Visual Transformer in CV

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering &
Artificial Intelligence

References

Main

- https://towardsdatascience.com/implementing-visualttransformer-in-pytorch-184f9f16f632
- https://jalammar.github.io/illustrated-transformer/

blog Sub

- https://towardsdatascience.com/implementing-visualttransformer-in-pytorch-184f9f16f632

Newly tutorials

- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html

Main

- https://github.com/lucidrains/vit-pytorch
- https://github.com/FrancescoSaverioZuppichini/ViT
- https://pypi.org/project/vision-transformer-pytorch/

2 DeiT (Data-efficient Image Transformers)

[Facebook AI]
Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles and Hervé Jégou,
"Training data-efficient image transformers & distillation through attention," arXiv 2021

facebookresearch/deit
https://github.com/facebookresearch/deit

Optimizing Vision Transformer Model for Deployment
https://pytorch.org/tutorials/beginner/vt_tutorial.html

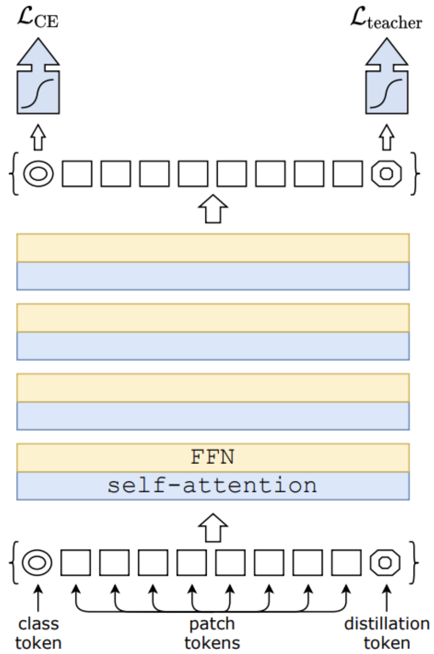
Abstract

- Recently, **neural networks purely based on attention** were shown to address **image classification**.
- These high performing **vision transformers** are **pre-trained with hundreds of millions of images** using a large infrastructure, thereby limiting their adoption.

❖ This work produces competitive **convolution-free transformers by training on Imagenet only**.
- **Train them on a single computer in less than 3 days**.
- **Our reference vision transformer (86M parameters) achieves top-1 accuracy of 83.1% (single-crop) on ImageNet with no external data**.
- Introduce a **teacher-student strategy specific to transformers**. It **relies on a distillation token ensuring that the student learns from the teacher through attention**.
- We show the interest of this **token-based distillation**, especially when using a convnet as a teacher. This leads us to report results competitive with convnets for both Imagenet (where we obtain up to 85.2% accuracy) and when transferring to other tasks.

2 DeiT (Data-efficient Image Transformers)

- **ViT** : Pre-trained with 100M images (large infrastructure) - limiting their adoption.
  - Excellent results trained with JFT-300M, 300M images
  - They concluded that transformers *"do not generalize well when trained on insufficient amounts of data"*.
- **DeiT** : Training on Imagenet only, **Single node with 4GPU in 3 days**
  - 86M parameters : top-1 accuracy 83.1% (single-crop) on ImageNet without external data
  - **Teacher-student strategy** : a **distillation token** ensuring that the student learns from the teach through attention



Model Zoo

We provide baseline DeiT models pretrained on ImageNet 2012.

name	acc@1	acc@5	#params	url
DeiT-tiny	72.2	91.1	5M	model
DeiT-small	79.9	95.0	22M	model
DeiT-base	81.8	95.6	86M	model
DeiT-tiny distilled	74.5	91.9	6M	model
DeiT-small distilled	81.2	95.4	22M	model
DeiT-base distilled	83.4	96.5	87M	model
DeiT-base 384	82.9	96.2	87M	model
DeiT-base distilled 384 (1000 epochs)	85.2	97.2	88M	model
CaiT-S24 distilled 384	85.1	97.3	47M	model
CaiT-M48 distilled 448	86.5	97.7	356M	model

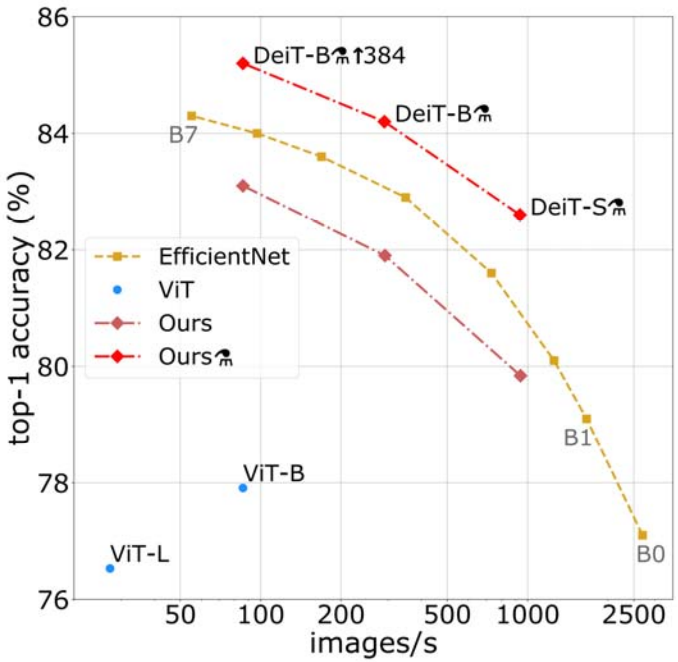


Figure 1: Throughput and accuracy on Imagenet of our methods compared to EfficientNets, trained on Imagenet1k only. The throughput is measured as **the number of images processed per second on a V100 GPU**. DeiT-B is identical to VIT-B, but the training is more adapted to a data-starving regime. It is learned in a few days on one machine. The symbol ⚗ refers to models trained with our transformer-specific distillation. See Table 5 for details and more models.

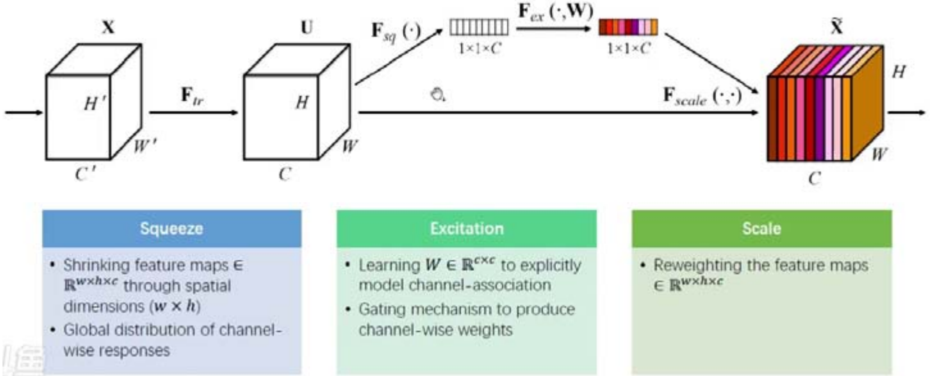
2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "[Training data-efficient image transformers & distillation through attention](#)," arXiv 2021

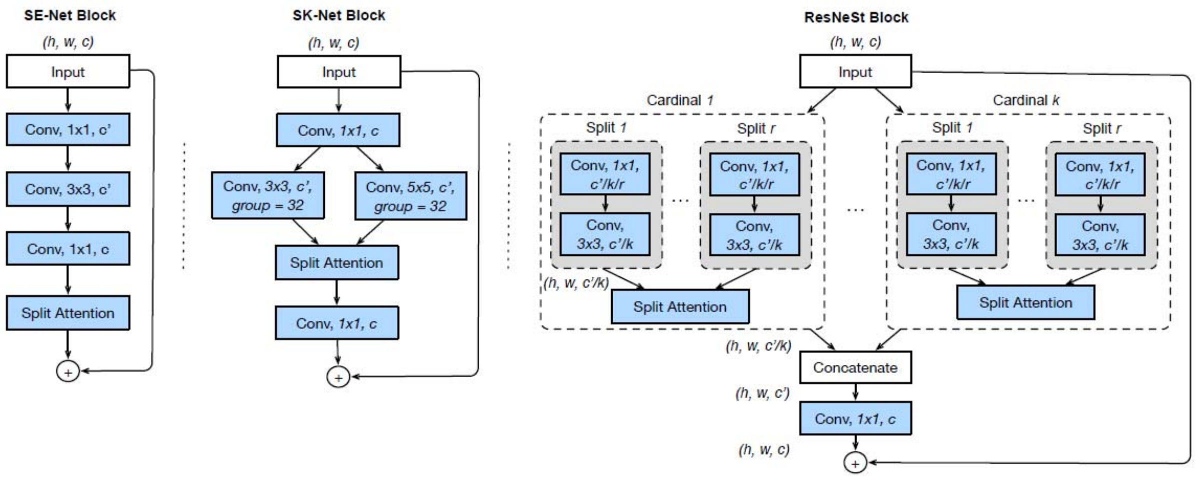
Related Works : Transformer Architecture

Transformer

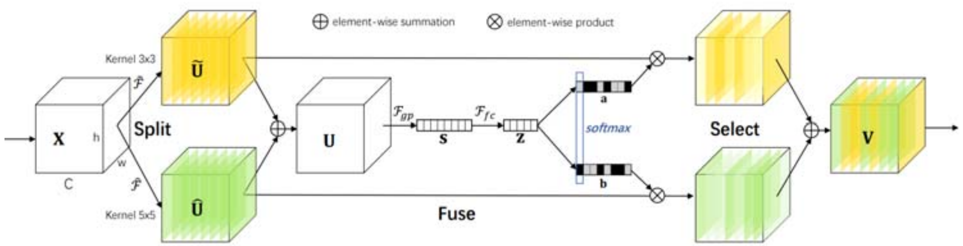
- Introduce by Vaswani et al. (Attention is All you need, 2017) for machine translation
- Currently the reference model for all NLP tasks
- Many improvements of convnets for image classification are inspired by transformers. Ex) Squeeze & Excitation, Selective Kernel, Split-attention networks exploit mechanism akin to transformers self-attention (SA) mechanism.



Squeeze-and-excitation (SE block) (2017)



Split-Attention Networks (2020)



Selective-Kernel Networks (SK-net) (2019)

2 DeiT (Data-efficient Image Transformers)

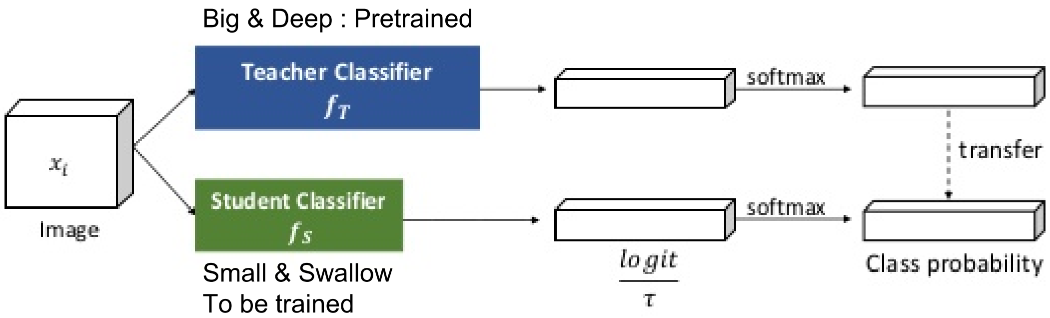
Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Related Works : Knowledge Distillation (KD)

- Introduce Hinton et al. (“Distilling the Knowledge in a neural network”, 2015)

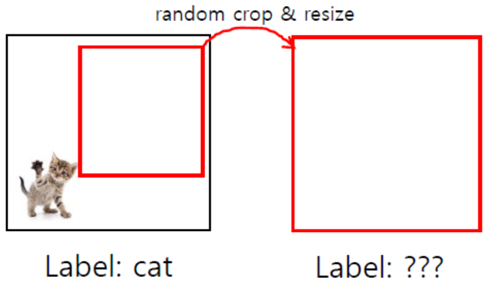
✓ **KD refers to the training paradigm in which a student model** leverages **"soft" labels** coming from **a strong teacher network** → This is the **output vector of teacher's softmax function**, rather than just the maximum of scores, which give **"hard" labels**

Objective: $\sum_{x_i \in \mathcal{X}} \mathrm{KL}\left(\mathrm{softmax}\left(\frac{f_T(x_i)}{\tau}\right), \mathrm{softmax}\left(\frac{f_S(x_i)}{\tau}\right)\right)$



[출처] slideshare, Wonpyo Park, "Relational knowledge distillation"
[출처] PR12 Paper Review, Jinwon Lee, PR-297 DeiT

- By Wei et al.[54] (“Circumventing Outliers of AutoAugment with Knowledge Distillation”, 2020)
- The teacher's supervision takes into account the effects of the data augmentation, which sometimes causes a misalignment between the real label and the image.



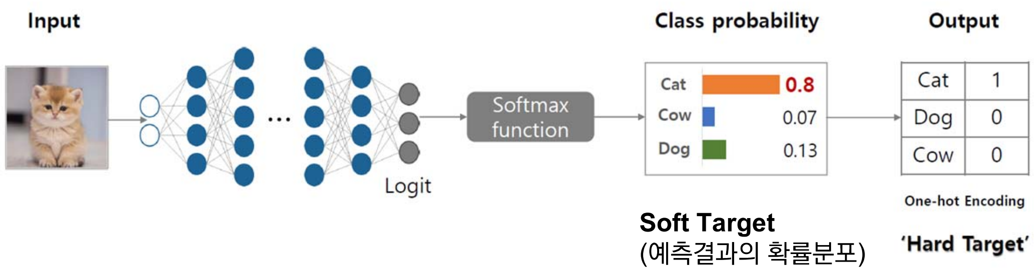
- Abnar et al. (Transferring Inductive Biases through Knowledge Distillation, 2020)
- KD can transfer inductive biases in a soft way in a student model using a teacher model where they would be incorporated in a hard way. → Useful to induce biases due to convolutions in a transformer model by **using a convolutional model as a teacher**.

2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "[Training data-efficient image transformers & distillation through attention](#)," arXiv 2021

Related Works : Knowledge Distillation (KD)

Knowledge : Soft Target

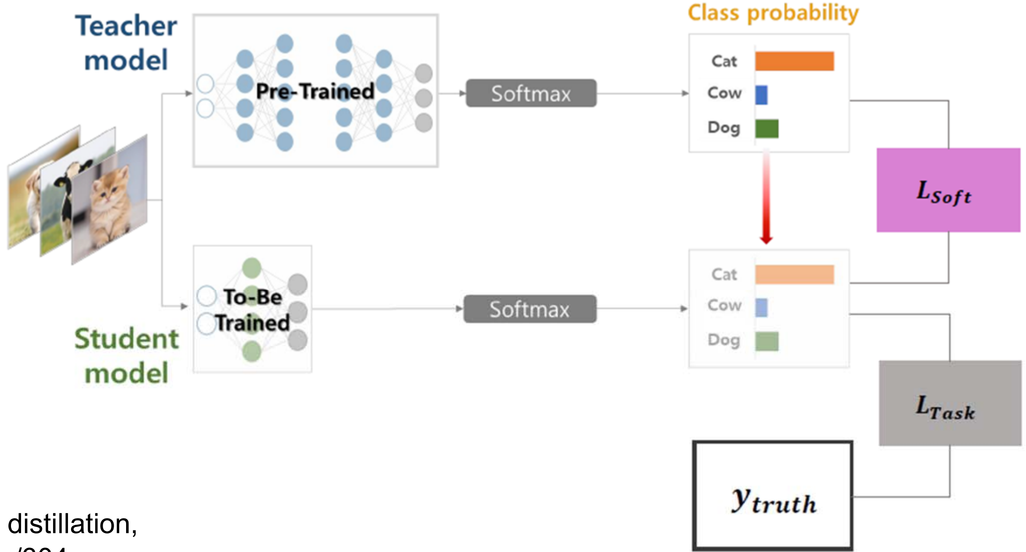


$$Softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \Rightarrow Softmax(z_i) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

τ $(Temperature)$: Scaling 역할의 하이퍼 파라미터

- τ = 1일 때, 기존 softmax function과 동일
- τ클수록, 더 soft한 확률분포

Distillation Method :
Offline distillation (Response-based)



- $f_T(x_i)$: Teacher 모델의 logit 값
- $f_T(x_i)$: Student 모델의 logit 값
- τ :Scaling 역할의 하이퍼 파라미터

$$L_{Soft} = \sum_{x_i \in X} KL(softmax(\frac{f_T(x_i)}{\tau}), softmax(\frac{f_S(x_i)}{\tau}))$$

$$L_{Task} = CrossEntropy\ (softmax(f_s(x_i)), y_{truth})$$

$$Student\ L_{Total} = L_{Task} + \lambda \cdot L_{Soft}$$

Response-Based
Feature-Based
Relation-Based

[출처] 황하은, Introduction to knowledge distillation, http://dmqm.korea.ac.kr/activity/seminar/304

DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Vision Transformer - Same Architecture as ViT

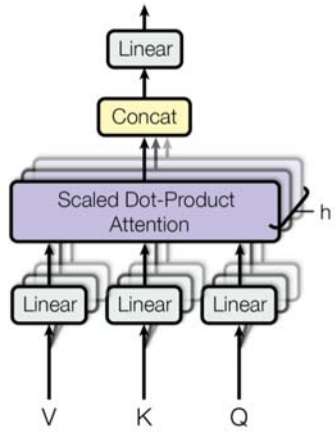
Multi-head Self-Attention Layers (MSA)

Multi-head Self Attention layers (MSA). The attention mechanism is based on a trainable associative memory with (key, value) vector pairs. A *query* vector $q \in \mathbb{R}^d$ is matched against a set of k *key* vectors (packed together into a matrix $K \in \mathbb{R}^{k \times d}$) using inner products. These inner products are then scaled and normalized with a softmax function to obtain k weights. The output of the attention is the weighted sum of a set of k *value* vectors (packed into $V \in \mathbb{R}^{k \times d}$). For a sequence of N query vectors (packed into $Q \in \mathbb{R}^{N \times d}$), it produces an output matrix (of size $N \times d$):

$$\text{Attention}(Q, K, V) = \text{Softmax}(QK^\top/\sqrt{d})V, \tag{1}$$

where the Softmax function is applied over each row of the input matrix and the $\sqrt{d}$ term provides appropriate normalization.

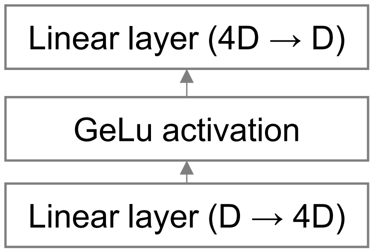
In [52], a Self-attention layer is proposed. Query, key and values matrices are themselves computed from a sequence of N input vectors (packed into $X \in \mathbb{R}^{N \times D}$): $Q = XW_Q$, $K = XW_K$, $V = XW_V$, using linear transformations W_Q, W_K, W_V with the constraint $k = N$, meaning that the attention is in between all the input vectors.
Finally, Multi-head self-attention layer (MSA) is defined by considering h attention "heads", *ie* h self-attention functions applied to the input. Each head provides a sequence of size $N \times d$. These h sequences are rearranged into a $N \times dh$ sequence that is reprojected by a linear layer into $N \times D$.



PxP : 16x16
N : 224/16 x 224/16 = 14x14
D : 16x16x3 = 768
h : 8, d = 96

Transformer block for images

- We add a **Feed-Forward Network (FFN)** on **top of MSA layer.**



D : 16x16x3 = 768

- Both MSA and FFN are operating as **residual operation** (thanks to skip-connection), and with a **layer normalization**.

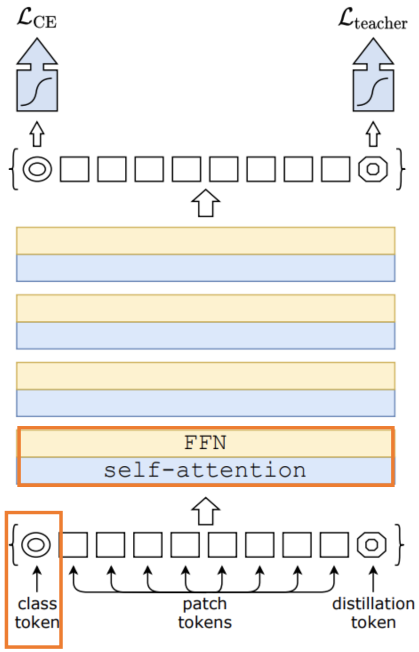
2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Vision Transformer - Same Architecture as ViT

Class token

- Trainable vector, appended to the patch token before the first layer, → goes through the transformer layer → projected with a linear layer to predict the class.
- Transformer process batches of $(N + 1)$ tokens of dimension D, of which only the class token is used to predict the output.
- Forces the self-attention to spread information between the patch tokens and the class token.
- At training time, the **supervision signal** comes only from the **class embedding**, while the **patch tokens** are the **model's only variable output**.



Fixing the positional encoding across resolution

- Tourvron et al. (*Fixefficientnet*, 2020) show that it is desirable to use a lower training resolution and fine-tune the network the larger resolution
  - ✓ Speed up the full training and improves the accuracy under prevailing data augmentation schemes.

→ When increasing the resolution of an input image, patch size does not change, therefore the number of input patches(N) does change. One need to adapt the positional embeddings.

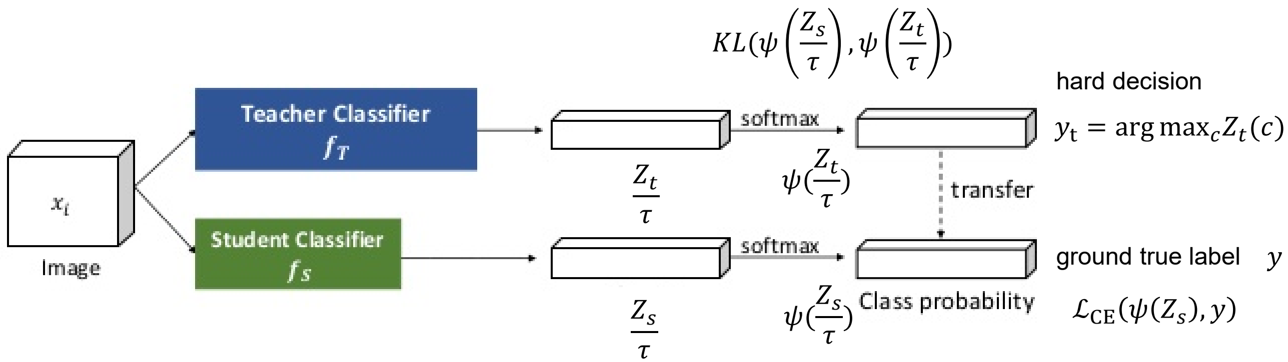
→ Dosovitaskiy et al. [15] (ViT) interpolate the positional encoding when changing the resolution. → Work with the subsequent fine-tuning stage.

DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "[Training data-efficient image transformers & distillation through attention](https://arxiv.org/abs/2012.12877)," arXiv 2021

Distillation through Attention

- Assume that **a strong image classifier as a teacher model**
- **Hard distillation** vs **Soft distillation**,
- **Classical distillation** vs **Distillation token**



1) Soft distillation

- Minimizes the Kullback-Leiber divergence between the teacher's softmax and the student's softmax.
- Distillation objective

$$\mathcal{L}_{\text{global}} = (1-\lambda)\mathcal{L}_{\text{CE}}(\psi(Z_s), y) + \lambda\tau^2 KL(\psi\left(\frac{Z_s}{\tau}\right), \psi\left(\frac{Z_t}{\tau}\right))$$

✓ Z_t , Z_s : the logits of teacher and student models

✓ τ : the temperature for the distillation

✓ λ : the coefficient balancing the KL divergence loss ($KL()$) and the cross-entropy ($\mathcal{L}_{CE}$) on ground truth labels y

✓ ψ : softmax ft.

2) Hard distillation

- Take the hard decision of the teacher ($y_t = \arg\max_c Z_t(c)$) as a true label
- Hard-label distillation based objective

$$\mathcal{L}_{\text{global}}^{\text{hardDistill}} = \frac{1}{2}\mathcal{L}_{\text{CE}}(\psi(Z_s), y) + \frac{1}{2}\mathcal{L}_{\text{CE}}(\psi(Z_s), y_t)$$

✓ For a given image, the hard label associated with the teacher may change depending on the specific data augmentation.

✓ The teacher prediction y_t plays the same role as the true label y.

- The hard-label can be converted into soft labels with label smoothing, where the true label is considered to have a probability $1-\epsilon$ (ϵ=0.1) and the remaining ϵ is shared across the remaining classes.

2 DeiT (Data-efficient Image Transformers)

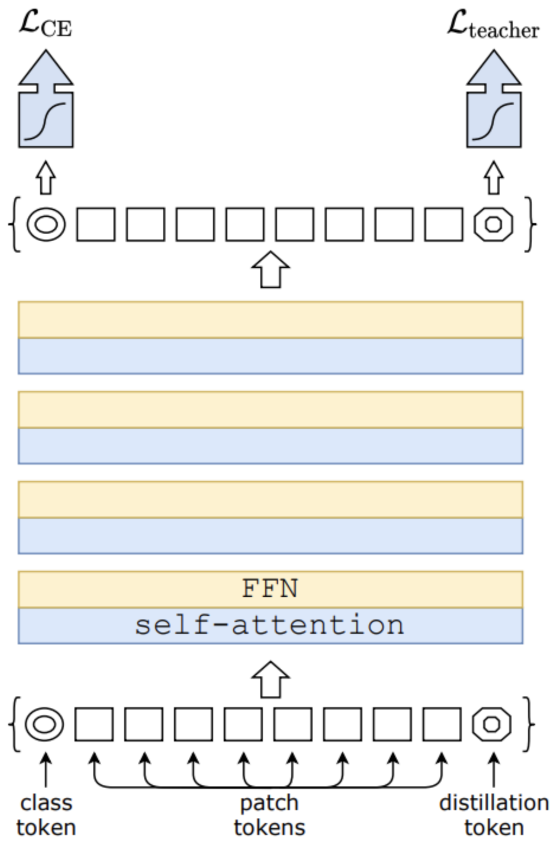
Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Distillation through Attention

3) Distillation token

- Add a **new distillation token** to class token/patch tokens.
- It **interacts with the class and patch tokens through the self-attention layers**.
- This distillation token is employed in a similar fashion as the class token, except that on output of the network, its objective is **to reproduce the (hard) label predicted by the teacher, instead of true label**. (The target objective is given by the distillation component of the loss.)
- Distillation embedding allows the model to learn from the output of the teacher, while remaining complementary to the class embedding.
- **Both the class and distillation tokens input to the transformers are learned by back-propagation**.
- The learned class/distillation tokens converge towards different vectors; the average cosine similarity between two tokens equal to 0.06. → at the last layer, their similarity equal to 0.93.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$



Our distillation procedure:

2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Distillation through Attention

4) Fine-tuning with distillation

- Use **both the true label and teacher prediction** during the fine-tuning stage at higher resolution.
- Use **a teacher with the same target resolution**, typically **obtained from the lower-resolution teacher**.

5) Classification with our approach: Joint classifier

- At test time, both *the class and distillation embeddings* (produced by transformer) are associated with *linear classifiers* and able to infer the image label.
- Our referent method is the late fusion of these two separate heads, for which we add the softmax output by two classifiers to make the prediction.

2 DeiT (Data-efficient Image Transformers)

Experiments

Transformer Models

- Architecture is identical to ViT with no convolution (ViT-B = DeiT-B)
- Only differences : Training strategy and Distillation token

- DeiT-B : Reference model (Same as ViT-B) → Parameters are fixed as D=768, h=12, d=D/h=64 (Keeping d=64)
- DeiT-B↑384 : Fine-tune DeiT at a larger resolution
- DeiT⚗ : DeiT with distillation (using distillation token)
- DeiT-S (Small), DeiT-Ti (Tiny) : Smaller models of DeiT

Model	ViT model	embedding dimension	#heads	#layers	#params	training resolution	throughput (im/sec)
DeiT-Ti	N/A	192	3	12	5M	224	2536
DeiT-S	N/A	384	6	12	22M	224	940
DeiT-B	ViT-B	768	12	12	86M	224	292

Table 1: Variants of our DeiT architecture. The larger model, DeiT-B, has the same architecture as the ViT-B [15]. The only parameters that vary across models are the embedding dimension and the number of heads, and we keep the dimension per head constant (equal to 64). Smaller models have a lower parameter count, and a faster throughput. The throughput is measured for images at resolution 224×224.

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

[출처] PR12 Paper Review, Jinwon Lee, PR-297 DeiT

Distillation : Convnets teachers

- Using a **convnet teacher** gives better performance than using a transformer
- Due to the **inductive bias** inherited by the transformer through distillation

Teacher Models	acc.	Student: DeiT-B⚗ pretrain	↑384
DeiT-B	81.8	81.9	83.1
RegNetY-4GF	80.0	82.7	83.6
RegNetY-8GF	81.7	82.7	83.8
RegNetY-12GF	82.4	83.1	84.1
RegNetY-16GF	82.9	83.1	84.2

- Default teacher is a RegNetY-16CF (84M parameter)

https://paperswithcode.com/method/regnety

2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

[출처] PR12 Paper Review, Jinwon Lee, PR-297 DeiT

Experiments

Distillation : distillation methods

- Hard distillation significantly outperforms soft distillation for transformers, even when using only a class token.
- The classifier on the two tokens is significantly better than the independent class and distillation classifiers
- The distillation token gives slightly better results than the class token. It is more correlated to the convnets prediction.

method ↓	Supervision label	Supervision teacher	Test tokens class	Test tokens distil.	ImageNet top-1 (%) pretrain	ImageNet top-1 (%) finetune 384
DeiT– no distillation	✓		✓		81.8	83.1
DeiT– usual distillation	✓	soft	✓		81.8	83.1
DeiT– hard distillation	✓	hard	✓		83.0	84.0
DeiT⚗: class embedding	✓	hard	✓		83.0	84.1
DeiT⚗: distil. embedding	✓	hard		✓	83.1	84.2
DeiT⚗: class+distillation	✓	hard	✓	✓	83.4	84.2

Table 3: Distillation experiments on Imagenet with DeiT, 300 epochs of pretraining. We separately report the performance when classifying with only one of the class or distillation embeddings, and then with a classifier taking both of them as input. In the last row (class+distillation), the result correspond to the late fusion of the class and distillation classifiers.

Distillation : Agreement with the teacher & inductive bias

- Does it inherit existing inductive bias that would facilitate the training?
- Below table reports the fraction of sample classified differently for all classifier pairs, i.e. the rate of different decisions.
- The distilled model is more correlated to the convnet than a transformer learned from scratch.

	groundtruth	no distillation convnet	no distillation DeiT	DeiT⚗ student (of the convnet) class	DeiT⚗ student (of the convnet) distillation	DeiT⚗ student (of the convnet) DeiT⚗
groundtruth	0.000	0.171	0.182	0.170	0.169	0.166
convnet (RegNetY)	0.171	0.000	0.133	0.112	0.100	0.102
DeiT	0.182	0.133	0.000	0.109	0.110	0.107
DeiT⚗– class only	0.170	0.112	0.109	0.000	0.050	0.033
DeiT⚗– distil. only	0.169	0.100	0.110	0.050	0.000	0.019
DeiT⚗– class+distil.	0.166	0.102	0.107	0.033	0.019	0.000

Table 4: Disagreement analysis between convnet, image transformers and distillated transformers: We report the fraction of sample classified differently for all classifier pairs, i.e., the rate of different decisions. We include two models without distillation (a RegNetY and DeiT-B), so that we can compare how our distilled models and classification heads are correlated to these teachers.

2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "[Training data-efficient image transformers & distillation through attention](#)," arXiv 2021

[출처] PR12 Paper Review, Jinwon Lee, PR-297 DeiT

Experiments

Distillation : Number of epochs

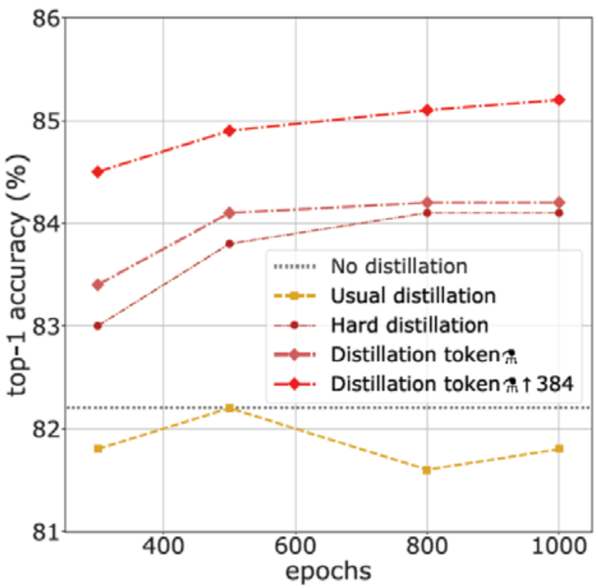
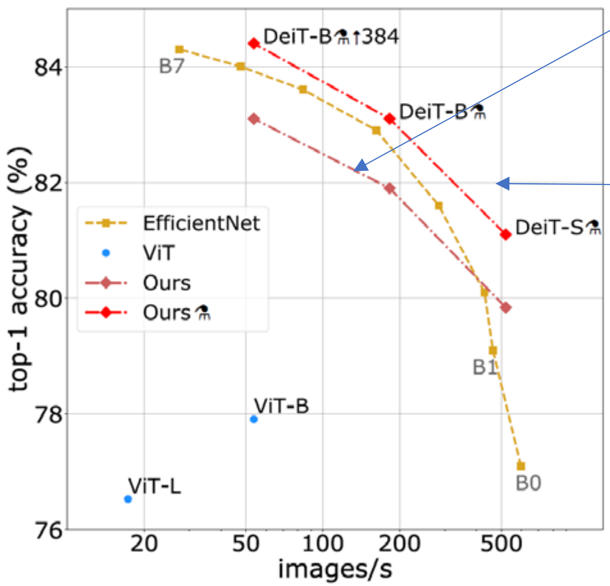


Figure 3: Distillation on ImageNet [42] with DeiT-B: performance as a function of the number of training epochs. We provide the performance without distillation (horizontal dotted line) as it saturates after 400 epochs.

With 300 epochs, our distilled network DeiT-B⚗ is already better than DeiT-B. But while for the latter the performance saturates with longer schedules, our distilled network clearly benefits from a longer training time.

Efficiency vs accuracy : Comparative study with convnets



- DeiT is slightly below EfficientNet, which shows that almost closed the gap between visual transformer and convnets when training with Imagenet only.
- These results are a major improvement (+6.3% top-1 in a comparable setting) over previous ViT models trained on Imagenet1k only.
- Furthermore, when DeiT benefits from the distillation from a relatively weaker RegNetY to produce DeiT⚗ , it outperforms EfficientNet.

2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "[Training data-efficient image transformers & distillation through attention](#)," arXiv 2021

[출처] PR12 Paper Review, Jinwon Lee, PR-297 DeiT

Experiments

Efficiency vs accuracy : Comparative study with convnets

- Compared to EfficientNet, one can see that, for the same number of parameters, the convnet variants are much slower. This is because large matrix multiplication offers more opportunity for hardware optimization than small convolutions.

Network	#param.	image size	throughput (image/s)	ImNet top-1	Real top-1	V2 top-1
Convnets						
ResNet-18 [21]	12M	224^2	4458.4	69.8	77.3	57.1
ResNet-50 [21]	25M	224^2	1226.1	76.2	82.5	63.3
ResNet-101 [21]	45M	224^2	753.6	77.4	83.7	65.7
ResNet-152 [21]	60M	224^2	526.4	78.3	84.1	67.0
RegNetY-4GF [40]⋆	21M	224^2	1156.7	80.0	86.4	69.4
RegNetY-8GF [40]⋆	39M	224^2	591.6	81.7	87.4	70.8
RegNetY-16GF [40]⋆	84M	224^2	334.7	82.9	88.1	72.4
EfficientNet-B0 [48]	5M	224^2	2694.3	77.1	83.5	64.3
EfficientNet-B1 [48]	8M	240^2	1662.5	79.1	84.9	66.9
EfficientNet-B2 [48]	9M	260^2	1255.7	80.1	85.9	68.8
EfficientNet-B3 [48]	12M	300^2	732.1	81.6	86.8	70.6
EfficientNet-B4 [48]	19M	380^2	349.4	82.9	88.0	72.3
EfficientNet-B5 [48]	30M	456^2	169.1	83.6	88.3	73.6
EfficientNet-B6 [48]	43M	528^2	96.9	84.0	88.8	73.9
EfficientNet-B7 [48]	66M	600^2	55.1	84.3	_	-
EfficientNet-B5 RA [12]	30M	456^2	96.9	83.7	-	-
EfficientNet-B7 RA [12]	66M	600^2	55.1	84.7	_	-
KDforAA-B8	87M	800^2	25.2	85.8	_	-
Transformers						
ViT-B/16 [15]	86M	384^2	85.9	77.9	83.6	-
ViT-L/16 [15]	307M	384^2	27.3	76.5	82.2	-
DeiT-Ti	5M	224^2	2536.5	72.2	80.1	60.4
DeiT-S	22M	224^2	940.4	79.8	85.7	68.5
DeiT-B	86M	224^2	292.3	81.8	86.7	71.5
DeiT-B↑384	86M	384^2	85.9	83.1	87.7	72.4
DeiT-Ti⚗	6M	224^2	2529.5	74.5	82.1	62.9
DeiT-S⚗	22M	224^2	936.2	81.2	86.8	70.0
DeiT-B⚗	87M	224^2	290.9	83.4	88.3	73.2
DeiT-Ti⚗ / 1000 epochs	6M	224^2	2529.5	76.6	83.9	65.4
DeiT-S⚗ / 1000 epochs	22M	224^2	936.2	82.6	87.8	71.7
DeiT-B⚗ / 1000 epochs	87M	224^2	290.9	84.2	88.7	73.9
DeiT-B⚗↑384	87M	384^2	85.8	84.5	89.0	74.8
DeiT-B⚗↑384 / 1000 epochs	87M	384^2	85.8	85.2	89.3	75.2

* : *Regnet* optimized with a similar optimization procedure as ours, which boosts the results. These networks serve as teachers when we use our distillation strategy.

Table 5: Throughput on and accuracy on Imagenet [42], Imagenet Real [5] and Imagenet V2 matched frequency [41] of DeiT and of several state-of-the-art convnets, for models trained with no external data. The throughput is measured as the number of images that we can process per second on one 16GB V100 GPU. For each model we take the largest possible batch size for the usual resolution of the model and calculate the average time over 30 runs to process that batch.

2 DeiT (Data-efficient Image Transformers)

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Training Details & Ablation

- Discuss **the DeiT training strategy** to **learn vision transformers in a data-efficient manner**.
- We build upon PyTorch [39] and the **timm** library [55]. (The timm implementation already included a training procedure that improved the accuracy of ViT-B from 77.91% to 79.35% top-1, and trained on Imagenet-1k with a 8xV100 GPU machine.)

❖ timm library
https://timm.fast.ai/
https://github.com/rwightman/pytorch-image-models

Initialization & hyper-parameters

- Transformers are relatively sensitive to initialization
- Follow Hanin et al. [20] to initialize the weights with a truncated normal distribution. Follow Cho et al. [9[] to select parameters τ=3.0, λ=0.1 for the usual (soft) distillation.

- We report the accuracy scores (%) after the initial training at resolution 224x224, and after fine-tuning at resolution 384x384. The hyper-parameters are fixed according to Table 9, and may be suboptimal.

Table 9: Ingredients and hyper-parameters for our method and Vit-B.

Methods	ViT-B [15]	DeiT-B
Epochs	300	300
Batch size	4096	1024
Optimizer	AdamW	AdamW
learning rate	0.003	$0.0005 \times \frac{\text{batchsize}}{512}$
Learning rate decay	cosine	cosine
Weight decay	0.3	0.05
Warmup epochs	3.4	5
Label smoothing ε	✗	0.1
Dropout	0.1	✗
Stoch. Depth	✗	0.1
Repeated Aug	✗	✓
Gradient Clip.	✓	✗
Rand Augment	✗	9/0.5
Mixup prob.	✗	0.8
Cutmix prob.	✗	1.0
Erasing prob.	✗	0.25

Hugo Touvron, et al. "Training data-efficient image transformers & distillation through attention," arXiv 2021

Training Details & Ablation

Ablation study

Table 8: Ablation study on training methods on ImageNet [42]. The top row (”none”) corresponds to our default configuration employed for DeiT. The symbols 3 and 7 indicates that we use and do not use the corresponding method, respectively. We report the accuracy scores (%) after the initial training at resolution 224x224, and after fine-tuning at resolution 384x384. The hyper-parameters are fixed according to Table 9, and may be suboptimal.

* indicates that the model did not train well, possibly because hyper-parameters are not adapted.

												top-1 accuracy	
Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	pre-trained 224^2	fine-tuned 384^2
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8 ±0.2	83.1 ±0.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3
	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

2 DeiT Pytorch

gihub

facebookresearch/deit : https://github.com/facebookresearch/deit
lucidrains/vit-pytorch : https://github.com/lucidrains/vit-pytorch
FrancescoSaverioZuppichini/DeiT : https://github.com/FrancescoSaverioZuppichini/DeiT

timm : pytorch image models
rwightman/pytorch-image-models : https://github.com/rwightman/pytorch-image-models

FrancescoSaverioZuppichini/glasses : His deep learning computer vision library
https://github.com/FrancescoSaverioZuppichini/glasses

2 DeiT Pytorch

[참고] FrancescoSaverioZuppichini/DeiT :
https://github.com/FrancescoSaverioZuppichini/DeiT

- Knowledge distillation

```
import torch
from torch import nn
import torch.nn.functional as F
from torch import Tensor

class HardDistillationLoss(nn.Module):
    def __init__(self, teacher: nn.Module):
        super().__init__()
        self.teacher = teacher
        self.criterion = nn.CrossEntropyLoss()

    def forward(self, inputs: Tensor, outputs : Tensor, labels: Tensor) -> Tensor:

        base_loss = self.criterion(outputs, labels)

        with torch.no_grad():
            teacher_outputs = self.teacher(inputs)

        teacher_labels = torch.argmax(teacher_outputs, dim=1)
        teacher_loss = self.criterion(outputs, teacher_labels)

        return 0.5 * base_loss + 0.5 * teacher_loss

# little test
loss = HardDistillationLoss(nn.Linear(100, 10))
_ = loss(torch.rand((8, 100)), torch.rand((8, 10)), torch.ones(8).long())
```

→ Modify by Attention Distillation

```
from typing import Union

class HardDistillationLoss(nn.Module):
    def __init__(self, teacher: nn.Module):
        super().__init__()
        self.teacher = teacher
        self.criterion = nn.CrossEntropyLoss()

    def forward(self, inputs: Tensor, outputs: Union[Tensor, Tensor], labels: Tensor) -
> Tensor:
        # outputs contains booth predictions, one with the cls token and one with the
dist token
        outputs_cls, outputs_dist = outputs
        base_loss = self.criterion(outputs_cls, labels)

        with torch.no_grad():
            teacher_outputs = self.teacher(inputs)

        teacher_labels = torch.argmax(teacher_outputs, dim=1)
        teacher_loss = self.criterion(outputs_dist, teacher_labels)

        return 0.5 * base_loss + 0.5 * teacher_loss
```

2 DeiT Pytorch

[참고] FrancescoSaverioZuppichini/DeiT :
https://github.com/FrancescoSaverioZuppichini/DeiT

- Distillation token

```
from einops import rearrange, reduce, repeat
from einops.layers.torch import Rearrange, Reduce

class PatchEmbedding(nn.Module):
    def __init__(self, in_channels: int = 3, patch_size: int = 16, emb_size: int = 768,
img_size: int = 224):
        self.patch_size = patch_size
        super().__init__()
        self.projection = nn.Sequential(
            # using a conv layer instead of a linear one -> performance gains
            nn.Conv2d(in_channels, emb_size, kernel_size=patch_size,
stride=patch_size),
            Rearrange('b e (h) (w) -> b (h w) e'),
        )
        self.cls_token = nn.Parameter(torch.randn(1,1, emb_size))

        # distillation token
        self.dist_token = nn.Parameter(torch.randn(1,1, emb_size))

        self.positions = nn.Parameter(torch.randn((img_size // patch_size) **2 + 1,
emb_size))
```

```
    def forward(self, x: Tensor) -> Tensor:
        b, _, _, _ = x.shape
        x = self.projection(x)
        cls_tokens = repeat(self.cls_token, '() n e -> b n e', b=b)
        dist_tokens = repeat(self.dist_tokens, '() n e -> b n e', b=b)
        # prepend the cls token to the input
        x = torch.cat([cls_tokens, dist_tokens, x], dim=1)
        # add position embedding
        x += self.positions
        return x
```

2 DeiT Pytorch

- Classification Head

```
class ClassificationHead(nn.Module):
    def __init__(self, emb_size: int = 768, n_classes: int = 1000):
        super().__init__()

        self.head = nn.Linear(emb_size, n_classes)
        self.dist_head = nn.Linear(emb_size, n_classes)

    def forward(self, x: Tensor) -> Tensor:
        x, x_dist = x[:, 0], x[:, 1]
        x_head = self.head(x)
        x_dist_head = self.dist_head(x_dist)

        if self.training:
            x = x_head, x_dist_head
        else:
            x = (x_head + x_dist_head) / 2
        return x
```

[참고] FrancescoSaverioZuppichini/DeiT :
https://github.com/FrancescoSaverioZuppichini/DeiT

Follows the ViT code

```
class MultiHeadAttention(nn.Module):
    def __init__(self, emb_size: int = 768, num_heads: int = 8, dropout: float = 0):
        super().__init__()
        self.emb_size = emb_size
        self.num_heads = num_heads
        # fuse the queries, keys and values in one matrix
        self.qkv = nn.Linear(emb_size, emb_size * 3)
        self.att_drop = nn.Dropout(dropout)
        self.projection = nn.Linear(emb_size, emb_size)

    def forward(self, x : Tensor, mask: Tensor = None) -> Tensor:
        # split keys, queries and values in num_heads
        qkv = rearrange(self.qkv(x), "b n (h d qkv) -> (qkv) b h n d",
h=self.num_heads, qkv=3)
        queries, keys, values = qkv[0], qkv[1], qkv[2]
        # sum up over the last axis
        energy = torch.einsum('bhqd, bhkd -> bhqk', queries, keys) # batch,
num_heads, query_len, key_len
        if mask is not None:
            fill_value = torch.finfo(torch.float32).min
            energy.mask_fill(~mask, fill_value)

        scaling = self.emb_size ** (1/2)
        att = F.softmax(energy, dim=-1) / scaling
        att = self.att_drop(att)
        # sum up over the third axis
        out = torch.einsum('bhal, bhlv -> bhav ', att, values)
        out = rearrange(out, "b h n d -> b n (h d)")
        out = self.projection(out)
        return out
```

2 DeiT Pytorch

[참고] FrancescoSaverioZuppichini/DeiT :
https://github.com/FrancescoSaverioZuppichini/DeiT

Follows the ViT code

```
class ResidualAdd(nn.Module):
    def __init__(self, fn):
        super().__init__()
        self.fn = fn

    def forward(self, x, **kwargs):
        res = x
        x = self.fn(x, **kwargs)
        x += res
        return x

class FeedForwardBlock(nn.Sequential):
    def __init__(self, emb_size: int, expansion: int = 4, drop_p: float = 0.):
        super().__init__(
            nn.Linear(emb_size, expansion * emb_size),
            nn.GELU(),
            nn.Dropout(drop_p),
            nn.Linear(expansion * emb_size, emb_size),
        )
```

```
class TransformerEncoderBlock(nn.Sequential):
    def __init__(self,
                 emb_size: int = 768,
                 drop_p: float = 0.,
                 forward_expansion: int = 4,
                 forward_drop_p: float = 0.,
                 ** kwargs):
        super().__init__(
            ResidualAdd(nn.Sequential(
                nn.LayerNorm(emb_size),
                MultiHeadAttention(emb_size, **kwargs),
                nn.Dropout(drop_p)
            )),
            ResidualAdd(nn.Sequential(
                nn.LayerNorm(emb_size),
                FeedForwardBlock(
                    emb_size, expansion=forward_expansion, drop_p=forward_drop_p),
                nn.Dropout(drop_p)
            )
            ))

class TransformerEncoder(nn.Sequential):
    def __init__(self, depth: int = 12, **kwargs):
        super().__init__(*[TransformerEncoderBlock(**kwargs) for _ in range(depth)])
```

2 DeiT Pytorch

DeiT model

```
class DeiT(nn.Sequential):
    def __init__(self,
                in_channels: int = 3,
                patch_size: int = 16,
                emb_size: int = 768,
                img_size: int = 224,
                depth: int = 12,
                n_classes: int = 1000,
                **kwargs):
        super().__init__(
            PatchEmbedding(in_channels, patch_size, emb_size, img_size),
            TransformerEncoder(depth, emb_size=emb_size, **kwargs),
            ClassificationHead(emb_size, n_classes)
        )
```

[참고] FrancescoSaverioZuppichini/DeiT :
https://github.com/FrancescoSaverioZuppichini/DeiT

To train, we can use a bigger model (ViT-Huge, RegNetY-16GF ...) as teacher and a smaller one (ViT-Small/Base) as student. The training code looks like this:

https://github.com/facebookresearch/deit

```
ds = ImageDataset('./imagenet/')
dl = DataLoader(ds, ...)

teacher = ViT.vit_large_patch16_224()
student = DeiT.deit_small_patch16_224()

optimizer = Adam(student.parameters())
criterion = HardDistillationLoss(teacher)

for data in dl:
    inputs, labels = data
    outputs = student(inputs)

    optimizer.zero_grad()

    loss = criterion(inputs, outputs, labels)

    loss.backward()
    optimizer.step()
```