APL745 - Deep Learning for Mechanics

Report 1

ViViT: A Video Vision Transformer

Shivam Gupta
2020AM10668
am1200668@iitd.ac.in
Department of Applied Mechanics
Indian Institute of Technology, Delhi

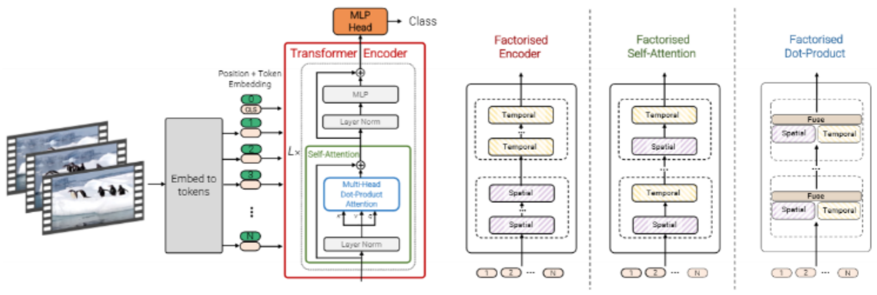
April 19, 2024

Abstract

We use Transformers for classification of videos, by dissecting the videos into spatial and temporal tokens, and then inputting this as embedding to the Transformer. The innovation here is in proposing some effective ways of factorising the spatio-temporal dimensions of the input video. And although transformer models are known to be useful only when a large dataset is available, we experiment with how we can effectively regularise the model during training and leverage pretrained image models to run on small datasets.

1 Introduction

With the introduction of AlexNet, which gave the idea of using a large scale dataset for training (using DCNN), have advanced SOTA for vision models. The coming out of Transformers in 2017, has greatly accelerated development in sequence to sequence models. Transformers are effective at modeling long range dependencies, and learning relationship between each input token, and overcome some of the problems in RNN, which lacked from vanishing gradient problem, and unable to remember long range dependencies. The success of transformers in sequence to sequence model has encouraged the viewpoint of modeling other modalities(image, video, audio) as also sequence to sequence, and one paper, ViT(Vision Transformer), used pure transformer based architecture to classify images. The original implementation of transformer from the 2017 paper was followed, and it was observed that only after training with a huge dataset, we observe its superior performance. Transformers lack some of the inductive biases present in CNN. We extend the same idea to videos, where we try to model video as sequence to sequence spatial and temporal tokens. The main operation of Transformer is attention, which tries to learn relationship between tokens in the input, and here the input is the spatial and temporal tokens of the input video. 2 main work is done here, we train to propose some methods of factorising our model along spatial and temporal dimensions to increase efficiency, and try to train model on small dataset by regularising the model. We have done analysis on



(a) **Fig 1**. Tranformer Architecture for Video classification

different tokenisation strategies of the spatio-temporal tokens, model architecture of the transformer, and regularisation methods to train it on small dataset. SOTA performance is observed on multiple video classification benchmarks, like Kinetics 400 and 600, Epic Kitchens 100, Something-Something v2, and Moments in Time.

2 Related Work

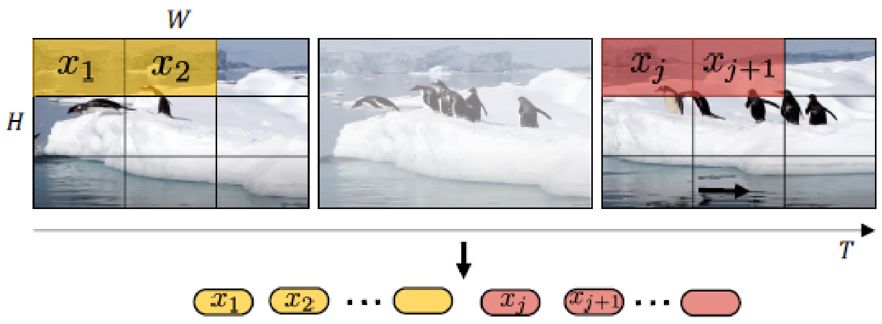
The success of AlexNet on ImageNet, led to the surgence of training on a large dataset, and also on CNN in the case of Vision models. And similarly, availability of large dataset of videos for classification, like Kinetics, led to training of spatio-temporal 3D CNNs for videos. And as videos have more parameters and hence need large datasets, improvement and innovations in factorising spatial and temporal dimensions was introduced. In ViT, Vaswani achieved SOTA by replacing CNN and RNN with transformers, which consisted only of self-attention, layer normalization and multilayer perceptron(MLP) operations. Many variants of transformers has come out, to reduce computational cost for different use cases and achieve superior performance. Self-attention in vision models has mainly been used as a layer at the end stages. Vit has inspire a large number of work, i.e., of using transformers for vision models. We develop pura transformer based models for video classification, using pre trained models to show that datasets need not be as large as was used in training ViT, and improvements in factoring the spatial and temporal tokens of the input video.

3 Overview of Vision Transformers

Transformers were first applied on images by dividing each image into N non-overlapping patches, x_i, each of dimension $h \times w$. A linear operation is then performed on each individual patch, x_i, and rasterized into 1D tokens, z_i, each of dimension d. Hence, the representation of an image can be expressed as:

$$Z = [z_{\text{cls}}, Ex_1, Ex_2, \ldots, Ex_N] + p,$$

where each x_i is projected by the matrix E, and z_{cls} is an additional token appended to this sequence. The representation of z_{cls} at the final layer of the encoder indicates the class to which the image belongs, i.e., its representation is used for classification. The token p



(b) **Fig 2**. Uniform frame Sampling: Embed n_t time frames independently following each 2D frame

is added for positional embedding. These tokens are then passed through the transformer layers.

It is observed that the Transformer can be applied to any sequence-to-sequence task. By converting an image to a sequence representation, and further proposing methods to tokenize video as a sequence-to-sequence representation, Transformers can be applied to classify the video.

4 Tokenizing Videos for Transformers

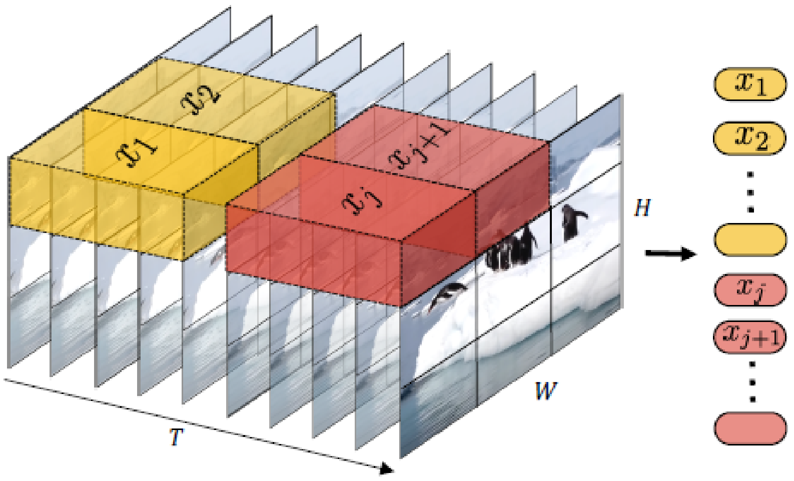
We present two methods for tokenizing videos to be fed as input to transformers:

1) Uniform Frame Sampling

This approach is similar to ViT, where we just concatenate the tokens of different time frames together. Let's say the video has n_t frames altogether, each frame is being represented by a token z. So, we concatenate n_t tokens to token z. Here, the total number of tokens are as follows: $n_h \times n_w$ patches are extracted from each image, and that multiplied by n_t (total number of time steps). So, a total of $n_t \times n_h \times n_w$ tokens. We then reshape these $n_t \times n_h \times n_w$ tokens to a linear dimension, which we input to the encoder of the transformer.

2) Tubelet Embedding

Here, instead of sampling from each frame and fusing the temporal tokens separately, we extract spatio-temporal tokens from the video. This is like 3D convolution, where we extract tokens in a 3D way (instead of the 2D way we were using for uniform rate sampling). This is ViT's extension to 3D. We first divide the input video, in batches, each of dimension $t \times h \times w$, into $n_t \times n_h \times n_w$, where n_t is $[T/t]$, n_h is $[H/h]$, n_w is $[W/w]$, where T, H, W is the total number of time frames, height of each frame, and width of each frame respectively. Here, each 3D embedding takes $n_h \times n_h$ patches, across n_t time frames, so a 3D matrix of dimension $n_t \times n_h \times n_w$ concatenated by the rest of the spatio-temporal embedding, is fed as input to the transformer.



(c) **Fig 3**.Tubelet embedding: we embed non-overlapping tubelets that span spatio-temporal input volume

5 How Transformer Works

Transformers were introduced to overcome the challenges of LSTMs. Some of the issues with LSTMs include their inability to capture long-term dependencies between tokens and time issues, as they operated in a sequential manner, which took a lot of time to calculate the overall dependency between tokens.

Transformers addressed these issues in the following ways:

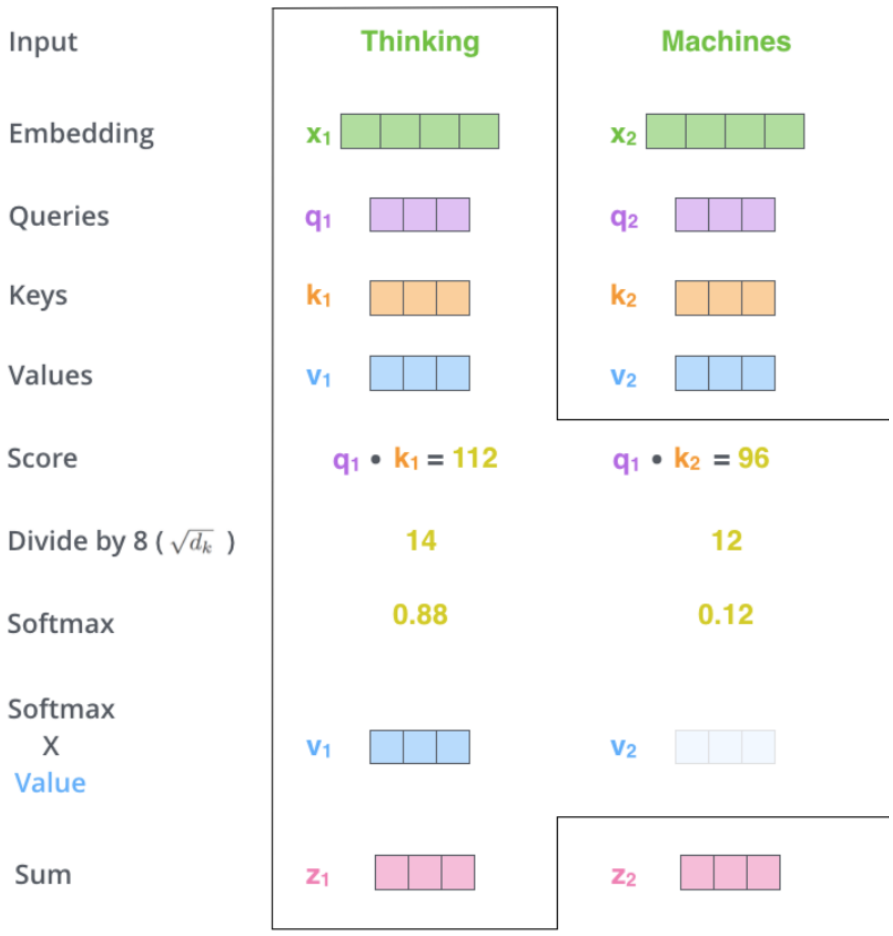
1. It calculates the dependency between tokens in $O(n^2)$ complexity, where all the tokens are fed at the same time to the transformer block. It eliminates the sequential operation, unlike LSTMs, and hence processes data in parallel.

2. It uses the attention mechanism, which tells how each token is related to the other token, to capture the long-term dependency between tokens.

The attention mechanism is at the heart of the Transformer architecture. The Transformer is essentially Multi-head Key-Value Self-attention. We compute multiple attentions for a given token sequence, using query, key, and value vectors, and the weights in this case are the query, key, and value matrix.

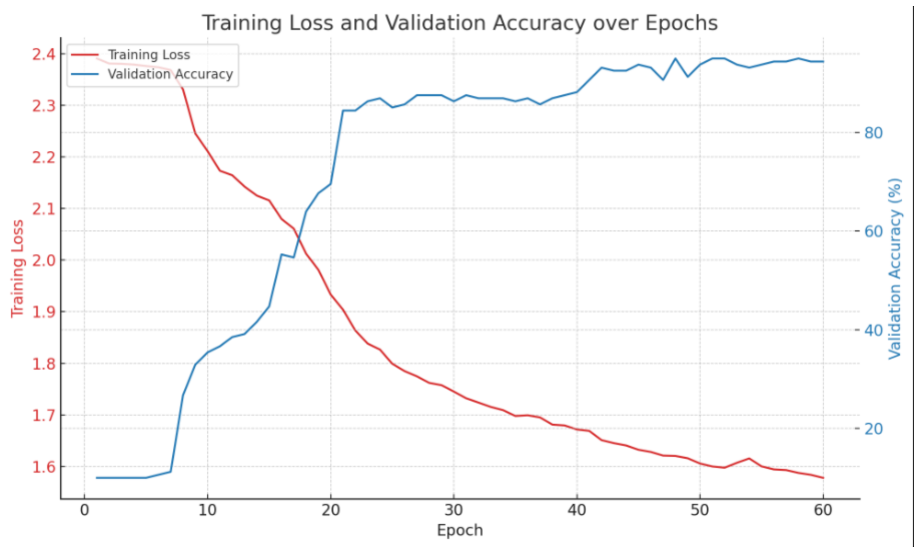
6 Multi-head Key-Value self-attention

The Encoder is broken down into 2 sub layers, the self attention block and the feed forward layer. The self attention block calculates dependency between different tokens, which we then pass forward to the feed forward network. The Decoder also has the same structure, but it also uses the output of the decoder(till t tokens which have been produced), in calculation of attention. We also have operation of residuals, where we add the output of attention skipping one layer, and normalization step, where we do layer normalization. Transformer is a bunch of encoder and decoder stacked together(6 each in this case).



(d) **Fig 4**.How Self-Attention works

Here's how we calculate self attention. Let's do it for 2 words. We first convert the words into vectors (using pretrained word2vec models), and using this vector, we calculate 3 vectors, which are obtained by multiplying the input embedding of each word with Query, Key, Value matrix(and we have to learn the weights of these matrices only). So we obtain 3 vectors, named query, key, value. The second step is calculating the score. We calculate the score of each word against all the other words(it tells how related different words in a sentence are, to a specific word). It is calculated by taking the dot product of query vector with the key vector of the rest of the words. Once, this is calculated, we divide it by 8, and the resulting output numbers which are formed, we do softmax operation over them. It gives an analogy of how much each word of a sentence will be expressed at that particular position. And after softmax score is calculated, we multiply each score by the respective value vectors at that position(the intuition being how much value each word amounts to, and drawing out irrelevant words). And finally, we sum the resultant value vectors, which produces the output of self attention layer at this position. We then send this vector to the feed forward neural network. And we perform the same operation in parallel using different query, key, value matrices, to get "multiple representation" of the same space. It helps to find more relations between the words.



(e) **Fig 4**.epoch vs training loss, validation accuracy

7 Training

We implement the PyTorch version of ViVIT's Keras implementation. For our example, we use the MedMNIST Dataset, a large-scale lightweight benchmark for 3D Biomedical Image Classification. We implement the Tubelet Embedding sampling method. Here, we sample 3D frames from a video clip and perform 3D convolution, which is an extension of ViT's method for images. For calculating attention, of the four methods suggested for its implementation, we use the Spatio-Temporal attention model. The hyperparameters used are as follows: We used a batch size of 32, dropout to be 0.1, and Adam Optimizer, with a learning rate of 1×10^{-4}, and weight decay to be 1×10^{-5}, and classify the videos into 11 classes. The input is of a **28×28×28** 3D image. We use 8 attention heads and 8 transformer layers.

We have 971 training samples, with each sample containing a video of dimension 28×28×28 and a label describing what that video is (we have 11 labels in total, so 11 classes). We have to convert these videos into sequence-to-sequence tokens. Here's how we do it: We load these videos in batches of size 32, with the input channel being 1 (as it's black and white only), and each video of size 28×28×28. We pass these videos through our Tubelet Embedder, which performs 3D Convolution on the images, to reduce it to a size of **[128,3,3,3]**, which means, we have 128 output channels, and each channel is 3×3×3. To do this, we used a patch size of 8×8×8, zero padding and stride of 8.

Calculations: Output size $=$ [Input Dim $-$ Kernel Size]/stride $+ 1 = [28 - 8]/8 + 1 = 3$.

We then flatten the final 3×3×3 3D Matrix into a **1D 27 dimensional vector**, and add position embeddings to this 27 dimensional vector.

This can be seen as, we have sequences of 27 words each, with the dimension of each being 128, and we have to classify it into 11 classes.

We then pass it through a Transformer Block, we used 8 encoder-only blocks (as it's a classification task only). After this, we perform a pooling operation across each of the 128 dimensions for 27 sequences. So the dimension after this operation is **[32,128]**, where 32 is the batch size. After this, we perform a simple Feed forward operation, and classify these 128 vectors into 11 classes, using a 128×11 weight matrix in total, and then apply softmax to select the output class. We used cross entropy loss.

Final results: We received the test accuracy to be **77%**, and training loss converges to 1.57 and validation accuray after 60 epochs comes around **94%** .

References

[1] https://arxiv.org/abs/2103.15691

[2] https://keras.io/examples/vision/vivit/

[3] https://arxiv.org/abs/2010.11929

[4] https://arxiv.org/abs/1706.03762

[5] https://jalammar.github.io/illustrated-transformer/

[6] https://www.cse.iitd.ac.in/ mausam/courses/col772/spring2024/lectures/11-transformer.pdf