Image Classification using Vision Transformers

Yenugula Hashish
EE20BTECH11056
ee20btech11056@iith.ac.in

Korapala Hrushikesh
EE20BTECH11022
ee20btech11022@iith.ac.in

Donthula Srinish
EE20BTECH11015
ee20btech11015@iith.ac.in

Ugranam Hema Chandar
EE20BTECH11062
ee20btech11062@iith.ac.in

Abstract

The Transformer architecture has established itself as the prevailing approach for addressing tasks in natural language processing (NLP). Nevertheless, its application in the field of computer vision has been fairly limited thus far. Within computer vision, attention mechanisms are often used in conjunction with convolutional neural networks (CNNs) or to replace certain components inside CNNs while maintaining the core CNN structure. Our research aims to disprove the widely held belief that CNNs are required for computer vision applications and that a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. This breakthrough is particularly evident when the Transformer model is pre-trained on extensive datasets and subsequently adapted to various mid-sized or smaller-scale image recognition benchmarks. Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

1. Problem Statement

In this project our objective is to implement Vision Transformers models and compare them with CNN models. We explored the possibility of using transformer-based models (ViT and Swin-T) for Image Classification. We have compared Vision Transformers and CNNs using benchmark datasets for image classification.

2. Introduction

In this report, we have described the architecture of Vision Transformers and Swin Transformers. We have implemented Vision Transformer models like ViT and Swin-T for image classification from scratch using Pytorch. We have trained the model on the CIFAR-10 dataset and compared it with ResNet50.

3. Dataset

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. The ten classes in the CIFAR-10 dataset are [airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck]. There are 50,000 training images and 10,000 test images.

4. Vision Transformer Architecture

Overall architecture is divided into 2 layers. Input Embedding and Encoder Layer. Brief description of each layer:

4.1. Input Embedding

In Transformers, input embedding is a fundamental component that helps in the conversion of discrete tokens into continuous vectors, semantic understanding, and positional information, reducing dimensionality, and allowing the model to operate with fixed-size inputs.

4.1.1 Patch Embeddings

The conventional Transformer model is designed to process input data in the form of a 1D sequence of token embedding. When working on 2D images, A different method is required to modify the transformer for image data. We reshape the image $x \epsilon R^{H \times W \times C}$ into a sequence of flattened 2D patches. Where, (H, W) is the resolution of the original image and (P, P) is the resolution of each image patch. We recast the image as a series of these flattened 2D patches to make it more fit for the Transformer. The Transformer's effective sequence length, denoted as N, is given by $N = \frac{HW}{P^2}$, an important parameter since it determines how the Transformer processes the image. In other words, the original image is divided into fixed-size chunks. For

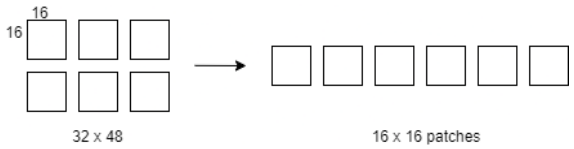

Figure 1. Reshaped Patch Embeddings

example, given below, we used a patch size of 16×16, indicating that the original image dimension will be 32×48 as illustrated in figure 1

4.1.2 Linear Projection of Flattened Patches

The authors of the paper [1] found a helpful preprocessing step while preparing the picture patches for input into the Transformer block. They used an approach that included a linear projection before putting these patches into the Transformer.

Here's how this works in more detail: Each image patch is extracted into a big vector. This vector is multiplied by the E matrix. This process produces what is known as $patched\ embeddings$. These patched embeddings are the data pieces that are eventually entered into the Transformer model. They are produced by multiplying each patch's vector by the embedding matrix. Along with this, positional embedding is included which provides information on the spatial positioning of each patch inside the image. A clear illustration is given in figure 2

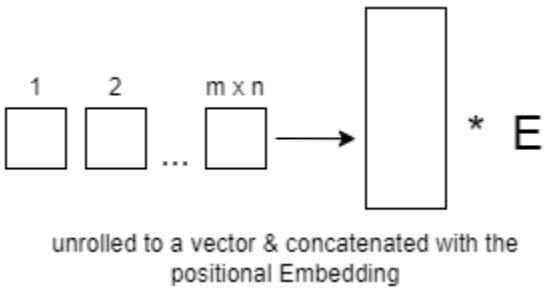

Figure 2. Linear Projection Block before feeding in Encoder

4.1.3 Positional Embeddings

Position embeddings are added to the patched embeddings to retain positional information. We explore different 2D-aware variants of position embeddings without any significant gains over standard 1D position embeddings. The joint embedding serves as input to the Transformer encoder. Each unrolled patch (before Linear Projection) has a sequence of numbers associated with it, in this paper the authors chose it to 1,2,3,4.... no of patches. These numbers are nothing but learnable vectors. Each vector is parameterized and stacked row-wise to form a learnable positional embedding table.

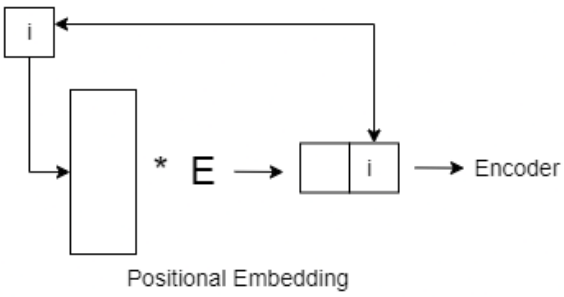

Figure 3. Positional Block to Encoder

Similar to BERT's [class] token, we prepend a learnable embedding to the sequence of embedded patches, whose state at the output of the Transformer encoder $({z_1}^0)$ serves as the image representation y. Both during pre-training and fine-tuning, the classification head is attached to ${z_1}^0$. Finally, the row number (initially sequenced number) associated with the patched embedding is picked up from the table (as positional embedding), concatenated, and fed to the Transformer encoder block.

4.2. Encoder Design

4.2.1 MSA

Multi-head Attention is a module for attention mechanisms that run through an attention mechanism several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension. Intuitively, multiple attention heads allow for attending to parts of the sequence differently (e.g. longer-term dependencies versus shorter-term dependencies).

The MSA block itself consists of a LayerNorm layer and the Multi-Head Attention Layer. The layer norm layer essentially normalizes our patch embeddings data across the dimension of the embedding. For a single image, each patch to get updated based on some similarity measure with the other patches. We do so by linearly mapping each patch to 3 distinct vectors: q, k, and v (query, key, value). Then, for a single patch, we are going to compute the dot product between its q vector with all of the k vectors, divide by the square root of the dimensionality of these vectors, softmax these so-called attention cues, and finally multiply each attention cue with the v vectors associated with the different k vectors and sum all up. In this way, each patch assumes a new value that is based on its similarity (after the linear mapping to q, k, and v) with other patches. This whole procedure, however, is carried out H times on H sub-vectors of our current patches, where H is the number of Heads. So,

the input shape to the MSA Block will be the shape of our patch embeddings. The output from the MSA layer will be of the same shape as the input.

4.2.2 MLP

The Machine Learning Perceptron (MLP) Block in the transformer is a combination of a Fully Connected Layer (also called a Linear Layer or a Dense Layer) and a non-linear layer. In the case of the Vision Transformer (ViT), this non-linear layer takes the form of a GeLU layer and Layer Norm (LN): This is added prior to each block as it does not include any new dependencies between the training images. This thereby helps improve the training time and overall performance. The transformer also includes a Dropout layer to improve model generalization and reduce overfitting. As a result, the MLP Block's makeup can be summarised as follows:
Input → Linear → GeLu → Dropout → Linear → Dropout

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{1}$$
$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \ldots L \tag{2}$$
$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \ldots L \tag{3}$$
$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

Figure 4. Two layers with GeLU

4.2.3 Why chose GeLU over other activation maps?

The GeLU consistently outperformed ELU and ReLU in terms of accuracy across the multiple datasets evaluated in [2], proving itself as a convincing alternative to earlier non-linear activation functions.

Authors defined Gaussian Error Linear Unit (GELU) as:

$$GeLU(x) = xP(X \leq x) = x\Phi(x) \tag{1}$$

We can approximate the GeLU with

$$0.5x(1 + \tanh\sqrt{2/\pi}(x + 0.044715x^3)) \tag{2}$$

or

$$x\sigma(1.702x) \tag{3}$$

Figure 5 shows the functions GeLU, ReLU, ELU.

From [2], GeLU nonlinearity is a smooth function with a unique characteristic: it's non-monotonic and curved at all points. This property makes it better at approximating complex functions than ReLU and ELU activations. It's nonlinearity is also related to a probabilistic interpretation, which is similar to the predicted transformation of a stochastic regularizer. This characteristic can help to reduce overfitting and improve generalization. Furthermore, unlike ReLUs, GELU weights inputs based on their magnitude rather than

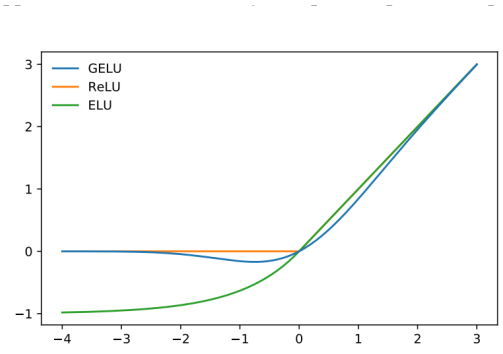

Figure 5. Positional Block to Encoder

their sign. This modification alleviates the vanishing gradient problem and improves gradient flow during backpropagation.

The authors in [2] evaluated the performance of different activation functions on the CIFAR-10 and CIFAR-100 datasets using two different CNN architectures: A shallow 9-layer CNN from [4] on CIFAR-10 and a deeper 40-layer CNN(WideResNet) [3] on CIFAR-100. Results are shown in Figure 7, and each curve is a median of three runs. Learning curves show training set error rates, and the lighter curves show the test set error rates.

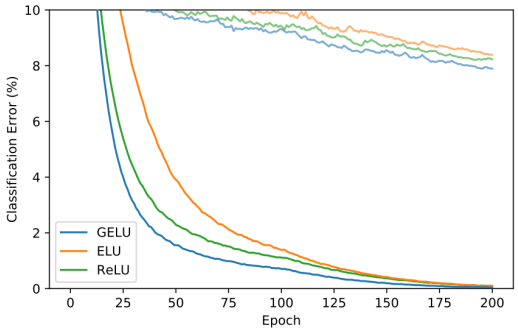

Figure 7. Positional Block to Encoder

Overall, the usage of GELU nonlinearity may provide advantages over other activation functions in terms of function approximation, regularization, and gradient flow, which may lead to improved training and convergence of deep neural networks

5. Swin Transformers

As discussed above, challenges in adapting Transformer from language to vision arise from differences between the two domains, such as large variations in the scale of visual entities and the high resolution of pixels in images com-

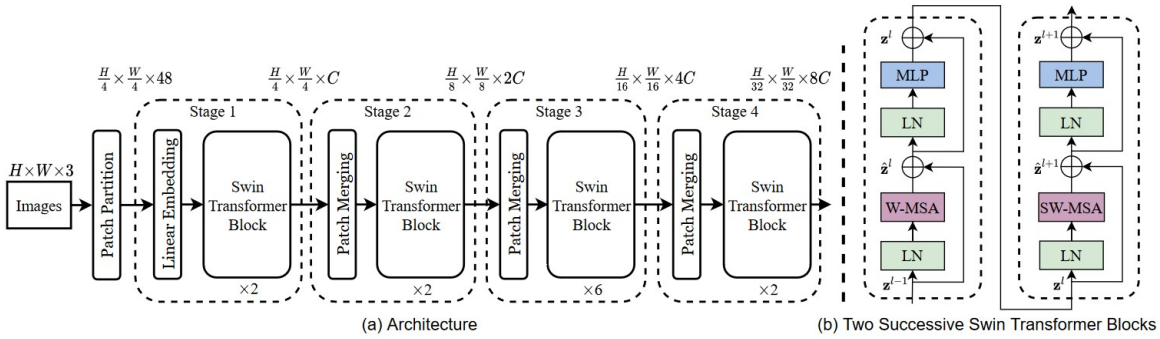

Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

Figure 6. Swin-T Architechture

pared to words in text. To address these differences, Swin Transformers use hierarchical Transformer whose representation is computed with Shifted windows. [5]

5.1. Patch Merging

Several Transformer blocks with modified self-attention computation (Swin Transformer blocks) are applied to these patch tokens. The Transformer blocks maintain the number of tokens H/4 × W/4, and together with the linear embedding are referred to as "Stage 1". To produce a hierarchical representation, the number of tokens is reduced by patch merging layers as the network gets deeper. The first block of patch merging and feature transformation is denoted as "Stage 2". The procedure is repeated twice, as "Stage 3" and "Stage 4".The first patch merging layer concatenates the features of each group of 2×2 neighboring patches, and applies a linear layer on the 4C - dimensional concatenated features. This reduces the number of tokens by a multiple of 2×2 = 4 (2× downsampling of resolution), and the output dimension is set to 2C.

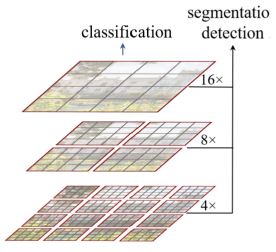

Figure 8. Patch Merging

5.2. Shifted Window

The window-based self-attention module lacks connections across windows, which limits its modeling power. To introduce cross-window connections while maintaining the efficient computation of non-overlapping windows, we propose a shifted window partitioning approach which alternates between two partitioning configurations in consecutive Swin Transformer blocks.

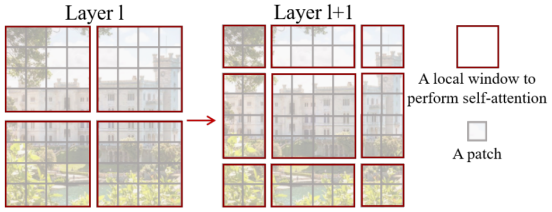

Figure 9. Shifting of Window

5.3. Overall Architechture

Swin Transformer constructs a hierarchical representation by starting from small-sized patches and gradually merging neighboring patches in deeper Transformer layers. With these hierarchical feature maps, the Swin Transformer model can conveniently leverage advanced techniques for dense prediction such as feature pyramid networks (FPN) or U-Net. A key design element of Swin Transformer is its shift of the window partition between consecutive self-attention layers. The shifted windows bridge the windows of the preceding layer, providing connections among them that significantly enhance modeling power.

Swin Transformer is built by replacing the standard multi-head self attention (MSA) module in a Transformer block by a module based on shifted windows, with other layers kept the same. A Swin Transformer block consists

of a shifted window based MSA module, followed by a 2-layer MLP with GELU nonlinearity in between. A LayerNorm (LN) layer is applied before each MSA module and each MLP, and a residual connection is applied after each module.

6. Results

We have trained CIFAR-10 Dataset on ViT, Swin Transformers and ResNet50. Below are the model summaries and results for each model. We have used Cross Entropy loss as metric.

$Loss = H(p,q) = -\sum p(x)\log(q(x))$

6.1. ViT

Below is the summary for Vision Transformer:

```
==========================================================================
Total params: 9,489,930
Trainable params: 9,489,930
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 26.71
Params size (MB): 36.20
Estimated Total Size (MB): 62.92
--------------------------------------------------------------------------
```

Figure 10. Vision Transformer summary

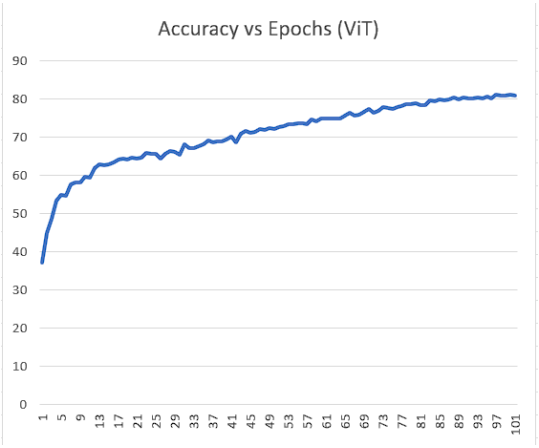

Figure 11. Vision Transformer Validation loss vs Epochs

Training Configurations:

- Learning Rate: 1e-3.
- Optimizer: Adam Optimizer.
- Batch Size: 1000.
- Image Size: 32.
- Number of Epochs: 200.
- Patch Size: 4.
- Dimension Head: 512.

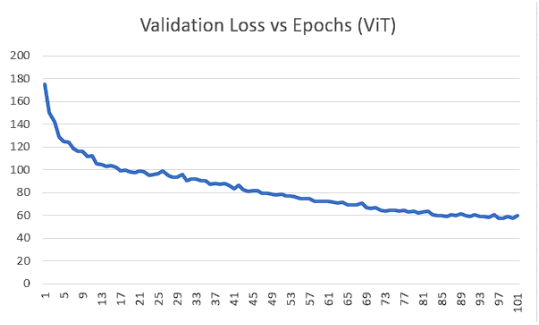

Figure 12. Vision Transformer Accuracy loss vs Epochs

We ran the model for 100 epochs and got best accuracy of 81.14%.

6.2. ResNet-50

Below is the model summary that we used:

```
==========================================================================
Total params: 23,520,842
Trainable params: 23,520,842
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 66.13
Params size (MB): 89.72
Estimated Total Size (MB): 155.86
--------------------------------------------------------------------------
```

Figure 13. ResNet50 model summary

ResNet50 Model Configuration:

- Architecture: ResNet50, a deep residual network with 50 layers. Composed of multiple residual blocks.
- Block Type: Basic Block: Consists of two 3x3 convolutions with batch normalization and ReLU activation.
- Skip Connections (Identity Shortcut): Identity shortcuts are used to skip one or more layers, aiding in the flow of gradients during training.
- Expansion Factor: Bottleneck blocks are used with an expansion factor of 4 (common for ResNet50).
- Global Average Pooling (GAP): Applied after the last residual block. Reduces the spatial dimensions to 1x1.
- Fully Connected (Linear) Layer: Takes the output of GAP and maps it to the number of classes (10 in the case of CIFAR-10).
- Batch Normalization: Applied before each convolution operation.

- Activation Function: Rectified Linear Unit (ReLU) is used as the activation function.
- ResNet50 Model Parameters: Number of Parameters: There are approximately 23.6 million parameters.

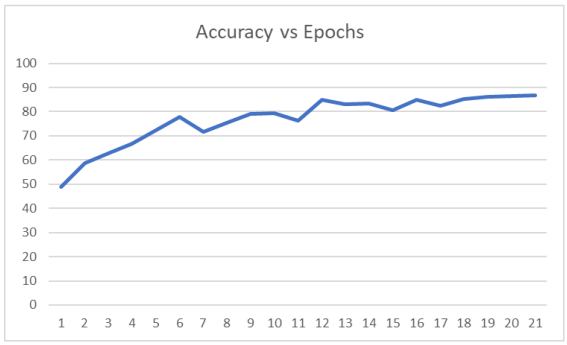

Figure 14. ResNet50 model accuracy

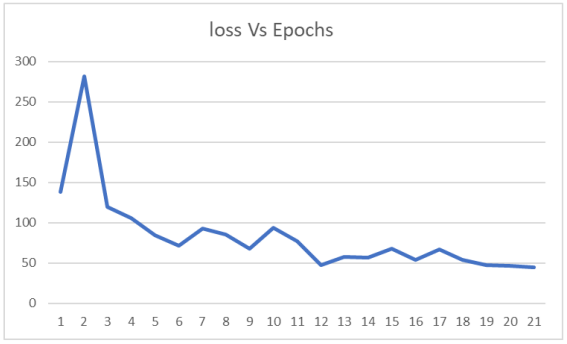

Figure 15. ResNet50 model loss

- Training Configuration:
  - Learning Rate: 1e-3
  - Optimizer: Adam optimizer is used.
  - Loss Function: Cross-entropy loss is used.
  - Data Preprocessing: Random cropping, resizing, and normalization for training. Resizing and normalization for testing.
  - Batch Size: 512
  - Image Size: 32x32 (CIFAR-10 size)
  - Number of Epochs: 200

For epoch 20, we got an accuracy of 86.81 % (which can get upto 91%).

6.3. SWIN-T Transformer

Below is the summary of swin t model

- Learning rate : 1e-3

```
        Linear-182          [-1, 4, 4, 2304]       1,769,472
        Linear-183           [-1, 4, 4, 768]         590,592
    CyclicShift-184           [-1, 4, 4, 768]               0
WindowAttention-185           [-1, 4, 4, 768]               0
       PreNorm-186           [-1, 4, 4, 768]               0
      Residual-187           [-1, 4, 4, 768]               0
     LayerNorm-188           [-1, 4, 4, 768]           1,536
        Linear-189          [-1, 4, 4, 3072]       2,362,368
          GELU-190          [-1, 4, 4, 3072]               0
        Linear-191           [-1, 4, 4, 768]       2,360,064
    FeedForward-192           [-1, 4, 4, 768]               0
       PreNorm-193           [-1, 4, 4, 768]               0
      Residual-194           [-1, 4, 4, 768]               0
     SwinBlock-195           [-1, 4, 4, 768]               0
    StageModule-196           [-1, 768, 4, 4]               0
     LayerNorm-197                 [-1, 768]           1,536
        Linear-198                  [-1, 10]           7,690
================================================================
Total params: 26,598,058
Trainable params: 26,598,058
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 25.34
Params size (MB): 101.46
Estimated Total Size (MB): 126.82
----------------------------------------------------------------
```

Figure 16. SWIN-T Transformer summary

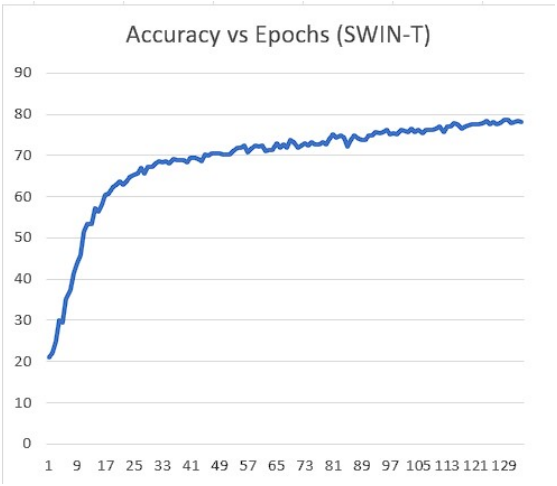

Figure 17. Swin-T Accuracy vs Epochs

- Optimizer : Adam
- Batch size : 1000
- Image size : 32x32
- Number of epochs : 200
- Patch size : 4
- Dimension of the attention head : 512

We ran the model for 131 epochs and got best accuracy of 78.61%.

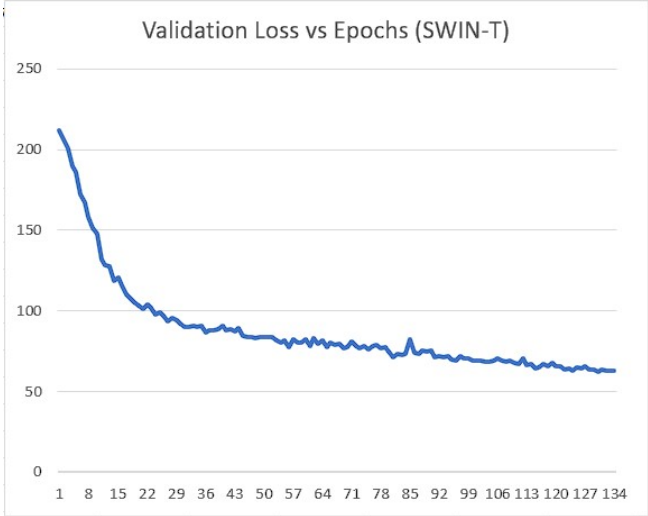

Figure 18. Swin-T Validation Loss vs Epochs

6.4. Observations and Conclusions

1. Best accuracy for ViT model is 81.14%, and the number of parmeters is 9.5 Million nearly.
2. Best accuracy for SWIN-T model is 78.61%, and the number of parmeters is 26 Million parameters.
3. Best accuracy for ResNet-50 model is 86.81%, and the number of parmeters is 23.5 Million nearly.
4. We also observed that ResNet was learning much faster and giving better results than both the ViT and Swin-T model.
5. When we further decreased the patch size to 2*2, the model was overfitting was at early epochs.
6. When we increased the patch size to 8*8 or 16*16 the model was not learning anything or the change in training is neglible.
7. We can safely conclude that we can vary the patch size to avoid high variance or high bias cases. Decreasing the patch size increases the number of features for the Transformer which increases the variance of the model.
8. Vision Transformers are largely dependent on the size of the dataset unlike CNN's. We observed that when we reduced the size of the dataset from 50,000 to 10,000, performance of ViT was much lower compared to ResNet50. When ViT are provided with big datasets, they are likely to outperform CNN's.

We have used the following references for our better understanding 6, 7, 8.

7. Individual Contributions

- Korapala Hrushikesh and Donthula Srinish have implemented 2 different scripts for Swin-T from scratch.
- Ugranam Hema Chandar and Yengula Hashish have implemented 2 different scripts for ViT from scratch.
- We have chosen best of ViT and Swin-T models for this report.

References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, andXiaohua Zhai Dirk Weissenborn, andMatthias Minderer Thomas Unterthiner, andMostafa Dehghani, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR 2021*. 2

[2] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415v3*. 3

[3] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*. 3

[4] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*. 3

[5] Ze Liu, Yutong Lin, and Yue Cao. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv:2103.14030v2*. 4