

SELF-SUPERVISED LEARNING

INTERN PROJECT TASK

The document contains Approaches used , Model Architectures, preprocessing steps, hyperparameters , Evaluation Results and Interpretation from the results

Prepared by:
Ishansh Sharma
Delhi Technological University
24/A12/050

AIM : To implement SSL methods for Visual Learning Models

One each for contrastive and MLM method and compare the results

What is Self Supervised Learning :

Self-supervised learning (SSL) is a machine learning technique where models learn from unlabeled data by generating their own supervisory signals from the data itself. Instead of relying on externally provided labels, SSL uses the data's inherent structure and relationships to create artificial labels, allowing the model to train on a task without explicit human guidance.

What are Vision Learning Models (VLMs) :

VLMs are models that learn to understand and relate images and text together. They use both visual and language inputs to perform tasks like captioning, visual QA, and retrieval.

VLMs use dual encoders (for image and text) or a unified encoder-decoder to jointly learn cross-modal representations. Core architectures often combine CNNs or Vision Transformers (ViTs) with Transformers for text.

Why is SSL needed in VLMs :

Self-supervised learning is needed for VLMs because we have massive amounts of unlabeled image and text data. It allows models to learn meaningful representations without relying on expensive manual annotations.

Different Methods of SSL :

Although various methods of SSL are available , some famous ones are as follows .

Masking (e.g., MAE, BERT)

Contrastive Learning (e.g., SimCLR, CLIP)

Predictive Learning (e.g., CPC)

Clustering-based (e.g., SwAV, DeepCluster)

Reconstruction (e.g., autoencoders)

Matching / Alignment (e.g., DINO, BYOL)

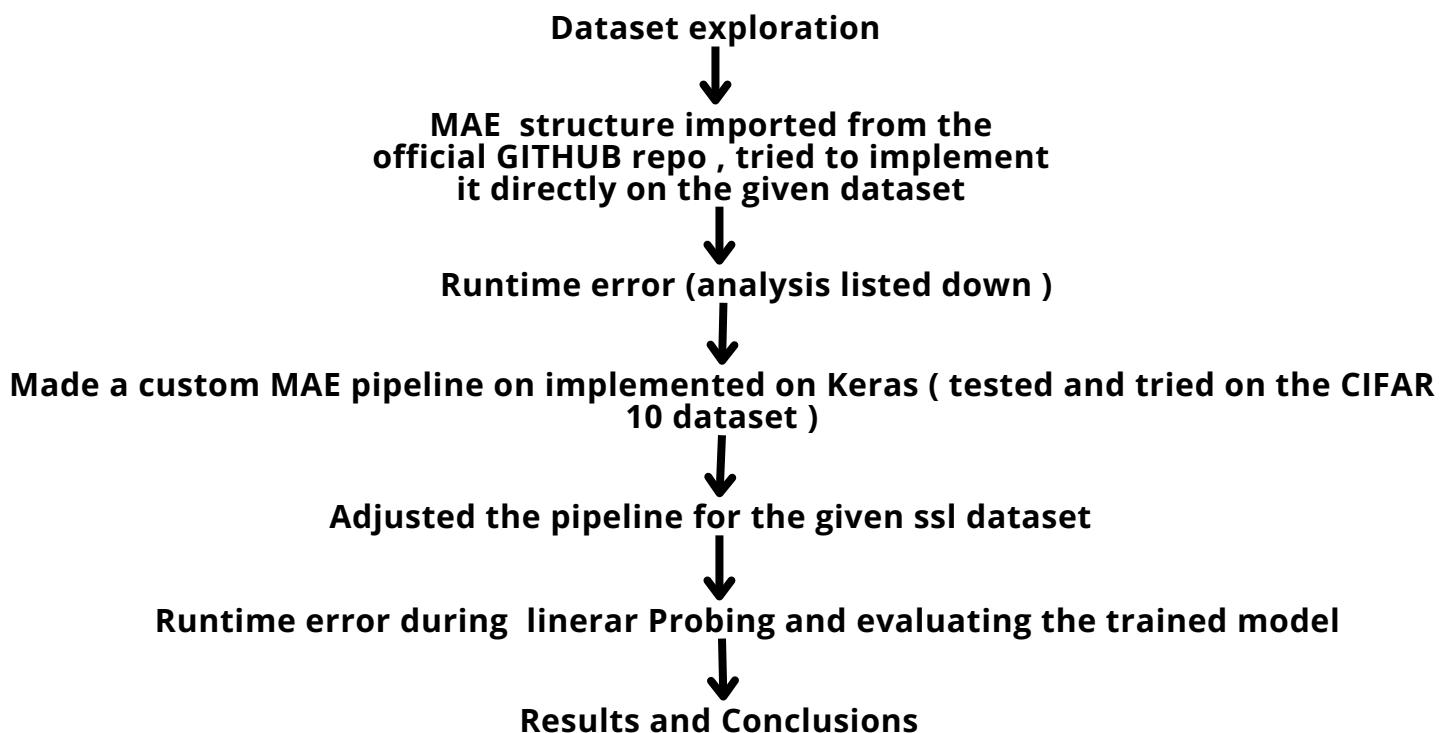
Approaches Used -

1. MAE on given dataset (MIM)
2. Barlow Twins method on given dataset (Matching/Alignment)
3. MAE on CIFAR 10 dataset (MIM)
4. NNCLR method on CIFAR dataset (Contrastive Learning)
5. SimSam method on CIFAR dataset (Contrastive Learning)

MAE on given dataset

Task done :

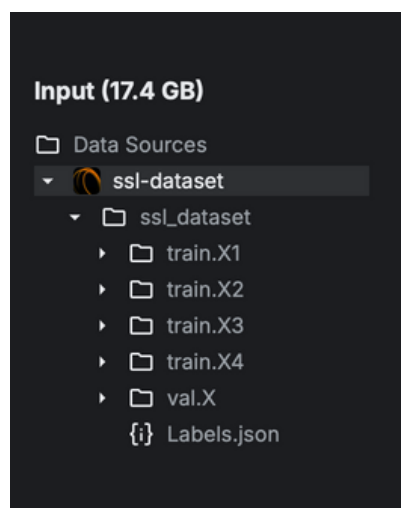
I tried to implement a Masked Autoencoder (MAE) on the provided SSL dataset for self-supervised pretraining. The model was trained using unlabelled data from multiple training splits (train.X1 to train.X4). After pretraining, I performed linear probing using a logistic regression classifier on extracted features from a separate validation set to evaluate the learned representations . However , it was not done without problems , I encountered various errors and bugs while trying to implement the above approach . The following flow chart describes the working



Details of all the steps are given below

Given Dataset Exploration

1. Data Access: The SSL dataset was initially downloaded from Google Drive and then uploaded to Kaggle for use in a notebook environment.
2. Dataset Structure: The dataset consists of a Labels.json file mapping class names to labels, along with image folders: train.X1 to train.X4 for training data and a valid folder for validation data. Each folder contains subdirectories named after class names, each holding the respective class images.
3. Validation Preparation: The valid folder, which was untouched during training, was used to create a labeled validation dataset for evaluation and linear probing.
4. Usage: Unlabeled training data was used to train a self-supervised model (MAE or Barlow Twins), and labeled validation data was used to assess feature quality via linear probing.



Model : Masked Autoencoders (MAE)

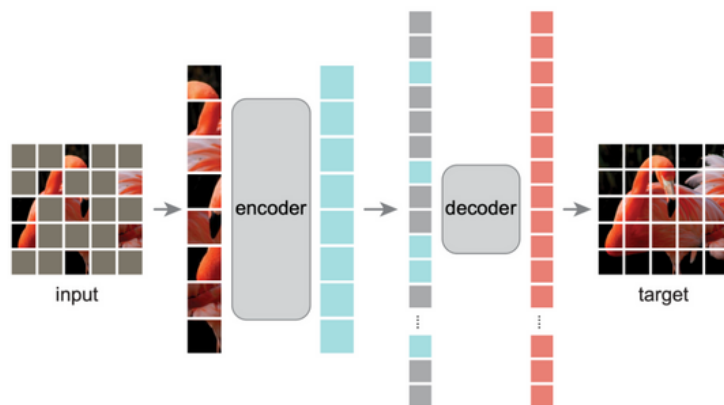
Masked autoencoders (MAE) are scalable self-supervised learners for computer vision.

They fall under the category of Masked Image Modelling Method for Self supervised Learning.

Masked Autoencoders (MAE) work by randomly masking a large portion of the input image and training a model to reconstruct the missing parts. The encoder processes only the visible patches, making it efficient, while the decoder tries to reconstruct the original image. This forces the model to learn meaningful representations of the image content. In self-supervised learning, MAE helps extract robust visual features without using labels.

Sub Approach 1 : MAE Implementation from the official Github Repo.

Architecture : The architecture is the same as described in the original paper on MAE (<https://arxiv.org/abs/2111.06377>) . Hence direct quote here “ Our MAE approach is simple: we mask random patches of the input image and reconstruct the missing pixels. It is based on two core designs. First, we develop an asymmetric encoder-decoder architecture, with an encoder that operates only on the visible subset of patches (without mask tokens), along with a lightweight decoder that reconstructs the original image from the latent representation and mask tokens. Second, we find that masking a high proportion of the input image, e.g., 75%, yields a nontrivial and meaningful self-supervisory task. ”

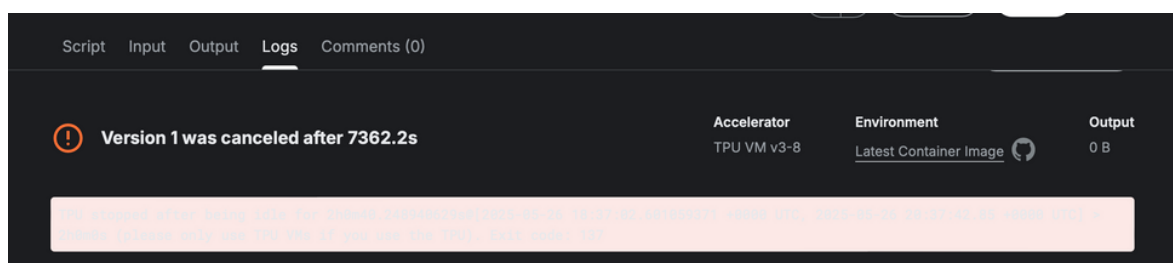


Approach : Directly Implemented the MAE. Imported the MAE model directly from the official [Facebook MAE GitHub repository](#). No preprocessing of the dataset was applied prior as it was not mentioned in the repo.

Problems faced : During training, a major issue encountered was that the browser tab kept closing, which repeatedly interrupted the training process on Kaggle. I experimented with various hyperparameters such as learning rate, batch size, and mask ratio, and searched through multiple discussions and official documentation for optimal MAE configurations

However, despite these efforts, the training consistently stopped midway due to session interruptions.

I tried to stop the training on the interactive sessions of the kaggle and tried to do it on the save version as well , however , it also stopped mid training .



While working with the official MAE model from Facebook's GitHub repo, I realized that I didn't have full visibility or control over the model's architecture. This limited my ability to tweak components or fully understand the inner workings of masked image modeling. To overcome this, I decided to build my own MAE pipeline from scratch using Keras and the CIFAR-10 dataset. This allowed me to design and experiment with the encoder, decoder, and masking logic myself. Considering the hands-on experience would not only deepened my understanding of self-supervised learning but also give me a stronger foundation for future research in this domain , I proceeded with this idea

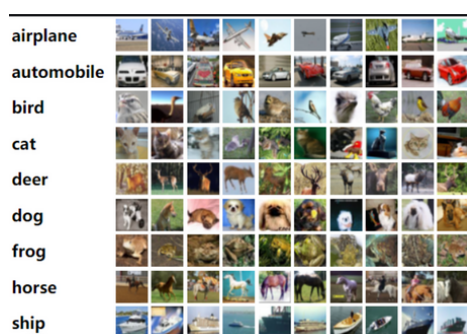
The whole approach is listed down , I have , then used the same pipeline on the given ssl dataset.

MAE pipeline implemented from Scratch

Dataset Used : CIFAR-10

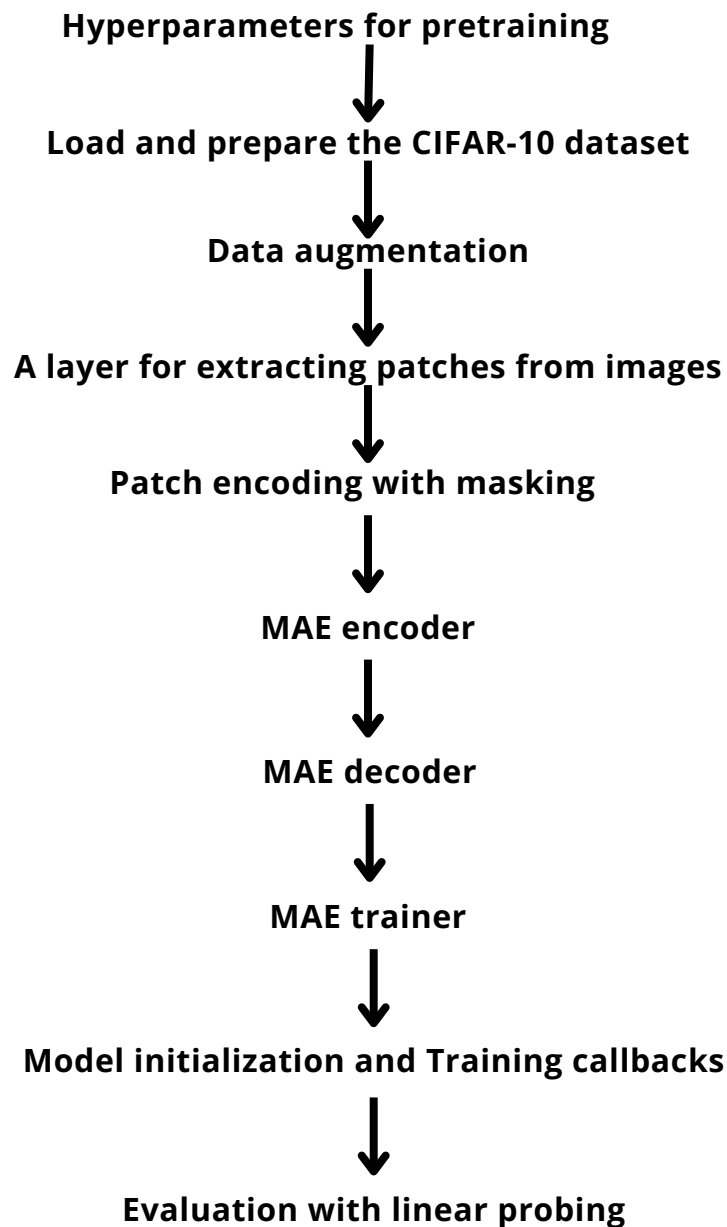
Advantage of using this dataset : Using the CIFAR-10 dataset for my custom MAE implementation offered several advantages. Being a lightweight and well-structured dataset, it was much easier to load and work with compared to the original 17GB SSL dataset, which often caused delays and resource issues. It was easy to load directly using Keras without needing to write any complex data loaders or deal with storage issues .This allowed me to focus more on building and experimenting with the core components of the MAE architecture—like the encoder, decoder, and masking strategies—rather than getting stuck on data handling. CIFAR-10 served as a great experimental ground.

Usage of dataset in pretraining and evaluation of Model : The CIFAR-10 dataset contains 60,000 color images of 10 different classes, each image sized 32x32 pixels. Since it is a labeled dataset, I discarded the labels during pre-training to focus on self-supervised learning, and later used the labels for evaluation through linear probing, which was straightforward to perform thanks to the available annotations.



Architecture of the pipeline for MAE

This flow chart represents the pipeline followed in the Approach (all the code for each of the steps is available on the github repo and hence , here I am presenting only the basic idea for each of the steps)



Hyperparameters : The hyperparameters have been set according to the original paper . for eg. masking is set to be 75% as discussed in the papers.

Different hyperparameters would present with different results , however , due to time constraints , I could not fine tune them , currently , as is advised to get the best results.

Data Augmentation : Data augmentation pipeline plays an important role in the SSL pipeline . However the authors of the paper point out that Masked Autoencoders do not rely on augmentations. They propose a simple augmentation pipeline of:

- Resizing
- Random cropping (fixed-sized or random sized)
- Random horizontal flipping

I have followed the same pipeline for the MAE approach .

A layer for extracting patches from images and path encoding with masking :

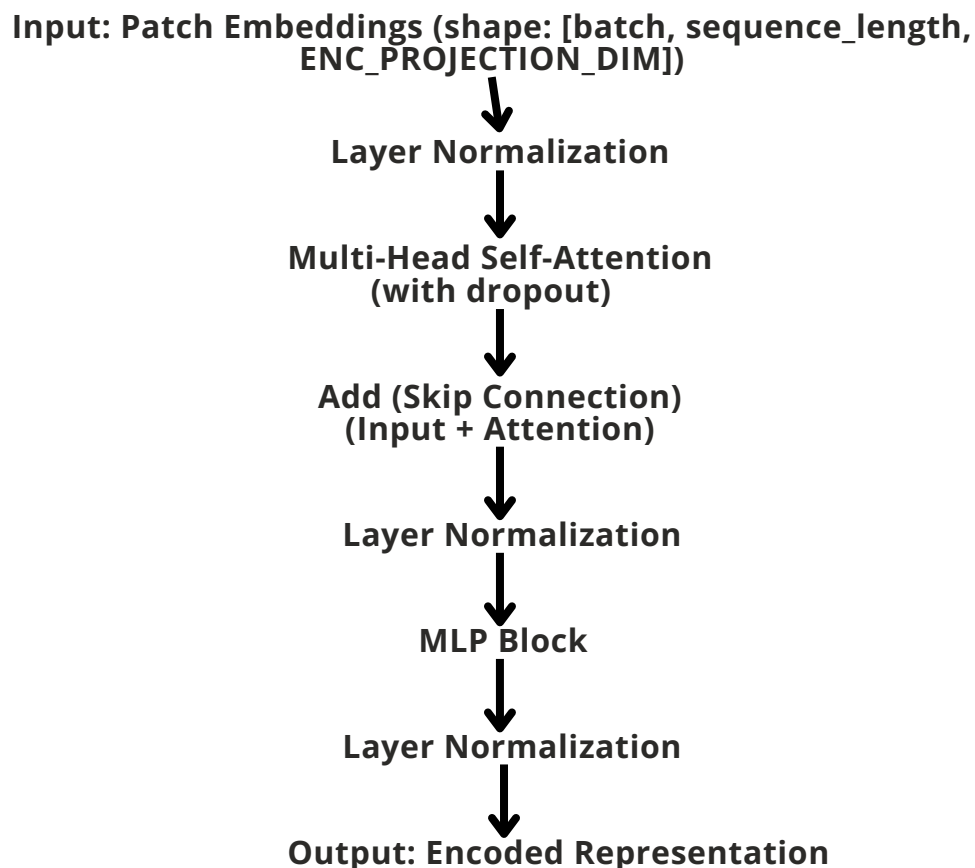
This layer accepts images as input and splits them into patches. It also provides two helper methods:

show_patched_image - accepts a batch of images along with their patches and displays a random image paired with its patches.

reconstruct_from_patch - takes a single set of patches and reassembles them back into the original image.

The path encoding method includes masking and encoding the patches..

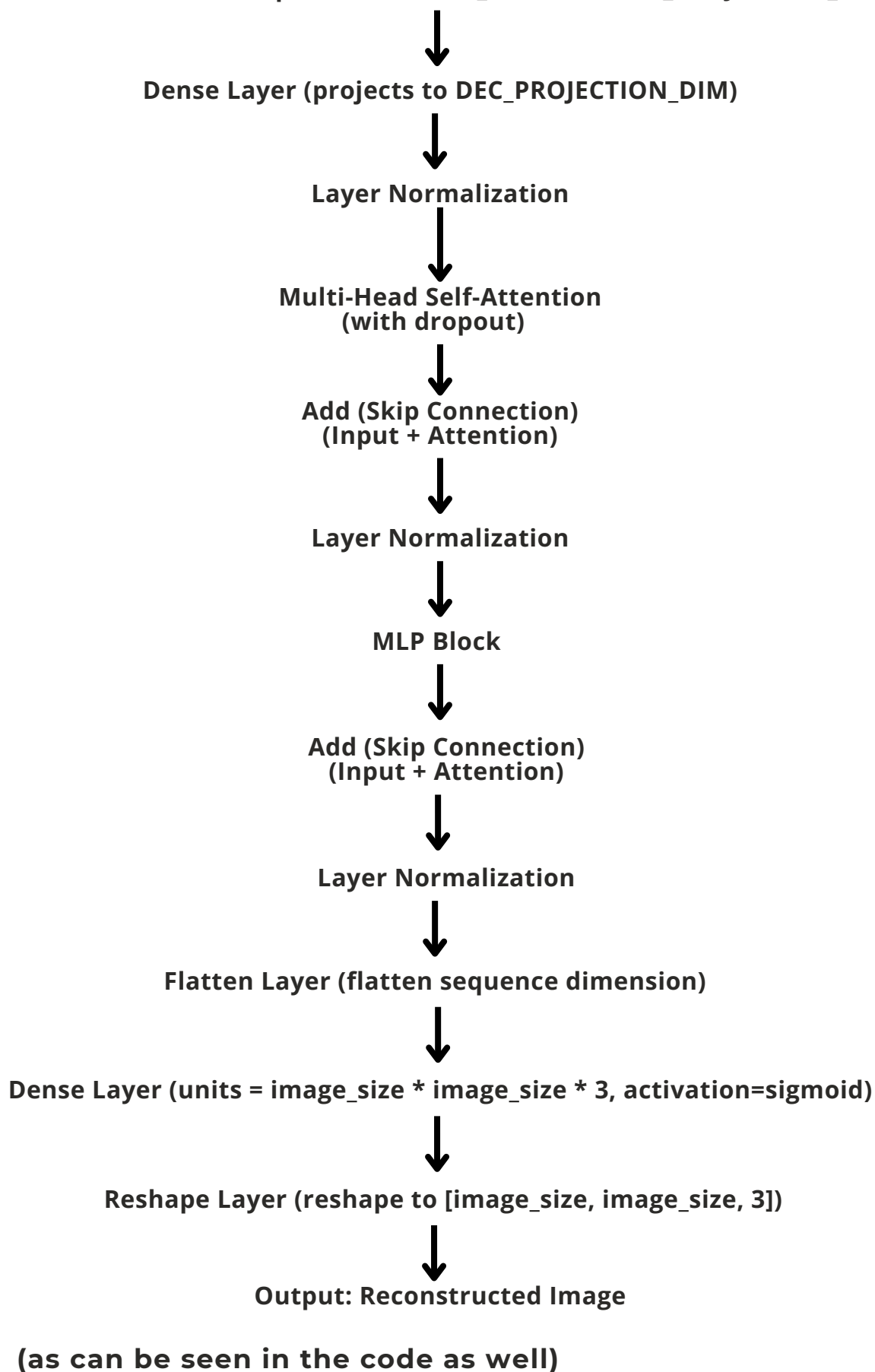
MAE encoder : Best explained by the diagram below-



(as can be seen in the code as well)

MAE decoder - Architecture is again best explained by the diagram

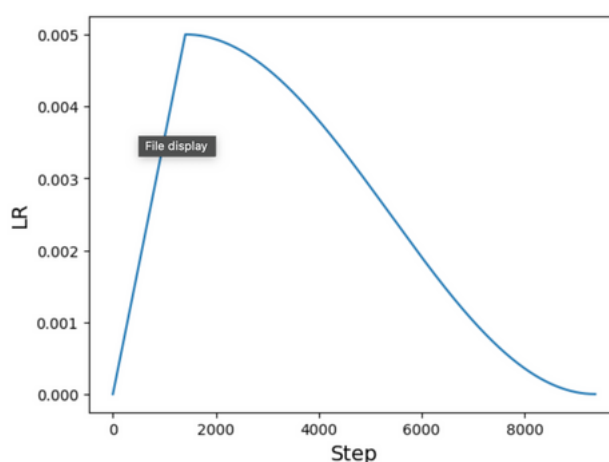
Input: Encoded Patches (shape: [batch, NUM_PATCHES, ENC_PROJECTION_DIM])



Comprehensive summary for the rest of model : The above encoder decoder architecture is used in the final MAE model . During traning , the number of epochs have been kept as 15 . !5 is not considered a very good very number of epochs for models involving vision tasks , however , owing to the time restrictions to try out more models and to get a idea of what the model is doing I have kept he epochs at only 15 here . In future I would increase it to around 100 to get the best results.

For Learning rate a **Wamup Cosine LR scheduler** has been implemented to help the model start training gently with a low learning rate, preventing unstable updates early on . However i started with a already comparitively low LR , since LR scheduler is installed , therefore I could possibly start witha higher LR to get better results.

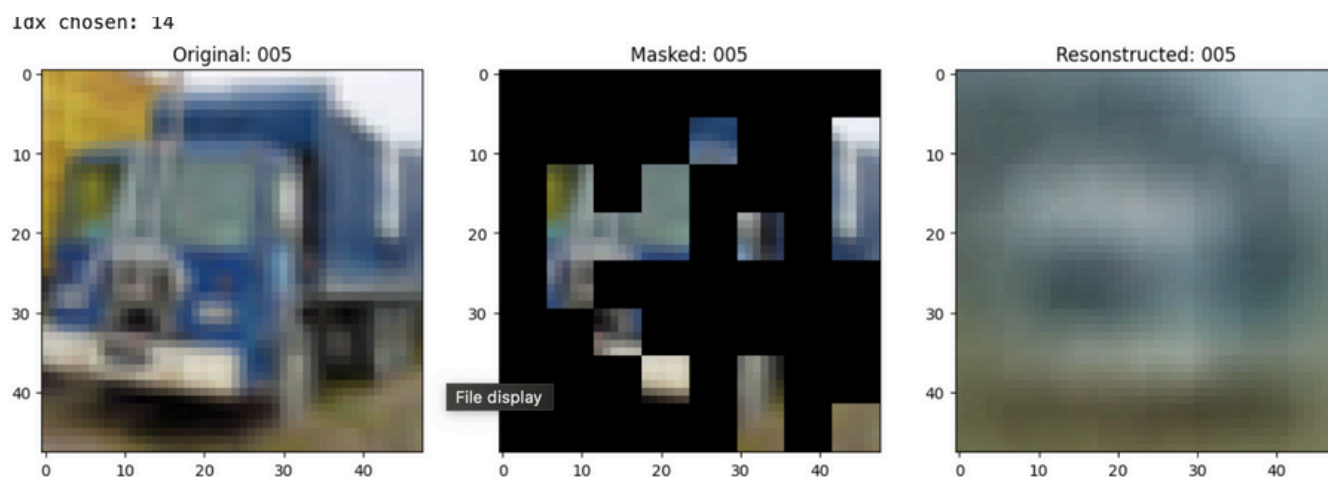
LR vs step graph is as below



Results after training the Model : Since i have already describe the model architecture , therefore , I would directly like to jump on the results achieved after training the epochs .

loss = 0.5662

mae = 0.0944



Linear Probe Model : The specifications are as follows :

Model: "linear_probe_model"

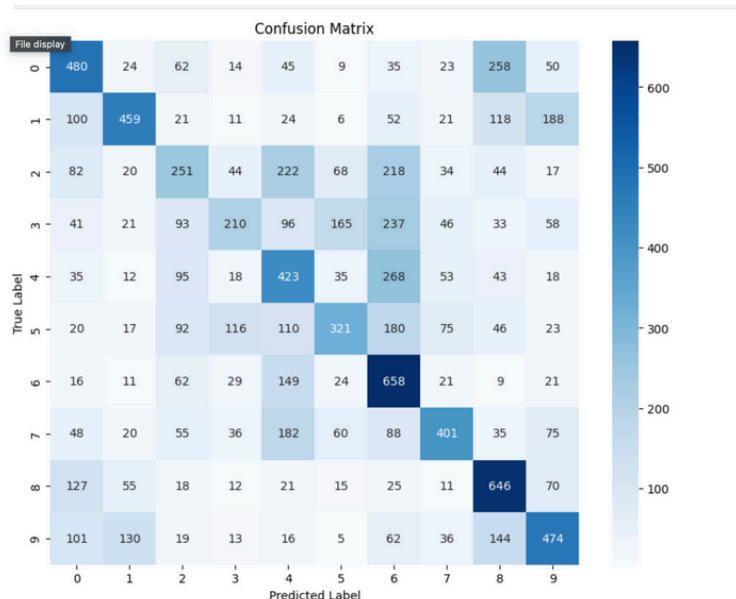
Layer (type)	Output Shape	Param #
patches_1 (Patches)	(None, 64, 108)	0
patch_encoder_1 (PatchEncoder)	(None, 64, 128)	22,144
mae_encoder (Functional)	(None, 64, 128)	1,981,696
batch_normalization (BatchNormalization)	(None, 64, 128)	512
global_average_pooling1d (GlobalAveragePooling1D)	(None, 128)	0
dense_20 (Dense)	(None, 10)	1,290

Total params: 2,005,642 (7.65 MB)

Trainable params: 1,290 (5.04 KB)

Non-trainable params: 2,004,352 (7.65 MB)

The Evaluation Result is as follows after the Evaluation :



Accuracy on the test set: 43.23%.

Classification Report:

	precision	recall	f1-score	support
0	0.4571	0.4800	0.4683	1000
1	0.5969	0.4590	0.5189	1000
2	0.3268	0.2510	0.2839	1000
3	0.4175	0.2100	0.2794	1000
4	0.3284	0.4230	0.3698	1000
5	0.4534	0.3210	0.3759	1000
6	0.3609	0.6580	0.4662	1000
7	0.5562	0.4010	0.4660	1000
8	0.4695	0.6460	0.5438	1000
9	0.4769	0.4740	0.4754	1000
accuracy			0.4323	10000
macro avg	0.4444	0.4323	0.4248	10000
weighted avg	0.4444	0.4323	0.4248	10000

More epochs and hyperparameter training should increase the accuracy and overall scores

MAE on Given Dataset

Sub Approach 2 :

MAE Implementation using the above mentioned pipeline using given dataset.

Architecture : The architecture is the same as described above hence would not write again.

Here i have just changed the dataset to the given ssl-dataset.

I have highlighted the points where there were some changes required due to various errors that occurred and why they were needed.

Hyperparameters : The hyperparameters were changed here than from the previous implementation , this was done as the ssl dataset is quite different in size and also for experimentation on various hyperparameters .

BATCH_SIZE = 256

changed to

BATCH_SIZE = 64

(It was observed in the training of subapproach 1 that a lower batch size helped in longer running of the model , hence to deal with the memory loss issue I trianed with sizes 128 and 64 (still got memory error) , however , in bs = 64 , the model did train completely .

And its weight could later be used in the linear probe evaluation (

This I did not think of , only later during training the barlowtwin model did I save it , after getting the feedback)

NUM_CLASSES = 10

(I took only 10% of the ssl-dataset , ie 13000 images , so did not change it)

NUM_PATCHES = (IMAGE_SIZE // PATCH_SIZE) ** 2

to

NUM_PATCHES = 9

(there was some sort of dimensional error occurring after applying in the decoder step , fixing num_ppatches fixed it to not throw the error)

(however , doing so changed my MASK_PROPORTION , which I believe did tamper with the final results , looking to some other soultuon for this problem instead of fixing the num_patches.)

DATA : Took only 10% of the ssl-dataset , split into train,test and valid

The results are as given below :

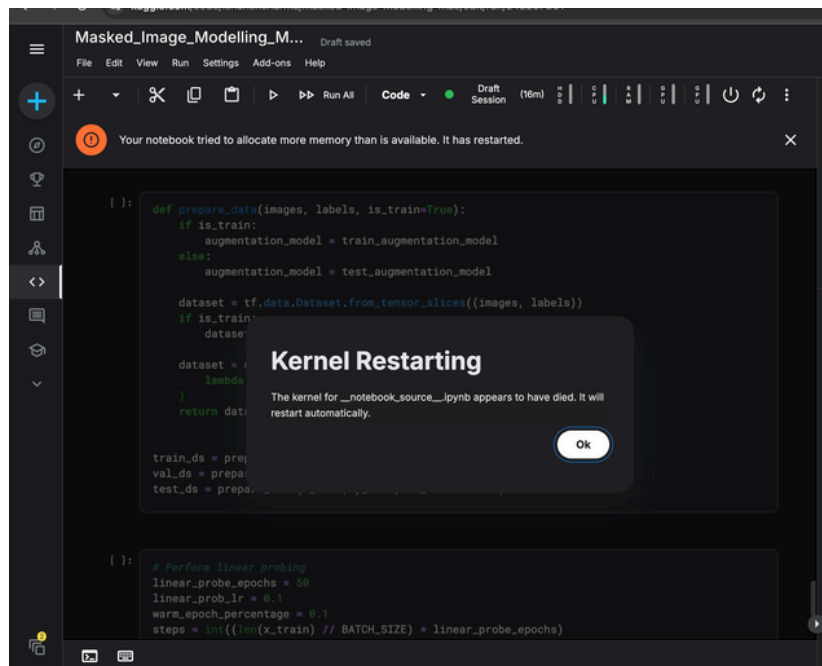
mae: 0.0972

loss: 0.0017

The evaluation results are not available here , as I encountered a memory error again , however , since the model did train , next time I could just save the weights and then use them in the linear probe.

One interesting thing to note is that , the chosen ssl dataset had less number of images than the CIFAR - 10 dataset , still I got the error here , which is strange to me .

I experimented with data loader in the next implementation of Barlow twin method on the same dataset.



After this , I decided to implement the contrastive learning method on the CIFAR-10 dataset only , and implemented two methods (which I talk about in the following document) .

However , I researched a little bit about other methods of SSL for VLMs and came across a few that I wanted to implement and test on the ssl-dataset .

The method is Barlow Twins method .

I proceed to explain everything about it in the following section , as we are already talking about the implementation on this dataset .

Barlow Twins method on given dataset

About the Barlow Twins method and why was it selected :

Introduced by Zbontar et al., the key idea behind Barlow Twins is to reduce redundancy between the learned features of two distorted views of the same image while maintaining invariance.

Barlow Twins applies redundancy-reduction — a principle first proposed in neuroscience — to self supervised learning

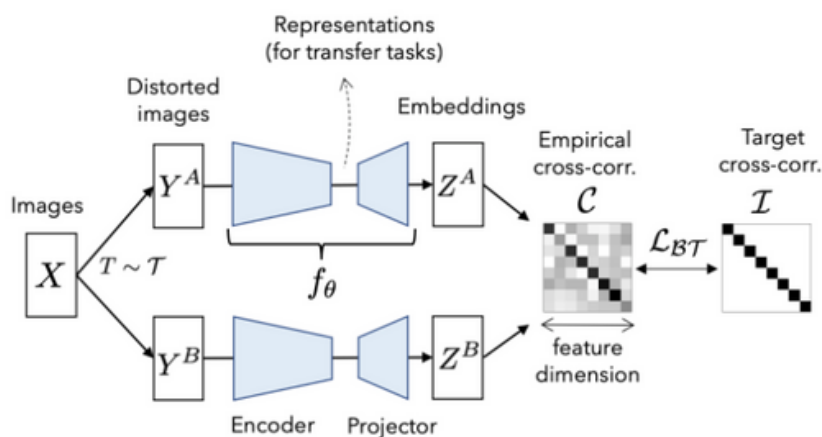
I chose this method because unlike contrastive methods like SimCLR or MoCo, it avoids the need for negative samples by reducing feature redundancy, allowing for stable training with smaller batch sizes—useful in resource-constrained environments like Kaggle.

Research has shown that Barlow Twins often outperforms other methods in downstream tasks when evaluated via linear probing , thus I thought that it may produce better results.

“Barlow Twins outperforms previous methods on ImageNet for semi-supervised classification in the low-data regime, and is on par with current state of the art for ImageNet classification with a linear classifier head, and for transfer tasks of classification and object detection.” - quoted from the paper linked below on the model.

Architecture :

Proposed architecture in the original paper ([link](#))

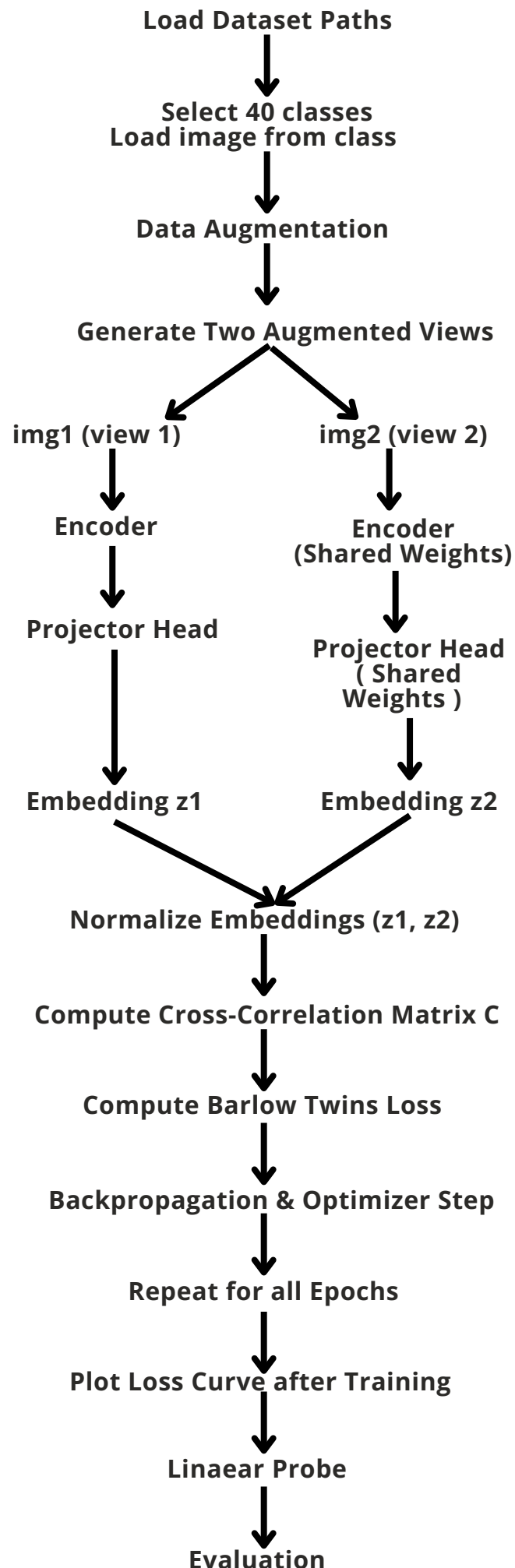


Backbone used - ResNet-50

Projection head is a 3 layer MLP(linear batch , normalisation and ReLU)

The architecture used in the code by me is not very different from the architecture proposed , just lighter , with a few changes (mainly to make it more computationally effective) . Hence , I would not describe them two different times , I will describe the architecture used by me in the details below.

Exact step by step architecture followed in my implementation of Barlow Twins method



Select Classes : Selected only 40 Classes from the given dataset containing 100 classes as I wanted to keep the training lightweight . The images for training were taken from the training folders and the images for the validation were taken from the validation folder , the labels from json file of labels.

The first 40 classes in the json file were selected.

Data Augmentation : The data augmentation pipeline is aligned with the one that was proposed in the original paper with only the minor change that it did not include the gaussian blur. However , I have come to the conclusion that it was not a good idea to exclude it as , the model seems to doing some kind of shortcut learning.

The rest of the pipeline is as follows :

RandomResizedCrop(224)
RandomHorizontalFlip()
ColorJitter
RandomGrayscale(p=0.2){here p is the probability of it being applied on the image , necessary to obtain two different images 1 and 2 }
Normalize

This augmentation step is essential to generate two distinct yet semantically similar views of the same image, encouraging the model to learn invariant representations.

More experiment with the augmentation step , I believe , should be done , as then only we can know if the model is receptive to some particular augmented images and performs better .

Encoder : The encoder used as the backbone here is **ResNet-18** . In the original paper they used the **ResNet-50**. However , I have used 18 due to computational efficiency , would recommend using 50 though .

One more important thing to note here is that I trained this Barlow Twin model two times , in the first time the weights of the ResNet 18 model were kept as "None" , while in the 2nd training they were kept as those weights from the ImageNet model .

Projector : A two layer MLP was used instead of the original 3 layer MLP in the paper.

Also , projector expands features to high-dim space (8192)
512→8192→8192

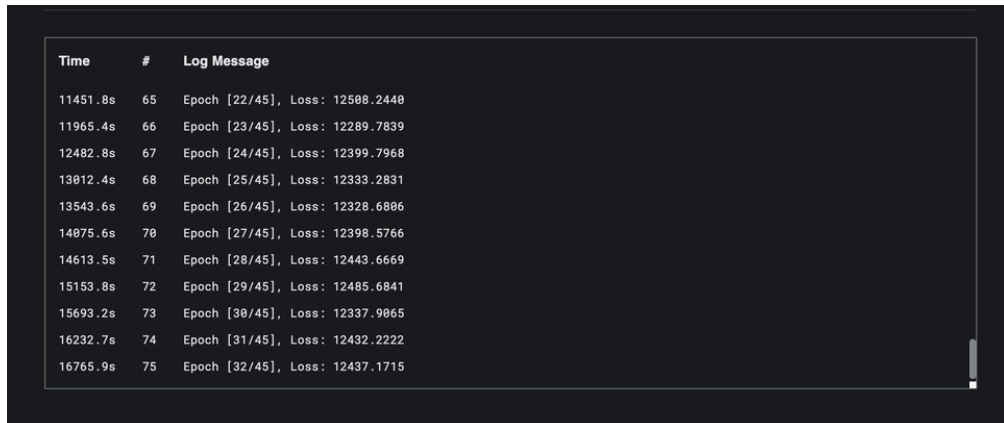
Barlow Twins Loss Function : It is used to implement the Barlow Twins objective:

Make representations invariant to augmentations (diagonal terms $\rightarrow 1$)

Reduce redundancy between features (off-diagonal terms $\rightarrow 0$)

Training Loop : I have used Adam optimizer with learning rate $1e-3$

However , I would also like to apply momentum as during the training of the model for the 2nd time , I felt that the gradient was stuck in a local minima . I would like to present the ss here for the same .



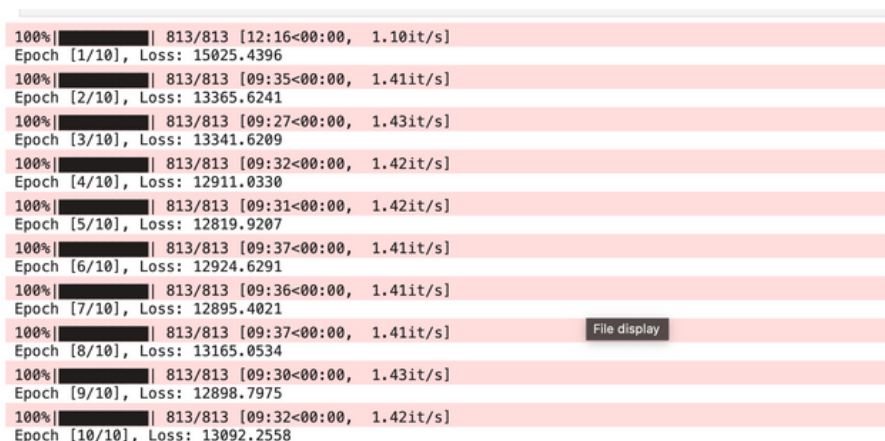
Time	#	Log Message
11451.8s	65	Epoch [22/45], Loss: 12508.2440
11965.4s	66	Epoch [23/45], Loss: 12289.7839
12482.8s	67	Epoch [24/45], Loss: 12399.7968
13012.4s	68	Epoch [25/45], Loss: 12333.2831
13543.6s	69	Epoch [26/45], Loss: 12328.6806
14075.6s	70	Epoch [27/45], Loss: 12398.5766
14613.5s	71	Epoch [28/45], Loss: 12443.6669
15153.8s	72	Epoch [29/45], Loss: 12485.6841
15693.2s	73	Epoch [30/45], Loss: 12337.9065
16232.7s	74	Epoch [31/45], Loss: 12432.2222
16765.9s	75	Epoch [32/45], Loss: 12437.1715

I trained for only 10 epochs (big mistake) in my 1st training , I feel this is the core reason for the bad performance more than anything else , as in the original paper they trained for 1000 epochs.

And also the model is not able to learn much in the 10 epochs.

In my 2nd training I ran for 100 epochs (which I by mistake closed the interactive session after 21 epochs) , but due to time and memory constraints I had to reduce it to 45 .

The results there were not great as well , though certainly better than the 1st training loop .

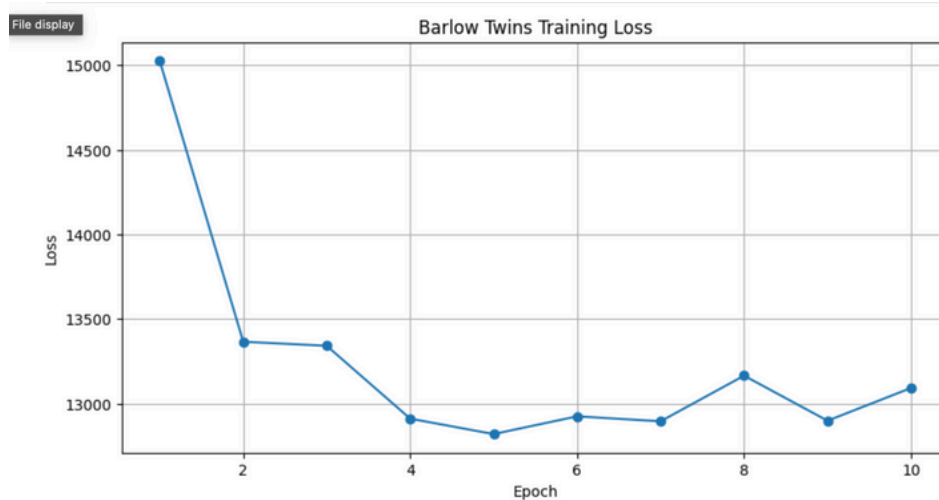


100% ██████████ 813/813 [12:16<00:00, 1.10it/s]
Epoch [1/10], Loss: 15025.4396
100% ██████████ 813/813 [09:35<00:00, 1.41it/s]
Epoch [2/10], Loss: 13365.6241
100% ██████████ 813/813 [09:27<00:00, 1.43it/s]
Epoch [3/10], Loss: 13341.6209
100% ██████████ 813/813 [09:32<00:00, 1.42it/s]
Epoch [4/10], Loss: 12911.0330
100% ██████████ 813/813 [09:31<00:00, 1.42it/s]
Epoch [5/10], Loss: 12819.9207
100% ██████████ 813/813 [09:37<00:00, 1.41it/s]
Epoch [6/10], Loss: 12924.6291
100% ██████████ 813/813 [09:36<00:00, 1.41it/s]
Epoch [7/10], Loss: 12895.4021
100% ██████████ 813/813 [09:37<00:00, 1.41it/s]
Epoch [8/10], Loss: 13165.0534
100% ██████████ 813/813 [09:30<00:00, 1.43it/s]
Epoch [9/10], Loss: 12898.7975
100% ██████████ 813/813 [09:32<00:00, 1.42it/s]
Epoch [10/10], Loss: 13092.2558

As you can see that it is taking around 9.5 minutes for each epoch , larger training becomes difficult.

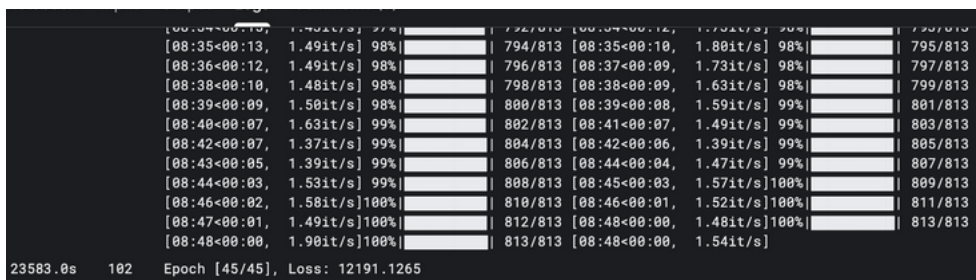
Results : Now since I have already talked about the input and the architecture of the model I would like to present with some results and their interpretation and methods to improve the model to get better results .

Loss VS the epochs for 1st training (10 epochs)



Final Loss for 1st training : 13092

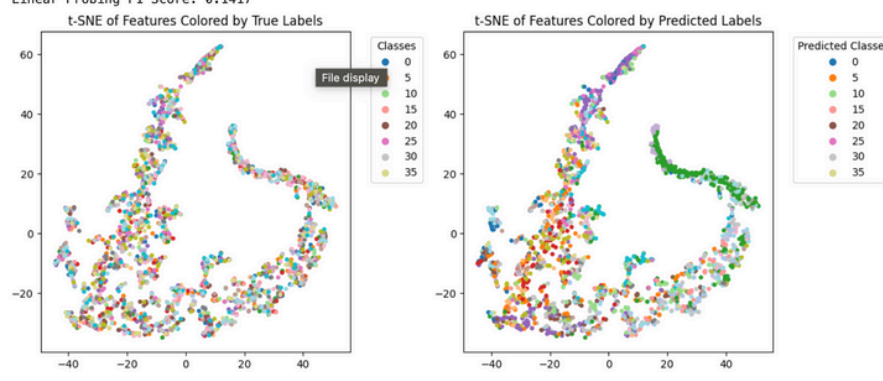
Final Loss for 2nd training (save version and run , did not find how to get the graphs) : 12191



Results after evaluation after Linear probing (These results are available only for training 1 , as there was some issue in loading the validation dataset to evaluate and hence error was thrown , however , I do have the model weights for the 2nd training and will use that to compare and evaluate later.

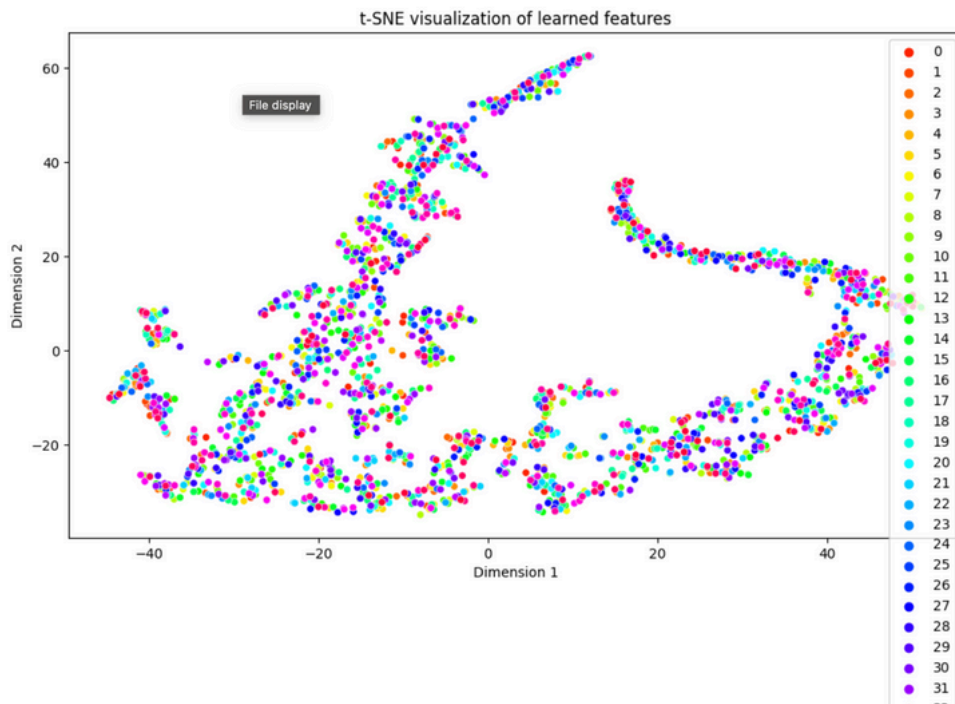
```
100%|██████████| 32/32 [00:00<00:00, 3.60it/s]
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Linear Probing Accuracy: 0.1515
Linear Probing F1 Score: 0.1417
```



Accuracy : 0.1515

F1 score : 0.1417



Proposed reasons for low scores and how to improve them :

Architecture :

Upgrade to ResNet-50 from ResNet-18 as this should provide for better feature extractions.

Add 3rd projector layer (512→8192→8192→8192)

Training:

Increase batch size , if the memory constraints allow .

Switch to LARS optimizer as the original paper had used this optimiser.

Implement stopping rounds , very necessary , slipped my mind when I was training the model .

Increase the number of epochs as it has been proven that it has decreased the loss.

Other :

We should augment the images more strongly as mentioned above as well.

Contrastive Learning Methods on CIFAR - 10 dataset.

The task had originally required to implement one contrastive learning method , I have however implemented 2 of them

SimSam
NNCLR

Contrastive Learning Methods for SSL : Self-supervised learning (SSL) using contrastive learning is a technique where a model learns to distinguish between similar and dissimilar data points without needing labeled data. The core idea is to create positive pairs (different augmented views of the same data sample) and negative pairs (views from different samples). The model is trained to map positive pairs closer together in the embedding space, while pushing negative pairs further apart

SimSam on CIFAR 10 dataset

SimSam Model and why I chose it :

I worked on CIFAR 10 dataset as I was able to implement and get results on this dataset for MAE method of MIM and to compare the two (contrastive learning method and Masked Image Modelling method) , I thought I should keep the dataset same.

About SimSam : The SimSiam method, introduced by Chen and He (2021), is a simple yet effective approach to self-supervised learning that eliminates the need for negative samples, large batch sizes, or momentum encoders—key components in earlier contrastive learning methods like SimCLR and BYOL.

SimSiam uses a Siamese network architecture with a stop-gradient operation to prevent collapsing solutions, allowing it to learn meaningful representations from positive pairs alone

Empirical results show that SimSiam achieves competitive or superior performance compared to SimCLR, especially in scenarios with shorter pre-training (e.g., under 100 epochs), and transfers well to downstream tasks such as object detection and segmentation([link](#)). Its simplicity leads to reduced computational overhead and architectural complexity while maintaining high accuracy, making it particularly attractive for practical applications

As is very clear by the above statements , this model seemed perfect for our case scenario , it has low computational cost due to no need for negative Sampling and better results for less number of epochs.

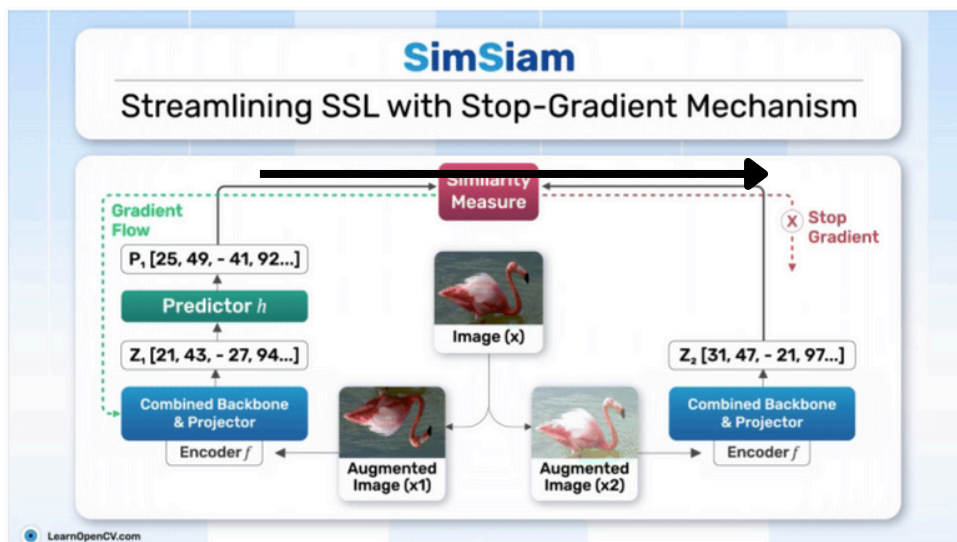
Architecture

The SimSiam architecture operates in three main steps:

Dual-Branch Processing: Two augmented versions of the same image are generated and passed through a shared encoder network (typically a ResNet backbone with a projection MLP), producing two feature representations.

Prediction and Stop-Gradient: One branch includes a prediction MLP that maps its feature to a prediction space, while the other branch applies a stop-gradient operation, preventing gradients from flowing backward and stabilizing training.

Similarity Optimization: The model is trained to maximize the similarity (typically using negative cosine similarity) between the predicted feature from one branch and the projected feature from the other, ensuring the representations align without collapsing to trivial solutions.



Used Pipeline

Load the CIFAR-10 dataset and define hyperparameters

data augmentation

Convert the data into TensorFlow Dataset objects

Defining the encoder and the predictor

Pre-training our networks

Evaluating our SSL method

Load and define the hyperparameters :

Total training examples: 50000

Total test examples: 10000

Data Augmentation : The augmentation pipeline is very important in a SSL task , it is usually observed that certain operations such as random crop are applied in almost all the other contrastive learning methods as well , therefore some of the augmentations has been similar.

Augmentation becomes crucial in SimSiam and similar self-supervised learning (SSL) methods because the core idea of these methods is to learn representations by comparing different views of the same image.

The following augmentations have been performed :

Random Flip (horizontal and vertical)

Random Crop

Random Brightness

Random Contrast

Random Saturation

Random Hue

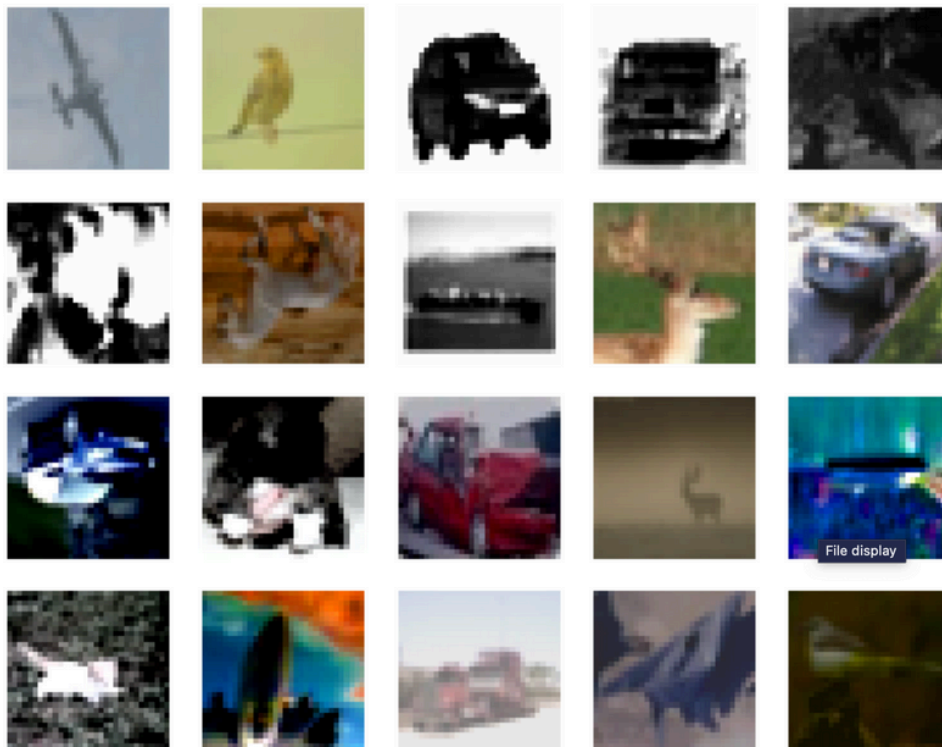
Grayscale Conversion

color_jitter is applied with 80% probability.

color_drop (grayscale) is applied with 20% probability.

Good augmentations help the model focus on semantically meaningful features.

Visualisation of the augmentations in the dataset



The different versions of the augmented dataset containing identical images are then zipped together so as to pass through the encoder predictor model.

Encoder : A ResNet-based architecture (resnet_cifar10_v2) is used as the feature extractor .

Images are rescaled from $[0, 255]$ to $[-1, 1]$

Global average pooling is applied to flatten the output feature maps.

A A 2-layer MLP (Multi-Layer Perceptron) acts as the projection head

This outputs the projected representation z .

Predictor : A shallow 2-layer MLP

Dense \rightarrow ReLU \rightarrow BatchNorm \rightarrow Dense

Loss Function : Negative Cosine Similarity Loss

SimSiam tries to maximize the agreement between the predicted vector p_1 and the projection z_2 of another augmented view (and vice versa).

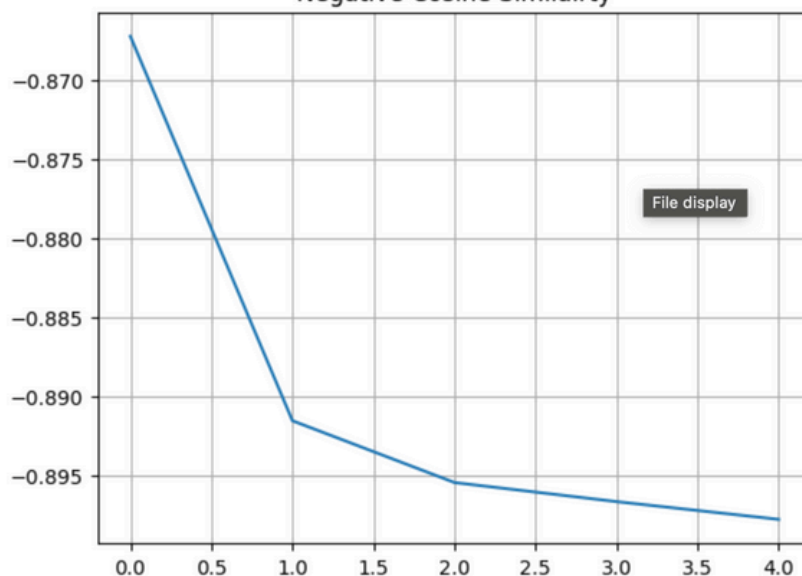
$\text{stop_gradient}(z)$ is used to prevent gradients from flowing into the branch that produces z , which helps avoid collapse.

Training : The traning uses a Cosine Decay Scheduler and Early Stopping monitors the loss and stops training if no improvement is seen for 5 consecutive epochs.

Results : presenting with the relevant scores and results

Epoch 1/5	
391/391	784s 2s/step - loss: -0.8179
Epoch 2/5	
391/391	768s 2s/step - loss: -0.8899
Epoch 3/5	
391/391	768s 2s/step - loss: -0.8948
Epoch 4/5	
391/391	794s 2s/step - loss: -0.8965
Epoch 5/5	
391/391	803s 2s/step - loss: -0.8979

Negative Cosine Simailrty



After training and evaluation on the linear classifier

Test accuracy: 22.86%

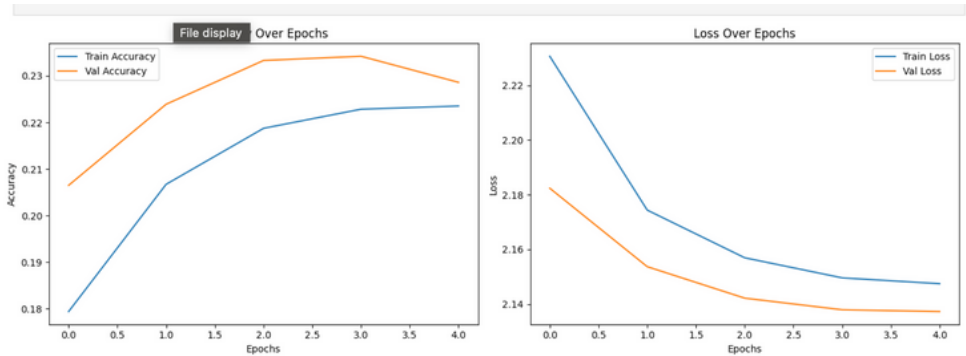
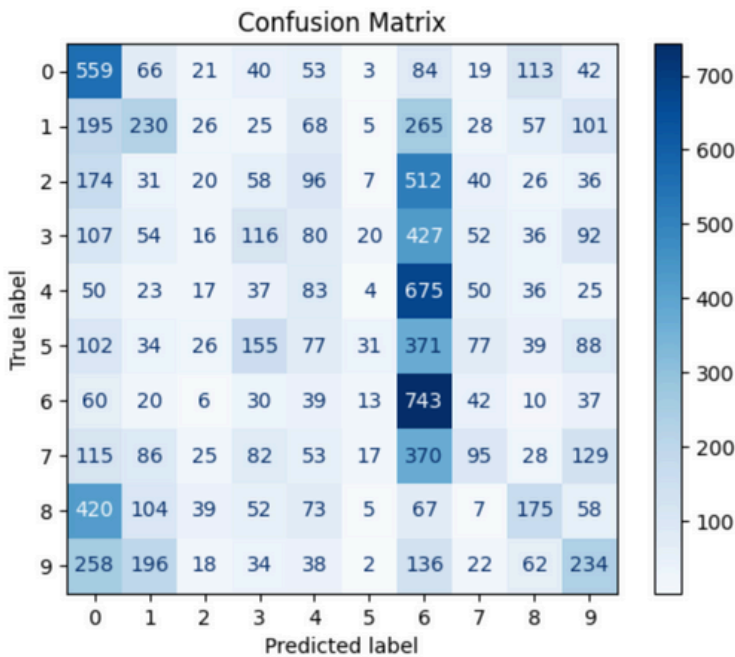
f1-score: 0.1876

precision : 0.2242

recall : 0.2286

	precision	recall	f1-score	support
0	0.2740	0.5590	0.3678	1000
1	0.2725	0.2300	0.2495	1000
2	0.0935	0.0200	0.0329	1000
3	0.1844	0.1160	0.1424	1000
4	0.1258	0.0830	0.1000	1000
5	0.2897	0.0310	0.0560	1000
6	0.2036	0.7430	0.3196	1000
7	0.2199	0.0950	0.1327	1000
8	0.3007	0.1750	0.2212	1000
9	0.2779	0.2340	0.2541	1000
accuracy			0.2286	10000
macro avg	0.2242	0.2286	0.1876	10000
weighted avg	0.2242	0.2286	0.1876	10000

File display



The loss and accuracy graphs vs the epochs suggest that more number of epochs would certainly help the model to perform better .

Also in the case of SimSam specific architecture , a larger training data (unlabelled of course) greatly benefits the results as is proposed in researches.

One can also observe that the MAE method performed well in the case of CIFAR 10 dataset as compared to the Simsam contrastive learning method.

The NNCLR implementation is not done properly as of now by me , however I have uploaded the file in the github , still working on it .

Now from all the scores that have been generated we can observe that the scores are very low , to use the SSL in real life to train the VLMs we will need more scores and better results and hence I am looking forward to this opportunity where I can contribute to improving SSL frameworks and help bridge the gap between theoretical potential and practical performance.