Ajou University

2024.12.04

# Enhancing Vision Transformer's Inference Speed

Integrating Hybrid Architectures and Novel Similarity Measures

**Department of Software & Computer Engineering**

JEONCHAN GANG, HYEONWON NAM, YONGHUN JEONG DONGYEOP HAN

# Table of contents

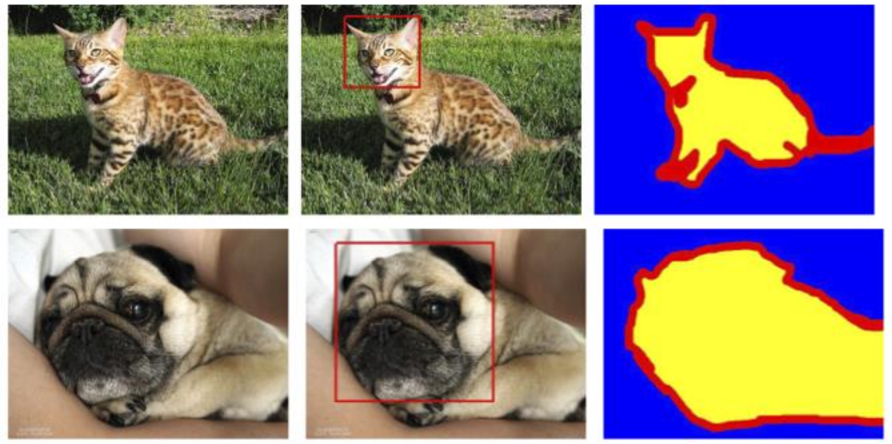
# 00 Learning Condition

## Minimum Performance guaranteed

| ImageNet (1000 classes) | Number of Images | Size (GB) |
|---|---|---|
| **Training** | 1,281,167 | 138GB |
| **Validation** | 50,000 | 6.3GB |
| **Test** | 100,000 | 12.57GB |

Batch size : 32
Learning rate : 1e-3
Epoch : 50

Initially, we tried to use ImageNet as Dataset,
but it was too big and no longer supported by Pytorch Dataset library.



| Oxford-IIIT Pet (37 classes) | Number of Images | Size (GB) |
|---|---|---|
| **Training** | 3,680 | 0.77GB |
| **Validation** | 3,669 | 0.77GB |
| **Test** | - | - |

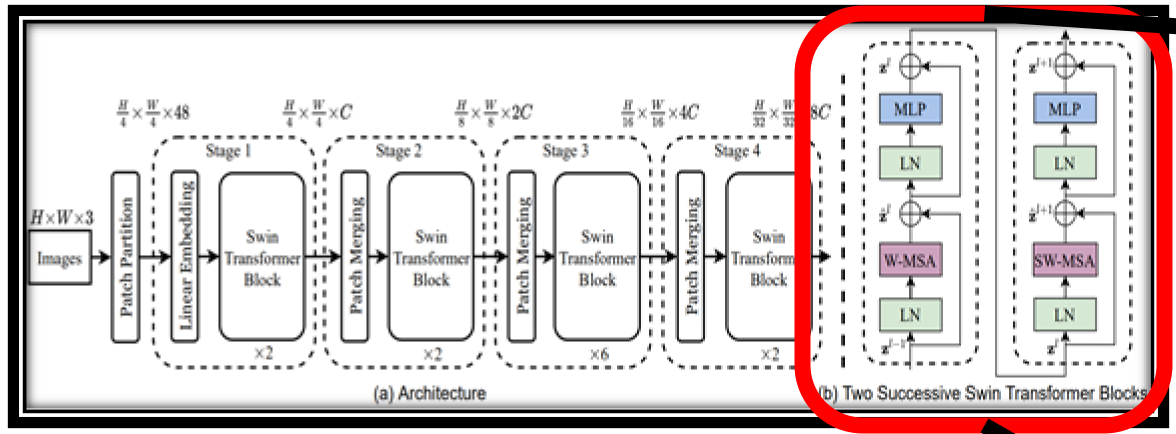
**7 : 1.5 : 1.5 Split**

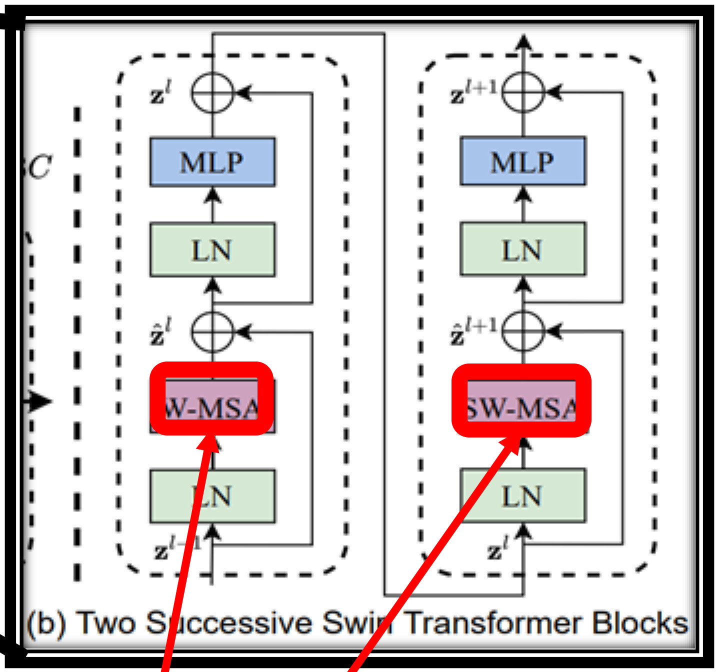
| Oxford-IIIT Pet (37 classes) | Number of Images |
|---|---|
| **Training** | 5,114 |
| **Validation** | 1,102 |
| **Test** | 1,103 |

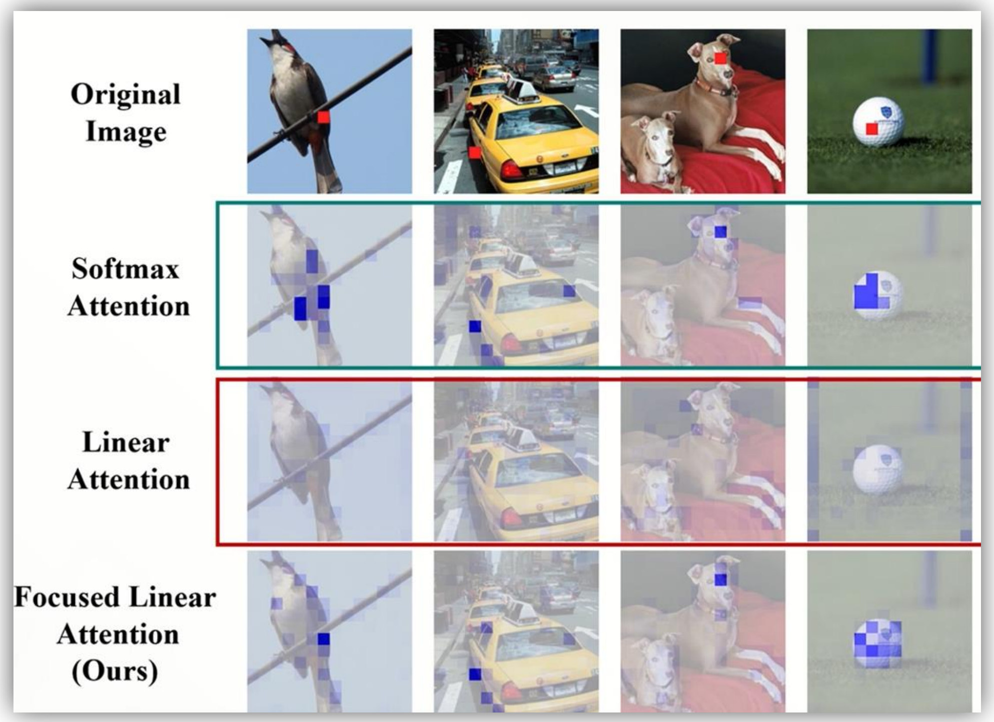
Among the things being supported by Pytorch,
we trained with a smaller dataset, Oxford-IIIT Pet (Pet37).

# 01 Swin Based Approach

## Swin + FLatten Transformer

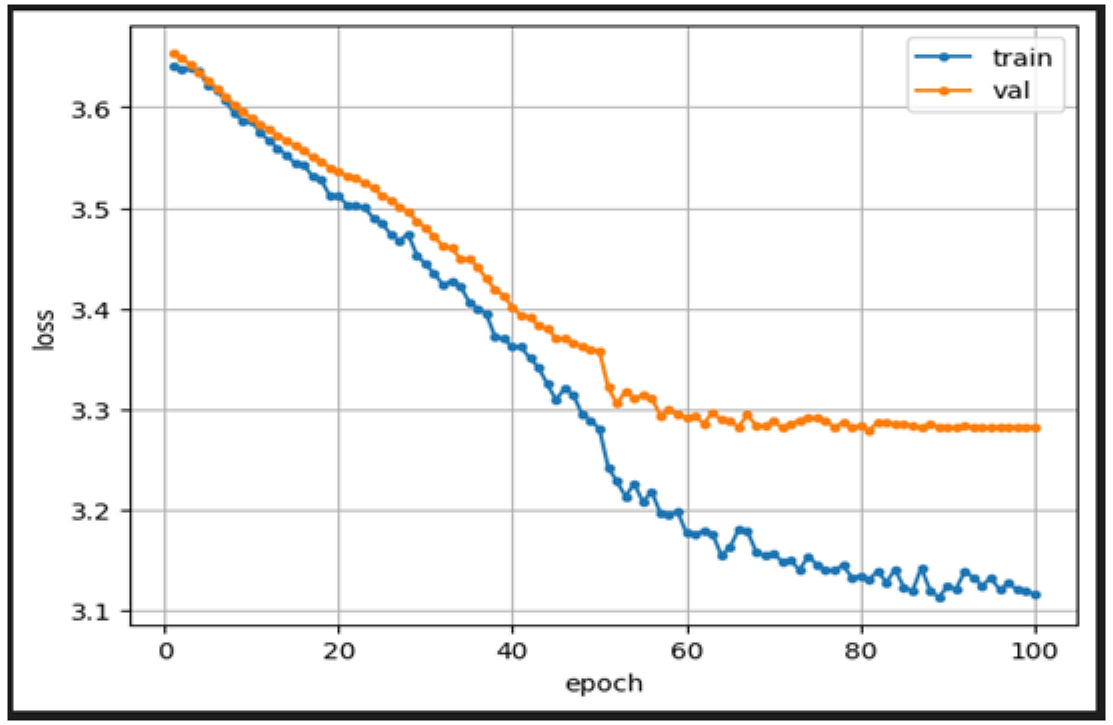



Replaced the self-attention
with focused linear attention.

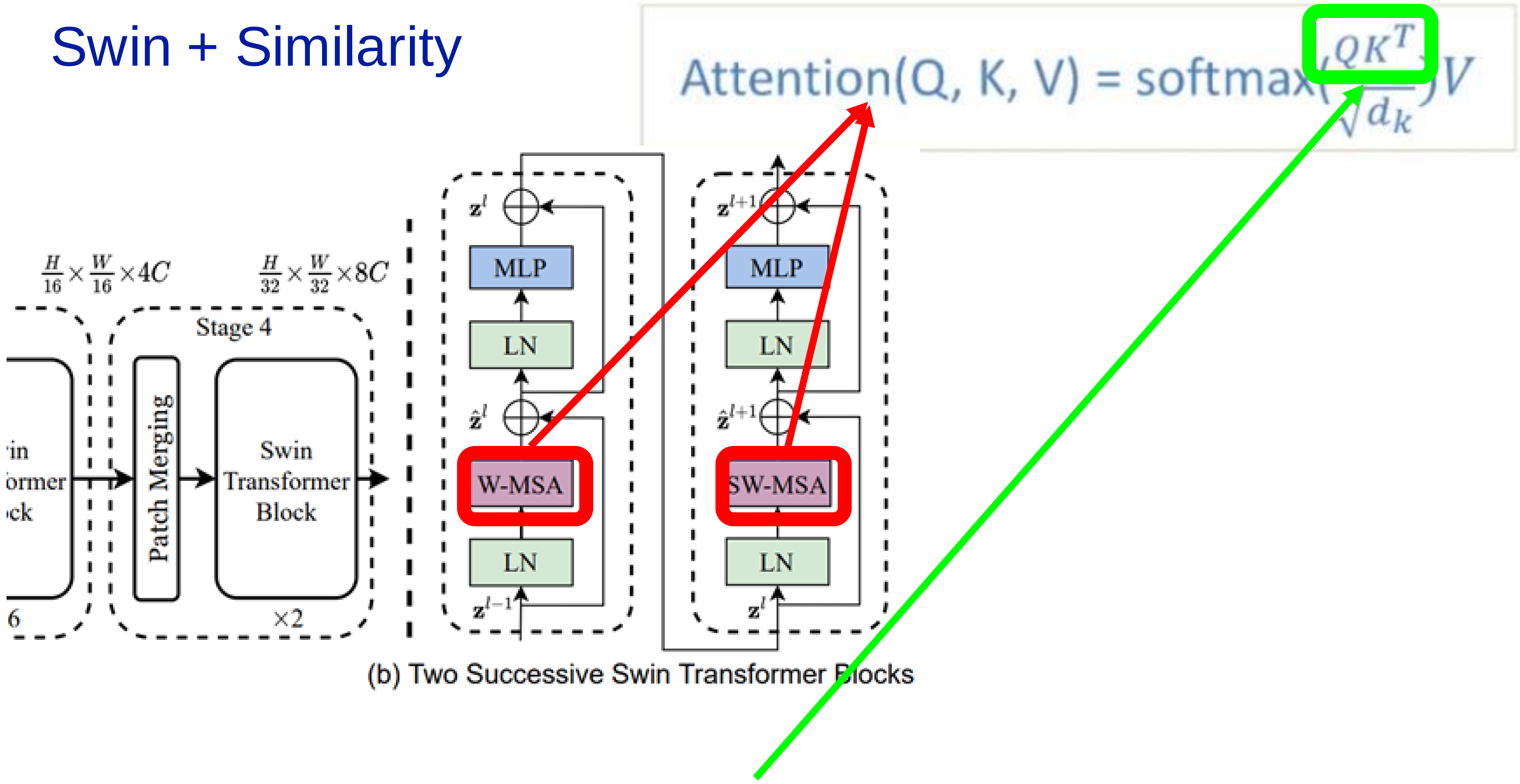
We have tested this combination
with different window sizes, focus factors, etc.




Overfitting was addressed,
but the model failed to train accurately.

# 01 Swin Based Approach

## Swin + Similarity



(b) Two Successive Swin Transformer Blocks

Change similarity of Q and K Measurement of Attention

1. Jaccard similarity function

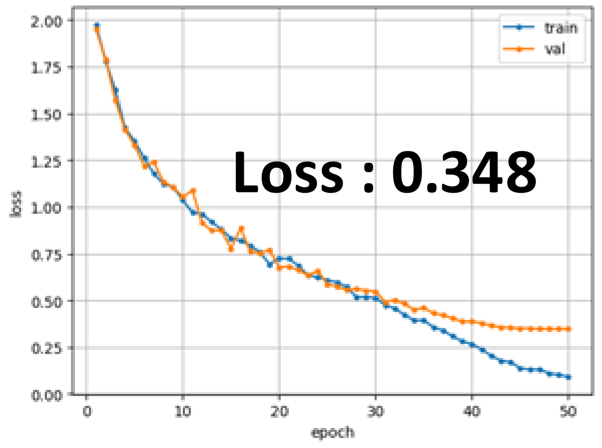
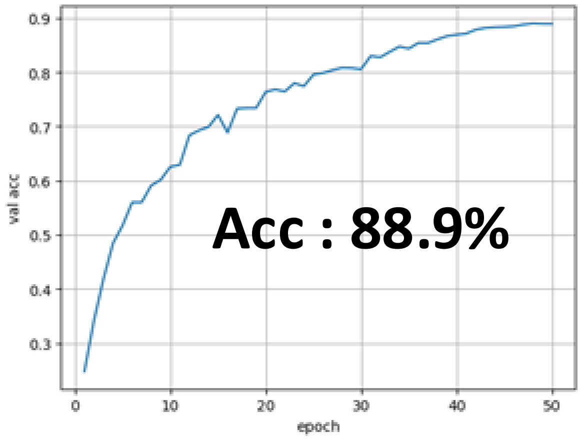
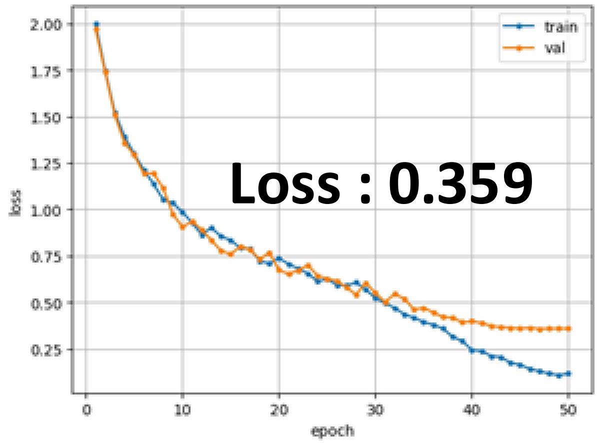
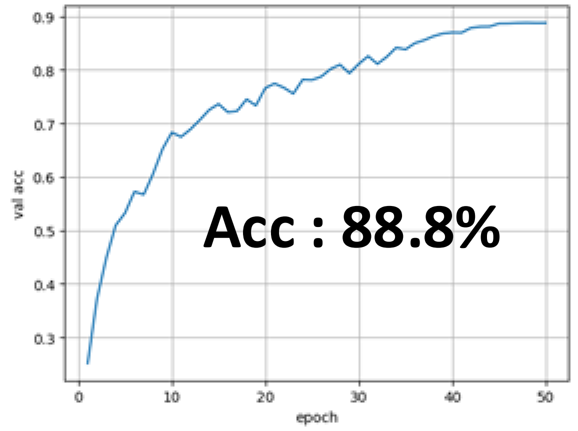
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

2. Euclidian similarity function

$$d(x,y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

We confirmed that the inference time was **similar or slightly faster** than when using the similarity function using the existing inner product.

This showed better accuracy than original. **Loss : 0.394, Acc : 87.9%**

| Set | Result |
|---|---|
| Swin + Jaccard |  Loss : 0.348  Acc : 88.9%<br>**Average inference time : 10.83ms** |
| Swin + Euclidian |  Loss : 0.359  Acc : 88.8%<br>**Average inference time : 9.79ms** |

Both similarity metrics shows good training without overfitting.

We used CIFAR-10 ( 32 X 32)

# Q: Does Swin improve the inference time?

| method | image size | #param. | FLOPs | throughput (image / s) | ImageNet top-1 acc. |
|---|---|---|---|---|---|
| **(a) Regular ImageNet-1K trained models** | | | | | |
| RegNetY-4G [48] | $224^2$ | 21M | 4.0G | 1156.7 | 80.0 |
| RegNetY-8G [48] | $224^2$ | 39M | 8.0G | 591.6 | 81.7 |
| RegNetY-16G [48] | $224^2$ | 84M | 16.0G | 334.7 | 82.9 |
| EffNet-B3 [58] | $300^2$ | 12M | 1.8G | 732.1 | 81.6 |
| EffNet-B4 [58] | $380^2$ | 19M | 4.2G | 349.4 | 82.9 |
| EffNet-B5 [58] | $456^2$ | 30M | 9.9G | 169.1 | 83.6 |
| EffNet-B6 [58] | $528^2$ | 43M | 19.0G | 96.9 | 84.0 |
| EffNet-B7 [58] | $600^2$ | 66M | 37.0G | 55.1 | 84.3 |
| ViT-B/16 [20] | $384^2$ | 86M | 55.4G | 85.9 | 77.9 |
| ViT-L/16 [20] | $384^2$ | 307M | 190.7G | 27.3 | 76.5 |
| DeiT-S [63] | $224^2$ | 22M | 4.6G | 940.4 | 79.8 |
| DeiT-B [63] | $224^2$ | 86M | 17.5G | 292.3 | 81.8 |
| DeiT-B [63] | $384^2$ | 86M | 55.4G | 85.9 | 83.1 |
| Swin-T | $224^2$ | 29M | 4.5G | 755.2 | 81.3 |
| Swin-S | $224^2$ | 50M | 8.7G | 436.9 | 83.0 |
| Swin-B | $224^2$ | 88M | 15.4G | 278.1 | 83.5 |
| Swin-B | $384^2$ | 88M | 47.0G | 84.7 | 84.5 |
| **(b) ImageNet-22K pre-trained models** | | | | | |
| method | image size | #param. | FLOPs | throughput (image / s) | ImageNet top-1 acc. |
| R-101x3 [38] | $384^2$ | 388M | 204.6G | - | 84.4 |
| R-152x4 [38] | $480^2$ | 937M | 840.5G | - | 85.4 |
| ViT-B/16 [20] | $384^2$ | 86M | 55.4G | 85.9 | 84.0 |
| ViT-L/16 [20] | $384^2$ | 307M | 190.7G | 27.3 | 85.2 |
| Swin-B | $224^2$ | 88M | 15.4G | 278.1 | 85.2 |
| Swin-B | $384^2$ | 88M | 47.0G | 84.7 | 86.4 |
| Swin-L | $384^2$ | 197M | 103.9G | 42.1 | 87.3 |

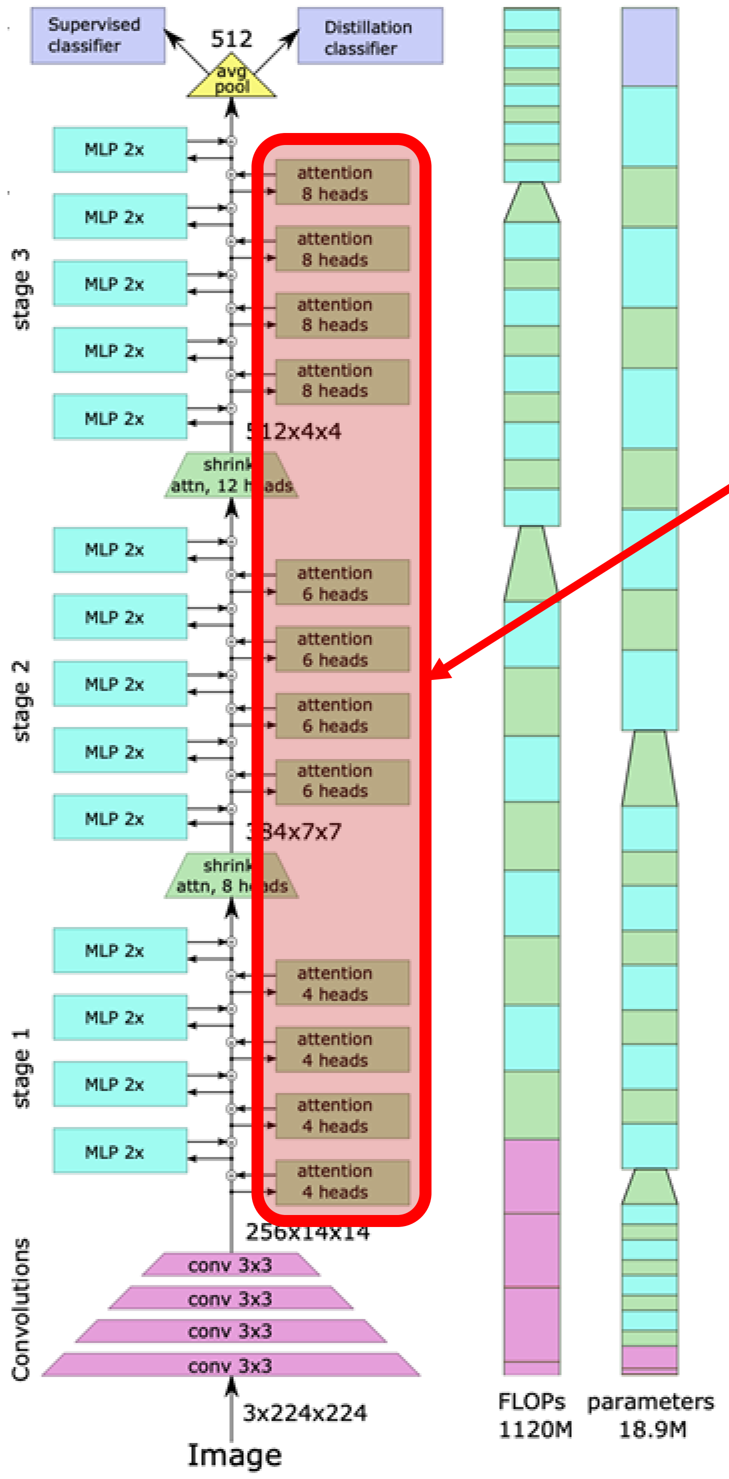
Table 1. Comparison of different backbones on ImageNet-1K classification. Throughput is measured using the GitHub repository of [68] and a V100 GPU, following [63].

Swin Transformer was proposed for ***higher accuracy*** compared to the models with similar size, params, throughput.

Since this architecture is not intended for faster inference, we decided to drop this and moved to LeViT architecture.

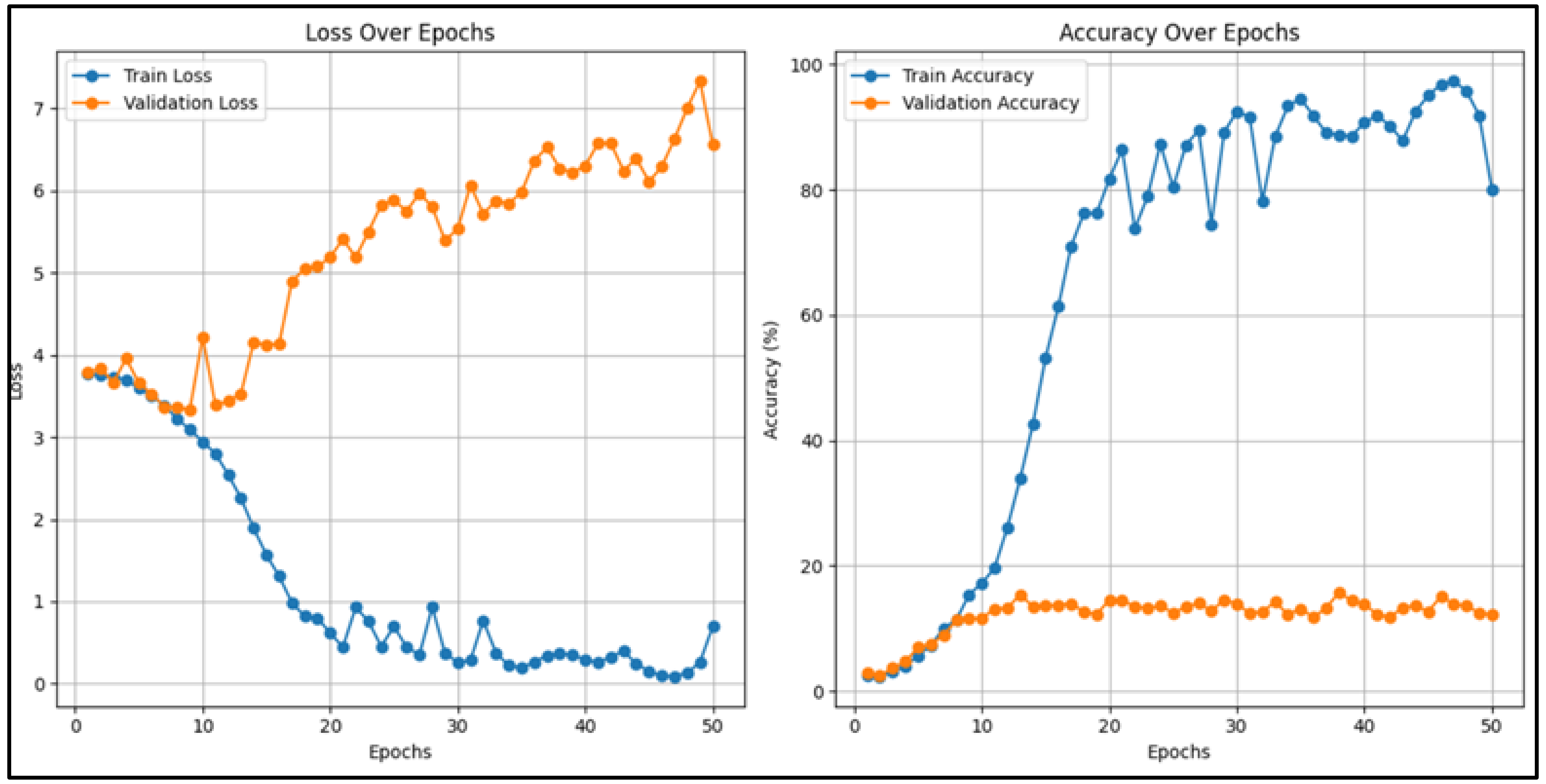
# 02 LeViT Based Approach

## LeViT + FLatten Transformer



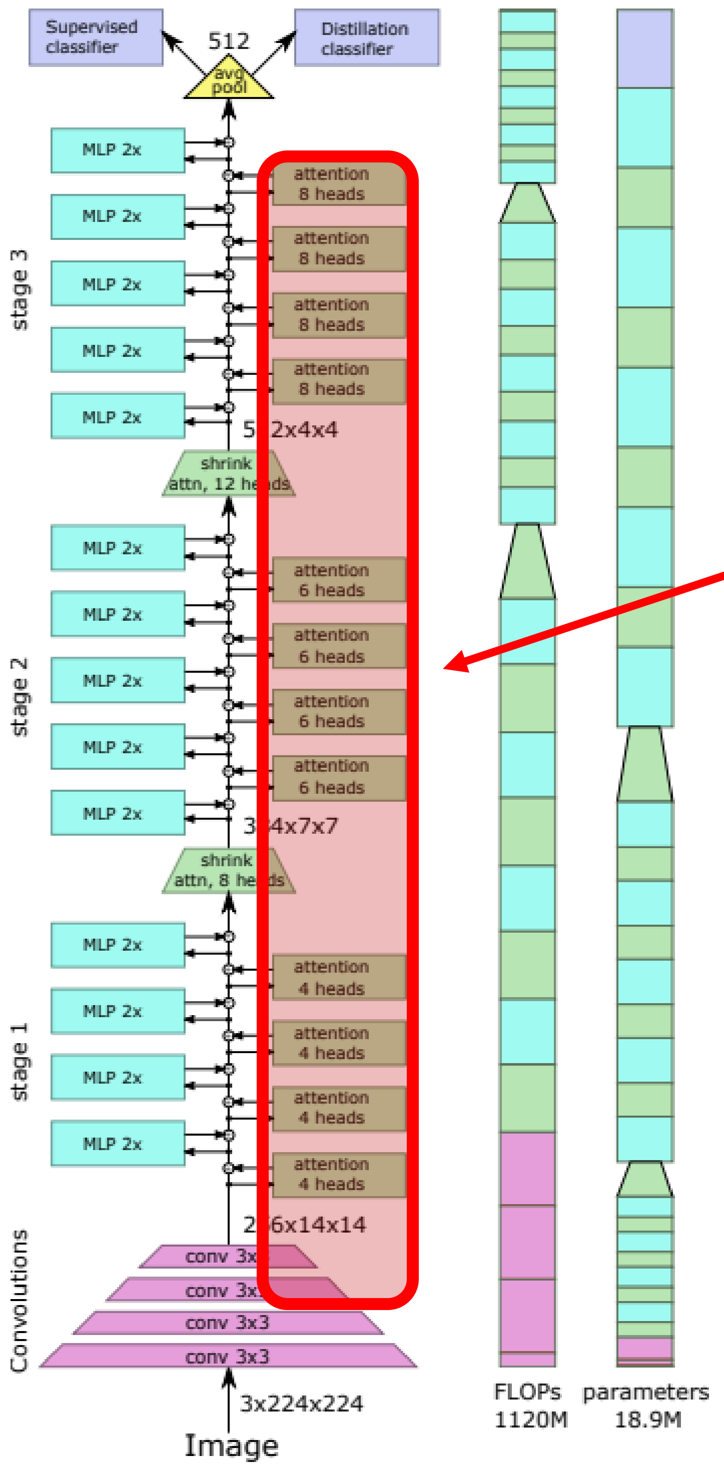

Replaced the attention block with focused linear attention.

Overfitting occurred, like the case with Swin + FLA.



# 02 LeViT Based Approach

## LeViT + GLA(Gated Linear Attention)




**+**

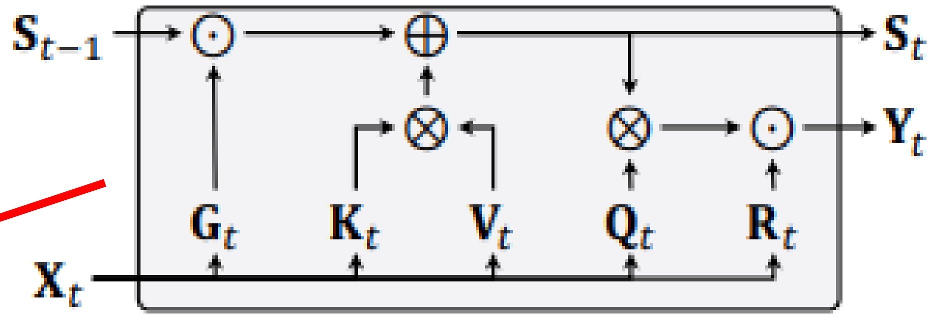



Figure 4. Pipeline of GLA.

$$S'_{t-1} = G_t \odot S_{t-1} \in \mathbb{R}^{d_k \times d_v}, \quad (7)$$
$$S_t = S'_{t-1} + K_t^\top V_t \in \mathbb{R}^{d_k \times d_v}, \quad (8)$$
$$O_t = Q_t^\top S_t \in \mathbb{R}^{1 \times d_v}, \quad (9)$$
$$R_t = \mathtt{Swish}(X_t W_r + b_r) \in \mathbb{R}^{1 \times d_v}, \quad (10)$$
$$Y_t = (R_t \odot \mathrm{LN}(O_t)) W_O \in \mathbb{R}^{1 \times d}, \quad (11)$$

※⊙ is the element-wise multiplication operation

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \mathbf{K} = \mathbf{X}\mathbf{W}_K, \mathbf{V} = \mathbf{X}\mathbf{W}_V, \quad (4)$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are linear projection weights. The dimension number of $\mathbf{Q}$, $\mathbf{K}$ is $d_k$, and $d_v$ is for $\mathbf{V}$. Next, GLA compute the gating matrix $\mathbf{G}$ as follows:

$$\mathbf{G}_t = \alpha_t^\top \beta_t \in \mathbb{R}^{d_k \times d_v}, \alpha = \sigma(\mathbf{X}\mathbf{W}_\alpha + b_\alpha)^{\frac{1}{\tau}} \in \mathbb{R}^{L \times d_k}, \quad (5)$$

$$\beta = \sigma(\mathbf{X}\mathbf{W}_\beta + b_\beta)^{\frac{1}{\tau}} \in \mathbb{R}^{L \times d_v}, \quad (6)$$

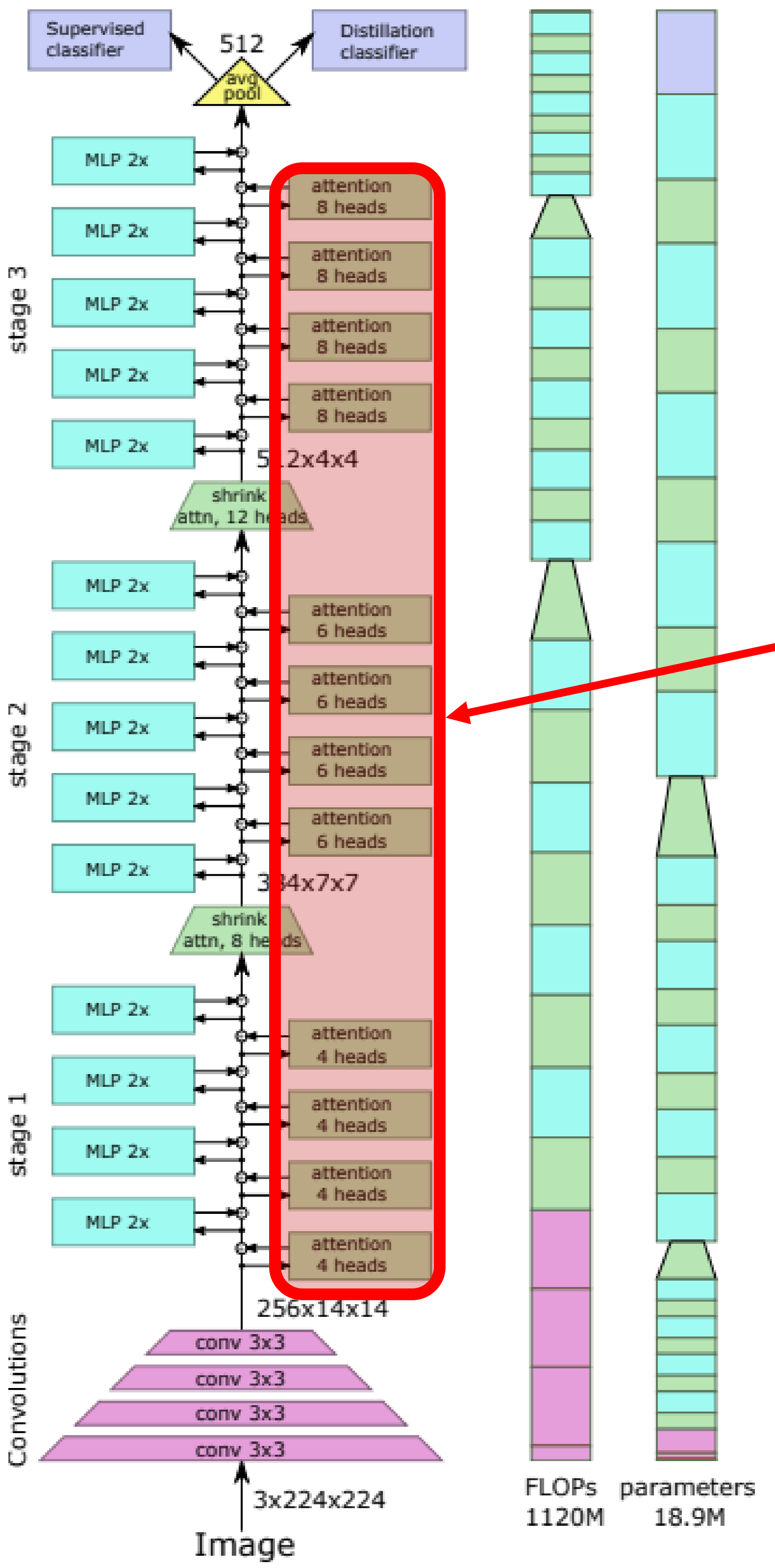
Similar to LSTM, GLA also uses a **gating mechanism**.

However, GLA achives computational efficiency by employing parallelized linear attention.

So we had replaced Attention blocks to GLA blocks to improve inference time.

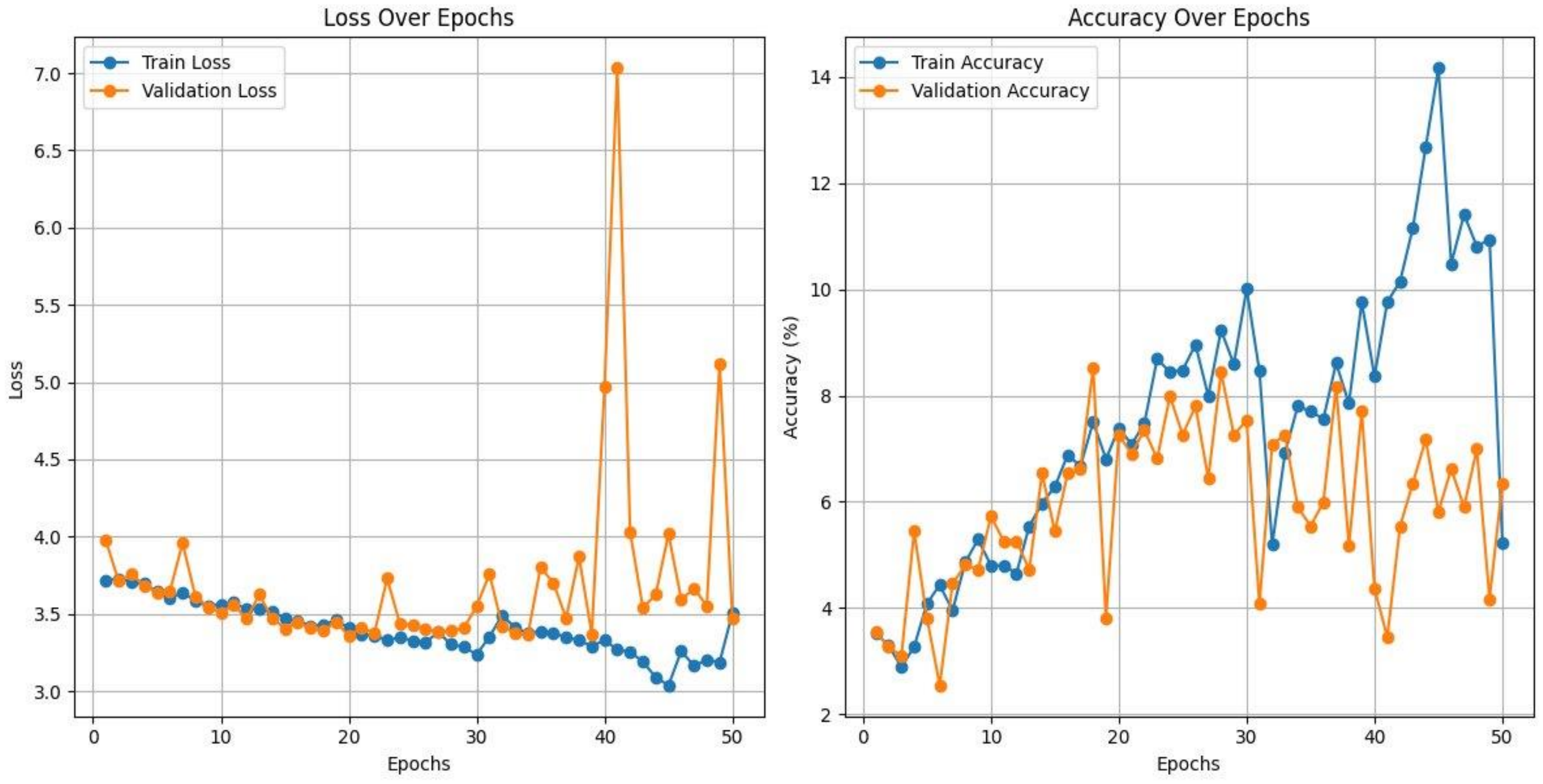
# 02 LeViT Based Approach

## LeViT + GLA




```
(0): LevitBlock(
  (attn): GatedLinearAttention(
    (q_proj): Linear(in_features=256, out_features=128, bias=False)
    (k_proj): Linear(in_features=256, out_features=128, bias=False)
    (k_gate): Sequential(
      (0): Linear(in_features=256, out_features=16, bias=False)
      (1): Linear(in_features=16, out_features=128, bias=True)
    )
    (v_proj): Linear(in_features=256, out_features=256, bias=False)
    (g_proj): Linear(in_features=256, out_features=256, bias=True)
    (out_proj): Linear(in_features=256, out_features=256, bias=False)
    (group_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=False)
  )
```

**replace attention mechanism to GLA**

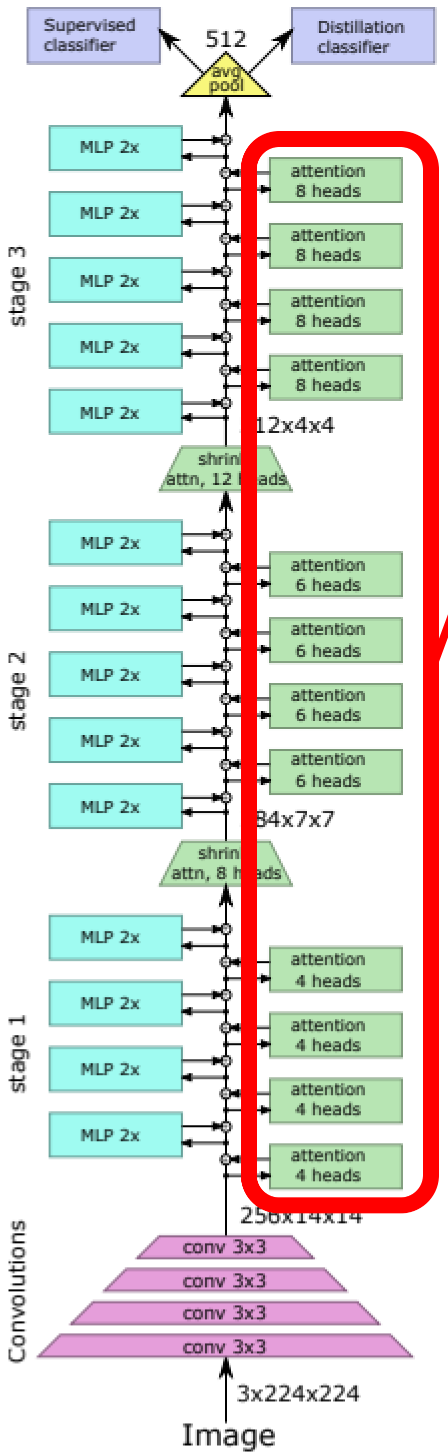



**LeViT_GLA's Average Inference Time = 14.13 ms**

Although the inference time has improved
compared to the original LeViT as shown in the plot above,
there is an extreme change in loss after 40 epochs,
and the accuracy also exhibits inconsistency.

# 02 LeViT Based Approach

## LeViT + Similarity




$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Same as Swin + Similarity

Change similarity of Q and K Measurement of Attention

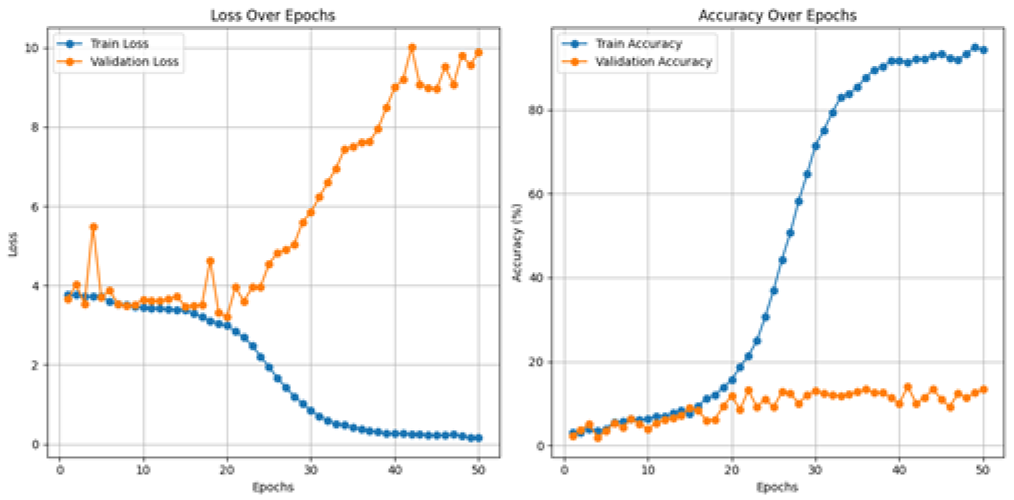
1. Jaccard similarity function

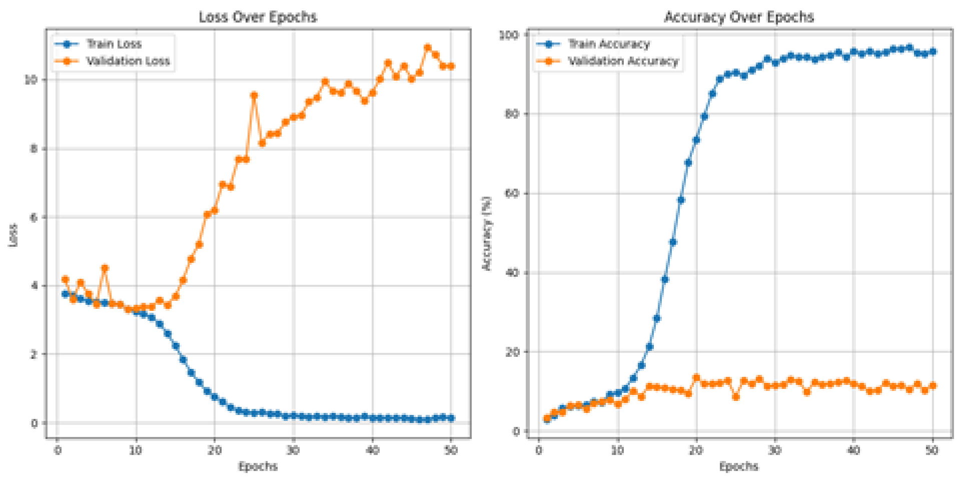
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

1. Euclidian similarity function

$$d(x,y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

**Unlike Swin + Similarity**

<table>
<tr><th>Set</th><th>Result</th></tr>
<tr><td>LeViT + Jaccard</td><td>


Average inference time : 84.42ms</td></tr>
<tr><td>LeViT + Euclidian</td><td>


Average inference time : 42.36ms</td></tr>
</table>

**Original LeViT Average inference time : 16.29ms**

REJECTED

# 03 Launcher Architecture

## Minimum Performance guaranteed

Apart from the inference time, the overall accuracy was low when trained without hyperparameter tuning at 50 epochs.

```
Final Test Evaluation
Test: 100%¦██████████¦ 35/35 [00:05<00:00,  6.21it/s]
Test Loss: 6.6629, Test Accuracy: 11.97%
```

LeViT + FLA implement

```
Final Test Evaluation
Test: 100%¦██████████¦ 35/35 [00:06<00:00,  5.35it/s]
Test Loss: 3.4992, Test Accuracy: 6.26%
```

LeViT + GLA implement

```
Test: 100%¦██████████¦ 35/35 [00:08<00:00,  4.12it/s]
Test Loss: 9.9227, Test Accuracy: 10.70%
Test Inference Time: 0m 8.50s
```

LeViT + Jaccard implement

If it is a problem with model architecture, it should be well trained for verified models taken from timm.

However, we can also see that the model taken from timm is also poorly trained when it is trained for 50 epochs in same dataset.

```
Final Test Evaluation
Test: 100%¦██████████¦ 35/35 [00:05<00:00,  5.87it/s]
Test Loss: 4.4714, Test Accuracy: 34.45%
```

ResNet50 from timm

```
Final Test Evaluation
Test: 100%¦██████████¦ 35/35 [00:08<00:00,  4.31it/s]
Test Loss: 3.5597, Test Accuracy: 4.62%
```

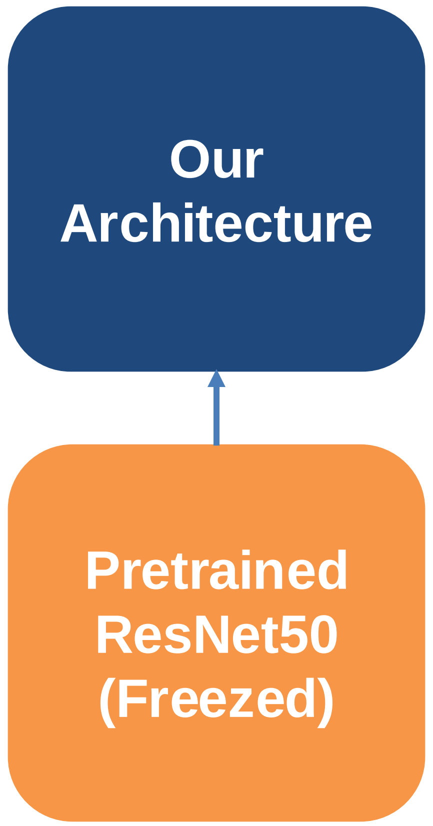
ViT from timm

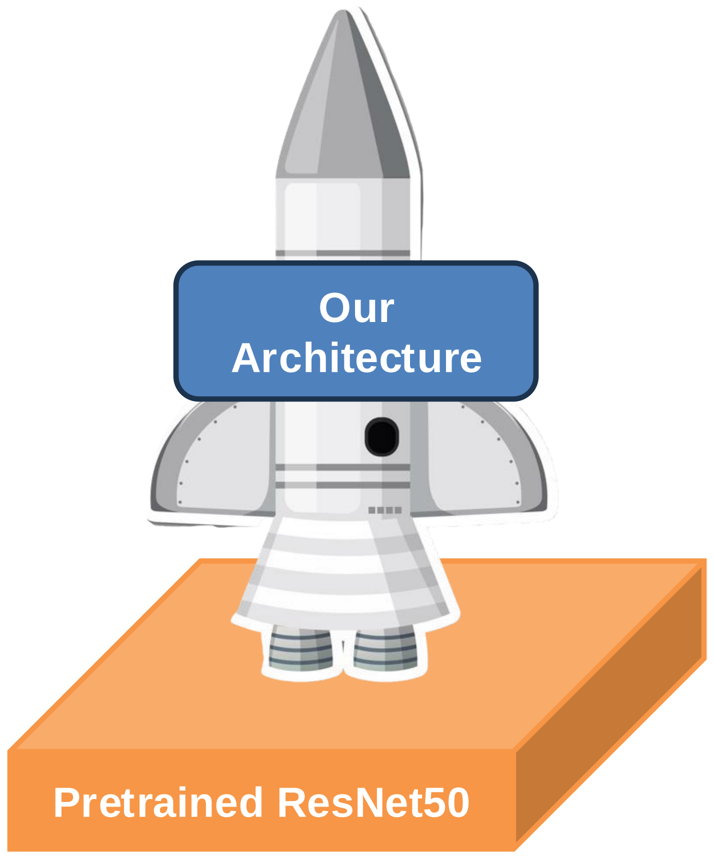
```
Final Test Evaluation
Test: 100%¦██████████¦ 35/35 [00:08<00:00,  4.29it/s]
Test Loss: 3.6203, Test Accuracy: 1.81%
```

SWIN from timm

# 03 Launcher Architecture

Minimum Performance guaranteed



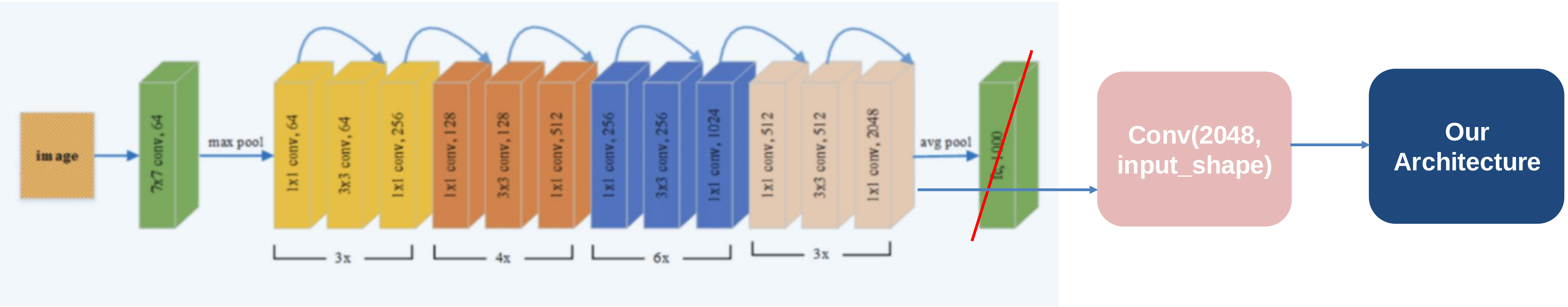




It's impossible to train with larger dataset on Colab.
So, we had to find another way to show that our model accuracy is guaranteed.

Pre-trained ResNet50 was placed in front of the architecture to maintain the accuracy.
We decided to call the model with pretrained ResNet50 as Launcher.

# 03 Launcher Architecture

Minimum Performance guaranteed



# 03 Launcher Architecture

## Minimum Performance guaranteed

| | LeViT implement | L_LeViT implement | L_LeViT + FLA | L_LeViT + GLA | L_LeViT + Jaccard |
|---|---|---|---|---|---|
| Top-1 Acc | 2.81% | **83.59%** | **84.30%** | 49.32% | **86.49%** |
| Loss | 3.9639 | 0.8333 | 0.7668 | 2.8617 | 0.8176 |

In the case of the Launcher with pre-trained ResNet50, we can see that accuracy is increased except for GLA.
This shows that the architecture we proposed does not lose class information.

For GLA however, it seemed prior class knowledge extracted by ResNet50 is lost during the process.

Technically, even though the Launcher model guarantees accuracy,
it does not mean that the architecture guarantees accuracy in full tuning.

It means that the architecture implemented in an environment with limited datasets
does not compromise the pre-learned information.

Therefore, we will show that the accuracy is guaranteed using the Laucher model in a limited environment
and try to compare the inference time.

# 04 New Ideas

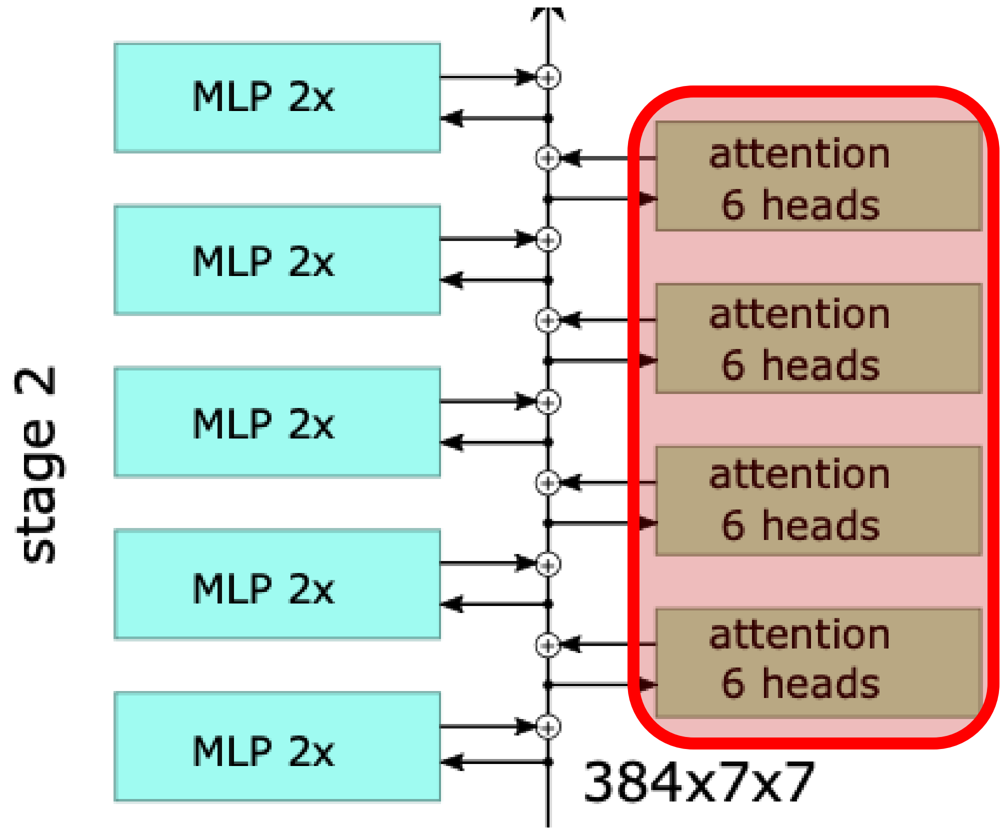
## IDEAS

None of the FLA, GLA, and similarity methods had a significant improvement on the inference time. So we have experimented with 5 ideas below.

1. Replaced some of the attention Blocks with ConvNd layer.
2. Checkerboard masking for attention input.
3. Decreased the number of attention blocks.
4. Changed attention-based downsampling block to convolution-based downsampling block.
5. Applied Dropout.

# 04 New Ideas

## 1. 1x1 Conv




```
Input (384 channels)
       |
  +--------------------+
  |    Conv2d          |  Kernel: 1x1, Stride: 1, Padding: 0
  |  (384 -> 512)      |
  +--------------------+
       |
  +--------------------+
  |  BatchNorm2d       |
  +--------------------+
       |
  +--------------------+
  |      ReLU          |
  +--------------------+
       |
  +--------------------+
  |    Conv2d          |  Kernel: 1x1, Stride: 1, Padding: 0
  |  (512 -> 512)      |
  +--------------------+
       |
  +--------------------+
  |  BatchNorm2d       |
  +--------------------+
       |
  +--------------------+
  |      ReLU          |
  +--------------------+
       |
Output (512 channels)
```

We replaced the attention block with Conv with matching input/output channel size.

Conv-Batch-ReLU is composed of one block. (Using 1x1 Conv)

# 04 New Ideas

## 1. 1x1 Conv

LeViT has stages including three attention blocks.

Stages 1 ~ 3 were compared with each other,
by replacing them with Conv respectively.

| | Control Group(L_LeViT) | Stage1 -> 1x1 Conv | Stage2 -> 1x1 Conv | Stage3 -> 1x1 Conv |
|---|---|---|---|---|
| Conv Block Param | - | 395,776 | 790,272 | 921,600 |
| Top-1 Acc | 83.59% | 1.54% | 86.22% | **82,77%** |
| Inference Time | 22.52ms | 19.21ms | 22.92ms | **21.14ms** |

By replacing the first stage with Conv,
model cannot extract features properly,
affecting the lower layer and showing a low accuracy.

Replacing stage 3 is faster, while maintaining accuracy.


```
====================================================================================================
Layer (type:depth-idx)                                 Output Shape              Param #
====================================================================================================
LevitDistilled                                         [32, 37]                  --
├─Stem16: 1-1                                          [32, 256, 14, 14]         --
│    └─ConvNorm: 2-1                                   [32, 32, 112, 112]        --
│    │    └─Conv2d: 3-1                                [32, 32, 112, 112]        864
│    │    └─BatchNorm2d: 3-2                           [32, 32, 112, 112]        64
│    └─Hardswish: 2-2                                  [32, 32, 112, 112]        --
│    └─ConvNorm: 2-3                                   [32, 64, 56, 56]          --
│    │    └─Conv2d: 3-3                                [32, 64, 56, 56]          18,432
│    │    └─BatchNorm2d: 3-4                           [32, 64, 56, 56]          128
│    └─Hardswish: 2-4                                  [32, 64, 56, 56]          --
│    └─ConvNorm: 2-5                                   [32, 128, 28, 28]         --
│    │    └─Conv2d: 3-5                                [32, 128, 28, 28]         73,728
│    │    └─BatchNorm2d: 3-6                           [32, 128, 28, 28]         256
│    └─Hardswish: 2-6                                  [32, 128, 28, 28]         --
│    └─ConvNorm: 2-7                                   [32, 256, 14, 14]         --
│    │    └─Conv2d: 3-7                                [32, 256, 14, 14]         294,912
│    │    └─BatchNorm2d: 3-8                           [32, 256, 14, 14]         512
├─Sequential: 1-2                                      [32, 9, 512]              --
│    └─LevitStage: 2-8                                 [32, 196, 256]            --
│    │    └─Identity: 3-9                              [32, 196, 256]            --           stage1
│    │    └─Sequential: 3-10                           [32, 196, 256]            1,583,616
│    └─LevitStage: 2-9                                 [32, 49, 384]             --
│    │    └─LevitDownsample: 3-11                      [32, 49, 384]             1,746,176    stage2
│    │    └─Sequential: 3-12                           [32, 49, 384]             3,555,072
│    └─LevitStage: 2-10                                [32, 9, 512]              --
│    │    └─LevitDownsample: 3-13                      [32, 9, 512]              4,211,072    stage3
│    │    └─Sequential: 3-14                           [32, 9, 512]              4,208,640
├─NormLinear: 1-3                                      [32, 37]                  --
│    └─BatchNorm1d: 2-11                               [32, 512]                 1,024
│    └─Dropout: 2-12                                   [32, 512]                 --
│    └─Linear: 2-13                                    [32, 37]                  18,981
├─NormLinear: 1-4                                      [32, 37]                  --
│    └─BatchNorm1d: 2-14                               [32, 512]                 1,024
│    └─Dropout: 2-15                                   [32, 512]                 --
│    └─Linear: 2-16                                    [32, 37]                  18,981
====================================================================================================
Total params: 15,733,482
Trainable params: 15,733,482
Non-trainable params: 0
Total mult-adds (G): 33.33
====================================================================================================
```

# 04 New Ideas

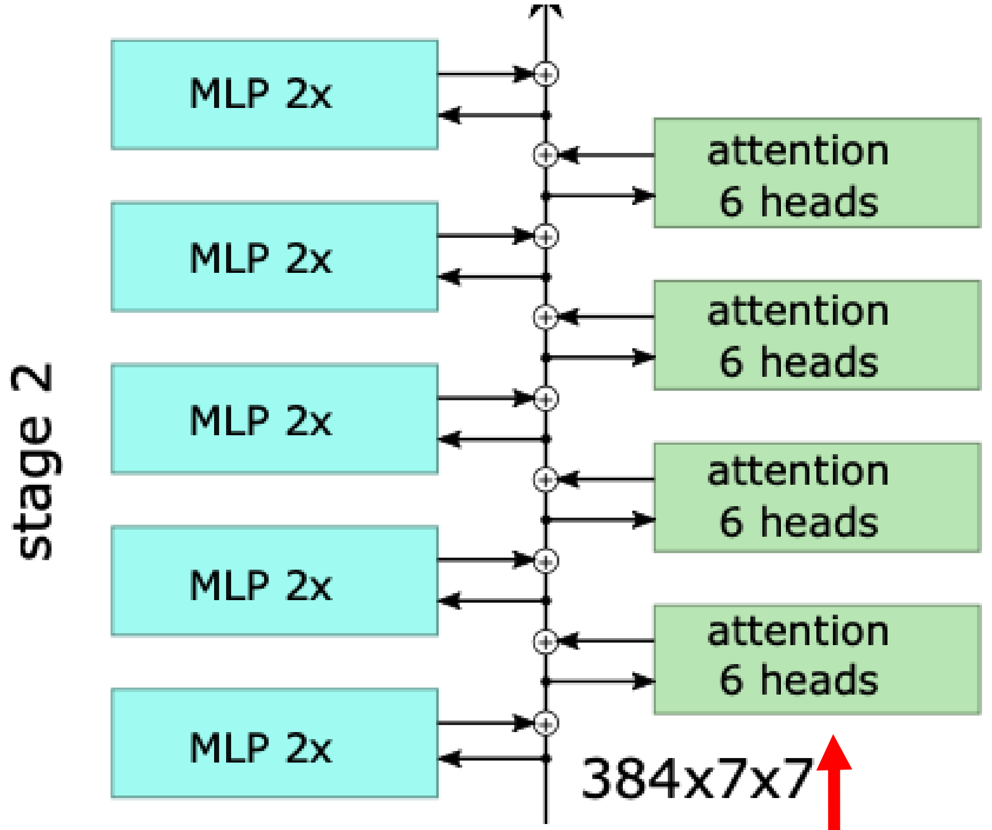
## 1. 1x1 Conv

| | Control Group(LeViT) | Stage3 -> 1x1 Conv |
|---|---|---|
| Inference Time | 10.50ms | **7.94ms** |

Additionally, we compared vanilla LeViT model with the modified one, which still showed faster inference time.

# 04 New Ideas

## 2. Checkerboard Masking




Checkerboard mask

$$\begin{matrix} 5 & 3 & 8 & 7 & 9 \\ 4 & 6 & 2 & 5 & 1 \\ 9 & 8 & 7 & 6 & 4 \\ 3 & 2 & 5 & 8 & 7 \\ 6 & 4 & 9 & 3 & 1 \end{matrix} \quad \text{element-wise multiplication} \quad \begin{matrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \end{matrix} = \begin{matrix} 5 & 0 & 8 & 0 & 9 \\ 0 & 6 & 0 & 5 & 0 \\ 9 & 0 & 7 & 0 & 4 \\ 0 & 2 & 0 & 8 & 0 \\ 6 & 0 & 9 & 0 & 1 \end{matrix}$$

| | Control Group(LeViT) | Masked LeViT |
|---|---|---|
| Inference Time | **10.50ms** | 11.55ms |

We thought masking the input to the attention block would reduce computation.

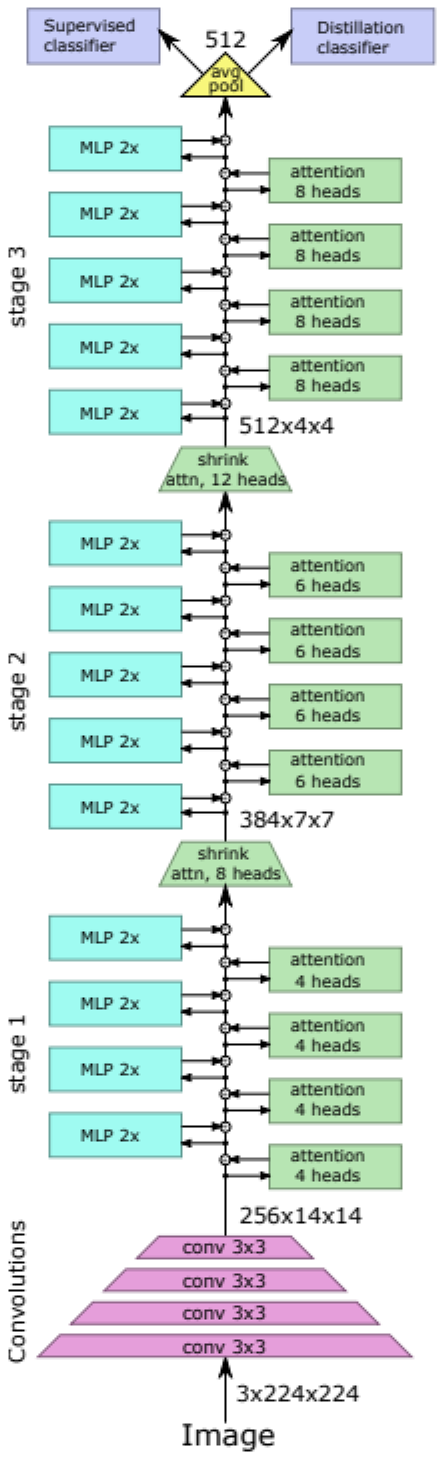
However, when we applied masking, it became slightly slower.

This could be because creating and applying the masking added extra computational overhead.

REJECTED

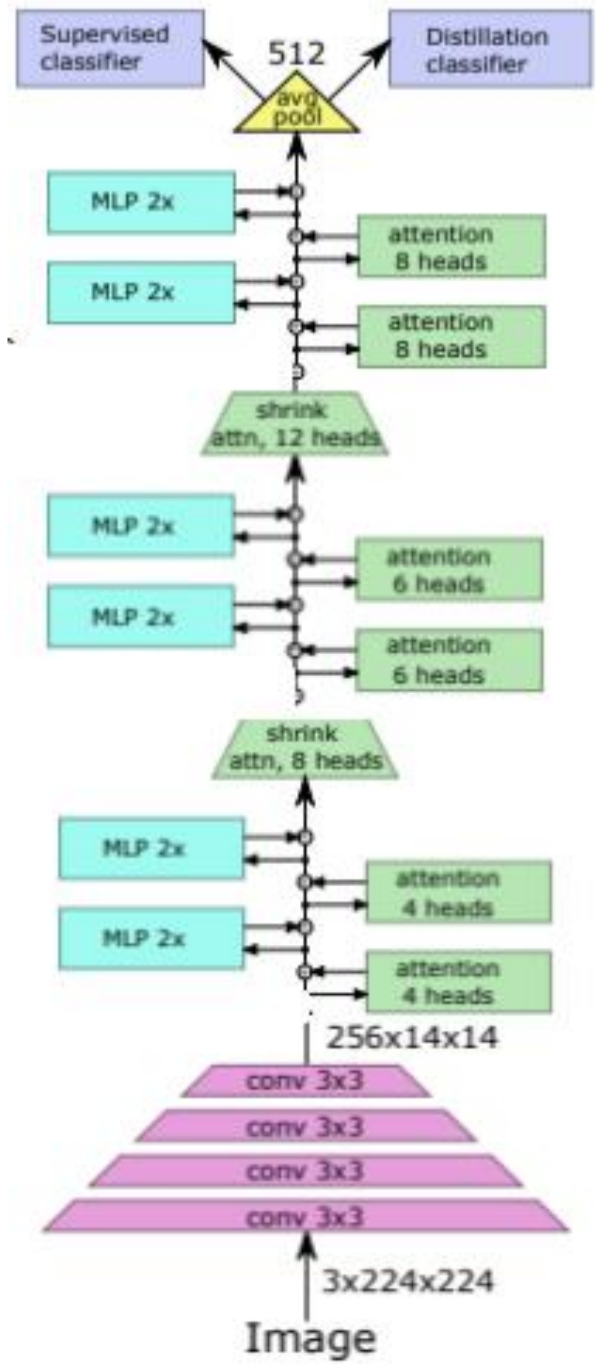
# 04 New Ideas

## 3. Reduce number of blocks




Original LeViT

Reduce




2 Blocks (per each stage)

| Num of blocks Stage1 + 2 + 3 | Result | | |
|---|---|---|---|
| | | Control Group(L_LeViT) | Control Group (LeViT) |
| Original LeViT 3 + 3 + 2 | Conv Block Param | - | 15,733,482 |
| | Top-1 Acc | 83.59% | 2.81% |
| | Inference Time | 22.52ms | 10.50ms |
| | | L_2Blocks | 2Blocks |
| Original LeViT 2 + 2 + 2 | Conv Block Param | 56,805,757 | 14,020,586 |
| | Top-1 Acc | **85.77%** | **13.78%** |
| | Inference Time | **20.92ms** | **8.74ms** |

When we **reduce # of blocks(3 → 2) per stage** both accuracy and inference speed increased.

APPROVED

# 04 New Ideas

## 4. Change Attention of Downsampling layer to Conv

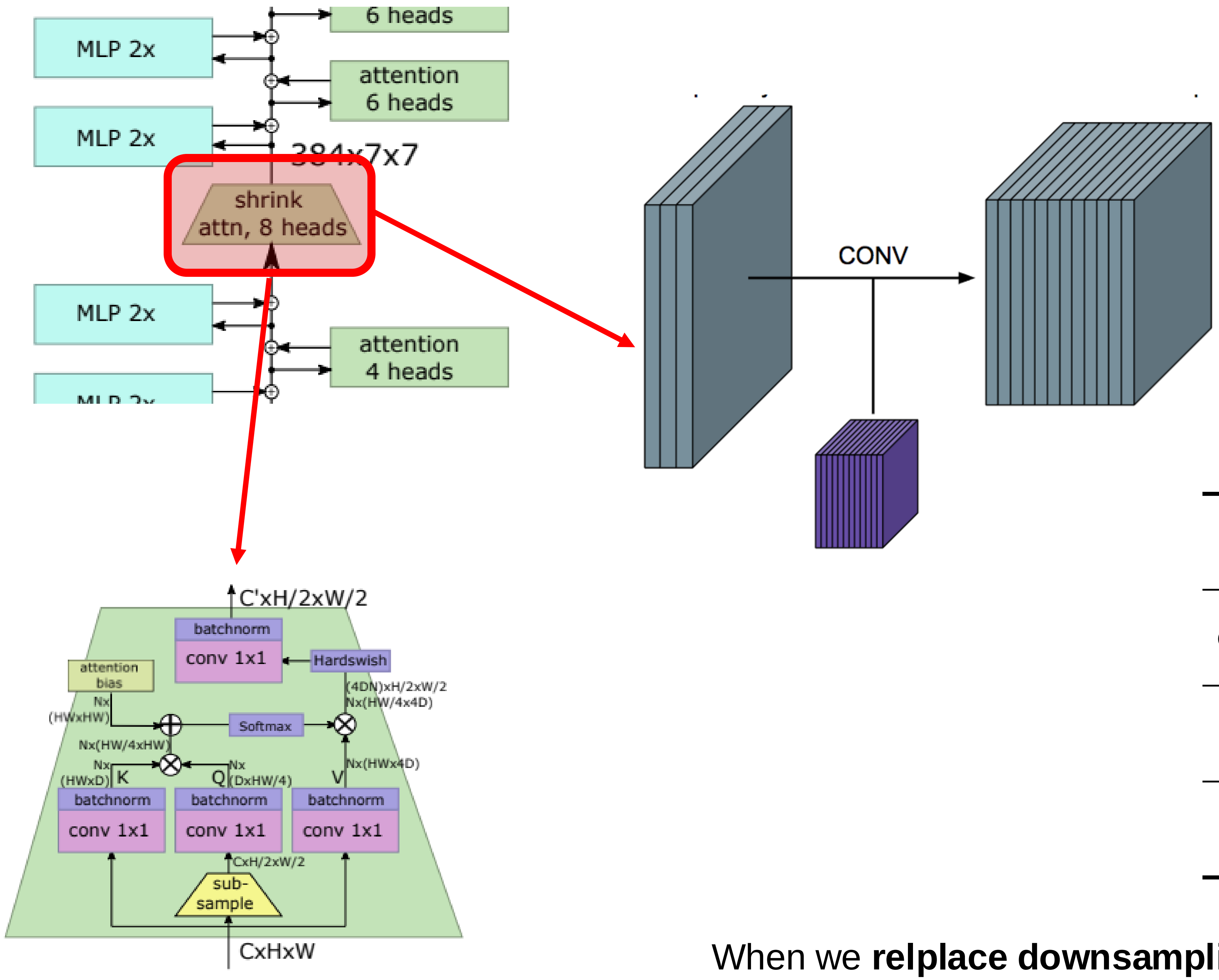
LeViT has a shrink attention block to downsample data.
The reason of downampling data is to connect stage and stage.

LeViT already has many attention calculation.
As reducing attention block showed better performance,

Higher attention computation cost = lower performance

So, we changed the Downsampling layer to Conv.

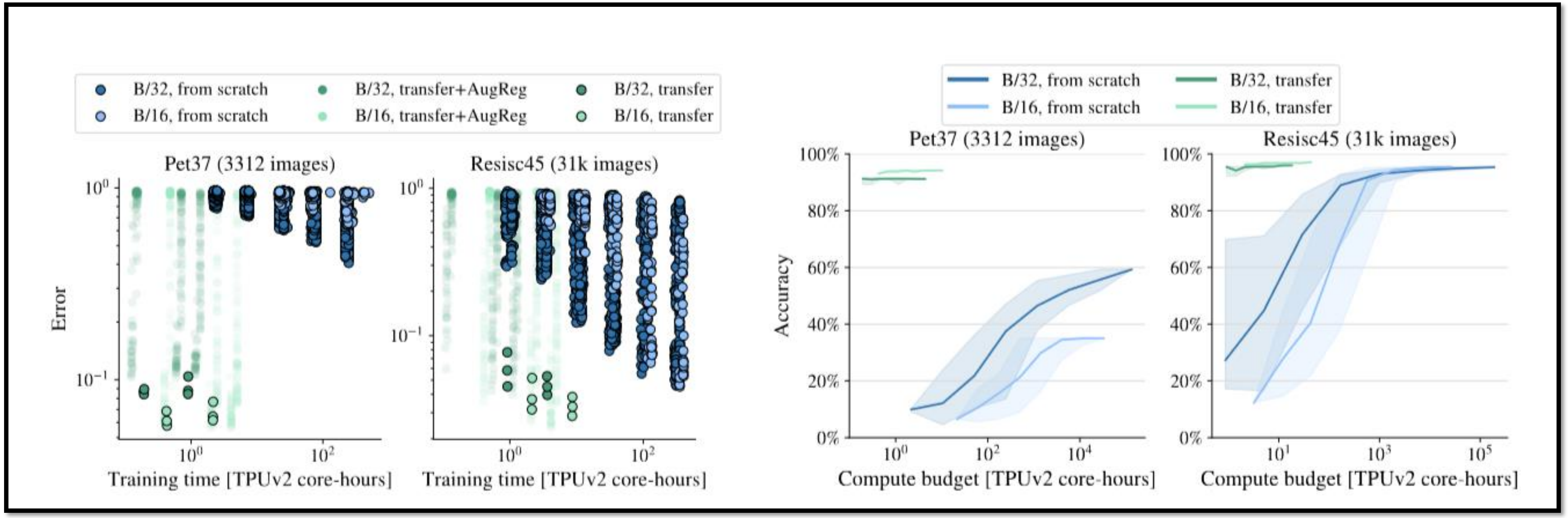
```
kernel_size=3, stride=2, padding=1
```




| | Control Group(L_LeViT) | Control Group(LeViT) | L_downsample with Conv | **Downsample with Conv** |
|---|---|---|---|---|
| Conv Block Param | - | 15,733,482 | 55,216,509 | 12,431,338 |
| Top-1 Acc | 83.59% | 2.81% | 86.22% | 22.76% |
| Inference Time | 22.52ms | 10.50ms | 19.27ms | **9.70ms** |

When we **relplace downsampling layer to Conv**
It showed the inference speed has increased and the accuracy has also increased

# Q: “Is it right to add pretrained models beforehand?”




*** How to train your ViT? Data, Augmentation, and Regularization on Vision Transformers – Steiner et al., IEEE TMLR 2022**

Previous research confirmed that **with small datasets, it is impossible to obtain *generic* models**.

Model trained *from scratch* with small datasets can’t reach accuracies
anywhere near the model compared with the larger datasets/transferred model.

Therefore, we added pretrained ResNet50 prior to our architecture input for improving accuracy.

# 05 Conclusion

## Why HoViT?

To ***improve inference time***
while maintaining and/or increasing performance of the vision transformer,

We made a new (ViT based) deep learning model.

HoViT → LeViT + Replacing some Attention block to Conv (1 X 1 Conv)

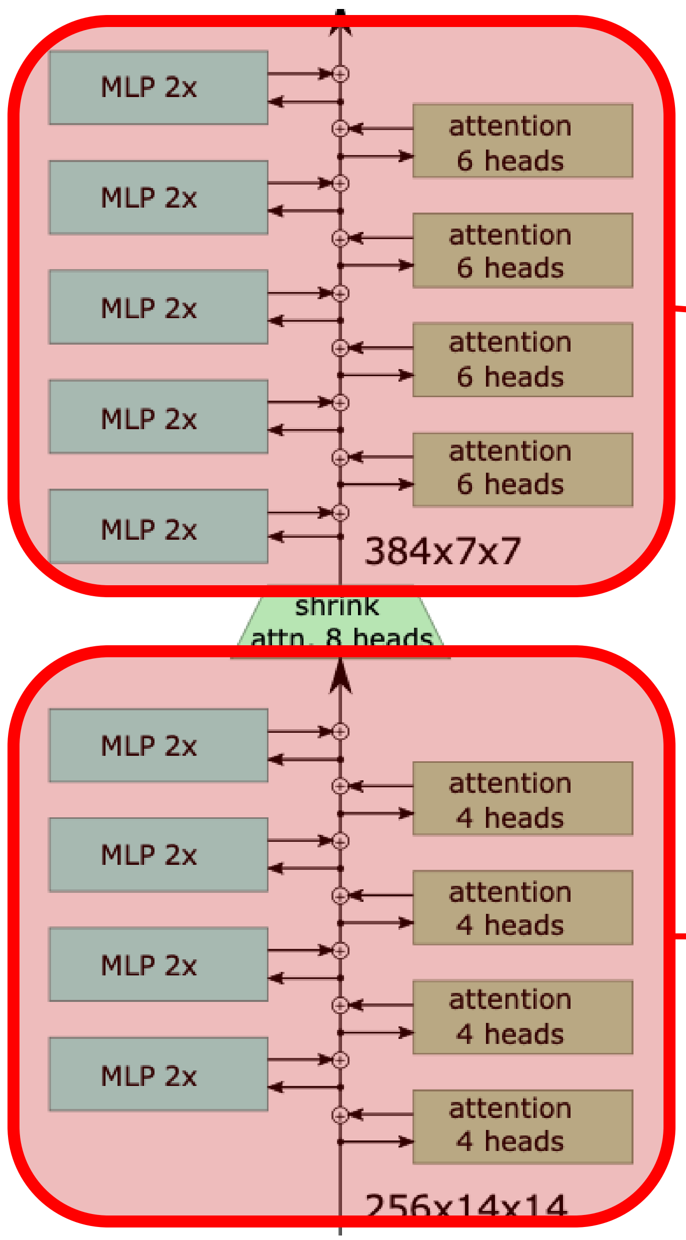
+ Reduce number of attention blocks (2 + 2 + 2)
+ Use Conv at Downsampling layer

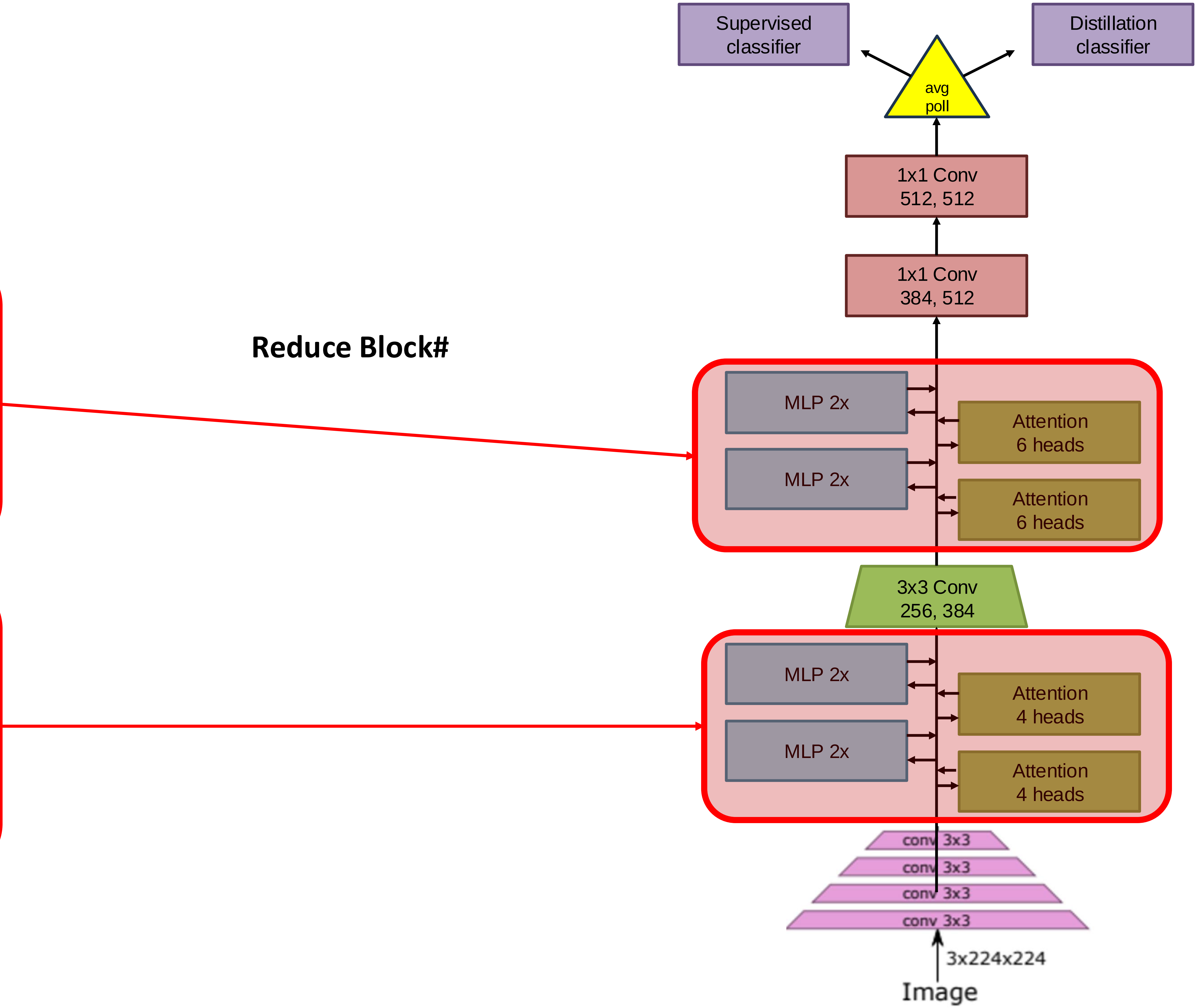
***H***ighly shrunk & ***o***ptimized ***Vi***sion ***T***ransformer – ***HoViT***

**Small but Fast..**

# 05 Conclusion

## HoViT Architecture

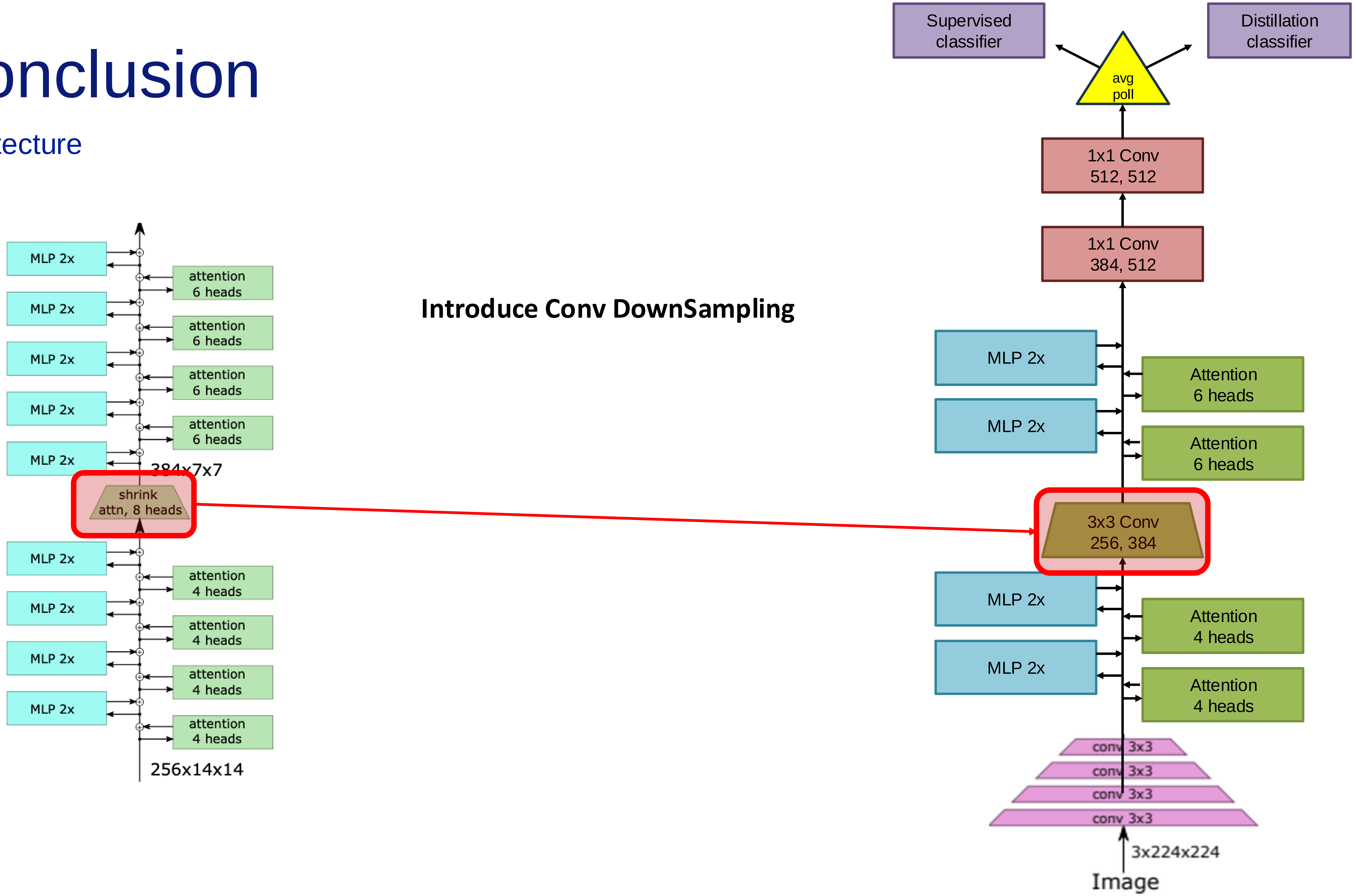


# 05 Conclusion

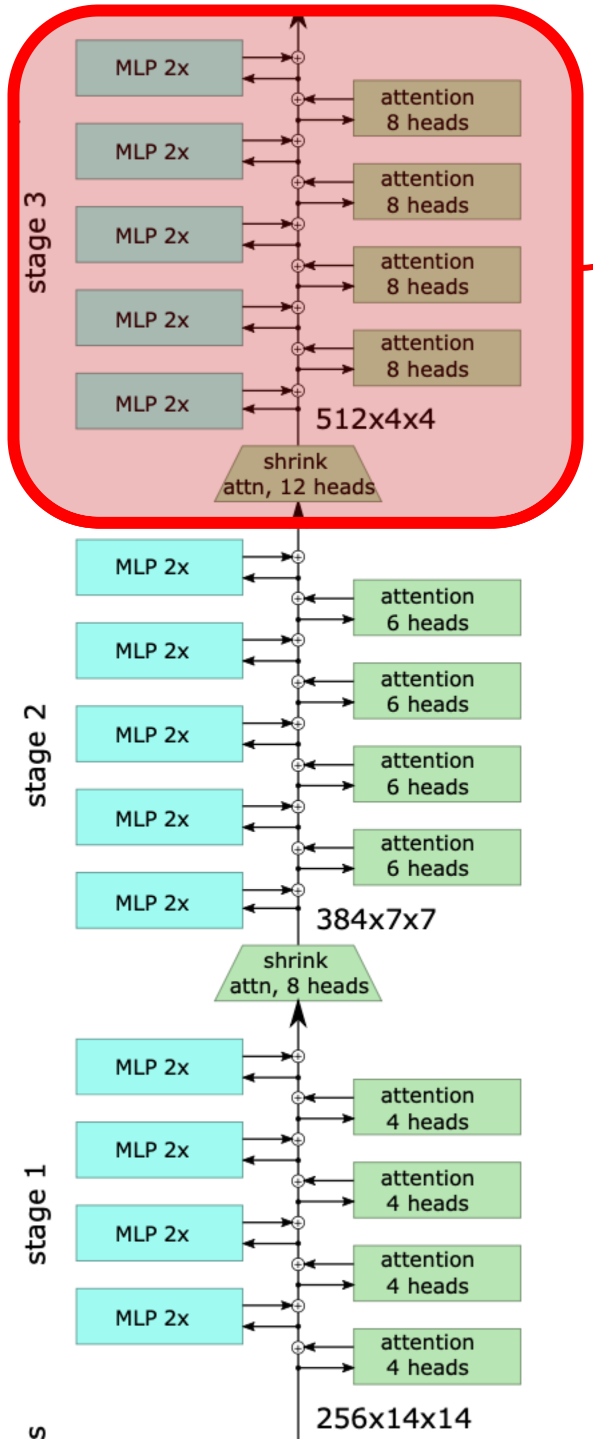
## HoViT Architecture

**Introduce Conv DownSampling**

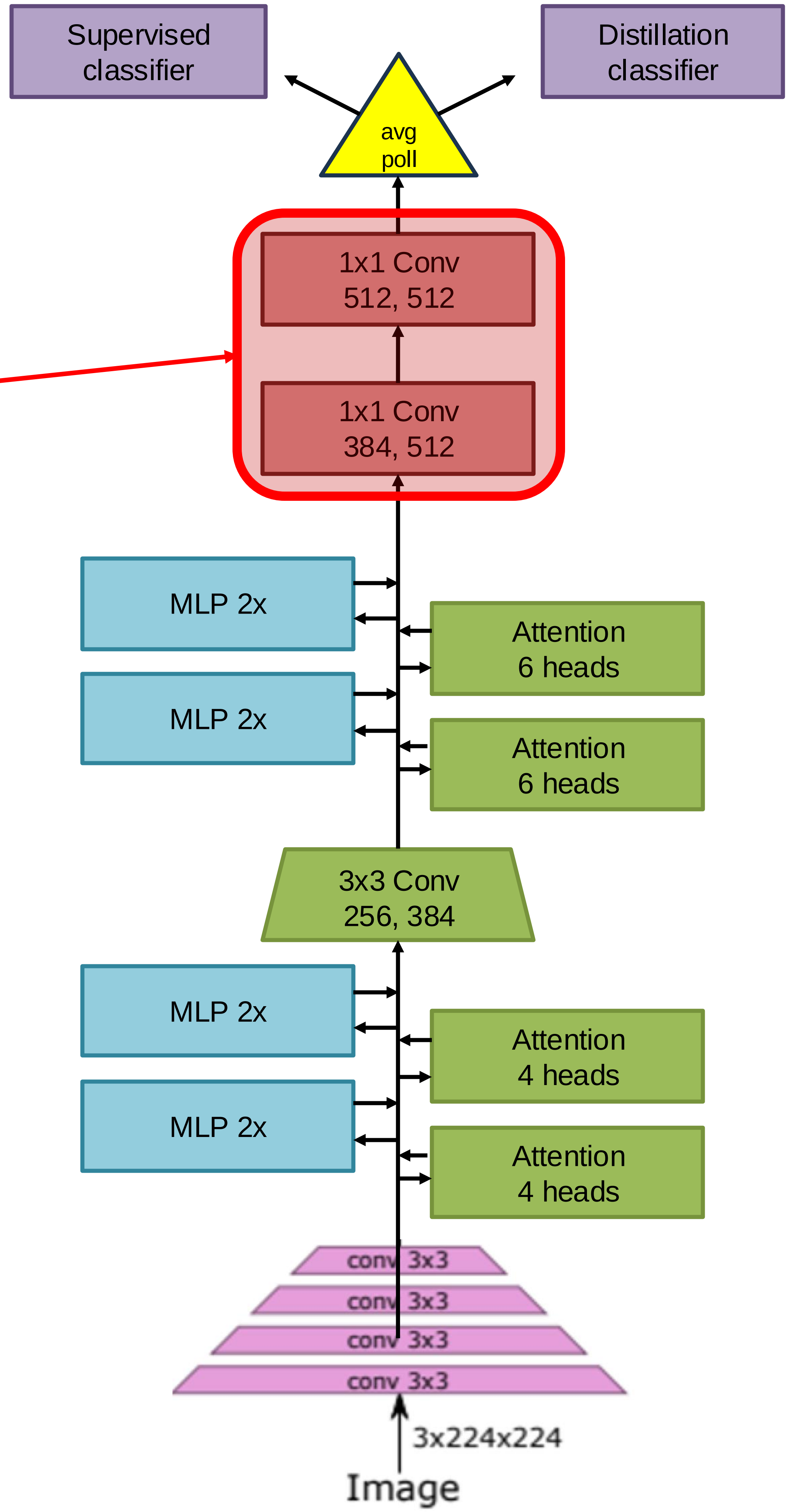


# 05 Conclusion

## HoViT Architecture

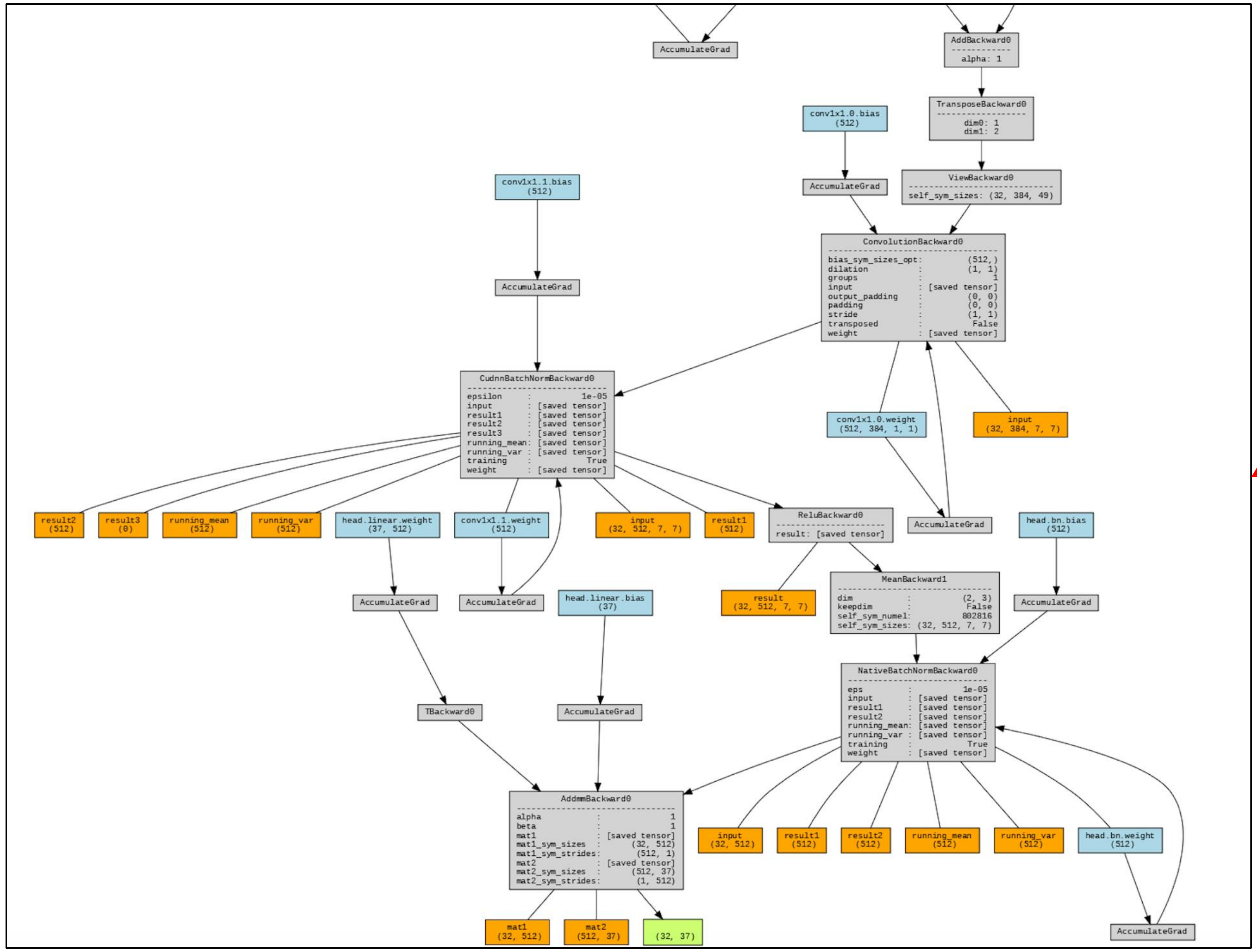


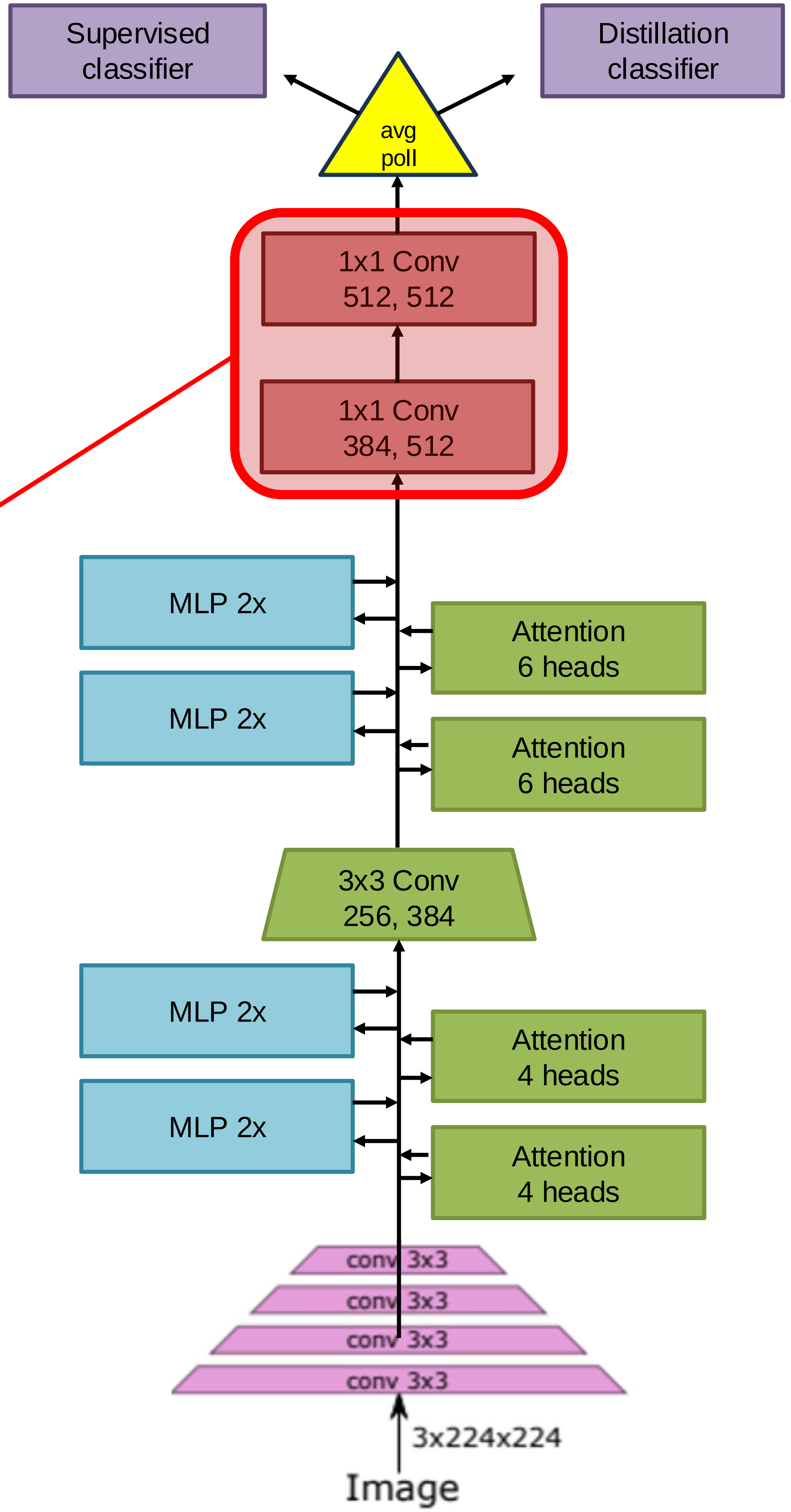

**Replace Shrink & Stage3 to 1x1 Conv**



# 05 Conclusion

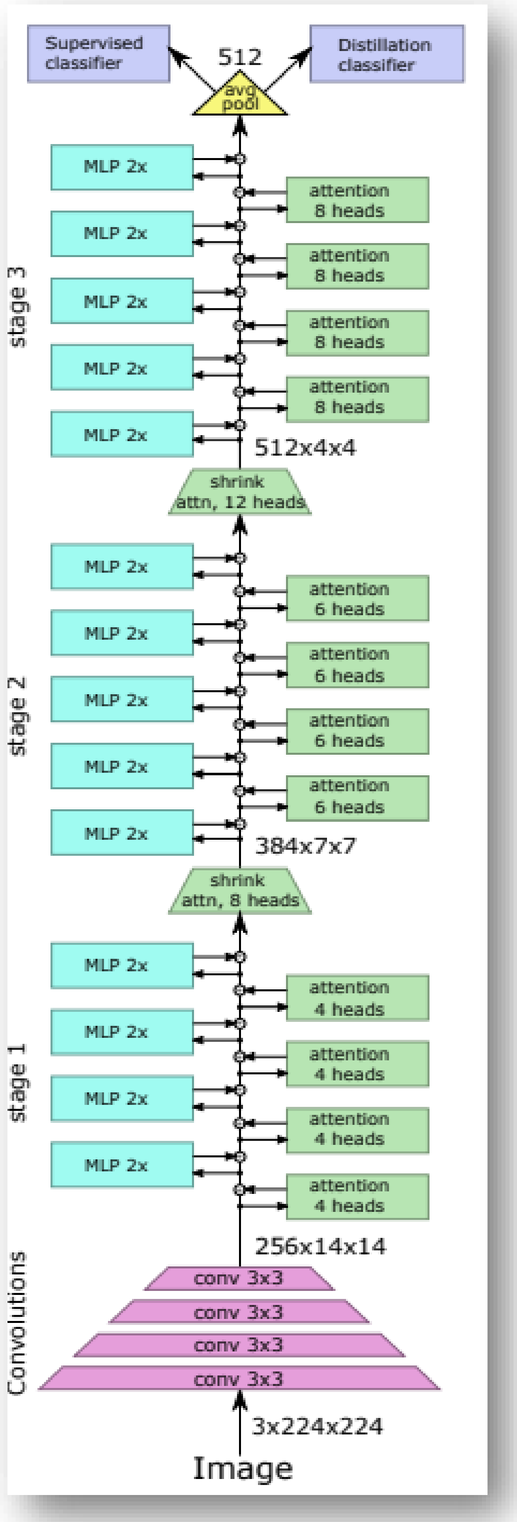
## HoViT Architecture

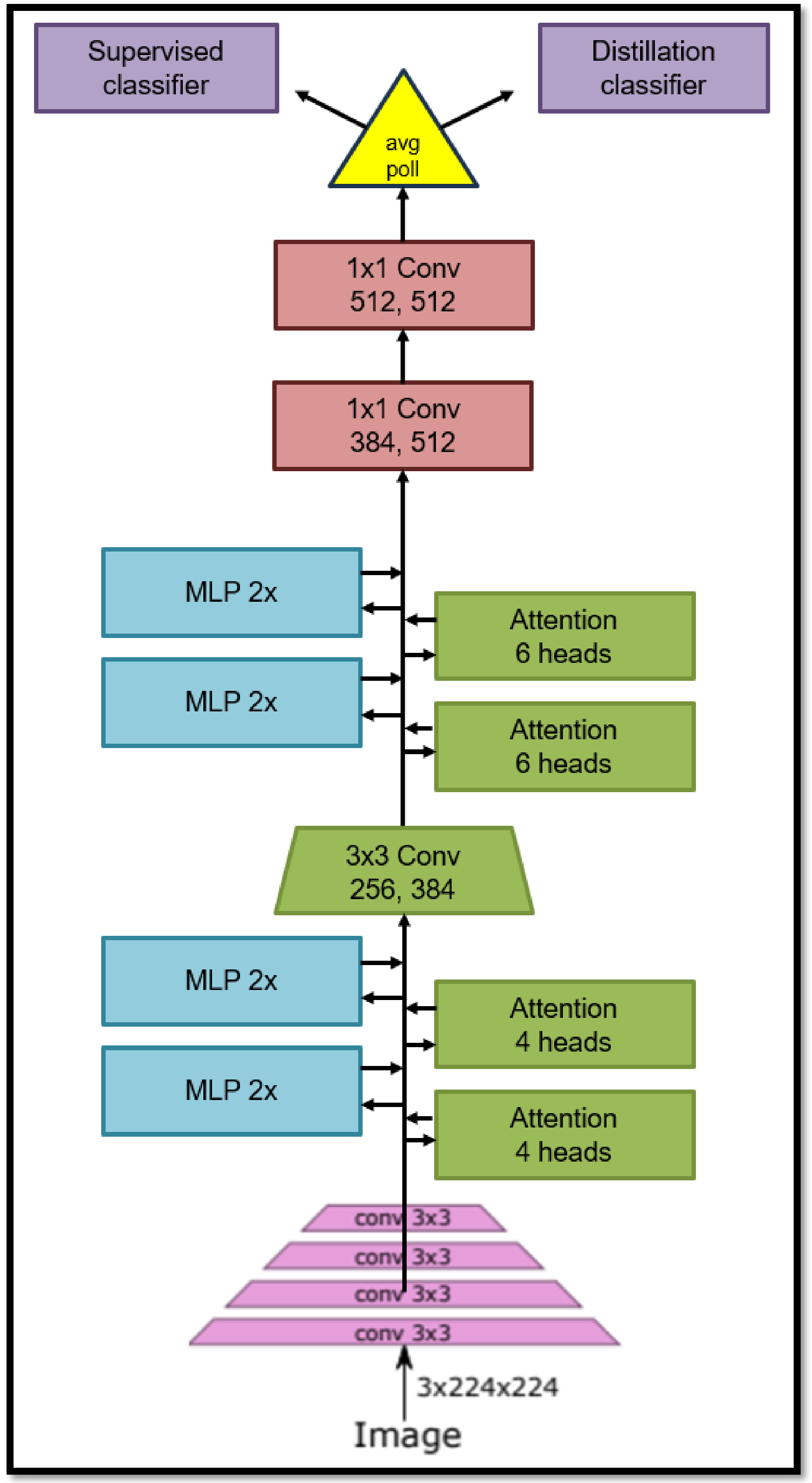




# 05 Conclusion

## HoViT architecture compared with LeViT



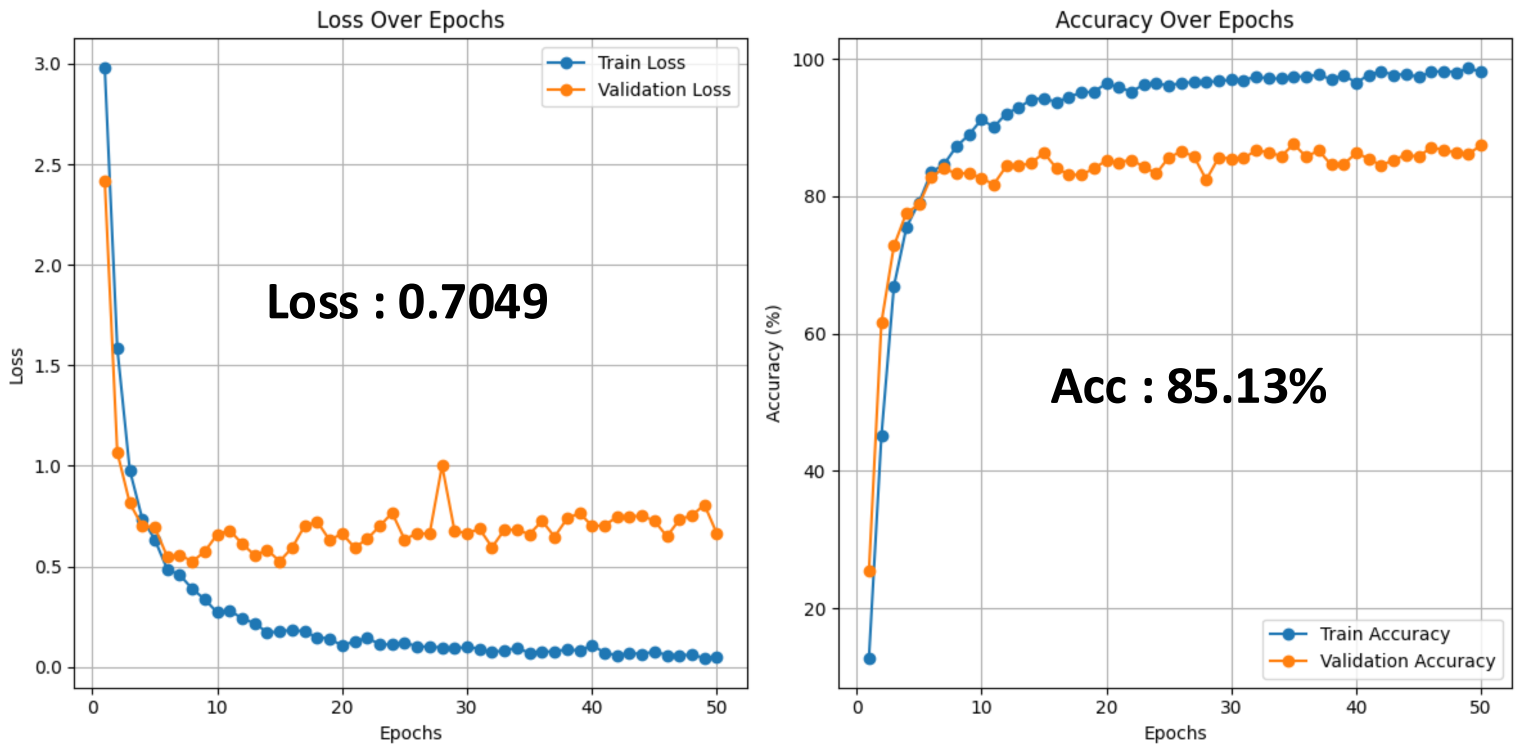



# 05 Conclusion

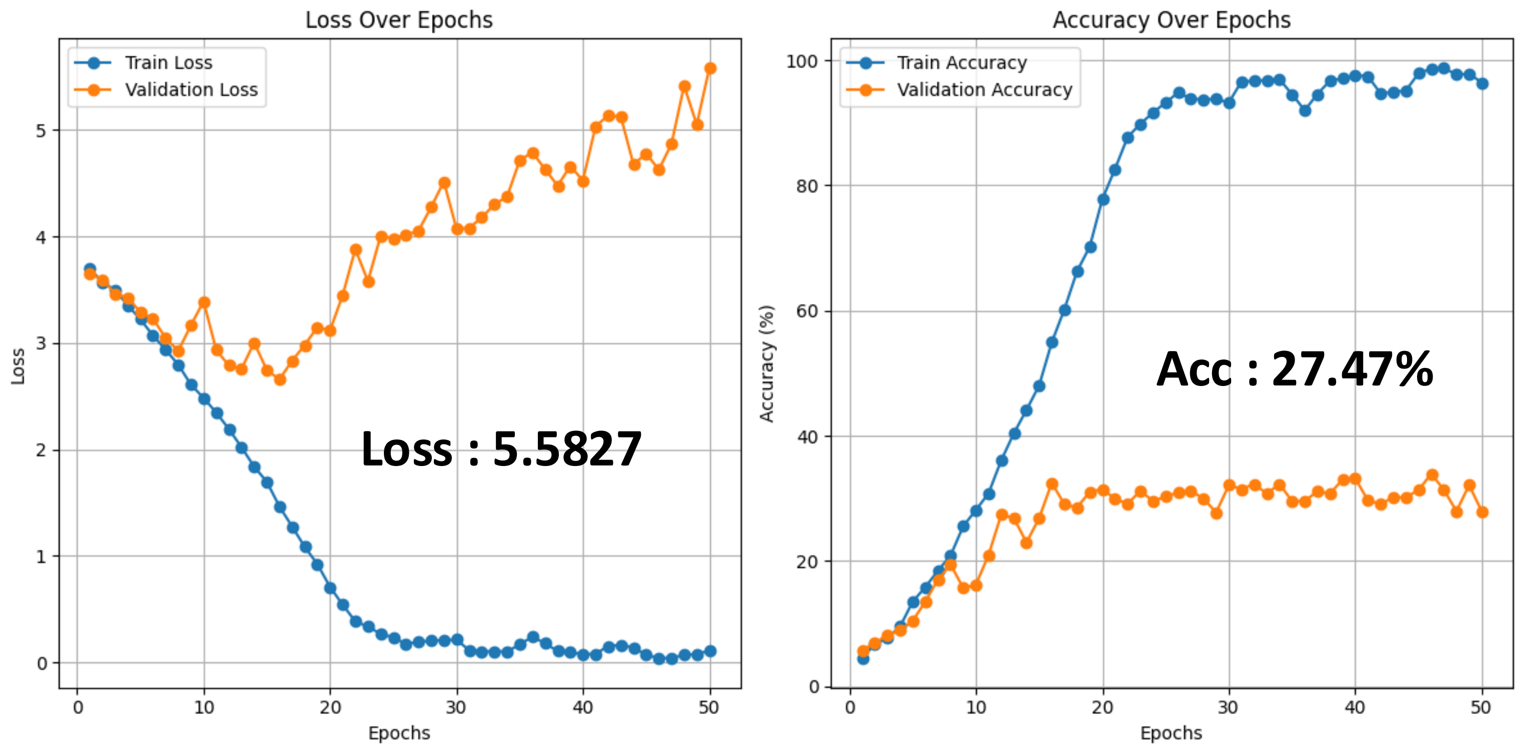
## HoViT result

Oxford-IIIT Pet with 50 epochs

### L_HoViT




### HoViT




| | Control Group(L_LeViT) | Control Group(LeViT) | L_HoViT | HoViT |
|---|---|---|---|---|
| Conv Block Param | - | 15,733,482 | 47,723,133 | 4,937,962 |
| Top-1 Acc | 83.59% | 2.81% | **85.13%** | 27,47% |
| Inference Time | 22.52ms | 10.50ms | **18.47ms** | **5.82ms** |

HoViT showed ***faster inference speed*** while maintaining performance

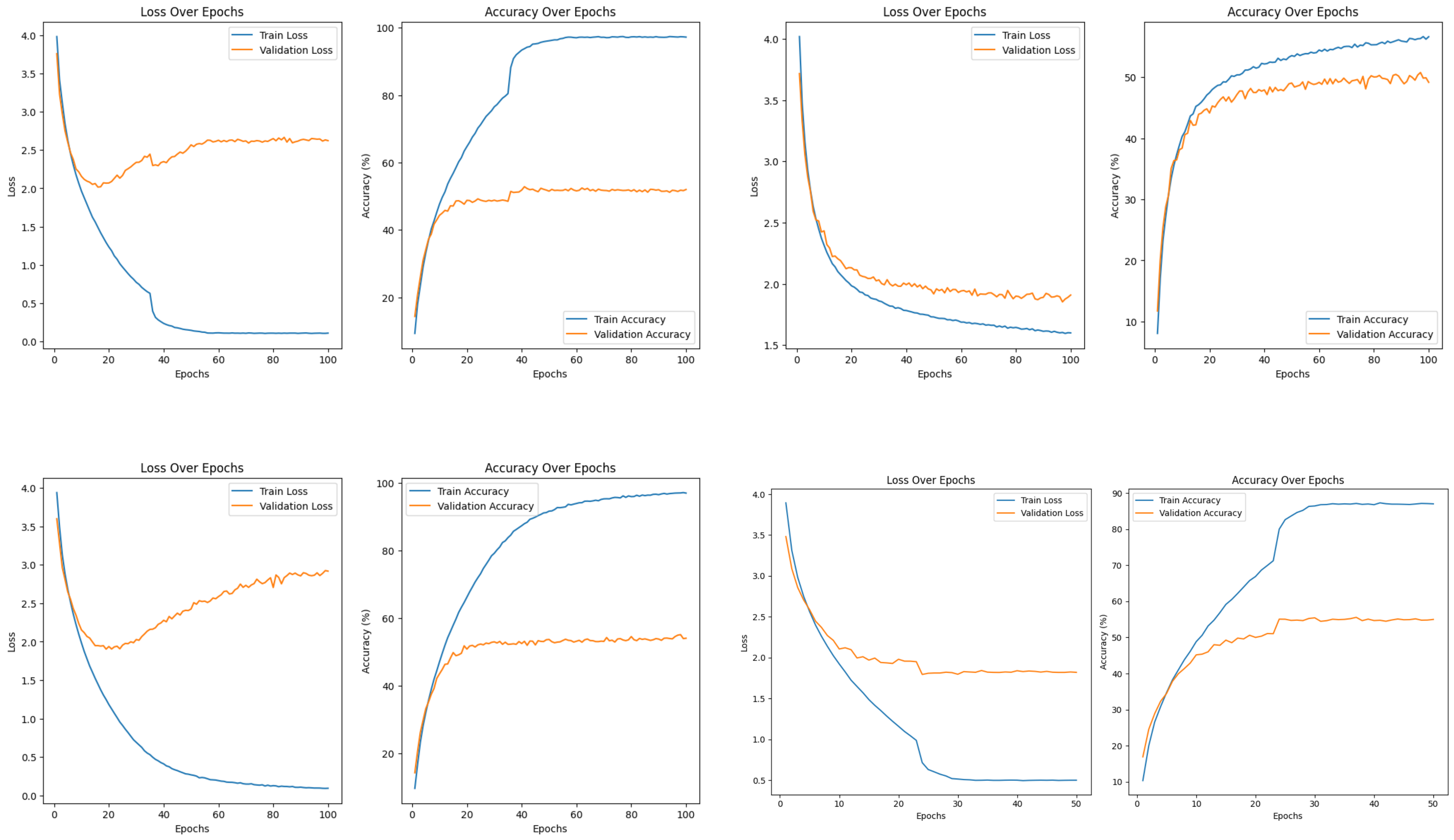
Single model inference time improved by **44.57%**

Parameters have been reduced by **68.62%**

# 05 Conclusion

## HoViT result

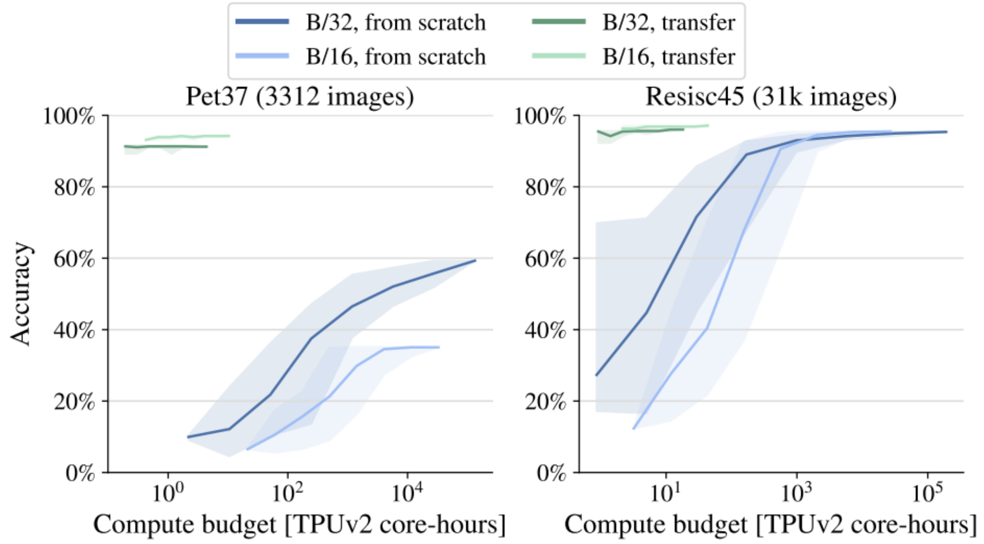
## Tried CIFAR-100 with 100 epochs




We tried CIFAR-100 dataset with 100 epochs
different learning rate, optimizer, scheduler, …

Accuracy stopped between **50% ~ 60%**

https://arxiv.org/abs/2106.10270


As shown in the paper mentioned above,
When you have a small dataset,
The ViT accuracy converges to around **50 or 60%**.

**Same symptoms with our proposed HoViT**

**So, we want to try with larger dataset (Like ImageNet)**

# 06 Future Work

## Discussion and Future Direction

**Jaccard Similarity**

Computing attention with Jaccard similarity showed stable training even on small datasets.

However, it comes with a trade-off in terms of complexity.

It is expected to be applicable to architectures designed for small datasets.

**GLA** (Gated Linear Attention)

For GLA, while it shows potential (As a method originally proposed for diffusion models),

further experiments on larger datasets are necessary.

**HoViT**

For HoViT, additional ablation studies are required to evaluate its effectiveness thoroughly.