FAST VISION TRANSFORMER TRAINING

Igor Drozdov 317336451 Talel Levi 201366515

In the following project we aim to shorten the training process of visual transformers. In order to do so, we followed the following project:

[libffcv/ffcv-imagenet: Train ImageNet \*fast\* in 500 lines of code with FFCV (github.com)](https://github.com/libffcv/ffcv-imagenet)

they proposed to use the ffcv library in order to improve the training procedure.

A picture containing text

Description automatically generated

The FFCV ( fast forward computer vision ) library is a drop-in data loading system that dramatically increases data throughput in model training:

[libffcv/ffcv: FFCV: Fast Forward Computer Vision (and other ML workloads!) (github.com)](https://github.com/libffcv/ffcv)

How does ffcv improve the training procedure?

1. Indexing the data into a unique data structure and then when loading the data, all the data is uploaded to the memory unlike the torch dataloader.
2. Converting input type to memory efficient type such as mixed precision and etc.
3. Reducing the resolution of the input as the epochs progress, as a result the forward and the backward pass is faster.
4. Implementing efficient augmentations and new augmented conv2d layer which is called BlurPoolConv2d

As the aim to use transformers, 3 is not applicable in our case as the number of tokens has to remain constant. All the other bullet points can be used in our project as well.

***Dataset***

We used ImageNette as our dataset due to time constraints. But our preliminary results have shown the same pattern on ImageNet.

***Architecture***

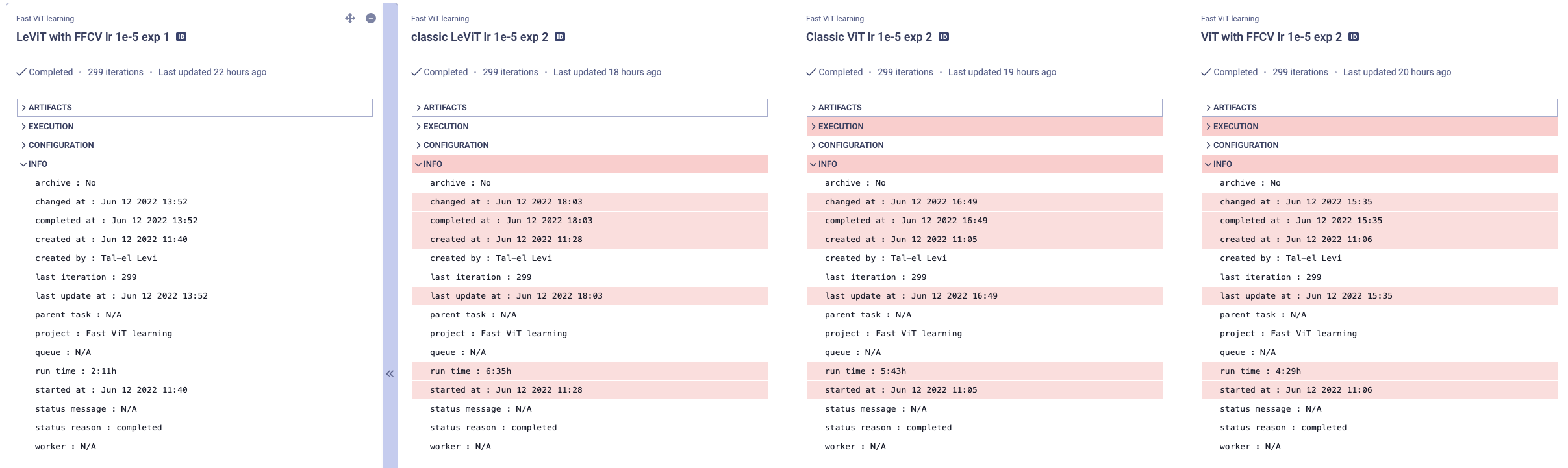
In the project we used the ViT and LeViT architectures. ViT is the ‘vanilla’ vision transformer while LeViT is a more efficient version of it.

**Results:**

(all experiments were conducted on Nvidia RTX 2080Ti GPU)

Since we were using the newton HPC servers we couldn’t “choose” which GPU we were assaigned to run our process. we ran the same experiment multiple times untill we had been assigned the same GPU for all experiments (Nvidia RTX 2080Ti) as different GPUS perforamnce could vastly change the results.

Training time:

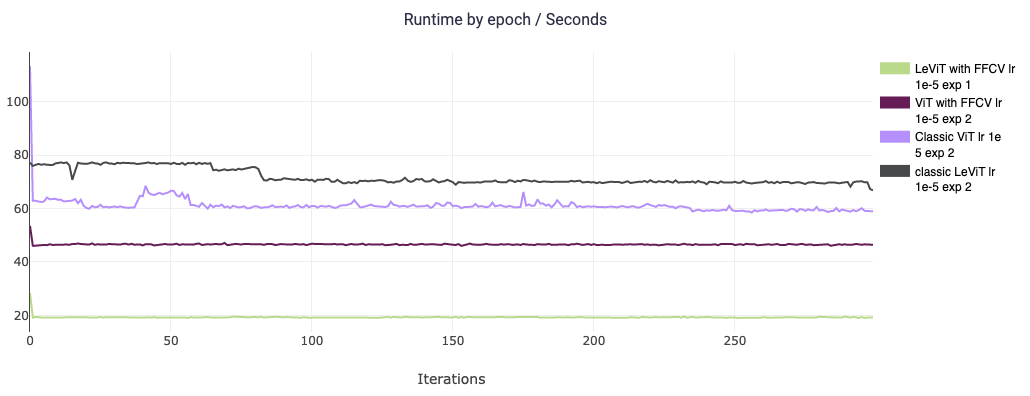


as we can see the runtime is vastly different between the models using FFCV and without it ( all models were training for 300 epochs. )



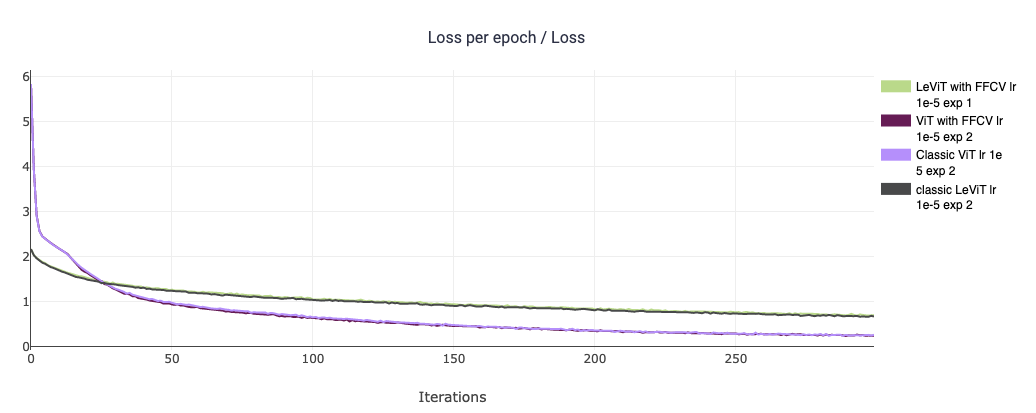
In the next graph we can see the runtime per epoch, as we can see the FFCV training is faster compared to the same model with out using FFCV.

One more benefit of using FFCV is the consistency of the training time per epoch as the dataloader hold the data in memory there is no hindernce to epoch training time caused by varying loading time of data.

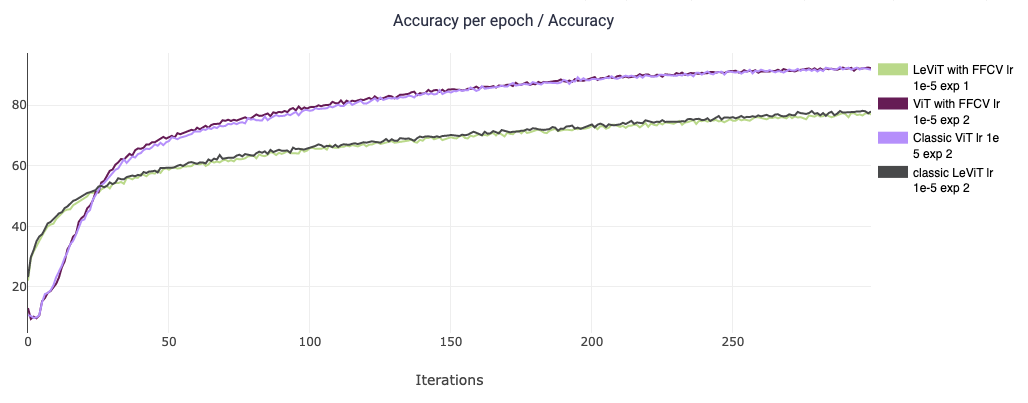
****

Impact on model’s performance:

In the next graphs we show that introduction of FFCV into the model doesn't hinders model’s performance. The loss behaves the same for the same architecture with and without using FFCV

****

so does the accuracy, we can see that the model’s accuracy isn’t hindered at all.

****

We weren’t focused on achiving high accuracy and did not optimize hyperparameters in order to do so.

The hyperparameters were the same for all experiments in order to make sure the only difference is the incorporation of FFCV into the models.

**Conclusions**

As showen in the graphs, incorporating the ffcv library into the traditional training procedure can significantly shorten the training times. Up to 20% times reduction in ViT and Up to **75%** times reduction in LeViT!!!

As we have shown the perforamnce of the network didn’t suffer any hinderence due to the incoropration of FFCV.

Lastly, as we can notice the same pattern in both architectures, we can also expect similar results on other visual transformers.

***Code***

[TalelLevi/ftt: attempt at fast transformer training (github.com)](https://github.com/TalelLevi/ftt)

***Future Work***

We propose to investigate new efficient transformer architectures that utilize elements of local attention such as NAT [[2204.07143] Neighborhood Attention Transformer (arxiv.org)](https://arxiv.org/abs/2204.07143)

and more efficient mechanism of self attention such as recently published MobileViTv2 [[2206.02680] Separable Self-attention for Mobile Vision Transformers (arxiv.org)](https://arxiv.org/abs/2206.02680) .

for imporoving inference time an easy approch could be:

<https://medium.com/pytorch/accelerating-pytorch-inference-with-torch-tensorrt-on-gpus-896e06ff1637>