An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

ALLOUACHE et ELMAM
Sorbonne Université

Introduction

Les Transformers sont appliqués sur plusieurs taches de NLP, ses application à la vision restent limitées. Dans cet article, on montre que pour une tache de classification d'image, on utilise un transformer pur appliqué sur des séquences de patchs d'image. Ainsi, le retour aux CNN n'est plus nécessaire On explore la reconnaissance d'images à des échelles plus grandes que l'ensemble de données standard d'ImageNet. On divise l'imge en un ensemble de patchs. Les patchs sont traités de la même manière que les mots dans une application de NLP.

Points-clés

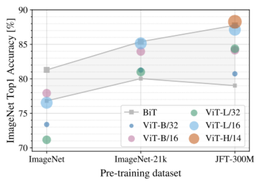
- Application de principe des Transformers pour la classification d'images.
- Représenton d'image sous forme d'une séquence de patchs.
- Limitation de la taille des patchs.
- La nécessité d'un large dataset pour l'entraînement.
- L'obtention de visualisation intermédiaire plus significatif.
- Les architectures hybrides (CNN+Transformers).
- Moins de ressources pour l'entraînement que les modèles CNNs.

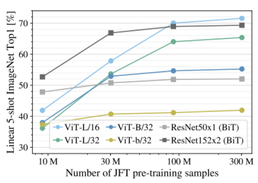
Experimentation

- **Datasets**: Entrainement sur JFT, $ImageNet$, ensuite application d'un transfert learning par les auteurs de l'article. Dans notre cas, on a entrainé le modèle sur CIFAR10.
- La configuration ViT utilisée par les auteurs est basée sur celles utilisées pour $BERT$ (Devlin et al., 2019).

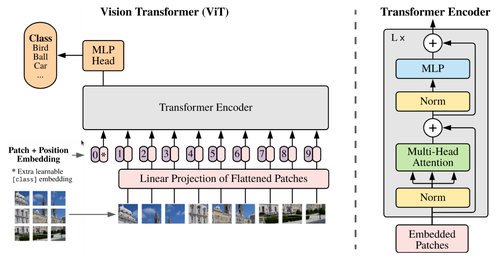
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

- ViT-L/16 signifie la variante "Large" avec une taille de patch d'entrée de 16×16.
- La longueur de la séquence du transformer est inversement proportionnelle au carré de la taille du patch, ce qui fait que les modèles avec une taille de patch plus petite sont plus lord en termes de calcul.
- Preprocessing: Basé sur celui utilisé dans *Kolesnikov et ai (2020)*.
- Fonction de loss: label smoothing qui favorise la généralisation.
- Self-Supervision: pour prédire les couleurs des patchs.
- Afin d'avoir de meilleures performances sur des petits datasets, on applique des regularisations: Adam avec un weight_decay = 0.1, linear schedulers.





Vision Transformer ViT



1 Embedding

- **Construction des Patchs**: Image $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ tel que (H, W) est la resolution de l'image et C est le nombre de channels. On aplatit cette image vers une séquence de patchs $\mathbf{x}_p \in \mathbb{R}^{N\times(P^2\cdot C)}$ avec $N = HW/P^2$ et P la taille de patch.
- **Projection linéaire des patchs (Patch Embedding):** Avec les paramètres $\mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D}$, on projette linéairement chaque patch $\mathbf{x}_p$ vers une dimension D qui est la dimension de modèle.
- **Le token de classification:** Commee le token $[class]$ de ***BERT***, on ajoute à la séquence des Embeddings des patch $\mathbf{z}_0^0 = \mathbf{x}_{\text{class}}$ dont l'état à la sortie de l'encodeur $\mathbf{Z}_L^0$ sera la représentation de l'image d'entrée.
- **Embedding des Position**: $\mathbb{E}_{pos} \in \mathbb{R}^{N+1\times D}$ Des Embeddings de position sont ajoutées aux Embeddings des patchs pour conserver les informations de position. On utilise une matrices de poids $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$ pour cet Embedding.

À la fin de l'étape de l'embedding, on obtient l'ensemble $\mathbf{z}_0$ qui sera l'entrée de l'encodeur:

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}}\,; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{\text{pos}}\,, \quad \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$$

2 Architecture Encodeur

Transformer encodeur consiste à alterner entre les couches Multi-Head-Self-Attention *(MSA)*, des blocs *(MLP)*. *LayerNorm (LN)* est appliquée avant chaque block, les connexions résiduelles sont appliqués après chaque bloc. *MLP* contient 2 couches linéaires et une activations non-linéaire *GELU*.

$$\begin{aligned}
\mathbf{z}'\ell &= \text{MSA}\left(\text{LN}\left(\mathbf{z}_{\ell-1}\right)\right) + \mathbf{z}_{\ell-1}, \quad \ell = 1\ldots L \\
\mathbf{z}_\ell &= \text{MLP}\left(\text{LN}\left(\mathbf{z}'\right)\right) + \mathbf{z}'_\ell, \qquad \ell = 1\ldots L \\
\mathbf{y} &= \text{LN}\left(\mathbf{z}_L^0\right)
\end{aligned}$$

2.1 MSA: Multi-Head Self-Attention

$$\begin{aligned}
[\mathbf{q}, \mathbf{k}, \mathbf{v}] &= \mathbf{z}\mathbf{U}_{qkv} && \mathbf{U}_{qkv} \in \mathbb{R}^{D\times 3D_h} \\
A &= \text{softmax}\left(\mathbf{q}\mathbf{k}^\top/\sqrt{D_h}\right) && A \in \mathbb{R}^{N\times N} \\
\text{SA}(\mathbf{z}) &= A\mathbf{v}.
\end{aligned}$$

Pour chaque élément d'une séquence $\mathbf{z} \in \mathbb{R}^{N\times D}$ est projeté vers une dimension $3 * D_h$ avec h le nombre de *heads* afin d'obtenir une représentation $[k, q, v]$, ensuite on calcule une somme pondérée sur toutes les valeurs **v** dans la sequence. Les poids d'attention A_{ij} sont basé sur la similarité entre les représentations $\mathbf{q}^i$ et $\mathbf{k}^j$.

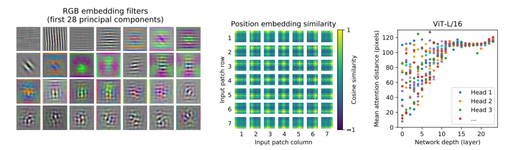
$$\text{MSA}(\mathbf{z}) = [\text{SA}_1(z); \text{SA}_2(z); \cdots; \text{SA}_k(z)]\,\mathbf{U}_{msa} \quad \mathbf{U}_{msa} \in \mathbb{R}^{k\cdot D_h\times D}$$

Multihead self-attention (MSA) c'est une extension de *SA* dans laquelle on exécute k self-attention appelée "$heads$", en parallèle, ensuite on projette leur concaténation vers D pour revenir à la dimension de l'encodeur. Pour le model ViT on choisit $D_h = D/k$.

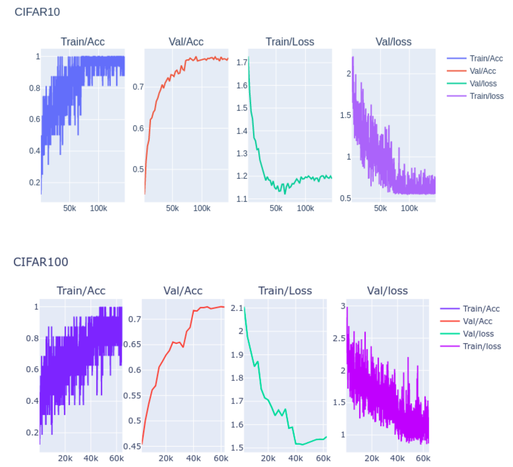
Résultats

Résultats obtenus par les auteurs de l'article:

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k



Nos résultats:



Conclusion

Nous avons exploré l'application directe de Transformers à la reconnaissance d'images et nous avons constaté que les résultats du modèle correspond ou dépasse l'état de l'art sur de nombreux ensembles de données de classification d'images.
Avec l'augmentation constante de la taille des ensembles de données, et le développement continu de méthodes non supervisées et semi-supervisées, le développement de nouvelles architectures de vision qui s'entraînent plus efficacement sur ces ensembles de données devient de plus en plus important. Nous pensons que ViT est une étape préliminaire vers des architectures génériques et évolutives qui peuvent résoudre de nombreuses tâches de vision, ou même des tâches provenant de nombreux domaines.

Informations supplémentaires

Code source: github.com/YacineAll/ViT-pytorch-lightning
Article: https://arxiv.org/abs/2010.11929