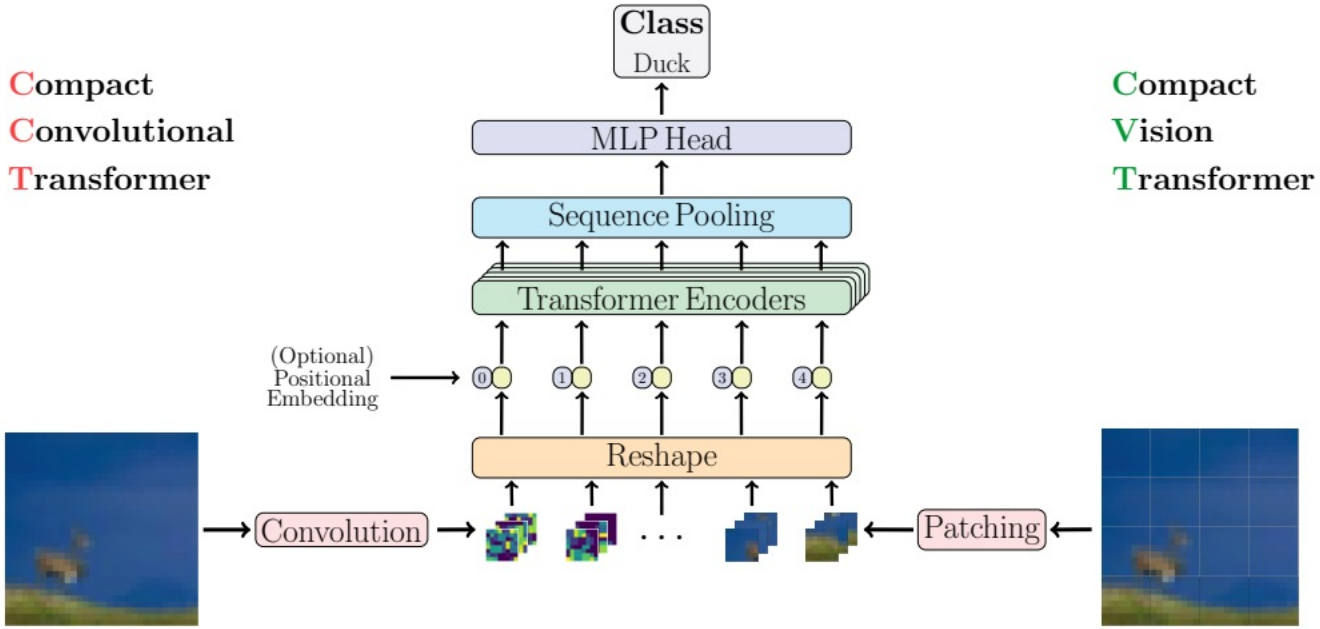
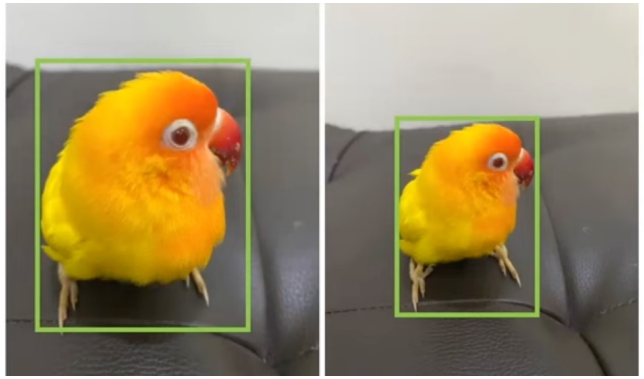
Escaping the Big Data Paradigm with Compact Transformers

- Transformers are not suitable for small sets of data.
- Compact Transformers with the right size, convolutional tokenization, transformers can avoid overfitting and outperform state-of-the-art CNNs on small datasets.

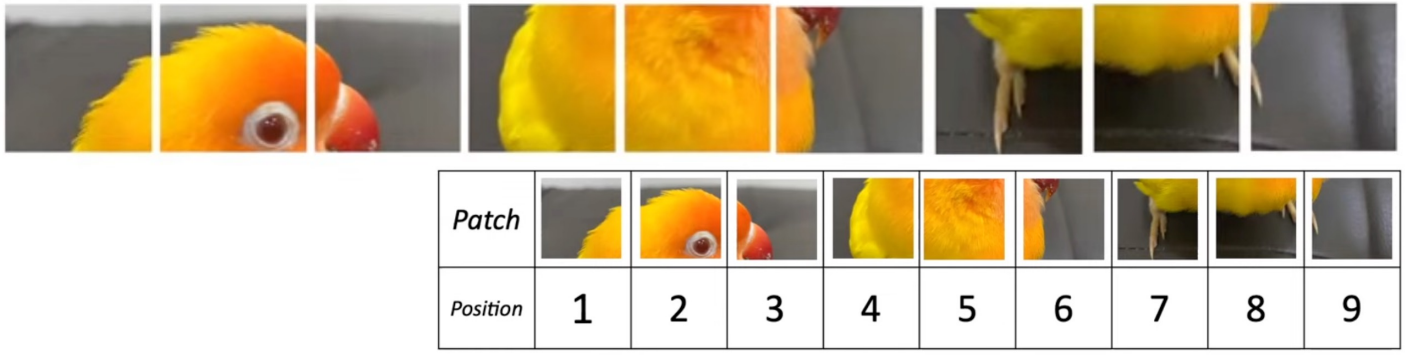
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

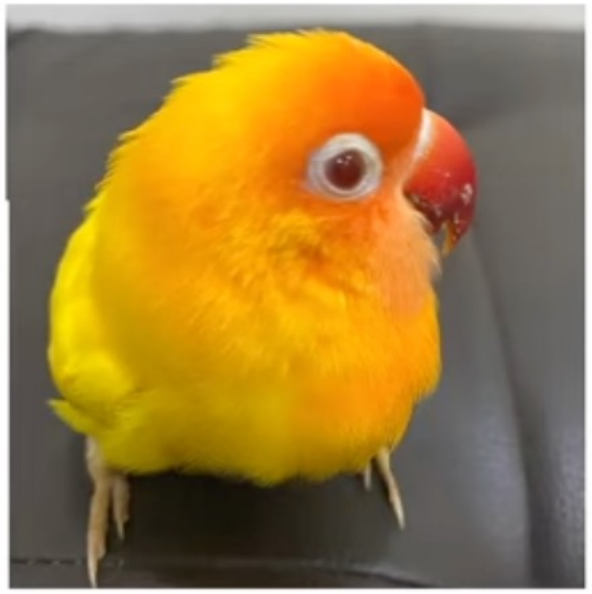


Introduction



- Convolution is translation and scale equivariant
- Pooling is translation and scale invariant
- Equivariance (Conv) + Invariance (Pool): Important for recognition, object detection and segmentation.
- ViT is Invariant to Position of Patches





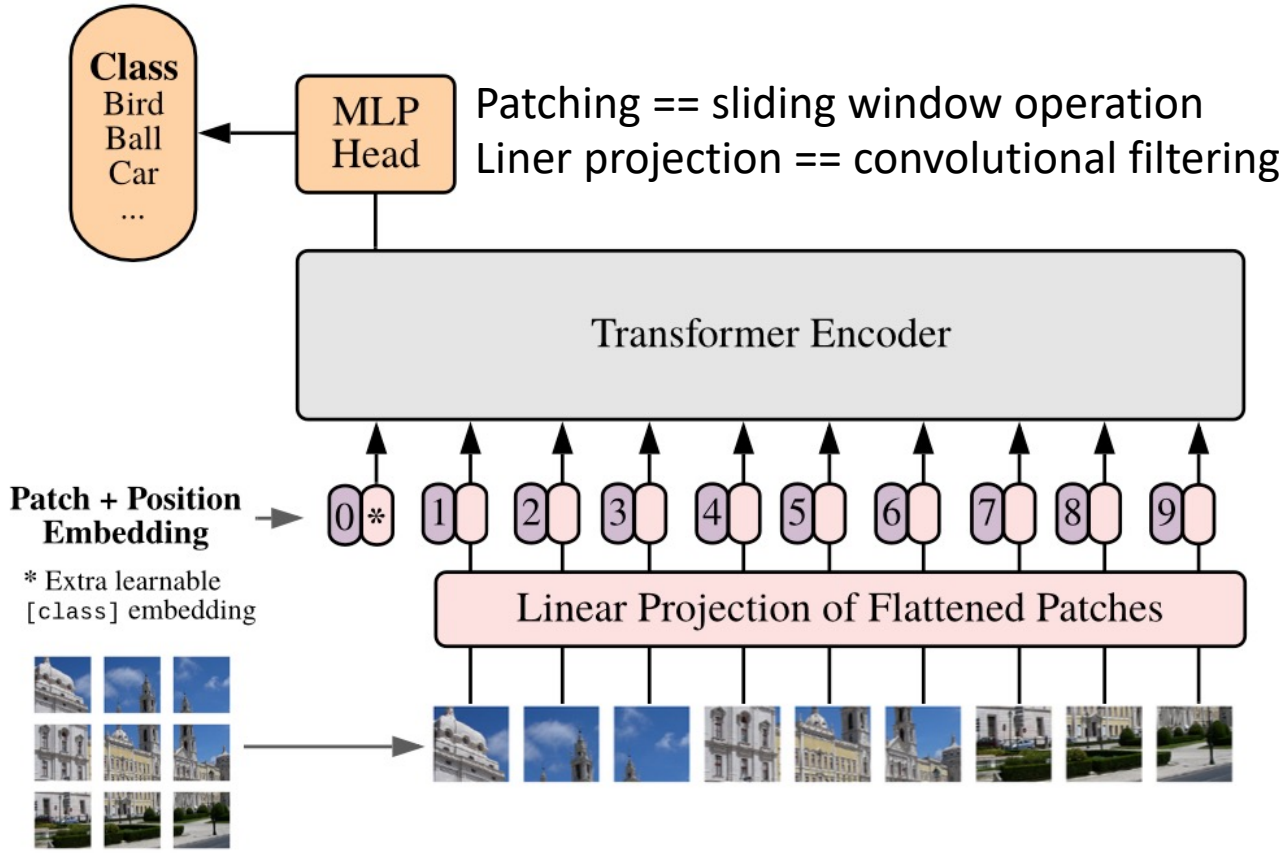
"Transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data."

Introduction

- CNNs are still the go-to models for smaller datasets because they are more efficient, both computationally and in terms of memory, when compared to transformers. Additionally, **local inductive bias** shows to be more important in smaller images.
- CNNs require less time and data to train while also requiring a lower number of parameters to accurately fit data. **However**, they do not enjoy the **long range interdependence** that attention mechanisms in transformers provide.

ViT

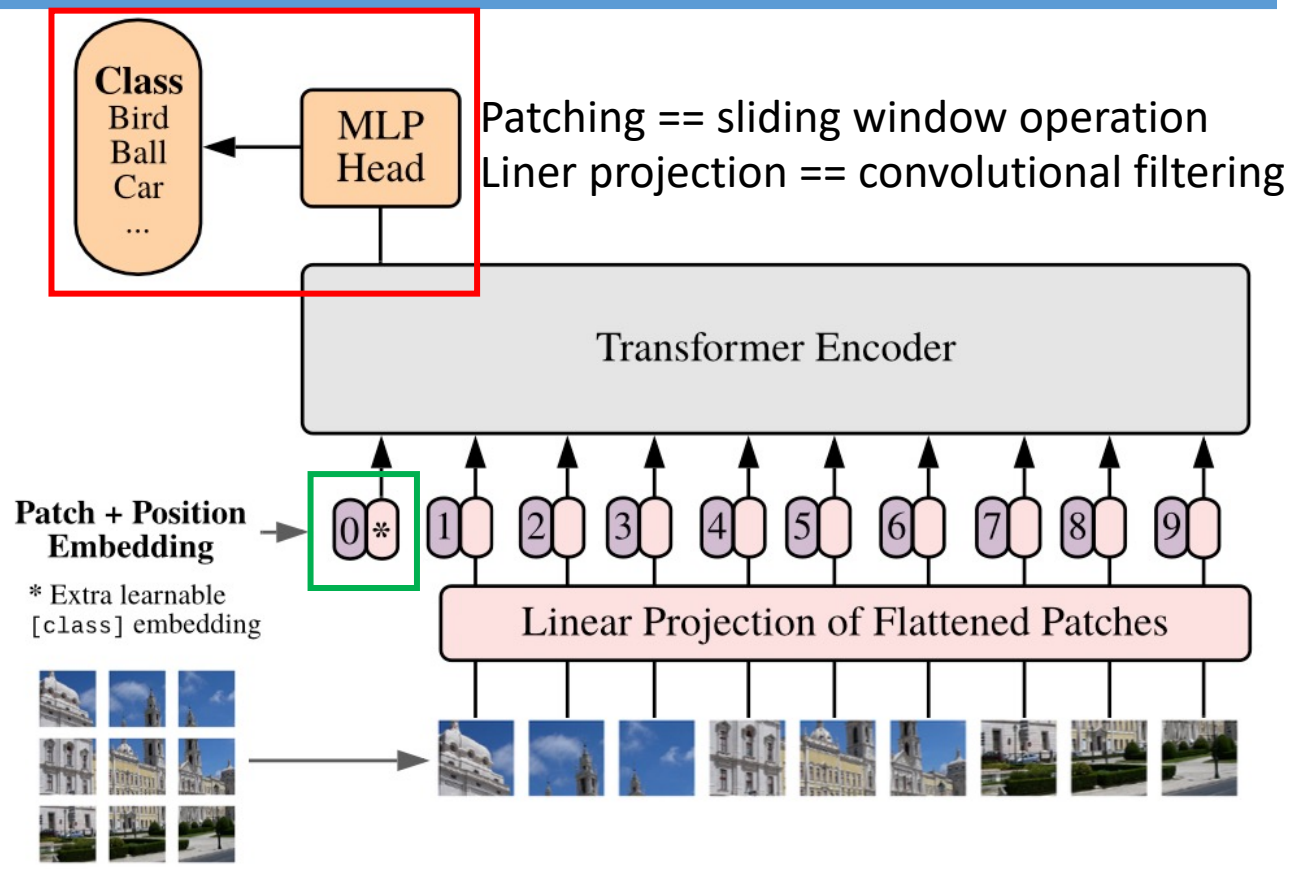
- **Image Tokenization**: ViT subdivides an image into non-overlapping square patches in raster-scan order. The sequence of patches, $xp \in R^{H\times(P2C)}$ with patch size P, are flattened into 1D vectors and transformed into latent vectors of dimension d.



- **Positional Embedding:** Positional embedding adds spatial information into the sequence. Since the model does not actually know anything about the spatial relationship between tokens.

ViT

- A learnable [cls] token is prepended to every sequence of patches. (D x 1 dimensional vector)
- After applying PE to the sequences, they are fed to the Transformer encoder, which outputs (N + 1) * D dimensional vector (N is the sequence length).



- At the classification stage only the first 1 x D row (corresponding to the learnable [cls] token) is taken and fed to the final classifier.

ViT

- ViT trained from scratch on ImageNet, the results were slightly worse. A pilot experiment: the ViT base model achieved only 69.82% Top-1 on CiFAR-10 if directly trained from scratch.
- ViT has achive a really great performance on a large dataset as ImageNet.

"*Transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data.*"

~ Dosoviskiy et al. 2020

Accuracy: 0.982222232553694, Precision: 0.982222232553694, Recall: 0.982222232553694, F1 score: 0.982222232553694	Accuracy: 0.6305799550480313, Precision: 0.6352797018157111, Recall: 0.6305799550480313, F1 score: 0.6224562724431356

Swin Transformer training results on my dataset: fine-tunning vs scratch

Compact Vision Transformer

2.4. Comparison

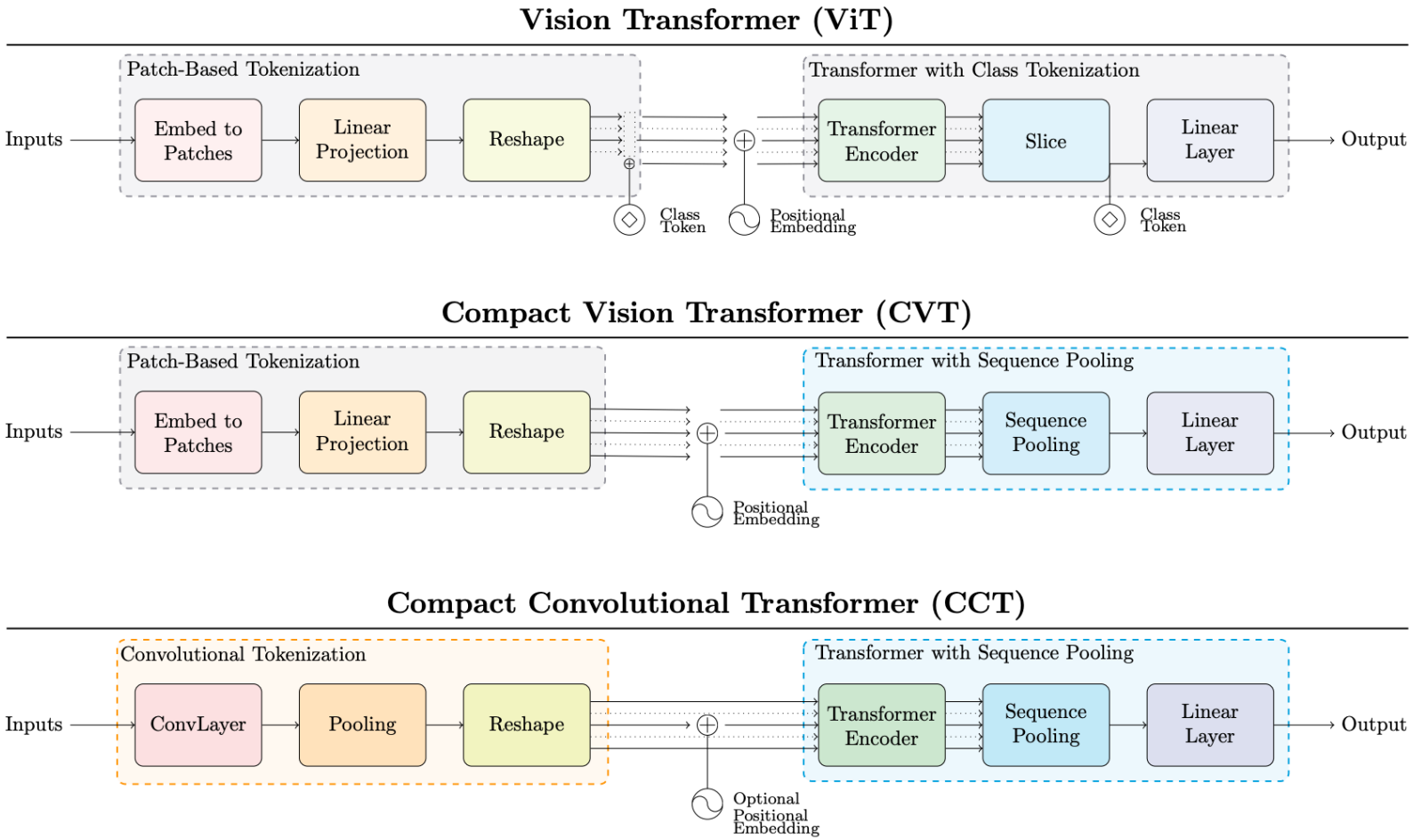
Our work differs from the aforementioned in several ways, in that it focuses on answering the following question: **Can vision transformers be trained from scratch on small datasets?** Focusing on a small datasets, we seek to create a model that can be trained, from scratch, on datasets that are orders of magnitude smaller than ImageNet. Having a model that is compact, small in size, and efficient allows greater accessibility, as training on ImageNet is still a difficult and data intensive task for many researchers. Thus our focus is on an accessible model, with few parameters, that can quickly and efficiently be trained on smaller platforms while still maintaining SOTA results.

Method

3 different models:

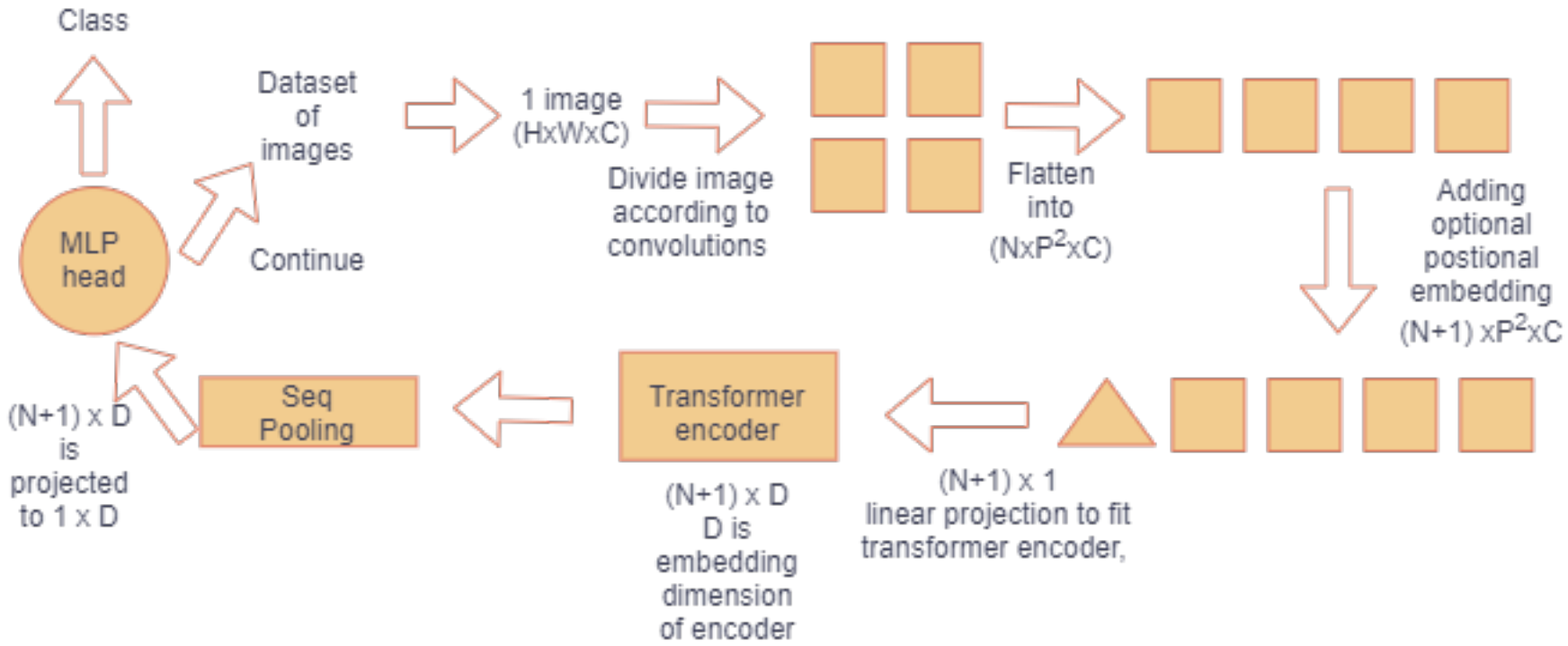
1. ViT-Lite is nearly identical to the original ViT: suitable size and patch size for small-scale learning.
2. CVT builds on ViT by using our Sequence Pooling method (SeqPool), that pools the entire sequence of tokens produced by the transformer encoder. SeqPool replaces the conventional [class] token.
3. CCT is builds on CVT and utilizes a convolutional tokenizer, generating richer tokens and preserving local information.

Method



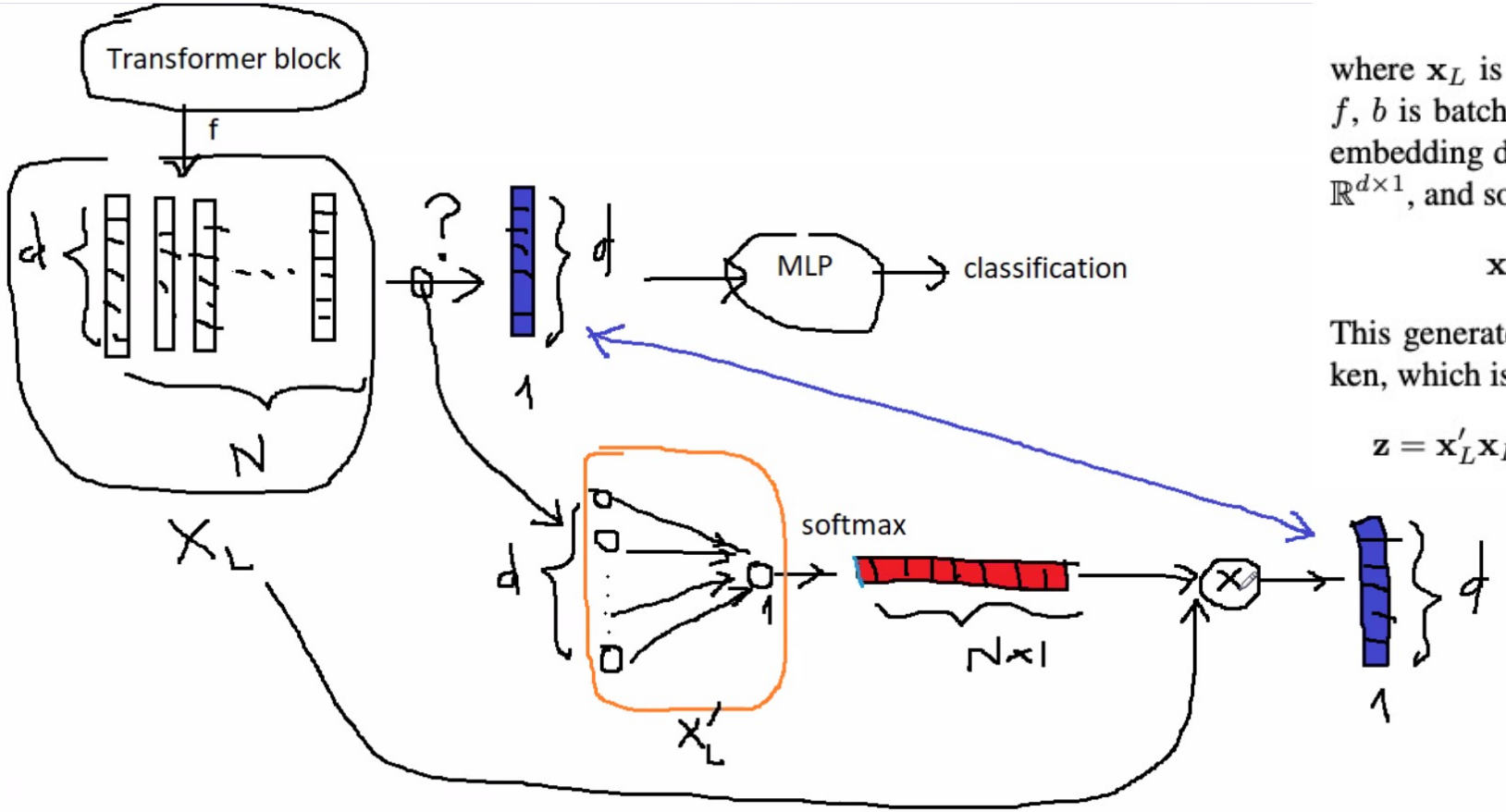
Transformer-based Backbone
Small and Compact Models
SeqPool

Method



[Shreyas-Bhat/CompactTransformers: Implementation of "Escaping the big data paradigm with Compact Transformers" by Ali Hassani et al. in Keras (github.com)]

Sequence Pooling



$$\mathbf{x}_L = \mathrm{f}(\mathbf{x}_0) \in \mathbb{R}^{b \times n \times d}$$

where $\mathbf{x}_L$ is the output of an L layer transformer encoder f, b is batch size, n is sequence length, and d is the total embedding dimension. $\mathbf{x}_L$ is fed to a linear layer $\mathrm{g}(\mathbf{x}_L) \in \mathbb{R}^{d \times 1}$, and softmax activation is applied to the output:

$$\mathbf{x}'_L = \mathrm{softmax}\left(\mathrm{g}(\mathbf{x}_L)^T\right) \in \mathbb{R}^{b \times 1 \times n}$$

This generates an importance weighting for each input token, which is applied as follows:

$$\mathbf{z} = \mathbf{x}'_L \mathbf{x}_L = \mathrm{softmax}\left(\mathrm{g}(\mathbf{x}_L)^T\right) \times \mathbf{x}_L \in \mathbb{R}^{b \times 1 \times d} \quad (1)$$

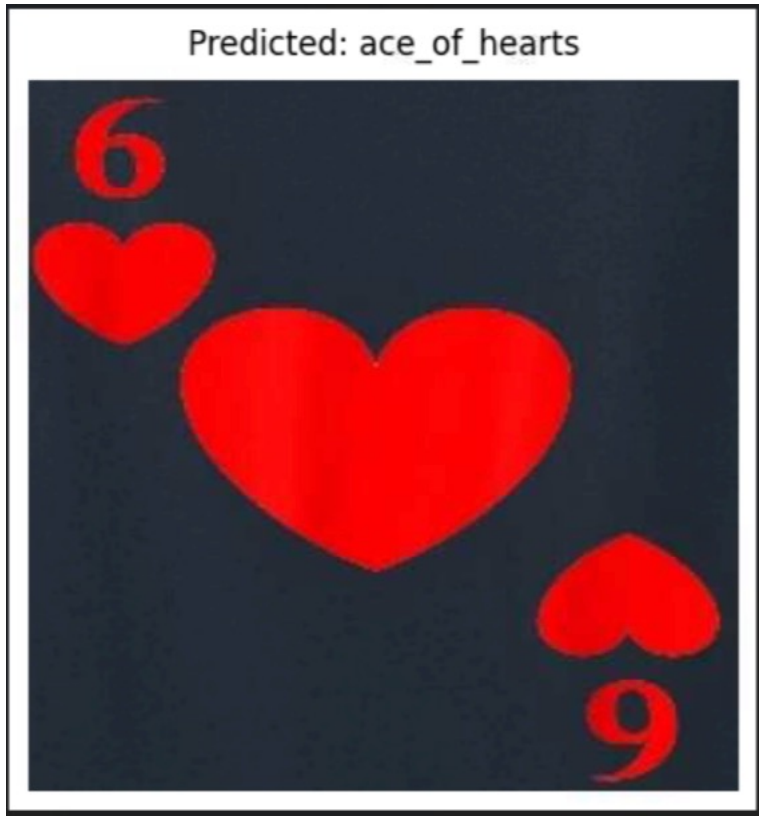
Method

- Instead of prepending a D dimensional [cls] token vector to the sequence, we place that vector as a D x 1 linear layer after the Transformer encoder. It therefore maps the N × D output matrix to an N x 1 vector.
- We apply softmax to the N × 1 output, transpose it and multiply it by the original N × D output, which will result in a 1 × D output. We pool over the entire sequence of data since it contains relevant information across different parts of the input image, making our model compact.
- The 1 × D is then taken and fed to the final classifier.

Method

Original ViT-Base	**CVT**	**CCT**
12 layers	6 layers	6 layers
Non-overlapping Convolutions $_{16x16}$	Non-overlapping Convolutions $_{4x4}$	Overlapping Convolutions $_{3x3}$
Class Token	SeqPool	Seqpool
69.82% Top-1	92.58% Top-1	94.81 % Top-1

Implementation



Accuracy: 0.744597527715894 9,
Precision: 0.7451575464672513,
Recall: 0.744597527715894 9,
F1 score: 0.7384613355000814