3: ImageNet result comparison of representative CNN and vision transformer models. Following [219], [148], the throughput is measured on NVIDIA V100 GPU and Pytorch, with 224×224 input size. Pure transformer means only using a few convolutions in the stem stage. CNN + Transformer means using convolutions in the intermediate layers.

Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
CNN				
ResNet-50 [89], [260]	25.6	4.1	1226	79.1
ResNet-101 [89], [260]	44.7	7.9	753	79.9
ResNet-152 [89], [260]	60.2	11.5	526	80.8
EfficientNet-B0 [213]	5.3	0.39	2694	77.1
EfficientNet-B1 [213]	7.8	0.70	1662	79.1
EfficientNet-B2 [213]	9.2	1.0	1255	80.1
EfficientNet-B3 [213]	12	1.8	732	81.6
EfficientNet-B4 [213]	19	4.2	349	82.9
Pure Transformer				
DeiT-Ti [55], [219]	5	1.3	2536	72.2
DeiT-S [55], [219]	22	4.6	940	79.8
DeiT-B [55], [219]	86	17.6	292	81.8
T2T-ViT-14 [260]	21.5	5.2	764	81.5
T2T-ViT-19 [260]	39.2	8.9	464	81.9
T2T-ViT-24 [260]	64.1	14.1	312	82.3
PVT-Small [232]	24.5	3.8	820	79.8
PVT-Medium [232]	44.2	6.7	526	81.2
PVT-Large [232]	61.4	9.8	367	81.7
TNT-S [85]	23.8	5.2	428	81.5
TNT-B [85]	65.6	14.1	246	82.9
CPVT-S [44]	23	4.6	930	80.5
CPVT-S-GAP [44]	23	4.6	942	81.5
CPVT-B [44]	88	17.6	285	82.3
Swin-T [148]	29	4.5	755	81.3
Swin-S [148]	50	8.7	437	83.0
Swin-B [148]	88	15.4	278	83.3
CNN + Transformer				
Twins-SVT-S [43]	24	2.9	1059	81.7
Twins-SVT-B [43]	56	8.6	469	83.2
Twins-SVT-L [43]	99.2	15.1	288	83.7
Shuffle-T [105]	29	4.6	791	82.5
Shuffle-S [105]	50	8.9	450	83.5
Shuffle-B [105]	88	15.6	279	84.0
XCiT-S12/16 [56]	26	4.8	781	83.3
CMT-S [77]	25.1	4.0	563	83.5
CMT-B [77]	45.7	9.3	285	84.5
VOLO-D1 [261]	27	6.8	481	84.2
VOLO-D2 [261]	59	14.1	244	85.2
VOLO-D3 [261]	86	20.6	168	85.4
VOLO-D4 [261]	193	43.8	100	85.7
VOLO-D5 [261]	296	69.0	64	86.1

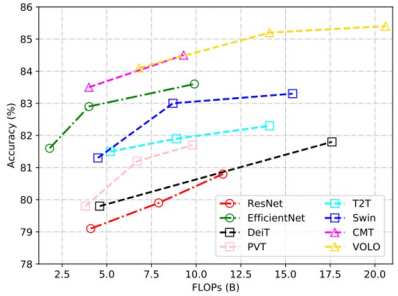


Fig. 7: FLOPs comparison of representative CNN and vision transformer models.

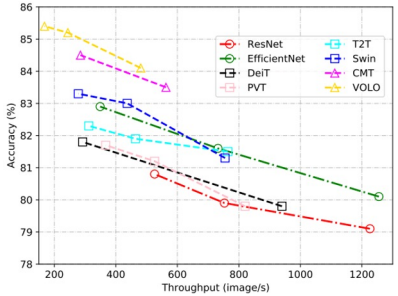


Fig. 8: Throughput comparison of representative CNN and vision transformer models.

Vision Transformer vs CNN

TABLE 4: Comparison of different transformer-based object detectors on COCO 2017 val set. Running speed (FPS) is evaluated on an NVIDIA Tesla V100 GPU as reported in [291]. †Estimated speed according to the reported number in the paper. ‡ViT backbone is pre-trained on ImageNet-21k. *ViT backbone is pre-trained on an private dataset with 1.3 billion images.

Method	Epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	#Params (M)	GFLOPs	FPS
CNN based										
FCOS [216]	36	41.0	59.8	44.1	26.2	44.6	52.2	-	177	23†
Faster R-CNN + FPN [186]	109	42.0	62.1	45.5	26.6	45.4	53.4	42	180	26
CNN Backbone + Transformer Head										
DETR [19]	500	42.0	62.4	44.2	20.5	45.8	61.1	41	86	28
DETR-DC5 [19]	500	43.3	63.1	45.9	22.5	47.3	61.1	41	187	12
Deformable DETR [291]	50	46.2	65.2	50.0	28.8	49.2	61.7	40	173	19
TSP-FCOS [210]	36	43.1	62.3	47.0	26.6	46.8	55.9	-	189	20†
TSP-RCNN [210]	96	45.0	64.5	49.6	29.7	47.7	58.0	-	188	15†
ACT+MKKD (L=32) [284]	-	43.1	-	-	61.4	47.1	22.2	-	169	14†
ACT+MKKD (L=16) [284]	-	40.6	-	-	59.7	44.3	18.5	-	156	16†
SMCA [71]	108	45.6	65.5	49.1	25.9	49.3	62.6	-	-	-
Efficient DETR [257]	36	45.1	63.1	49.1	28.3	48.4	59.0	35	210	-
UP-DETR [49]	150	40.5	60.8	42.6	19.0	44.4	60.0	41	-	-
UP-DETR [49]	300	42.8	63.0	45.3	20.8	47.1	61.7	41	-	-
Transformer Backbone + CNN Head										
ViT-B/16-FRCNN‡ [10]	21	36.6	56.3	39.3	17.4	40.0	55.5	-	-	-
ViT-B/16-FRCNN* [10]	21	37.8	57.4	40.1	17.8	41.4	57.3	-	-	-
PVT-Small+RetinaNet [232]	12	40.4	61.3	43.0	25.0	42.9	55.7	34.2	118	-
Twins-SVT-S+RetinaNet [43]	12	43.0	64.2	46.3	28.0	46.4	57.5	34.3	104	-
Swin-T+RetinaNet [148]	12	41.5	62.1	44.2	25.1	44.9	55.5	38.5	118	-
Swin-T+ATSS [148]	36	47.2	66.5	51.3	-	-	-	36	215	-
Pure Transformer based										
PVT-Small+DETR [232]	50	34.7	55.7	35.4	12.0	36.4	56.7	40	-	-
TNT-S+DETR [85]	50	38.2	58.9	39.4	15.5	41.1	58.8	39	-	-
YOLOS-Ti [64]	300	30.0	-	-	-	-	-	6.5	21	-
YOLOS-S [64]	150	37.6	57.6	39.2	15.9	40.2	57.3	28	179	-
YOLOS-B [64]	150	42.0	62.2	44.5	19.5	45.3	62.1	127	537	-

Vision Transformer object detection on COCO datasets

End of the Document