Reproduction of "A Simple Framework for Contrastive Learning of Visual Representations"*

1st Yilin Ye
Columbia University
yy3152@columbia.edu

1st Zhenyi Yang
Columbia University
zy2540@columbia.edu

1st Haoyu He
Columbia University
hh2982@columbia.edu

***Abstract*—In this project, we focused on implementing the SimCLR (Simple Framework for Contrastive Learning of Visual Representations) framework using TensorFlow and training it on the CIFAR100 dataset. Our goal was to replicate the original SimCLR study's methodology and results. Following the training, we conducted a comparative analysis against the outcomes reported in the original paper. This analysis primarily centered on identifying and understanding the discrepancies between our results and those documented in the original research. Key areas of focus included model performance metrics, data preprocessing techniques, and hyperparameter settings. Our findings provide insights into the application of SimCLR on CIFAR100 and contribute to the broader understanding of contrastive learning in the domain of computer vision.**

I. Introduction

In recent years, we have witnessed significant advancements in self-supervised learning specifically tailored for computer vision problems. Among these developments, a noteworthy contribution is the paper by Chen et al., titled 'A Simple Framework for Contrastive Learning of Visual Representations' [1]. This paper introduces a framework known as SimCLR, which stands out for its simplicity and effectiveness in contrastive learning of visual representations. In this report, we delve into the study of this paper, endeavoring to replicate some of the experimental results and discuss our findings in detail.

II. Summary of the original paper

A. SimCLR

Fig.1 shows the core components of the SimCLR framework, which is a contrastive learning method for visual representations. In this framework, an original image 'x' undergoes augmentation to produce two correlated views 'xi' and 'xj'. These augmentations, represented by 'τ ', are randomly sampled from a predefined augmentation strategy, which, according to the literature, is critical for effective performance and includes random cropping and color distortions. The base encoder network 'f(.)' computes representations of the augmented images, which are then mapped into a space where contrastive loss can be applied through a nonlinear projection 'g(.)'. The goal is to maximize agreement between these projections 'zi' and 'zj' using a contrastive loss that encourages representations of augmentations from the same image to be closer together than representations from different images, thereby learning useful features from the data in an unsupervised manner.

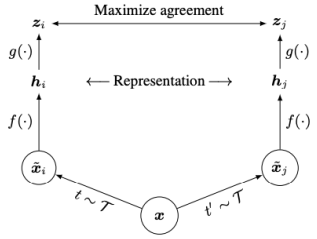


Fig. 1. Illustration of SimCLR

B. Data Augmentation

In the original paper, the authors sequentially apply three simple augmentations: random cropping followed by resizing to the original size, random color distortions, and random Gaussian blur. They note that the combination of random cropping and color distortion is crucial for achieving good performance. In our project, we have employed a similar approach, and to enhance accuracy, we have attempted to introduce additional augmentations.

C. Loss Function

In the SimCLR framework,the author used NT-Xent loss function in a sensible way.

For two augmented images xi and xj, the cosine similarity s(i,j) is calculated using the projected representations zi and zj, with the formula:

$$s_{i,j} = \frac{z_i^T z_j}{\tau ||z_i|| ||z_j||}$$

where τ is a temperature parameter that scales the inputs

Then, they used Noise Contrastive Estimation Loss formulation to calculate and the loss for each pair is determined by taking the logarithm's negative of the previously mentioned cosine similarity score.

$$l(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$$

	Food	CIFAR10	CIFAR100	BIRDSNAP
linear evalustion:				
SimCLR	68.4	90.6	71.6	37.4
Supervised	72.3	93.6	78.3	53.7
Fine-tuned:				
SimCLR	88.2	97.7	85.9	75.9
Supervised	88.3	97.5	86.4	75.8
Random init	86.9	95.9	80.2	76.1

TABLE I
A SIMPLE FRAMEWORK FOR CONTRASTIVE LEARNING OF VISUAL REPRESENTATIONS

Finally, the total loss L over a batch of size N is the average of the loss over all positive pairs:

$$L = \frac{1}{2N}\sum_{k=1}^{N}[l(2k-1,2k) + l(2k,2k-1)]$$

D. Global BN

In the original paper, the researchers chose not to use a memory bank for convenience and instead varied the training batch size from 256 to 8192. However, the standard methods like SGD/Momentum under these conditions would become unstable, so the author employed the LARS optimizer for improving stability. Also, they utilized Cloud TPU, expanding from 32 to 128 cores depending on the batch size, and implemented global batch normalization to prevent local information leakage.

E. Key Results

From Table.I , it is observed that on the CIFAR-100 dataset, SimCLR achieved an accuracy of 71.6 in linear evaluation, compared to 78.3 for supervised learning. After fine-tuning, SimCLR improved to 85.9, while the supervised model reached 86.4. This shows that although SimCLR underperforms pure supervised learning without fine-tuning, its performance significantly increases with fine-tuning, narrowing the gap with the supervised model.

According to the paper, SimCLR would be beneficial from larger data and longer training, thus it would perform well even when transferred to different datasets and fine-tuning tasks. This aspect drew our attention, leading us to focus on its transfer capabilities in our research.

III. METHODOLOGY

A. Objectives and Technical Challenges

Our plan is to assess the effectiveness of the representations learned from the SimCLR model using a 3-layer linear classifier. For this purpose, we have chosen the CIFAR-100 dataset as our primary dataset. This dataset will be instrumental in evaluating the nuanced features learned by the SimCLR model through its self-supervised learning process.

B. Data augmentation in our project

In our project, we use many data augmentation methods which is shown below.

Random Crop and Resize: This augmentation randomly crops an image and then resizes it to a target height and width.This method introduces variability in the scale and position of objects within the image.

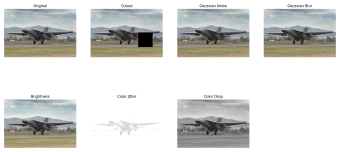


Fig. 2. Examples for some data augmentation in the project

Random Crop, Resize, and Flip: Extending the previous technique, this method additionally applies a horizontal flip to the image with a certain probability. This augmentation simulates the variability in object orientation, enhancing the model's ability to recognize objects in different orientations.

Center Crop: This method crops the central region of an image to a specified height and width. It helps the model focus on the most significant part of the image, potentially improving its ability to recognize central features.

Color Distortion (Drop and Jitter): Color distortion encompasses two effects - 'drop', which converts an image to grayscale, and 'jitter', which randomly alters brightness, contrast, saturation, and hue. These manipulations help the model become robust against variations in lighting and color in the input images.

Rotation: This technique rotates the image by a random angle (90, 180, or 270 degrees). Rotation introduces variability in the orientation of objects, aiding the model in learning orientation-invariant features.

Cutout: This technique applies a mask of zeros (blackout) to a random location within the image, forcing the model to focus on less prominent features and thereby improving its ability to generalize.

Gaussian Noise: Adding Gaussian noise to an image simulates the effect of sensor noise in real-world scenarios, helping the model to become more robust against such noise in practical applications.

Gaussian Blur: This method applies a Gaussian blur to the image, which can simulate out-of-focus scenarios. It helps the model in handling images with varying levels of sharpness.

Sobel Filtering: Sobel filtering is used to detect edges in the images. By training on images with emphasized edges, the model can become better at recognizing shapes and contours, which are critical for object detection and classification tasks.

Fig.**??** displays some examples and for more details, please look at https://github.com/d29f/E4040-SimCLRthis GitHub link..

Here are the pseudo code descriptions for the provided functions:

`gaussian_noise` function:

1) Adds Gaussian noise to an image tensor with a given standard deviation.
2) The function applies this noise with a specified probability.

`gaussian_blur` function:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
resnet50 (Functional)	(None, 2048)	23587712
dense_1 (Dense)	(None, 128)	262272
dense_2 (Dense)	(None, 50)	6450

Total params: 23856434 (91.01 MB)
Trainable params: 23803314 (90.80 MB)
Non-trainable params: 53120 (207.50 KB)

Fig. 3. Overall Model Structure

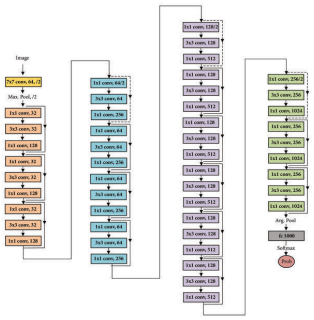


Fig. 4. Block diagram of Resnet-50

1) Applies Gaussian blur to an image tensor based on a randomly chosen sigma value within a specific range.
2) The function performs this operation with a given probability, respecting guidelines from the referenced paper.

For the details of implementation and to include these functions in your report, please refer to the code provided or the TensorFlow documentation for methods like `tf.random.normal`, `tf.clip_by_value`, `tf.nn.depthwise_conv2d`, and other used TensorFlow operations.

IV. Implementation

A. Data

In this project, following the methodology outlined in the referenced paper, we used pretrained weights from the ImageNet dataset, which are readily available on TensorFlow. Subsequently, we fine-tuned our model using the CIFAR-100 dataset. The CIFAR-100 is recognized as a critical benchmark in the computer vision domain, frequently used to assess image recognition algorithms. The CIFAR-100 dataset comprises 60,000 color images, each measuring 32x32 pixels. It is organized into 100 classes, with each class containing 600 images. This composition ensures an even distribution across all classes, providing a balanced variety of data for model training and evaluation.

B. Deep Learning Network

Fig.3 outlines our Overall Model Structure, showcasing the architecture of our neural network. The ResNet-50 architecture is detailed depicted in Fig.4 (Available from https://www.researchgate.net/figure/Block-diagram-of-Resnet-50-1-by-2-architecture$_{fig4_326198791}$).

For the part of the training algorithm for the SimCLR model, we illustrate the training algorithm with several key

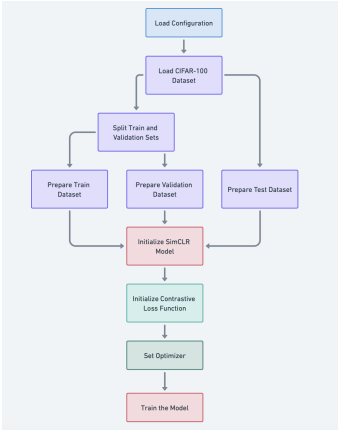


Fig. 5. top level flow chart

steps. Initially, the `augment_image` function applies image augmentation, performing resizing, cropping, and color distortion during training; for testing, it only resizes. Next, the `preprocess_for_simclr` function normalizes the images and creates two augmented versions for training, or resizes them for testing. These images are loaded into a TensorFlow Dataset using the `load_dataset` function, which applies shuffling, batching, and prefetching. The model employs a contrastive loss function (`contrastive_loss`) using NT-Xent loss to measure the similarity between augmented image representations. Training configurations such as batch size, dimensions, learning rate, and epochs are specified in a YAML file. Finally, the model, instantiated with the `ResNetSimCLR` class, is trained using an SGD optimizer alongside a cosine decay learning rate schedule.

As for the data used in the project, we began by loading the CIFAR-100 dataset using TensorFlow's `tf.keras.datasets.cifar100.load_data()` function, which provided us with 50,000 training and 10,000 test color images, each 32x32 in size and labeled across 100 fine-grained classes. We then processed this dataset, first by splitting the training data into new training and validation sets through our `split_train_validation` function, ensuring we had a separate set for model evaluation during training. Next, we prepared the datasets for training, validation, and testing phases using our custom `load_dataset` function.

C. Software Design

Fig.5 shows the low-level model training process in our project, starting with the initial setup and data preparation while Fig.6 introduces low level model training.

For the detailed algorithm, please see the comments in the code for step by step implementation description.

Here is some pseudo code for our procedures:

- **augment_image function:**
  1) Applies a series of data augmentations to an image tensor.

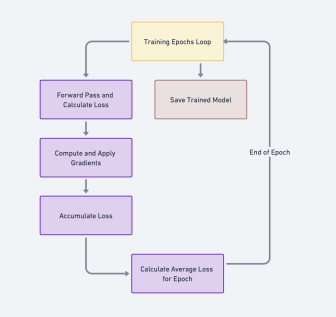


Fig. 6. low level model training flow chart

2) Includes resizing, cropping, color distortion, and potentially other augmentations like rotation and cutout.

- **preprocess_for_simclr function:**
  1) Normalizes an image tensor and applies the augment_image function twice to create two augmented versions of the image tensor for training.
- **load_dataset function:**
  1) Creates a TensorFlow dataset from image and label tensors.
  2) Applies the preprocess_for_simclr function to each element in the dataset.
  3) Shuffles and batches the dataset for training purposes.
- **contrastive_loss function:**
  1) Defines the loss function used in contrastive learning (NT-Xent loss).
  2) Calculates the loss between two sets of embeddings from the augmented images.
- **Model Training:**
  1) Iterates through epochs and batches of the dataset.
  2) For each batch, computes the forward pass and the contrastive loss.
  3) Performs a backward pass to update the model's weights.
  4) Saves the model after training.

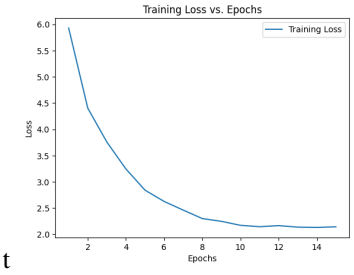
You can see more information at https://github.com/d29f/E4040-SimCLRthis GitHub link.

V. Results and Discussion

A. Project Results

In our endeavor to replicate the SimCLR, we chose the CIFAR100 dataset, considering the intensive time and resources required for training on the entire ImageNet dataset as in the original paper. The training loss stopped decreasing after 15 epochs, and the final loss value is 2.14.

To evaluate the effectiveness of the learned representations from our SimCLR model, we employed a 3-layer linear classifier. This classifier was designed to test the quality of the features extracted by the SimCLR model from the CIFAR100 dataset. The classifier consisted of three dense



t

Fig. 7. Training loss curve of our model

layers with intermediate dropout layers for regularization. the test accuracy achieved was 6.9%, which is notably lower than the benchmarks set by the original SimCLR model that is pretrained on ImageNet and fine-tuned on CIFAR100. The plot of training loss is shown in Fig.7

B. Comparison of the Results

In the original SimCLR paper, the authors achieved remarkable results by pretraining the model on the full ImageNet dataset and then utilizing this pretrained model for transfer learning tasks. Specifically, they reported an accuracy of 80.2% on CIFAR100 in the linear evaluation task, and an impressive 89.0% accuracy in the fine-tuning task. In contrast, our project, constrained by the unavailability of the pretrained SimCLR model weights on ImageNet, took a different approach. We adapted the ResNet50 architecture, incorporating pretrained weights from ImageNet. Focusing solely on linear evaluation, we extracted feature representations from the CIFAR100 test dataset using our model. These features were then fed into a linear classifier, ultimately achieving an accuracy of 6.9%.

C. Discussion

Our implementation of the SimCLR framework exhibits notable deviations from the original paper, primarily driven by differences in data sources, model training, and augmentation strategies. The original paper's approach was to train the SimCLR model on the expansive ImageNet dataset, followed by fine-tuning or evaluation on CIFAR100. Our adaptation, however, leveraged the ResNet50 architecture with weights pre-trained on ImageNet, diverging from the original model's training methodology.

A critical difference lies in the optimizer and batch size used. The original study employed a LARS optimizer with an exceptionally large batch size of 4096, optimizing the learning process for a vast dataset like ImageNet. In contrast, our implementation utilized the SGD optimizer with a significantly smaller batch size of 512. This alteration in the training environment could substantially impact the model's ability to converge to optimal solutions and learn effective representations. Furthermore, the learning rate and training duration were markedly different; the original paper used a learning rate of 4.8 over 100 epochs, whereas our approach utilized a learning rate of 1.0 for just 15 epochs. This reduction

in both the learning rate and the number of training cycles likely contributed to the disparities in model performance.

The implementation of data augmentation techniques also varied considerably. The original paper's methodology included random crops with flips and resizing, color distortion, and Gaussian blur, applied to ImageNet's 224x224 sized images. These augmentations are well-suited for larger images that contain abundant information. However, applying similar augmentations to the CIFAR100 dataset, which comprises 32x32 sized images, proved challenging. The smaller image size of CIFAR100, offering much less detailed information compared to ImageNet, introduced complexities and instability in the training process when subjected to equivalent augmentation techniques. This fundamental difference in data quality and augmentation adaptability is a significant factor contributing to the differing results between the original study and our project.

VI. Future Work

Looking ahead, we plan to try multiple methods to improve our model's performance. A key focus will be on adjusting and optimizing the data augmentation functions, an aspect that significantly influences the efficacy of the SimCLR model. We plan to find the optimal combination of data augmentation techniques for training CIFAR100 dataset.

Additionally, we plan to delve deeper into tuning various hyperparameters, such as learning rate and batch size. We will also apply LARS optimization to the model and compare its performance with SGD and Adam.

Finally, we aim to align more closely with the methodology outlined in the original SimCLR paper by undertaking the challenge of pretraining the SimCLR model using the ImageNet dataset. This approach, despite being resource-intensive, is anticipated to provide a robust foundation for the model, enhancing its ability to learn transferable and generalizable features.

VII. Conclusion

In this project, we delved into the SimCLR algorithm, a key player in the field of self-supervised learning, implementing it and comparing our results with those from the original paper. This endeavor not only enhanced our understanding of self-supervised learning but also brought to light the complexities involved in replicating advanced machine learning models.

While our implementation revealed certain limitations, particularly in data augmentation and parameter tuning, it provided invaluable insights and a strong foundation for future improvement. Our plan is to refine our approach, aligning more closely with the methodologies of the original study, especially in terms of pretraining on larger datasets like ImageNet.

In summary, this project was a significant learning journey, highlighting both the potential and challenges of self-supervised learning algorithms. We anticipate further advancements in our implementation, guided by the lessons learned from this experience.

VIII. Acknowledgement

We would like to express our profound gratitude to Professor Turkcan for his exceptional guidance throughout the Neural Networks and Deep Learning course. His expertise and dedication to fostering an encouraging academic environment have been a driving force behind this research project. Our sincere thanks also extend to Chen et al, for their groundbreaking work in contrastive learning of visual representations, which has significantly influenced and shaped the direction of our study. Lastly, we acknowledge the support and camaraderie of our peers, whose collaborative spirit has contributed to the success of this endeavor.

IX. References

References

[1] Chen, T., Kornblith, S., Norouzi, M. & Hinton, G. A Simple Framework for Contrastive Learning of Visual Representations. (2020)

X. Appendix

A. Individual Student Contributions in Fractions

- **UNI**: yy3152
- **Last Name**: Ye
- **Fraction of (useful) total contribution**: 1/3
- **What I did 1**: Write the code for the SimCLR model, and the jupyter notebook for training.
- **What I did 2**: Perform training and testing.
- **What I did 3**: Write the abstract, the results, the Future Work, and the Conclusion part of the report.
- **UNI**: zy2540
- **Last Name**: Yang
- **Fraction of (useful) total contribution**: 1/3
- **What I did 1**: Write parts of the code for the augmentation
- **What I did 2**: Write introduction, summary of the original paper part, Methodology part of the final report.
- **What I did 3**:Debug parts of the code
- **UNI**: hh2982
- **Last Name**: He
- **Fraction of (useful) total contribution**: 1/3
- **What I did 1**: Write parts of the code for the augmentation
- **What I did 2**: Manage github repo and write readme page.
- **What I did 3**: Write Implementation part of the final report.

B. Support Material

1) https://github.com/sthalles/SimCLR-tensorflow/tree/master
2) https://github.com/google-research/simclr