Replication of Swin Transformer

Yutao Zhou
Department of Electrical Engineering
Columbia University
New York, USA
yz4359@columbia.edu

***Abstract*—This report presents the entire process of reimplementing the Swin Transformer. The generic architecture has been re-implemented, which can be used to achieve different variants of Swin Transformer, such as Swin-T, Swin-B, etc, by specifying different sets of parameters. I tested this archetecture on two datasets - the imagenet1K [1] and the cifar100 dataset [2]. However, due to resource constraints, I cannot train the imagenet1K efficiently. On Cifar100, I achieved 46.7%(80 epochs) total accuracy with the Top 5 accuracy being 72.96%(80 epochs). In this report, the original Swin Transformer paper was summarized with a comparison with the literature. Next, I elaborated on the implementation details. In the end, I presented the result of the experiment and discussed the challenges I faced during the implementation.**

***Index Terms*—TensorFlow, Image Classification, Swin Transformer, Cifar100**

I. Introduction

This project is a replica of the original "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows" paper. I studied the original work and replicated the Swin Transformer architecture by using the deep learning library - TensorFlow.

II. Summary of the Original Paper

The transformer has a long history of being great at machine translation since 2017 [3]. Before the appearance of the transformer, it is hard to learn dependencies that are distant [**?**] without using Recurrent Neural Network(RNN) or Convolution [3]. The transformer is good at effectively learning distanced dependency. This characteristic makes recurrent make it shine in Natural Language Processing (NLP) domain. In 2021, A research group at Google introduced Vision Transformer(ViT) to apply transformer-based architecture to Computer Vision(CV) domain [2]. They improved on existing architectures and used a larger large dataset(ImageNet-21k and JFT-300M) to train the model [2]. Subsequently, Vision Transformer could outperform state-of-the-art CNNs in image classification. Further, the Swin Transformer uses Vision Transformer as a cornerstone and creates a model that could accomplish more general computer vision tasks like image classification, object detection, and semantic segmentation. All in all, a transformer has proven effective in both NLP and CV.

[1]https://image-net.org/challenges/LSVRC/2012/index.php
[2]https://www.cs.toronto.edu/~kriz/cifar.html

A. Methodology of the Original Paper

Convolutional neural networks (CNNs) have dominated the computer vision domain. Whereas, transformers have always been the backbone of natural language processing (NLP) models. These two architects have been extraordinarily successful in their own domain. However, scholars have always sought a way to make cross-filed applications. In the original paper, the authors are trying to apply transformers to computer vision. This adaptation would be challenging because the structures of natural language and images are different in many ways.

Transformer use tokens to process word input, and use tokenized words for model training. Tokenization is efficient because words in natural language are similar and would not expand to a wide range. Depending on the Tokenization method, a fixed-size dictionary might be used. But, this is not true in images. Pixies could vary substantially in scare and sample from the sample. It would not be possible to build such a fixed scale 'dictionary' efficiently. Another problem is images have higher resolution compared to words. For example, a 720x720 image would have 518.4K pixels not to mention there are usually three to four channels for an image. This would be the length of a novel in natural language case [1]. What's more some vision applications like semantic segmentation require pixel-level predictions. As a result, the transformer would struggle to perform efficiently.

B. Key Results of the Original Paper

The original paper proposed Swin Transformer overcome the performance issue. It combined ideas from CNN and Transformer and achieved great, more than 80% accuracy, geniality in image classification, object detection, and semantic segmentation. Also, it has a lower real-life latency compared to the original Transformer. One of the largest advantages of the Swin Transformer is its efficiency. The previous Transformer based architectures have quadratic complicity. This is because it uses a single-resolution feature map. In contrast, Swin Transformer achieves linear efficiency by using non-overlapping windows. The number of patches in each window is fixed, as a result, a linear complexity would be achieved.

In traditional sliding window-based self-attention approaches, the connectivity is preserved because of the use of one feature to slide on the entire window. On account of non-overlapping windows, the connectivity between windows has been eliminated. To address this problem, the original paper designed shifted window approach. The architects always have

two consecutive Swin transformer blocks. In the first block, the entire window is partitioned into four equal size windows by cutting in the middle of the width and height of the image. The attention in block one is going to be on local windows only. In the second block, the entire window would be shifted by half of the window size. After shifting the three sections of the original image that is outside of the new window would be used to fill in the blank space in the new window position. Now, the attention is going to be on the newly formed windows. Nevertheless, the new window would have some segments that are not connected in the original image. So, a mask would be used to constrain the attention to only part of the window that is connected in the original image. With a shift window and two continuous block approaches, the connectivity problem has been solved.

III. Methodology

I kept the exact structure as the original paper. However, I did train it on a smaller dataset because of the limited computational resource I have. After starting the project, I realize the resource problem in GCP and the stableness of the remote Jupyter Notebook are not as good as the local machine. Therefore, I move most of the training, to the local machine and only use GCP for environment compatibility checks. The original ImageNet 22K was not used because of the limited computational power we have[3]. I will discuss this in detail in the discussion section. We have chosen to use the CIFAR-100 and ImageNet2012 subset datasets instead. I would discuss more regarding the reason later in the discussion part.

A. CIFAR-100

I have mainly conducted my experiment on CIFAR-100 because this is a standard dataset that my GPU could handle efficiently. I have done some pre-filtering on the parameters that I am going to adjust. First, I tested different layer numbers for the base model. I tried to keep the exact layer numbers as the Swim-T, which is 2,2,6,2 for each stage, version in the paper. However, I realized no matter what parameters I set, the model is barely learning anything with the training accuracy below 10%. I suspect this is because my model is too expressive for CIFAR-100. The original paper used a way larger dataset(137GB) than CIFAR-100(162MB). I will discuss the model capacity more in the discussion session. So, I reduce the layer number dramatically. Eventually, I observed the best layer number for CIFAR-100 is 1,1,2,1. Then, I tested the number of heads and fixed it to 2,2,4,8 which is very small compared to Swin-T(3,6,12,24). Still, this setup is used to limit model capacity. These two steps are used to make sure we have a reasonable size model for our dataset. Then, I tested three learning rates at (0.01, 0.001, 0.0001). I kept the best learning rate of 0.001. I did not perform any further experiments on the learning rate and formally start my experiment. The general experiment method is to control variables with independent variable tuning. For every group of experiments, there would only be one parameter to be changed. After I have found the best parameter for one variable I would generally keep it fixed for the rest of the experiment.

There are six parameters been experimented with. They are Batch Size, Window Size, Embedding Dimension, Max Dropout Layer Probability, Dropout Probability, and Attention Dropout Probability. A detailed table of parameter values tested is shown in TABLE I.

TABLE I
DETAILS OF PARAMETERS SETTING THAT HAS BEEN TESTED.

Parameters	Values Experimented
Batch Size	128, 256, 512, 1024
Window Size	2, 4, 8
Embedding Dimension	16, 32, 64, 128
Max Dropout Layer Probability	0.1, 0.15, 0.2, 0.25, 0.3, 0.35
Dropout Probability	0.1, 0.15, 0.2, 0.25, 0.3, 0.35
Attention Dropout Probability	0.05, 0.1, 0.15, 0.2, 0.25, 0.3

B. ImageNet2012 subset

For the ImageNet2012 subset, I have trained two models. The ImageNet2012 subset is a 20GB dataset with each image size that is a lot larger than 32 by 32. Therefore, I have increased the model capacity greatly, with the same number of blocks per layer, or stage, setting as Swin-T. For the first model, I trained 40 epochs. This takes about 6 hours on 2 GTX 1080Ti. After inspecting the training history. I validation loss is still decreasing, and validation accuracy is still increasing. In other words, the model has not converged and did not overfit after 40 epochs. Thus, I retrained the model and set the number of epochs to 200 while keeping other parameters the same. The training took about 32 hours. The detailed parameters setting are in TABLE II.

TABLE II
DETAIL OF PARAMETERS SETTING FOR IMAGENET2012 SUBSET.

Paramiters	Values Experimented
Learning Rate	0.0001
Batch Size	72
Number of Epochs	40(200)
Window Size	7
Embedding Dimension	96
Number of Blocks(for each stage)	2, 2, 6, 2
Number of Heads(for each stage)	3, 6, 12, 24
Weight_decay	0.0001

C. Objectives and Technical Challenges

The object of this project is to build a deep learning model that could recreate the original experiment result. More specifically, I focused on the image classification tasks in the original paper.

The largest problem I have faced in my recreation is the shortage of computational resources problems. After I have finished building the model on my local machine, I plan to migrate all of the codes to GCP and train the model there. I first used the class VM setup to train the model with ILSVRC

[3]On VM I have 1 V100 GPU. On the local machine I have access to 2 1080Ti GPUs

2012 dataset. The class VM setup is 2vCPU, 7.5GB memory, and NVIDIA Tesla T4 with 100GB of Storage. With this setup, the original data, which is 137GB, is larger than the storage. Therefore, I create another VM with the same setup and 500GB of Storage. After I download and preprocess the dataset. I faced the main issue that I did not find a good way to solve. The original paper used a batch size of 1024. However, I was only able to set to batch size to 64. A larger batch size would cause the GPU to run out of memory. With the batch size at 64, the model is training extremely slowly. This is understandable because the dataset is very large. Therefore, I decided to create a more powerful VM for the training.

I tried to create a VM with NVIDIA A100 40GB GPU, and 12vCPU, 85GB memory. However, only very limited zones have the A100 40GB GPU, and within those, there is not enough CPU resource. All possible zones had been tried. But, none of them have enough resources to create the VM. Thurs, I created a VM with NVIDIA V100 GPU this is the only selection available that is closest to A100. But, it only has 16GB of Memory. I tried to run the model with ILSVRC 2012 dataset again. Still, the training is extremely slow.

Then, I tried to use local machines for training. My local machine has 2 GeForce GTX 1080Ti GPUs which add up to a total of 22GB of memory. But, running the model on a subset(20GB) of ILSVRC 2012 with a batch size of 64 and 40 epochs would still take a day. The original paper has trained for 300 epochs. I need to tune the model's parameters meaning to get the correct parameters which would lead to multiple runs. This is clearly not possible with the time limit of this project. Considering the reasons above, I decided to use the CIFAR-100 dataset which is 162MB in size. Also, I moved back to GCP since my VP could handle the CIFAR-100 dataset efficiently.

One detail I have to point out is I have used imagenet2012_subset combining it with the ILSVRC 2012's validation dataset. The original validation dataset has been used as the training data and imagenet2012_subset has been used as a validation set. I combine them to create a larger dataset. I swap them because the training date should be more than the validation data. Otherwise, those validations that could be used to train data had been wasted.

D. Problem Formulation and Design Description

To overcome my limited resource problem, I have decided to use a local machine for my project. My local machine has 2 1080Ti GPUs which would add up to 24 GB of RAM. Also, I have decided to use CIFAR-100 and ImageNet2012 subsets if time permits. Both datasets are a lot smaller compare to ImageNet 22K the original paper used. I prioritize tuning my model on the CIFAR-100 dataset because it is faster and I could implement more experiments. Training results are needed for the parameter tunning, there would be a higher chance that I could get a good result with a smaller dataset. The system and software for this project are fairly simple. There would be a few python files that form the layer component and utility that are necessary for the Swin Transformer. Then There would be the main python file called 'swimtransformer' that used components and utilities I mentioned before to build a Swin Transformer Block and model. Last but not least, there would be the main Jupiter notebook that acts as a user interface. It would interact with the main python file and trigger training.

IV. Implementation

In this section, I am going to present my replication of the original Swin Transformer model. In my implementation, I have replicated the entire structure of the Swin Transformer. I will first present the data set I used. Then, I will present an overview of my architecture. After that, I will present my architecture in detail by mimicking how my architecture will deal with image inputs to achieve different tasks, such as image classification. Last but not least I will talk about the Software Design I used.

A. Data

Datasets have been used in my project. The first one is CIFAR-100. The second one is magenet2012Subset with 10pct.

The CIFAR-100 is a standard dataset. It contains 100 classes of images with 600 images in each class [4]. Among the 600 images in each class, there are 500 images for training and 100 for testing [4]. Also, there are 20 superclasses to which each of the 100 classes would belong [4]. The dataset has a fixed image size of (32, 32), Therefore I do not need to do preprocessing. I could simply load it from Keras.

The ImageNet 2012 subset is a subset of the original ILSVRC2012 dataset. Same as CIFAR-100 it is a standard dataset. It shares the validation dataset with the ILSVRC2012. it contains 128116 training examples and 50000 validation examples [6]. I pre-processed this dataset by first converting the raw image to TensorFlow data type float32. The original ImageNet datasets contain images of various sizes. I need to unify the size to (224,224,3) to follow the original paper. First, I resized the image to (256,256,3) in my case. Then, I use CenterCrop to crop the image to size (224,224,3) Lastly, a Normalization layer is applied.

B. Deep Learning Network

The same as the architecture in the paper, roughly speaking, my architecture consists of an image partition block and four consecutive stages. The image partition block is used to generate patches of a given size and embeddings of each patch. The four consecutive stages (we call a Swin Transformer layer in our implementation) are very similar in terms of architecture. In each stage, there will be a number of Swin Transformer blocks and patch merging steps to achieve down-sampling. By this down-sampling step (with a down-sampling rate of 2), the architecture can capture the hierarchical structure of images. Different variants of the Swin Transformer proposed in the paper have different numbers of blocks. For instance, Swin-T has 6 blocks in the third stage and Swin-S has 18 blocks. Usually, there is a multiplier of two consecutive Swin

Transformer Blocks. The main difference between the two consecutive Swin Transformer blocks lies in that an attention layer is applied to partitioned windows in the first block while the attention layer is applied to shifted windows in the second block. Put differently, in the second block, we will first shift images before applying masked attention such that the architecture can capture cross-window connections. The first block will be used to capture connections over non-overlapping local windows. In such a setting, this architecture can capture both inner and intra-window attention A concrete overview of my architecture is shown in Figure 1.

First of all, we would have an image partition block that partitions the input images into patches. This block is achieved by using *tensorflow.keras.layers.Conv2D*. I set the *kernel size* to be *(patch size, patch size)* and the stride to be *patch size*. As a result, I can get the embedding for each patch. Then there would be a dropout layer used in the training process to avoid over-fitting.

Then we would get to the four stages (we call it four layers), which are the main blocks for the architecture. As we mentioned before, the four stages are very similar except that the last layer will not have a down-sampling layer. In each Swin Transformer layer, there would be multiple Swin Transfomer Blocks (a multiplier of 2 usually). Roughly speaking, each block is a standard transformer block, though, it is performed over local windows instead of the entire image input. Inside the block, I will first need to partition the input into windows of a pre-specified size. Then after performing a layer normalization, an attention layer is applied to local windows obtained from the previous step, followed by a normalization layer and a fully connected layer. In the end, I will reverse the partition step to recover the original input shape. This first block architecture only captures the connections between non-overlapping local windows. The second block architecture will be used to capture cross-window connections. In order to achieve this goal, the original paper proposed to perform a cyclic shift over the input image first and then re-perform the previous block over the shifted image. However, one problem is that cells in the current partitioned local windows may not be neighbors in the original image. Following the original paper, we need to generate an attention mask, which is used to limit the attention to a portion of the window so that only cells that are adjacent would be considered. Concretely, I will add 0 to the attention scores of the cells that are adjacent to the target cell in the original image whereas I will add a large negative number (in my implementation, I use -1000) to the attention score of those that we want to omit. After I perform softmax over the modified attention scores, non-adjacent cells will have an attention score close to 0. In such a way, I can only attend to adjacent cells even if the image is shifted.

In a nutshell, in this block, I first apply a normalization layer. Then if this window needs to be shifted I would roll the windows and refill blanks, inspired by the explanation in this blog [4]. Then, the image would be partitioned. Meanwhile, if the image is shifted, I will generate neighboring indices for each cell and then generate attention masks[5]. After that, the (masked) attention layer I mentioned before would be applied over local windows. Next, the image would be recovered by reversing the partition. Also, I would reverse the roll if this is a shifted window. The dropout layer and multi-layer perception will follow. This concludes the architecture of an entire Swin Transformer Stage. As mentioned, there will be four stages like this. As a result, this architecture would be applied four times with different parameters (such as different attention heads).

I will now talk about the downsampling process in detail. Following each Swin Transformer layer, we would have a downsampling layer(except for the last stage). In the down-sampling, we use a down-sampling rate of 2. In detail, we first partition an image into 2-by-2 windows. In each 2*2 window, we then split it into 4 cells and stack them along channels. As shown in Figure 2, an image is partitioned into four windows (i.e., x0, x1, x2,x3). For each such window, we initially have X000, X001, X010, and X011 each with C channels. Then, these four cells are stacked along channels, which leads to 4C channels. Then they will go through the normalization layer and the number of channels will be reduced to 2C channels by using a dense layer. As a result, this down-sampling processing halves the height and width of images and doubles the number of channels.

A detailed structure of the multi-head attention module is shown in Figure 3. When an image comes in, it will first be normalized. Then we use the parameter *shift size* to denote whether we need to shift the image and create an attention mask. If the *shift size* is larger than 0, we will shift the image and create an attention mask such that only nearby cells will be used in attention. Then we will partition the (shifted) image and calculate attention scores. Then, again, if the image has been shifted, we will reverse the shift. Then we will reassemble the windows in order to recover the original size of the image. In the end, we will use the dropout, normalization, and MLP to finish the entire block.

Figure 4 is an illustration of how the attention mask and shift are implemented. The matrix on the left represents the original images. Then the entire image had been shifted by 2x2 toward the top-left direction. This would result in the yellow, pink, and blue section is outside of the image size. So, we will move those three patches to the bottom right and place them in the position shown in the middle of the graph. So far, the image is shifted. Then, the image is partitioned into four windows again where those black lines on the rightmost image represent the border of those four windows. As we can observe, in three of the windows there are pixels that are not from the same patch. As a result, we need to use masks to rule out the influence of pixels that are not in the same patch. The numbers in the image will help me distinguish whether a pixel is in the same patch or not.

After the four transformer layers, there will be a normalization layer. Then, global average pooling would be applied to

[4]https://amaarora.github.io/2022/07/04/swintransformerv1.html

[5]when the image is not shifted, no attention mask is needed.

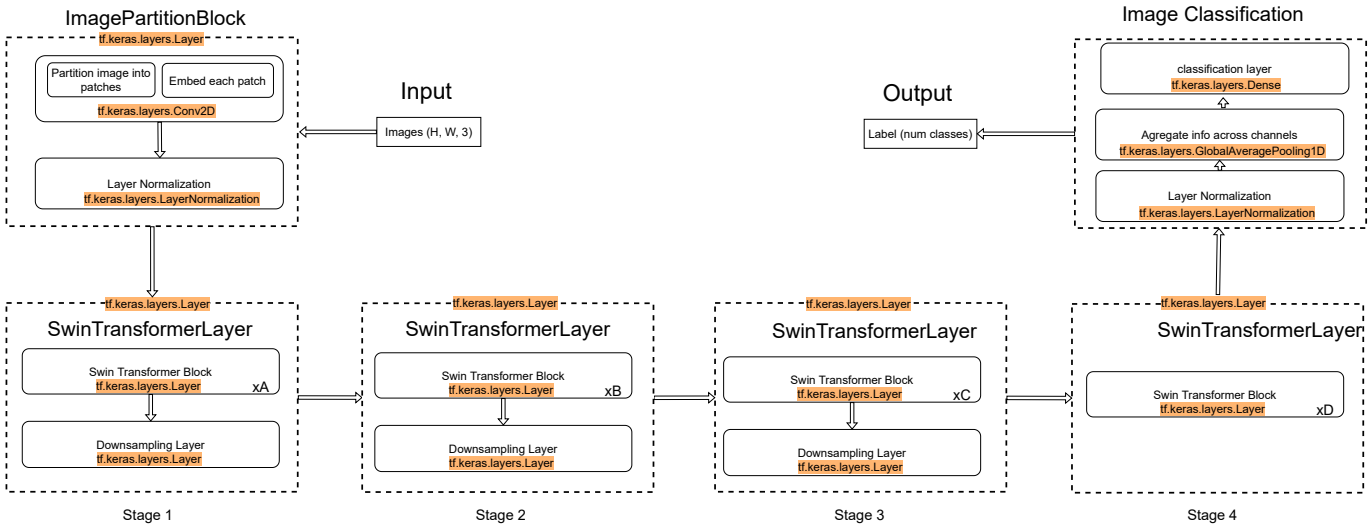

Fig. 1. Illustration of the implemented architecture.

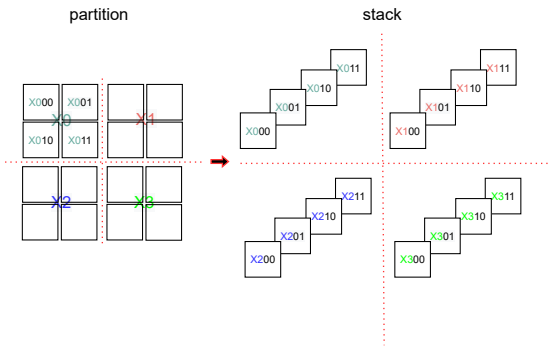

Fig. 2. Illustration of the down-sampling process.

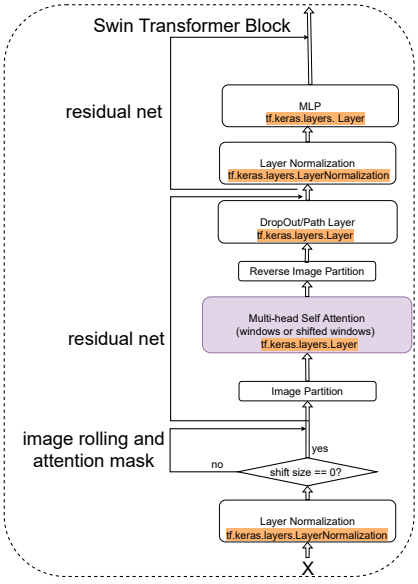

Fig. 3. Illustration of the multi-head attention module.

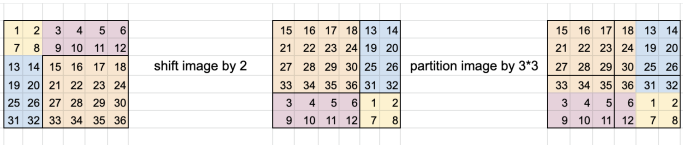

Fig. 4. Illustration of generating the attention mask.

aggregate information across channels. Finally, a dense layer with an activation function *softmax* will be used to achieve the image classification task.

C. Software Design

My software system is shown in figure 5. On the very top, I used a Jupyter Notebook [6] as a user interface. This is where all of the results would be visualized and the training process been triggered. After we start training the mode, the Jupyter Notebook would call the main Swin Transformer Python File [7]. Then the main Swin Transformer Python File would use functions and classes in 'layers.py' [8] and 'utils_func.py' [9] to construct the model. Then the dataset would be used to train my model. After the training, the model would be saved locally. In the future, the user could load those models back to make predictions.

V. Result

A. Project Results

I would show the results respectively separated by datasets.

[6]https://github.com/ecbme4040/e4040-2022Fall-Project-DLNN-yz4359/blob/main/main_experiment_cifar100.ipynb

[7]https://github.com/ecbme4040/e4040-2022Fall-Project-DLNN-yz4359/blob/main/swimtransformer.py

[8]https://github.com/ecbme4040/e4040-2022Fall-Project-DLNN-yz4359/blob/main/layers.py

[9]https://github.com/ecbme4040/e4040-2022Fall-Project-DLNN-yz4359/blob/main/utils_func.py

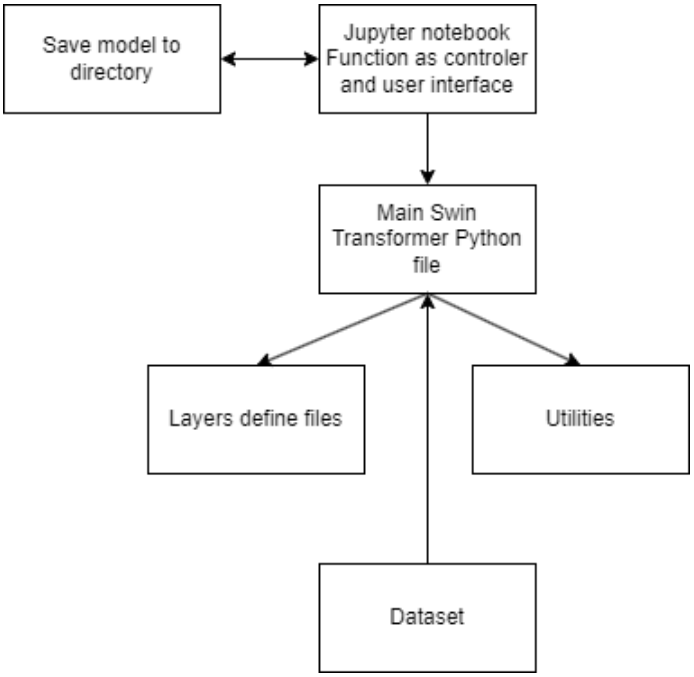

Fig. 5. Top level flow chart for my software.

B. CIFAR-100

I will first show the result for the CIFAR-100 dataset grouped by parameters. Then, I will show all of the results in a table and present the best model among 24 models. All of these models have been saved to the liondrive.

All of the CIFAR-100 models have a learning rate of 0.001. The Number of Blocks for each stage is set to 1,1,2,1. The Number of Heads for each block is set to 2, 2, 4, 8. **Batch Size** TABLE III is the influence of Batch Size on the model. All the models have the same parameter(except the Batch Size) with Window Size=2, Embedding Dimension=32, Max Drop Layer Probability=0.15, Dropout=0.1, and Attention Dropout=0.05.

TABLE III
MODEL ACCURACY GROUP BY THE BATCH SIZE.

Batch Size	Accuracy	Top5-Accurarcy
128	0.4143	0.6872
256	0.402	0.6667
512	0.3965	0.66
1024	0.3882	0.6559

∗TrainedonCIFAR100datasetwith2GTX1080Ti

Window Size

TABLE IV is the influence of Window Size on the model. All the models have the same parameter(except the Window Size) with Batch Size=512, Embedding Dimension=32, Max Drop Layer Probability=0.15, Dropout=0.1, and Attention Dropout=0.05.

Embedding Dimension

TABLE V is the influence of the Embedding Dimension on the model. All the models have the same parameter(except the Embedding Dimension) with Batch Size=512, Window Size=4, Max Drop Layer Probability=0.15, Dropout=0.1, and Attention Dropout=0.05.

TABLE IV
MODEL ACCURACY GROUP BY THE WINDOW SIZE.

Window Size	Accuracy	Top5-Accurarcy
2	0.3965	0.66
4	0.4146	0.6731
8	0.398	0.6594

∗TrainedonCIFAR100datasetwith2GTX1080Ti

TABLE V
MODEL ACCURACY GROUP BY THE EMBEDDING DIMENSION.

Embedding Dimension	Accuracy	Top5-Accurarcy
16	0.3878	0.6803
32	0.4146	0.6731
64	0.4218	0.6752
128	0.01	0.05

∗TrainedonCIFAR100datasetwith2GTX1080Ti

Max Drop Layer Probability

TABLE VI is the influence of the Embedding Dimension on the model. All the models have the same parameter(except the Max Drop Layer Probability) with Batch Size=512, Window Size=4, Embedding Dimension=64, Dropout=0.1, and Attention Dropout=0.05.

TABLE VI
MODEL ACCURACY GROUP BY THE MAX DROP LAYER PROBABILITY.

Max Drop Layer Probability	Accuracy	Top5-Accurarcy
0.1	0.4162	0.6709
0.15	0.4218	0.6752
0.2	0.01	0.05
0.25	0.4318	0.6897
0.3	0.4304	0.688
0.35	0.4304	0.6888

∗TrainedonCIFAR100datasetwith2GTX1080Ti

Dropout

TABLE VII is the influence of the Embedding Dimension on the model. All the models have the same parameter(except the Max Drop Layer Probability) with Batch Size=512, Window Size=4, Embedding Dimension=64, Max Drop Layer Probability=0.3, and Attention Dropout=0.05.

TABLE VII
MODEL ACCURACY GROUP BY THE DROPOUT.

Dropout	Accuracy	Top5-Accurarcy
0.1	0.4162	0.6709
0.15	0.44	0.7045
0.2	0.4448	0.7086
0.25	0.4383	0.7109
0.3	0.4396	0.7125
0.35	0.4381	0.7207

∗TrainedonCIFAR100datasetwith2GTX1080Ti

Attention Dropout

TABLE VIII is the influence of the Embedding Dimension on the model. All the models have the same parameter(except the Max Drop Layer Probability) with Batch Size=512, Window Size=4, Embedding Dimension=64, Max Drop Layer Probability=0.3, and Dropout=0.3.

TABLE VIII
MODEL ACCURACY GROUP BY THE ATTENTION DROPOUT.

Attention Dropout	Accuracy	Top5-Accurarcy
0.05	0.4396	0.7125
0.1	0.4406	0.7125
0.15	0.4495	0.7301
0.2	0.4418	0.7287
0.25	0.4442	0.7193
0.3	0.4423	0.7147

∗TrainedonCIFAR100datasetwith2GTX1080Ti

All Experiment result for CIFAR-100

All of the experiment I have performed on CIFAR-100 is in TABLE IX below. The best model parameters are Batch Size=512, Window Size=4, Embedding Dimension=64, Max Drop Layer Probability=0.3, Dropout=0.3, and Attention Dropout=0.15. This would achieve overall accuracy of 44.95% and top-5 accuracy of 73.01%.

C. ImageNet2012 subset

For the ImageNet2012 subset, I have the first train is 40 epochs, and the second train is 200 epochs. The parameters described in TABLE II were used in both runs. For the first train, I got an accuracy of 10.08% with the top 5 accuracy being 25.5%. For the second train, I got an accuracy of 13.79% with the top 5 accuracy being 31.15%.

D. Comparison of the Results Between the Original Paper and Students' Project

My best model achieves an overall accuracy of 44.95% and top-5 accuracy of 73.01%. The original paper has achieved an accuracy of 81.3% with Swin-T variance on image classification on ImageNet 1K dataset [1]. ImageNet 1K is a much larger dataset than CIFAR-100. As a result, the parameters the original paper used does not apply to my dataset. This is why I have reduced the size of my Swin Transformer greatly to only having 1,1,2,1 block per stage. Note that the original Swin-T has 2,2,6,2 blocks per stage. After all of the experiments with CIFAR-100, I realized my model would still be overfitting badly disregarding that I have tried to limit its capacity. The CIFAR-100 dataset is simply too small for my model. I have tried a lot more sets of parameters than those that are presented in this report. However, none of them performed significantly better.

Also, it would not be fair to compare two models on a different dataset. Therefore, I compared it to another paper that uses CIFAR-100 and tested it with Swin Transformer. In the paper "ML-Decoder: Scalable and Versatile Classification Head", they have achieved an accuracy of 95.1% with the Swin L and ML Decoder [5]. However, their paper is mainly focusing on ML-Decoder. They did not provide details about their implementation of Swim L. However, I think the differences in the performance between these two models lie in three aspects at least. First, they used a larger model - namely, Swim L is a larger variant with greater model capacity compared to Swin T, while the largest model I could set up in my GCP/local machine is even smaller than Swin-T [1]. Second, due to the same issue of the limited computational power, I can only set the batch size to 64/128 at most. Usually, the batch size should be 1024 in the original paper [1]. Third, I notice that they might resize the images in CIFAR-100 to a larger size. However, they did not mention that the size has been changed [5]. I have tried to resize it as well; however, my computational power failed to handle such a big model with such high-resolution images. There may be other reasons for the differences. If they could provide enough details on how they performed the classification task over cifar100, I will be more confident in discussing the difference in performance. Additionally, I realized that data processing and augmentation will impact the model performance significantly.

When applying my model to the ImageNet2012 subset, my model was still overfitting. For the 200 epochs run, the final training accuracy reached 44.75%. This is not contradictory to my hypothesis above because I have increased model capacity significantly for the ImageNet2012 subset. I may have overestimated the capacity needed and did not set the right parameters. I did not successfully find the more optimal parameters for both datasets. This was not expected. I tried very hard given the limited computational resources. GCP is unusable for the ImageNet2012 subset. It could generate the subset properly. But, before starting training the model, the kernel would die. What's more, I tried to train the model directly in the ssh terminal. However, TensorFlow would kill the program because of the memory limit. When I trained the model with ImageNet 2012, each training would take substantial time. This makes optimizing the model unpractical given I only have myself on the team.

E. Discussion / Insights Gained

One of the most important things I learned from this project is the Swin Transformer is computationally demanding. The original paper was able to pre-train the model with ImageNet 22K for 90 epochs with a batch size of 4096. This is unimaginable for my setup. I could not possibly achieve this because I do not have access to machines that could handle the batch size of 4096. As mentioned above, the largest batch size that would not crush my GPUs is 74. Even if a very small batch size was used, I still would not be able to pre-train the model efficiently one epoch for the entire dataset would cost a few days. This really makes me think that the development of deep learning would speed up greatly with the evolution of hardware. In 2016, a top-of-the-line GPU GTX1080Ti has 3584 CUDA Cores. Today, only 6 years later the newest 4090 has 16384 CUDA Cours. Not to mention architecture improvements like memory speed and size difference(11GB vs 24GB). This cross-domain influence is fabulous and amazing.

TABLE IX
DETAILED MODEL ACCURACY FOR CIFAR-100

Batch Size	Window Size	Embedding Dimension	Max Drop Layer Probability	Dropout	Attention Dropout	Accuracy(%)	Top5-Accurarcy(%)	Loss
512	4	64	0.3	0.3	0.2	44.18	72.87	2.86
256	2	32	0.15	0.1	0.05	40.2	66.67	3.16
512	4	64	0.1	0.1	0.05	41.62	67.09	3.09
512	4	64	0.3	0.3	0.25	44.42	71.93	2.91
1024	2	32	0.15	0.1	0.05	38.82	65.59	3.23
512	4	64	0.3	0.2	0.05	44.48	70.86	2.95
512	4	32	0.15	0.1	0.05	41.46	67.31	3.11
512	4	64	0.3	0.3	0.15	44.95	73.01	2.85
512	4	64	0.3	0.25	0.05	43.83	71.09	2.97
512	4	64	0.2	0.1	0.05	1	5	4.61
512	4	64	0.15	0.1	0.05	42.18	67.52	3.08
512	4	64	0.3	0.35	0.05	43.81	72.07	2.86
512	4	64	0.3	0.3	0.1	44.06	71.25	2.94
512	4	64	0.3	0.3	0.05	43.96	71.25	2.93
512	4	64	0.3	0.15	0.05	44	70.45	2.96
512	4	128	0.15	0.1	0.05	1	5	4.61
512	4	64	0.3	0.1	0.05	43.04	68.8	3.03
512	4	64	0.35	0.1	0.05	43.04	68.88	3.05
512	4	64	0.3	0.3	0.3	44.23	71.47	2.9
512	2	32	0.15	0.1	0.05	39.65	66	3.19
512	4	16	0.15	0.1	0.05	38.78	68.03	2.91
512	8	32	0.15	0.1	0.05	39.8	65.94	3.21
512	4	64	0.25	0.1	0.05	43.18	68.97	3.04
128	2	32	0.15	0.1	0.05	41.43	68.72	3.08

∗TrainedonCIFAR100datasetwith2GTX1080Ti

The influence of pre-train the model would also influence the experiment results a lot. The original paper has pre-trained the model with ImageNet-22K [1]. This would improve the baseline accuracy of the model because we are randomly initiating weights and did not use pre-train (we are unable to pre-train the model with the setting of the original paper due to limited resources). It would not be possible to achieve the same experiment result without pre-training. Another key influence is the pre-processing methods for the experiment. Without knowing the detailed methods of how the data was pre-processed, I could not recreate the experiment. Also, pre-processing would increase the accuracy of the model. This would be one of the reasons my model did not perform as well.

Another insight I got is a lot of experiment details could influence the experiment result. Even if the detailed structure was presented, a lot of detailed parameters would usually be omitted. For example, the original paper did not mention the parameters they used for dropout or normalization layers. The setting of these detailed parameters would influence the experiment result. My experiment might use different settings or methods from the original paper. This is the reason that in the Computer Science domain open sources are encouraged. So, all of the details could be found in the source code. However, there may not be corresponding descriptions in the paper that refer to detailed parameters in the source code.

Last but not least, a tremendous amount of computational power is needed in projects like Swin Transformer. A person usually would not have access to such a quantity of computational resources. Therefore, a large machine learning project would need a team or a center to perform the experiment and work together. It is not a good idea to work alone in the Deep learning and machine learning domain.

VI. FUTURE WORK

There is plenty of future work to do. I have completely implemented the entire structure of the Swin Transformer. My model could be pre-trained with better hardware on the original ImageNet 22K datasets. Then it could be finetuned on ImageNet 1K. In addition, all four variances that were mentioned in the original paper could be rebuilt and compared. These are the main work that could be done to recreate the original experiment result.

VII. CONCLUSION

All in all, in this project the entire architecture of the Swin Transformer has been recreated. On Cifar100, my model achieved 46.7%(80 epochs) total accuracy with the Top 5 accuracy being 72.96%(80 epochs). The objective accuracy is not achieved. But, my model is working. The main constrain are not solvable by myself. In the future, it would be amazing if my model could be reused to train on the original dataset with capable machines.

VIII. ACKNOWLEDGEMENT

I would take this time to acknowledge my professor Doctor Zoran Kostic. He is a fabulous professor who makes deep learning clear and intriguing. Also, our TA team is amazing. With their hard work in answering questions on Ed and giving recitations. Your work is very helpful.

References

[1] Z. Liu et al., 'Swin transformer: Hierarchical vision transformer using shifted windows', in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 10012–10022.

[2] A. Dosovitskiy et al., 'An image is worth 16x16 words: Transformers for image recognition at scale', arXiv preprint arXiv:2010. 11929, 2020.

[3] A. Vaswani et al., 'Attention is all you need', Advances in neural information processing systems, vol. 30, 2017.

[4] A. Krizhevsky, G. Hinton, and Others, 'Learning multiple layers of features from tiny images', 2009.

[5] T. Ridnik, G. Sharir, A. Ben-Cohen, E. Ben-Baruch, and A. Noy, 'Ml-decoder: Scalable and versatile classification head', arXiv preprint arXiv:2111. 12933, 2021.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, 'Imagenet: A large-scale hierarchical image database', in 2009 IEEE conference on computer vision and pattern recognition, 2009, pp. 248–255.

IX. Appendix

A. Individual Student Contributions in Fractions

I work alone without teammates. Thurs, all of the work are contributed by myself. Also, I would skip the table because there is no need to do so.