ALL WE DO IS SWIN: EVALUATION OF VISION TRANSFORMERS AND ITS VARIANTS ON IMAGE CLASSIFICATION TASKS

JEFFREY LI [LIJEFF@SEAS.UPENN.EDU],
DONTR LOKITIYAKUL [DANTEHL@SEAS.UPENN.EDU],
YASH NAKADI [YNAKADI@SEAS.UPENN.EDU],
THITI PREMRUDEEPREECHACHARN [TPREMRUD@SEAS.UPENN.EDU]

ABSTRACT. There has been a recent interest in applying Transformers to tasks in computer vision, such as image segmentation and image classification. In particular, Vision Transformers (ViT) and Hierarchical Vision Transformers via Shifted Windows were used to compare the performance of Convolutional Neural Networks (CNNs) in the image classification task on the CIFAR-100 dataset. With regularization, data augmentation, constructing hybrid models, and tuning hyperparameters, we found that the Swin architecture with increased number of Swin blocks outperforms all model and experimental configurations at 77.1% test accuracy. This can be compared to the best performance of the CNN model (67%) and the ViT (54.5%). Given its performance and hierarchical structure, we also explored its potential as a basis unit for encoder and decoder in generative modeling, notably Variational Autoencoder, in the Appendix.

1. INTRODUCTION

Convolutional neural networks (CNN) remain the dominant architecture for computer vision problems. The recent success of self-attention-based architectures for natural language processing (NLP) inspired the design and implementation of the Vision Transformers (ViT) for image classification tasks [1, 2]. In the pioneering work, Dosovitskiy et al. (2020) split images into non-overlapping patches and input them into a Transformer architecture for image classification tasks, achieving comparable or even superior performance to traditional architectures like ResNet [1, 3]. Transformers lack some of the inductive biases inherent in CNNs; however, the group overcame these shortcomings through training their models on larger datasets (14M to 300M images), notably JFT-300M and ImageNet21k [1]. Liu et al. (2021) introduced a hierarchical approach to image processing for Transformer-based architectures through shifted windows mechanism [4]. The Swin Transformer architecture increases efficiency by restricting self-attention computations to non-overlapping local windows while enabling cross-window attention, thus effectively capturing local and global information [4].

For this project, we extract and implement design choices to improve Transformer-based architectures on classification problems with smaller datasets, namely CIFAR-100 [5]. We also include a standard CNN architecture as a baseline for our model performances.

1.1. Contributions.

By validating the performance of three different architectures on image classification task on CIFAR-100 dataset with modifications, we found that the hierarchical vision transformers or Swin outperforms CNN, and CNN also outperforms the vision transformers or ViT with accuracies of 77.1 %, 67 %, and 54.5 %, for Swin, CNN, and ViT, respectively.

2. BACKGROUND AND RELATED WORKS

Transformers were first proposed by Vaswani et al. (2017) in *Attention Is All You Need* and have since become a widely popular architecture in natural language processing (NLP) problems [6]. A simple transformer model has a stack containing multiple encoder layers and a stack containing multiple decoder layers [7]. The encoder stack enables the model to capture the contextual relationships between words and tokens in the input text, and the decoder stack takes the output of the final encoder layer and uses it to generate a sequence of tokens [7]. Each encoder and decoder stack contains a layer for embedding which converts input text into a vector and positional encoding which provides information about the position of each token or word in an input sequence [7].

Both the Encoder stack and the Decoder stack possess the Self-Attention Layer whereas the Decoder stack contains the Encoder-Decoder Attention layer [7]. The Self-Attention layer enables the model to focus on other words in the input that are closely related to that word [7]. The Encoder-Decoder Attention layer allows the decoder to selectively attend to different parts of the encoder output when generating the output sequence.

Dosovitskiy et al. (2020) closely followed the implementation of the original Transformer during model design [1, 6]. For the Vision Transformer (ViT) input, 2D images are divided into a grid of square patches. Each patch is then flattened into a single vector by concatenating the channels of all pixels in a patch and then linearly projecting it to the desired input dimension. Positional embeddings are added to each patch of the model to retain positional information[1], which traditional transformers cannot do. For performing classification tasks, an extra learnable classification token (CLS token) is added to the patch embedding vectors [1]. CLS tokens are widely used in NLP classification problems to represent the target class that the model is attempting to predict. The embedded vectors are fed into a standard encoder layer, where multi-headed attention is used to capture contextual information about the input sequence and layer normalization standardizes the activations across the features dimension of a layer. The features are then passed through a multi-layer perceptron (MLP) consisting of fully-connected linear layers which allow the model to learn global contextual information and positional information [1]. Predictions of class probabilities are outputted from the MLP [7].

The ViT requires large-scale training datasets to attain competitive performance among CNNs [1, 4]. This is because transformers lack some of the inductive biases inherent to CNNs, including translation equivariance and locality, and thus generalizes poorly given insufficient training data [1]. Additionally, ViTs struggle with high-resolution images as the model's computational complexity is quadratic to the image size. Furthermore, the fixed scale tokens in ViTs are unsuitable in problems where the visual elements are of variable scale [4].

Liu et al. (2021) addresses the latter two concerns through the Swin Transformer, which constructs hierarchical feature maps and has linear computational complexity to image size [4]. The Swin Transformer inputs small-sized patches and gradually merges neighboring patches in deeper Transformer layers. Linear computation is achieved by computing self-attention locally within non-overlapping windows that partition an image. The key design implementation is the shifted window partition between consecutive self-attention layers, which captures relationships between patches within local windows while maintaining some contextual awareness across the entire image [4]. Note that all the variants presented in the original paper consisted of 4 stages of a varying number of Swin blocks and latent dimension C.

3. Approach

3.1. **Vision Transformer (ViT).** The original implementation of Vision Transformer is in Google's machine learning framework (ViT) JAX [1]. We sought inspiration and guidance in navigating the architecture-building process in PyTorch through helpful GitHub repositories [8, 9, 10]. For our ViT experiments, we followed a more progressive approach, as we incrementally changed network architecture and implemented design choices to improve ViT's performance in classification tasks. For our control, we closely followed the original implementation and parameter choices followed by the authors for *ViT-Base* [1]. We did this because models with more learnable parameters reportedly under-performed on the smallest dataset, ImageNet, despite moderate regularization [1].

The sequence of steps following initial experiments with the *ViT-Base* include: (1) regularization, data augmentation, and other best practices, (2) implement changes in architecture, and (3) design hybrid *ResNet-ViT* models. Steiner et al. (2022) performed the first major comprehensive study on model regularization, data augmentation, and training data size for ViTs. From their paper, we experimented with random resizing and cropping random rotation and random horizontal flipping, dropout of 0.1, and introduced gradient clipping at maximum norm = 1 [11]. Other design changes included using *OneCycleLR* with a set parameter of total steps as the size of training dataset replacing cosine learning rate schedule with a linear warmup, and *AdamW* with beta = (0.9, 0.999), weight decay of 0.01 [12].

For the second stage, we experimented with the number of heads in multi-headed attention and the number of encoder layers. Dosovitskiy et al. (2020) originally implemented the classification head using a MLP with one hidden layer [1]. However, Steiner et al. (2022) removed the layer entirely, reporting that it empirically does not lead to increased accuracy in models and often resulted in optimization instabilities [11]. In place of the existing MLP-head, we had a GELU activation between two fully-connected layers and subsequently applied a Dropout layer.

For the final round of experiments, we implemented hybrid *ResNet-ViT* models using the default *ResNet34* module from PyTorch, as hybrid models were reportedly shown to improve pure Transformers for smaller model sizes [1]. Feature maps of a CNN can be fed into ViT as initial patch embeddings [1, 11]. Similar to the original paper, all training is done with resolution of 224 [1]. To reduce the complexity of our network, we deviated from using the typical *ResNet50* module as reported in papers and instead utilized the first five convolutional layers of *ResNet34* to process our image [1, 11].

3.2. **Convolutional Neural Networks (CNN).** Along with our Vision Transformer models, we designed and trained a CNN model to compare performance. Our CNN architecture is heavily inspired by the *ResNet* architecture [3] and

follows a similar implementation procedure to the best-performing model found here [13]. Since many *ResNet* models are primarily suited for 224×224 images, the architecture was adjusted to account for the 32×32 images we use. For instance, a typical *ResNet* model starts with a 7×7 convolutional kernel with stride 2, but a kernel of this size is not suitable for CIFAR100 images because after this layer our datum would be of size 16×16, and later become 8×8 after the first Maxpooling layer. Every other kernel in the ResNet architecture is of size 3×3, so we decided to use these kernels for our model. This is better illustrated in the block diagramA.1. Another point of emphasis is that in a typical ResNet model, the number of channels jumps from 3 to 64 to 128, 256, and 512. However, since our images are small and do not take many blocks before the dimensions are 1×1 (ignoring channels), we decided to jump from 3 channels to 256 immediately.

A block of our model typically consists of a 3×3 convolutional filter followed by ReLU activation and batch normalization in respective order. This sequence of operations is repeated twice before performing MaxPooling and Dropout with a probability of 0.5. We perform the operations in the block until we obtain features of size 1×1 with 512 channels. Following the convolutional layers, the features are passed through two fully-connected layers before outputting probabilities for the 100 target classes. Furthermore, data augmentation was considered and performed, namely random horizontal flip with default $p = 0.5$ and Random Crop. In the standard ResNet blocks, the batch normalization layer is applied before the ReLU activation [3]. Applying the operations in this orders entails half of the features *h* be negative before applying ReLU activation, which will set them to zero. We performed tests with both orderings to see the effects on our model performance. This is to confirm that ReLU before Batch Normalization works better; all our model results follow this procedure (ReLU before BatchNorm).

3.3. **Hierarchical Vision Transformer via Shifted Windows (Swin).** While Swin's official implementation by Microsoft Research is in PyTorch, our implementation has taken inspiration from another GitHub repository [14, 15]. By modifying the hyperparameters and the architecture components directly, we evaluated its robustness in performing image classification performance. We also explored its application by using each stage of Swin architecture as a basis unit for the VAE, which the experiments and its results are discussed further in section C.2. Our evaluation encompasses classification performance on CIFAR-100 in three stages: (1) confirming default feasibility with Swin_T, (2) adjusting complexity by varying Swin blocks, and (3) introducing an MLP block in addition to using different activation functions for the whole network [5].

For the second stage, the number of Swin blocks in the third stage of the Swin architecture and the latent dimension C are the primary hyperparameters in the original paper. In this case, we chose to vary the number of Swin blocks as it increases the complexity significantly more than C. Then, for the third stage, we used ReLU as the baseline for activation functions to compare with Swish, which improves ImageNet accuracy, and GELU, which reduces CIFAR-100 error rates compared to ReLU [16, 17].

4. Experimental Results

4.1. **Vision Transformer (ViT).** Our experimental results for ViT are summarized in the Table 1. Training and testing curves are in the Appendix B.1. For all ViT models, we used the following hyperparameters: initial learning rate of 0.0001, *OneCycleLR* with a set parameter of total steps as the size of training dataset, and *AdamW* with beta = (0.9, 0.999) and weight decay of 0.01, batch size of 256, number of workers = 2, and normalize the images.

Table 1. Testing Accuracy of ViT Models After 100 Epochs

Trial	Augment	Hybrid	MLP-Head	Patch Size	Embed Dim	Num Layers	Num Heads	Test Acc
A	No	No	Original	16	768	12	12	27.7%
B	No	No	Original	4	768	12	12	31.7%
C	No	No	Original	4	512	12	16	31.6%
D	Yes	No	Modified	4	512	8	16	31.9%
E	Yes	Yes	Original	7	512	8	16	47.7%
F	Yes	Yes	Modified	7	512	12	16	54.5%

At first glance, the most effective combination of design choices and implementations involved adding *ResNet34* features and modifying the MLP-head. performing data augmentation, reducing patch size and embedding dimension, and increasing the number of heads in multi-headed attention. Processing the image input through ResNet34 convolutional layers provided the most noticeable impact on validation accuracy, followed by decreasing the patch size. The behavior

is reasonable as smaller patch sizes yield greater information sharing by the self-attention mechanism during training [1]. Additionally, since we are working with a smaller resolution dataset, a smaller patch size will allow the model to attain some degree of local connectivity. However, using a smaller patch size is costly, as training time is longer.

For the hybrid ResNet-ViT model, the patch size is scaled down to 7 as it is divisible by the input resolution (i.e., 224). Using *ResNet34* as a backbone for processing input images aided in model performance. Transformers lack the inductive biases inherent in CNNs, namely translational invariance and locality [1]. Without having access to large amounts of training data and computational resources, CNN backbones allow ViT to perform better in relatively smaller datasets. Lastly, modifying the MLP head appears to be more beneficial to the hybrid model and has minimal effects on the normal ViTs. The addition of data augmentation during preprocessing does not appear to have a significant effect on validation performance, rather it slows down training time. Further robust testing is required to understand the tradeoffs between data augmentation and regularization and the relationship between all the ViT components.

4.2. **Convolutional Neural Networks (CNN).** The baseline model trained on CIFAR-100 underwent tests on ImageNet 64x64 for robustness. Utilizing PyTorch's `torchvision.datasets`, we efficiently implemented preprocessing steps, including converting images to Tensors, normalizing using ImageNet means and standard deviations, and augmenting the training dataset. Three DataLoaders were created: one without augmentations, one with augmentations, and one for testing. Each DataLoader had a batch size of 64, and the `CrossEntropyLoss` was used for the classification problem. Training employed the Adam optimizer with a learning rate of 5×10^{-4} and a learning rate scheduler decreasing by 0.1 every 25 epochs. The standard beta values were $(0.99, 0.999)$. Performance comparisons included CNNs trained with and without augmentations and CNNs compared to a trained ResNet34 model.

The overall results as a function of the number of epochs for our baseline CNN model can be seen in section A.1 starting with the training loss results shown in Figure A.3.

The first thing to observe is that for models that are not trained on images with data augmentation, the training loss decreases much quicker. This insight is relatively intuitive, as these images are easier to interpret by the model, and while the loss settles rather quickly, these models are not as adept as models that have augmented images as part of their training data at correctly classifying images that have different perspectives, rotations, etc. In other words, the data augmentation assists the CNN training process in achieving generalization.

As shown in Table 2, our best model is our CNN model with data augmentation transformations applied to the training dataset, with a 67% accuracy on the testing set. Moreover, it outperforms a standard ResNet34 model through 100 epochs, even though ResNet34 is pre-trained on ImageNet data. At first, we suspected that this was due to ResNet being trained on images of size 224x224, but scaling up the size of the CIFAR100 images to size 224 did not yield much improvement for the ResNet model, nor did altering the batch size or beta values for the Adam optimizer, and other such hyperparameters. Visually shown in Figure A.4 A.4, one can observe a sudden increase in the testing and training accuracies at the 25th epoch, which can be attributed to how we implemented our learning rate scheduler. Because no similar bumps in accuracy occur at the 50th and 75th epoch, it is reasonable to assume that the model has found an ϵ-neighborhood region for the local minimum.

TABLE 2. Test Accuracy of Baseline Models

Dataset Type	Our CNN	ResNet34
Augmentation	67%	62%
No Augmentation	54%	58%

The overall results are better outlined in the plots shown in Figure A.4 such that all four models begin to settle at their highest accuracy value around the same epoch; the most interesting insight is that the ResNet results are much less volatile to the inclusion of augmented images, suggesting that this model is more robust to changes in training data. Nonetheless, with the proper preprocessing steps, our model performs notably better than a standard ResNet model and thus provides the baseline for us to compare with the ViT and SWiN models.

4.3. **Hierarchical Vision Transformer via Shifted Windows (Swin).** To evaluate Swin's feasibility as the model of choice for the image classification task, we perform two main experiments by first varying the complexity of the model, with the other experiment focusing on varying the activation function. In addition, we also explored its feasibility as a basis unit for the VAE model compared to the original VAE implementation discussed in the appendix C.2 [18]. The results are shown in Table 3, and the performance curves (i.e., accuracy and loss) for each experiment are shown in section C.

TABLE 3. Testing Accuracy of Swin Models after 300 epochs

Name	Configuration	Activation function	Test Acc
GELU Original / Original	`[2, 2, 6, 2]`	GELU	76.8%
Simple	`[2, 2, 2, 2]`	GELU	76.38%
Complex	`[2, 2, 10, 2]`	GELU	77.1%
GELU	`[2, 2, 2, 2]`	GELU	75.2%
ReLU	`[2, 2, 2, 2]`	ReLU	73.4%
Swish	`[2, 2, 2, 2]`	Swish	73.7%

4.3.1. *Varying complexity of Swin architecture.* We utilized variants of the Swin transformer model, focusing on the simplest configuration, Swin_T ($[2, 2, 6, 2]$ blocks). GPU memory limitations led us to this choice, as more complex variants require additional memory during training. To balance training time and model complexity, all experiments centered around Swin_T. We trained and tested on the CIFAR-100 dataset using a batch size of 256, a target learning rate of 10^{-3} with *AdamW* or Adam with weight decay, over 300 epochs, following the Swin paper's settings on classifying ImageNet-10K dataset except their batch size was 1024 which we are limited by computational resources [12]. A cosine annealing scheduler was employed with 20 warm-up epochs and 30 decay epochs. After the initial testing, we modified the architecture, adjusting the number of Swin blocks in the third stage to 2 (*Simple*) and 10 (*Complex*), compared to the original configuration (*Original*). Label smoothing cross entropy loss and accuracy were used for evaluation. Results in Figure C.1 show no significant differences in accuracy and loss among the three configurations (i.e., the widest gap of 0.8 % difference). Training results in Figure C.2, however, indicate the expected trend: *Complex* $>$ *Original* $>$ *Simple* in terms of accuracy, signifying models with increasing complexity without overfitting which is expected to increase the accuracy as it can capture more nuisances presented in the Nature.

4.3.2. *Varying activation functions.* After exploring changes in performance by varying the number of Swin blocks, we focus on the choice of activation functions in this subsection, with the original using GELU. Notably, we introduced a 2-layer MLP before mean pooling and the final linear layer for classification. The *Simple* Swin architecture variants were used for computational efficiency. Results in Figure C.5 and Figure C.4 compare GELU, ReLU, and Swish activation function performances, along with the original *Simple* Swin architecture without the additional 2-layer MLP (*GELU_original*). GELU slightly outperformed ReLU and Swish in accuracy and loss, but *GELU_original* showed better test performance than GELU, indicating potential overfitting due to the additional 2-layer MLP at test time. As GELU has a stricter shape in the negative domain compared to Swish and is more lenient in letting negative values through compared to ReLU, using GELU is expected to perform better. At the same time, the overfitting suggests caution in model complexity.

5. DISCUSSION

We evaluated the usability of Transformers-based architectures for vision tasks (i.e., image classification on the CIFAR-100 dataset) such that the ViT fails to outperform the CNN, which is the current primary choice for computer vision problems. Vision Transformers lack the inherent inductive biases in CNNs, and the we acknowledge the improvement in performance following the addition of a *ResNet34* backbone.

As Swin architecture outperforms the other models, though with a heavy computational complexity due to the requirement of calculating the local attention windows, we would like to perform the experiments in the same vein with what we did with Swin_T but for Swin_L (i.e., configuration = `[2,2,18,2]`, $C = 192$), which we fail to do so due to computational resources limitations [4]. Moreover, we would like to explore combining Swin with models such as *ResNet34* as it was done with CNN and ViT, even though it is worth noting that Swin already has some residual connections [4]. Moreover, as discussed in section C.2, which we only trained the models for 50 epochs, we would like to train them with more epochs (e.g., 300 epochs, which is akin to that of the image classification tasks) since 50 epochs did not result in the intuitive decoded image output for VAE, in addition to increasing the complexity of the original VAE by adding more linear layers.

Given more time and computational resources, we would like to perform more robust tests to understand the relationship between components in transformer architectures as well as study design implementations from other state-of-the-art models, such as Data-efficient Image Transformers (DEIT) and CrossViT for vision tasks [19, 20].

REFERENCES

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*. [Online]. Available: https://arxiv.org/abs/2010.11929

[2] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang, and A. Dosovitskiy, "Do vision transformers see like convolutional neural networks?" in *Neural Information Processing Systems*. [Online]. Available: https://arxiv.org/abs/1706.02413

[3] K. He, X. Zhang, Shaoqing, and R. J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition*. [Online]. Available: https://arxiv.org/abs/1512.03385

[4] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. [Online]. Available: https://arxiv.org/abs/2103.14030

[5] A. Krizhevsky. The cifar-100 dataset. Available at https://www.cs.toronto.edu/~kriz/cifar.html.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Neural Information Processing Systems*. [Online]. Available: https://arxiv.org/abs/1706.037624

[7] J. S. Lee. Transformers: a primer. Available at http://www.columbia.edu/~jsl2239/transformers.html.

[8] P. Lippe. Tutorial 15: Vision transformers. Available at https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial15/Vision_Transformer.html.

[9] P. Wang. vit-pytorch. Available at https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py.

[10] A. Khanna, V. Mittal, and R. Bansal. Vision-transformer. Available at https://github.com/ra1ph2/Vision-Transformer/tree/main.

[11] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, "How to train your vit? data, augmentation, and regularization in vision transformers," in *Transactions on Machine Learning Research*. [Online]. Available: https://arxiv.org/abs/2106.10270

[12] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*. [Online]. Available: https://arxiv.org/abs/1711.05101

[13] L. T. Anh, "Cnn cifar100," Available at https://github.com/LeoTungAnh/CNN-CIFAR-100.

[14] F. Wang. Swin-transformer. Available at https://github.com/WangFeng18/Swin-Transformer.

[15] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin-transformer. Available at https://github.com/microsoft/Swin-Transformer.

[16] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2023.

[17] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[18] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[19] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers distillation through attention," in *International Conference on Machine Learning*. [Online]. Available: https://arxiv.org/abs/2012.128770

[20] C.-F. Chen, Q. Fan, and R. Panda, "Crossvit: Cross-attention multi-scale vision transformer for image classification," in *IEEE International Conference on Computer Vision*. [Online]. Available: https://arxiv.org/abs/2103.14899

[21] J. Kang. Vae-tutorial. Available at https://github.com/Jackson-Kang/Pytorch-VAE-tutorial.

APPENDIX A. APPENDIX

A.1. **Figures for Baseline Model.** Plots below

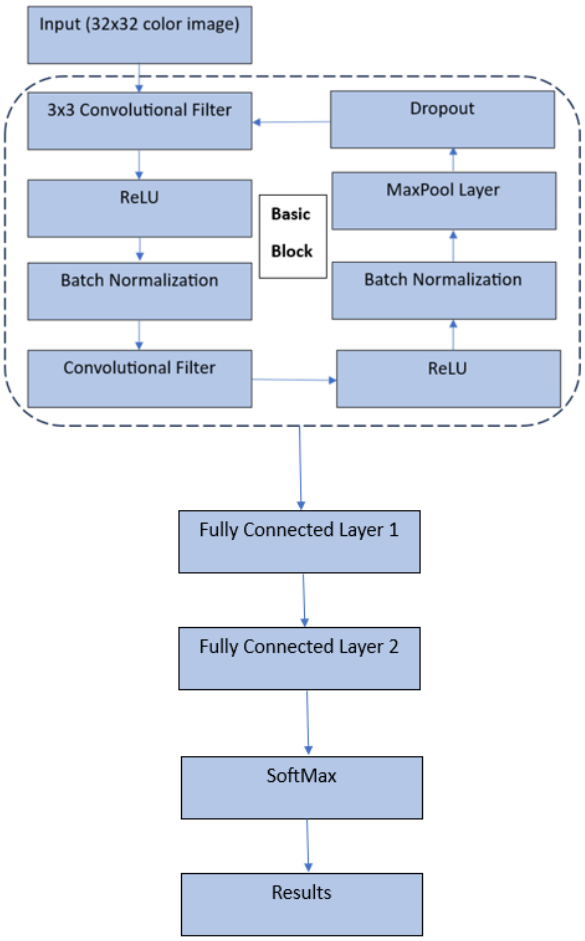

FIGURE A.1. Baseline Model Block Diagram

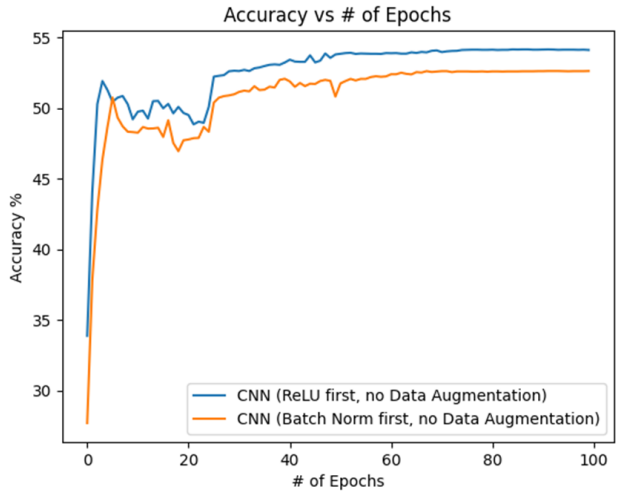

FIGURE A.2. Comparing Ordering of Layers in CNN Block

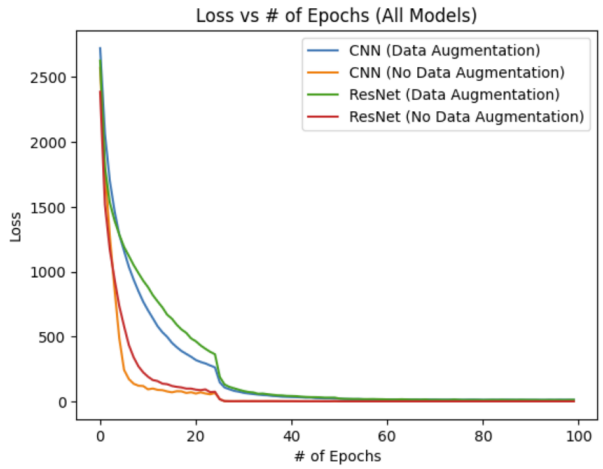

FIGURE A.3. Training Loss of Baseline Models

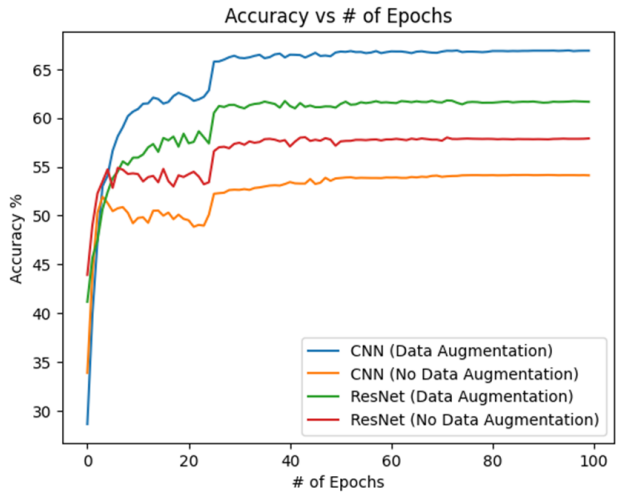

FIGURE A.4. Testing Accuracy of Baseline Models

APPENDIX B. VISION TRANSFORMER TRAINING AND TESTING CURVES

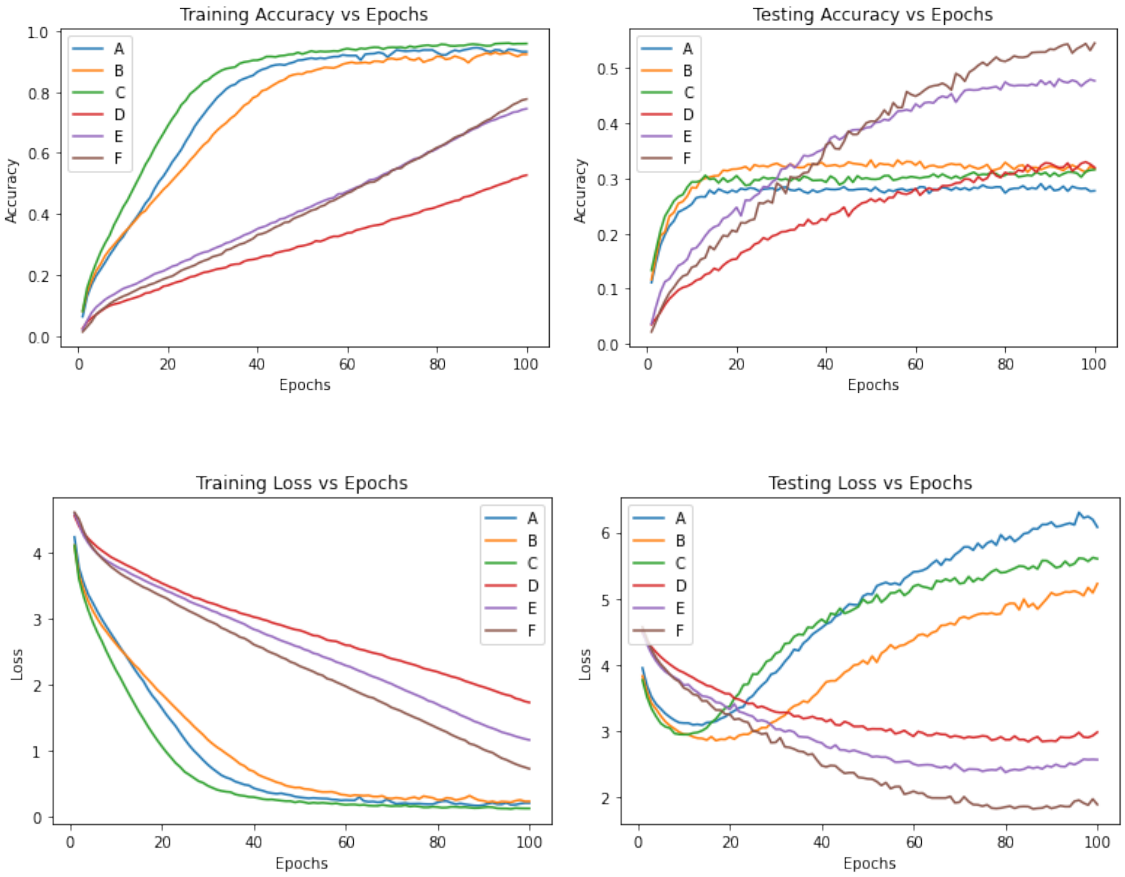

FIGURE B.1. Training (Bottom Left) and Testing (Bottom Right) Loss Curves of ViT Models. Training (Top Left) and Testing (Top Right) Accuracy Curves of ViT Models.

APPENDIX C. SWIN RESULTS AND POSSIBLE APPLICATION AS A BASIS UNIT OF VAE

C.1. **Additional Figures for Results.**

C.1.1. *Varying Complexity Results.* Recall that these are the keys to interpreting different variants of Swin:

- Complex: `[2,2,10,2]`
- Original: `[2,2,6,2]`
- Simple: `[2,2,2,2]`

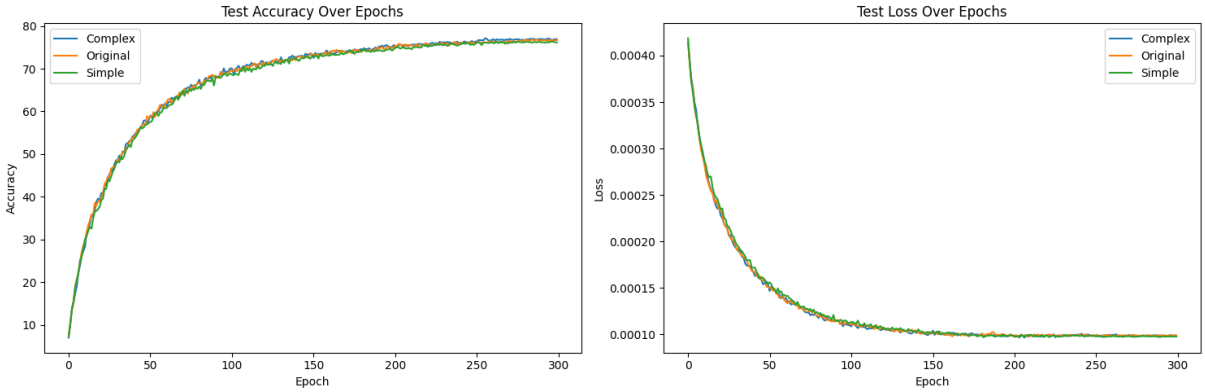

FIGURE C.1. Comparison of different Swin architecture variants based on complexity via number of Swin blocks for test set

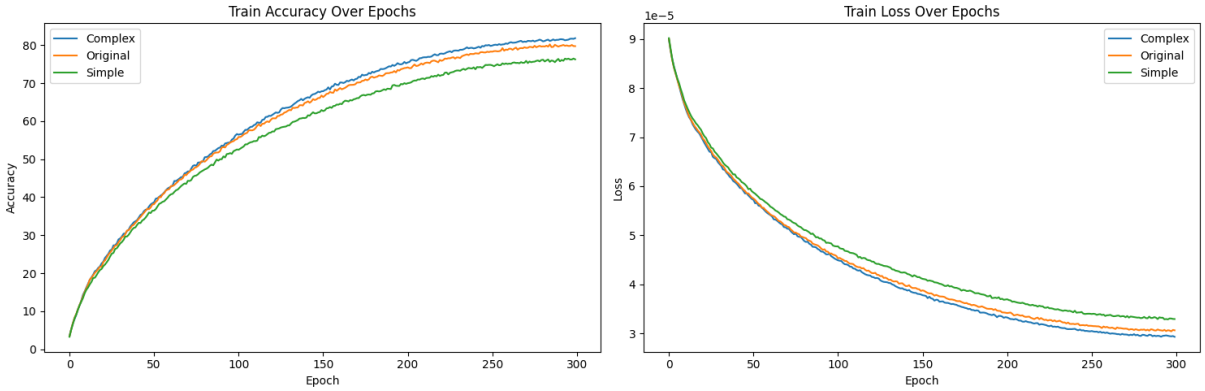

FIGURE C.2. Comparison of different Swin architecture variants based on complexity via number of Swin blocks for train set

C.1.2. *Varying Activation Function Results.* Recall that the GELU Original refers to the case where the architecture configuration is `[2,2,2,2]` and there is no added 2-layer MLP block to the architecture before the mean-pooling operator.

C.2. **Swin as a basis unit of VAE model.** While performing a literature review on Swin architecture, we realized that the Swin architecture utilizes a hierarchical feature map such that the height and width dimensions of the images are reduced by two-fold after every step due to the `Rearrange` tool and moves them into the latent dimension C, multiplying that by four-fold. We used that observation, combining with the implementation of the Variational Autoencoder (VAE) from the class lecture and homework, along with the implementation from GitHub repository, to build an encoder-decoder model using the Swin architecture stages as the basis of the model, instead of the fully-connected neural networks or linear layers, which we coined the architecture as Swin-VAE [21, 18]. We approached this idea by mirroring the pattern of Swin architecture to upsample the height and width dimensions two-fold after

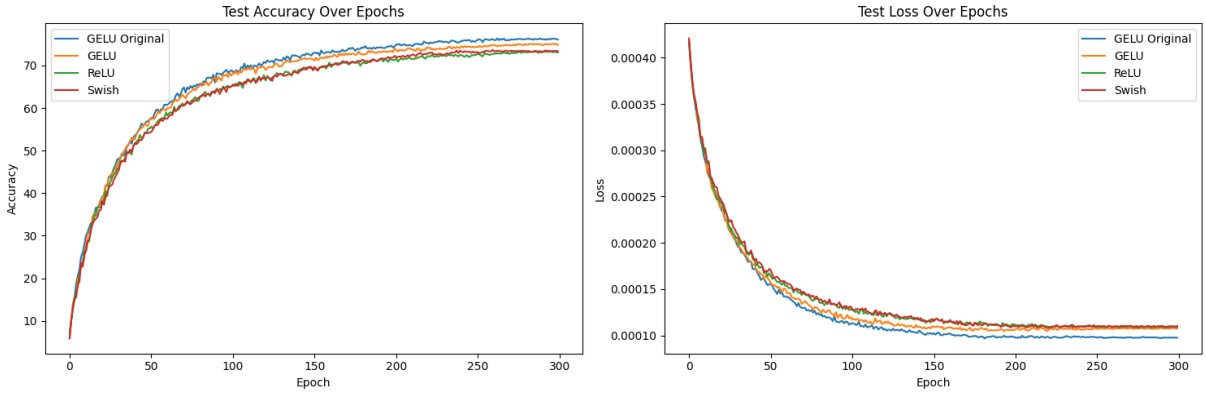

FIGURE C.3. Comparison of different activation functions used in Swin architecture with additional 2-layer MLP for test set

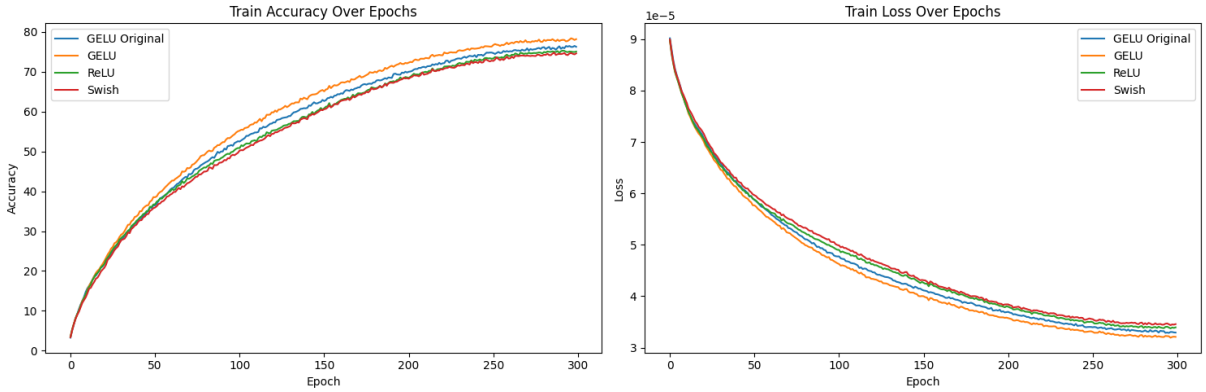

FIGURE C.4. Comparison of different activation functions used in Swin architecture with additional 2-layer MLP for train set

each stage for the decoder part, with the final output being an image. Specifically, for the network configuration of `[2,2,6,2]`, we will have a total of 8 stages with the configuration as `[2,2,6,2,2,6,2,2]`. The evaluation process using the sum of reconstruction loss and KL Divergence loss uses the same implementation as that of the VAE, which we also implemented using the inspiration from the GitHub repository [21]. In addition, since this model creates more overhead, we reduced the batch size to 128 and the number of epochs to 50 for both Swin-VAE and VAE for the model to be trainable with the available resources.

With the resultant images shown in Figure C.5, the Swin-VAE performed well such that the reconstruction is visually better than the VAE's. We suspect that this is because we have trained the model for only 50 epochs, and also, the VAE model is not complex enough to capture all the mean and variance as the output from the encoder. In addition, we also have the reconstruction loss and KL Divergence loss in Figure C.6 and the total loss in C.7

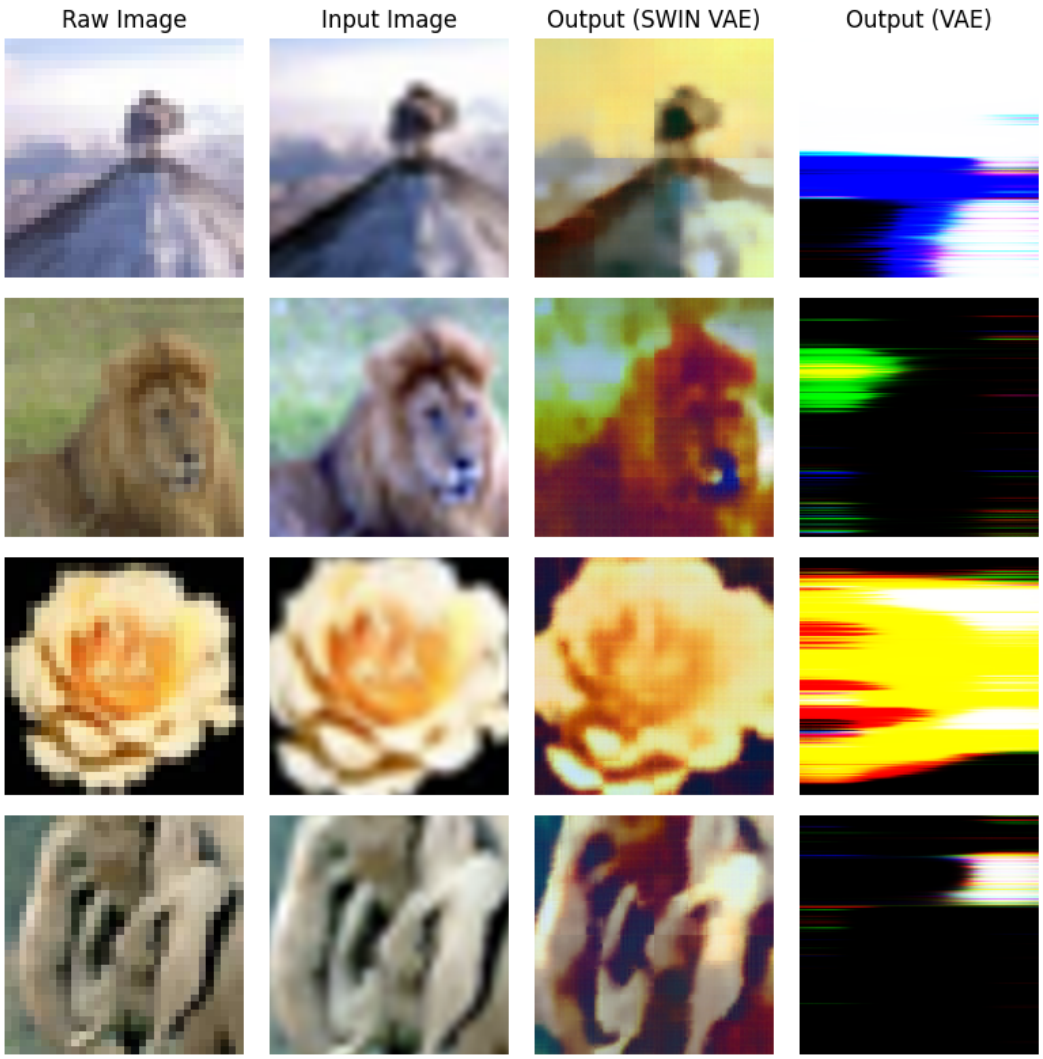

FIGURE C.5. Output visualization for VAE using Swin stages as the basis units and VAE using linear layers as the basis units

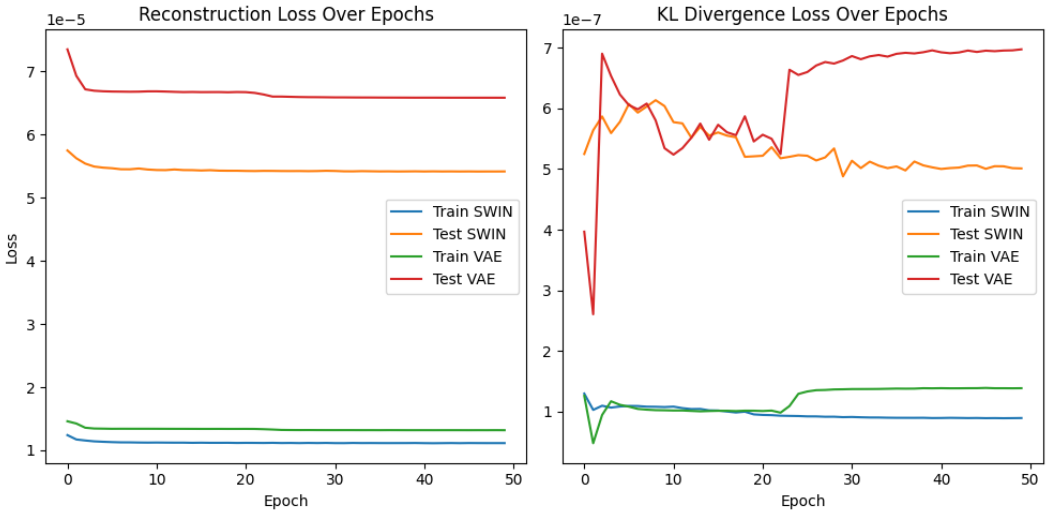

FIGURE C.6. Original VAE (VAE) and Swin-VAE (Swin) Reconstruction and KL Divergence loss versus epochs

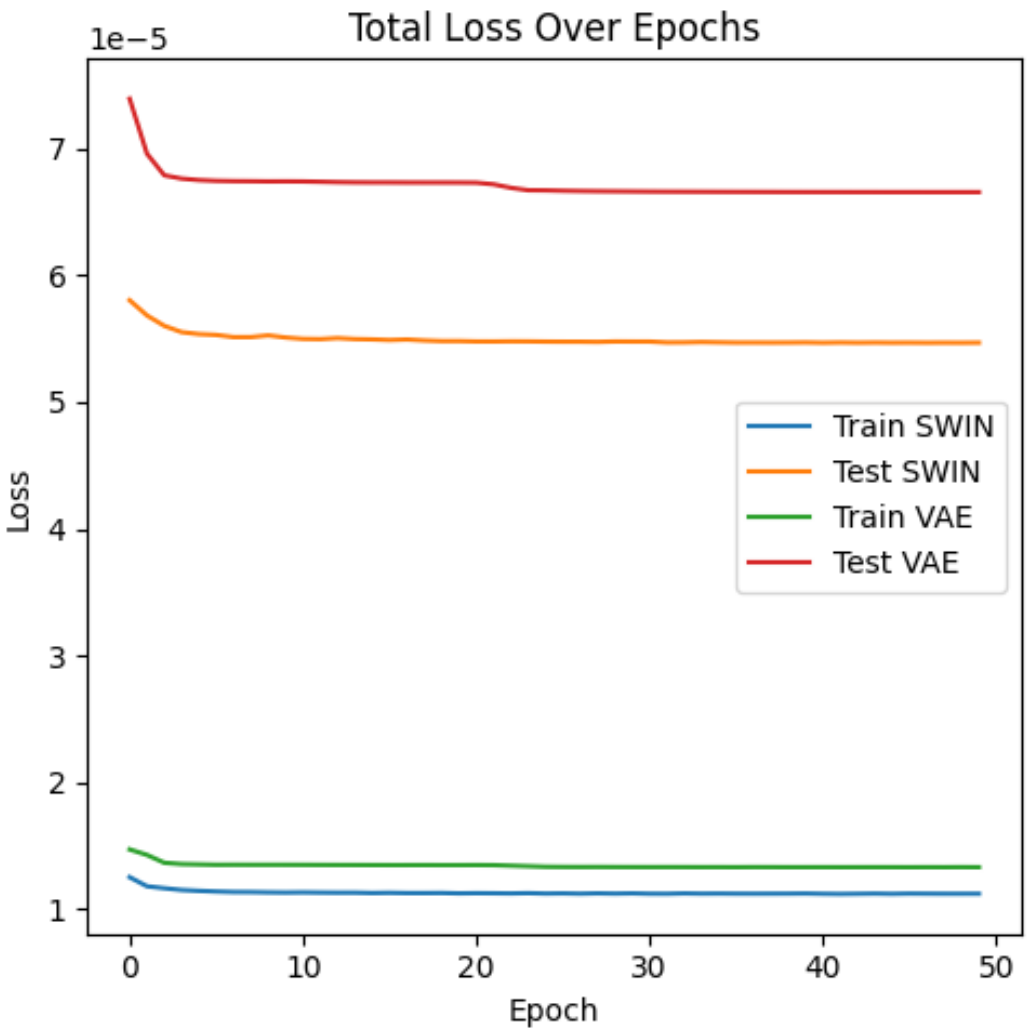

FIGURE C.7. Original VAE (VAE) and Swin-VAE (Swin) Total loss versus epochs