# Machine Learning for Science - DeepLens

## Self-Supervised Learning for Strong Gravitational Lensing

Project Proposal for Google Summer of Code 2023

## Personal Information

Display Name: yaashwardhan
Name: Yashwardhan Deshmukh
Email: yaashwardhan@gmail.com
Github: github.com/yaashwardhan
Country of Residence: India (UTC + 5:30)
Resume: CV_Yashwardhan_Deshmukh.pdf

## Project Title

Title: Self-Supervised Learning for Strong Gravitational Lensing
Mentors: Michael Toomey, Pranath Reddy, Sergei Gleyzer, Emanuele Usai, Saranga Mahanta and Karthik Sachdev
Number of Tests Completed: **7** (GitHub)

## Project Summary

Deep learning techniques effectively use their feature space to extract meaningful latent variables from supervised lensing data [(1)]. The amount of this data that is produced is staggering and unprecedented and manually labeling it would be exhausting and unsustainable. Hence a follow-up, unbiased dark matter analysis with a variational autoencoder approach was used in [(2)] which showed success towards possibilities beyond existing theories by capturing the key morphological features of dark matter substructures in the latent space in an unsupervised manner. However, an innovative unsupervised approach named self-supervised learning is used to learn representations from data by generating its own supervisory signals.

In a recent paper, the authors Stein et.al [(3)] applied self-supervised contrastive learning with a convolutional neural network (CNN) on simulated strong gravitational lens images to learn representations for identifying lenses. The results showed that the self-supervised approach significantly outperformed traditional supervised methods. However, instead of using CNNs, a more modern approach using a Hybrid-Transformer model promises a further improvement in the results for representational learning but it has not been addressed by the community yet.

Hence in this project, I will implement a custom hybrid transformer architecture that can be used for representational learning and then finetune it for specific tasks (such as regression and classification). I will then also construct equivariant transformers for the same to efficiently learn spatial hierarchies and exploit symmetries in the input data to improve generalization.

[1] Alexander, S., Gleyzer, S., McDonough, E., Toomey, M. W., & Usai, E. (2020). Deep Learning the Morphology of Dark Matter Substructure. The Astrophysical Journal, 893(1), 15. doi:10.3847/1538-4357/ab7925

[2] Alexander, S., Gleyzer, S., Parul, H., Reddy, P., Toomey, M. W., Usai, E., & Von Klar, R. (2020). Decoding Dark Matter Substructure without Supervision. ArXiv. /abs/2008.12731

[3] Stein, G., Blaum, J., Harrington, P., Medan, T., & Lukić, Z. (2022). Mining for Strong Gravitational Lenses with Self-supervised Learning. The Astrophysical Journal, 932(2), 107. doi:10.3847/1538-4357/ac6d63

This hybrid architecture will consist of twin network blocks having the same architecture and weights but will be trained on two different augmentations of the input image to learn meaningful broader representations through unsupervised means (self-supervised representational learning), while also being robust to noise.

The Twin architecture mainly consists of the Swin and ViT transformers that work together to capture local as well as global attention followed by a finetuning of choice. A residual is optional.

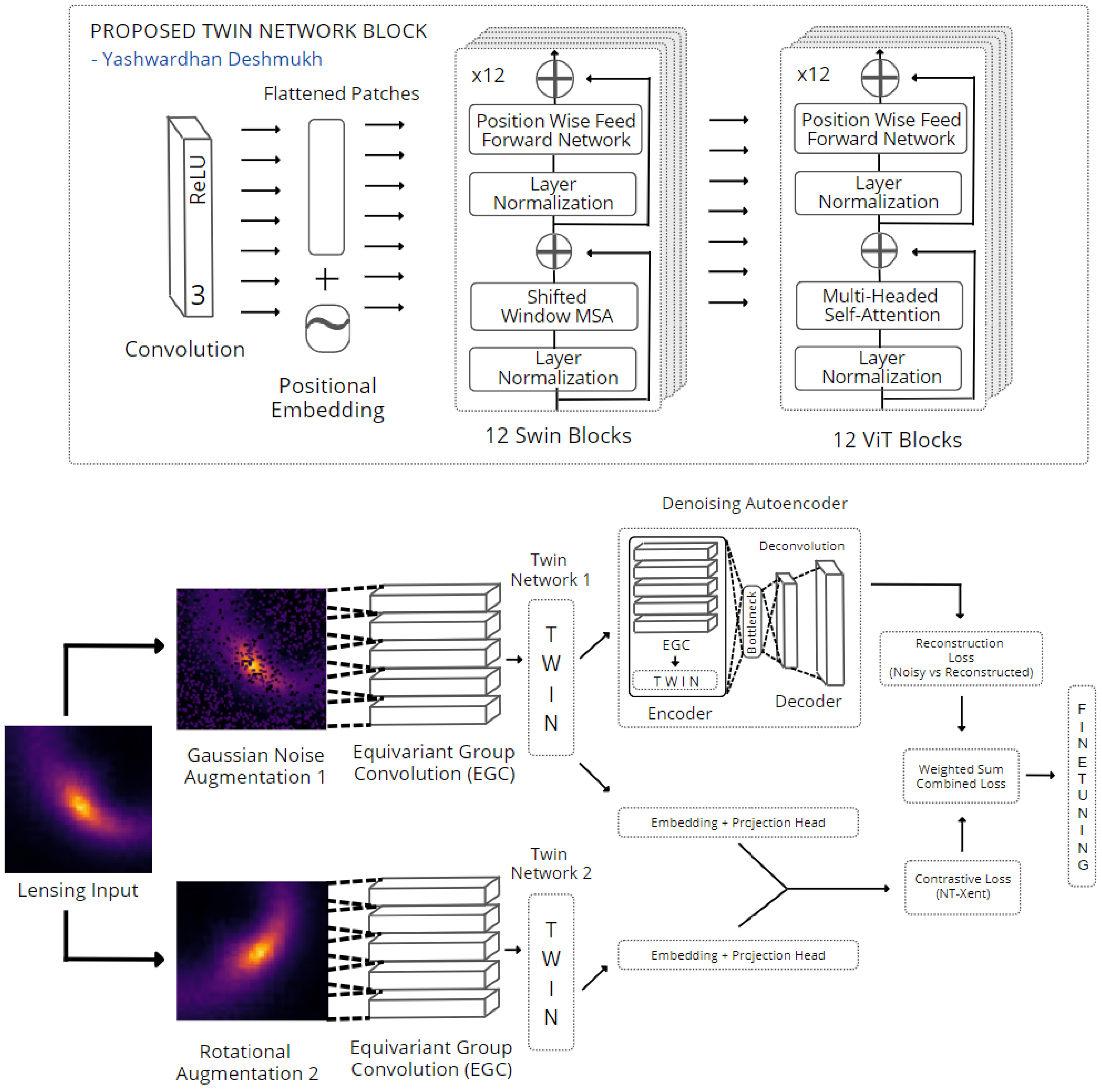
PROPOSED TWIN NETWORK BLOCK
- Yashwardhan Deshmukh
Flattened Patches
ReLU
3
Convolution
+
Positional Embedding
x12
Position Wise Feed Forward Network
Layer Normalization
Shifted Window MSA
Layer Normalization
12 Swin Blocks
x12
Position Wise Feed Forward Network
Layer Normalization
Multi-Headed Self-Attention
Layer Normalization
12 ViT Blocks
Denoising Autoencoder
Deconvolution
Bottleneck
EGC
TWIN
Encoder
Decoder
Twin Network 1
TWIN
Lensing Input
Gaussian Noise Augmentation 1
Equivariant Group Convolution (EGC)
Reconstruction Loss (Noisy vs Reconstructed)
Weighted Sum Combined Loss
FINETUNING
Embedding + Projection Head
Contrastive Loss (NT-Xent)
Embedding + Projection Head
Twin Network 2
TWIN
Rotational Augmentation 2
Equivariant Group Convolution (EGC)

**Fig. 1** Proposed Self-Supervised Architecture (Designed by Yashwardhan Deshmukh)

## Table of Contents

## Insights Gained from Tests

I have successfully **completed 7 tests** for the DeepLens project and have achieved desired results. I have even implemented multiple approaches for some tests.

I have hosted the notebooks at: github.com/yaashwardhan/Evaluation-Test-DeepLense . Along with a ReadMe.md file containing all the results.

The trained models (with the notebooks) can be found here: https://drive.google.com/drive/folders/1x5gm4ywOQ8brxMn11KjLBxsehQuxtx19?usp=sharing

Test VIII consists of two tasks, both of which require to train a self-supervised representation learning model for classification and regression. For this task, I developed a RotationalConv2D custom layer to apply convolutions in a rotationally invariant manner, which helps the model learn features independent of the input orientation. Using this, and some multi-head self-attention mechanisms, I have created an equivariant transformer architecture. Then I set up a contrastive loss function that pushes the model to learn similar features for positive examples and dissimilar features for negative examples.

From Tests I and II on multi-label classification and lens finding, I was able to infer to channel-wise attention mechanisms, to get inspired to build a more effective self-attention block due to its positive effect on feature importance, modularity, and adaptive weighing.

Tests III and IV introduced equivariance, a property I implemented for our task in order to promote generalization and develop the RotationalConv2D layer.

Moving on, Test V demonstrates the benefits of using vision transformers which treat images as sequences of patches and learning long-range dependencies using multi-headed self-attention mechanisms. We will see this used more in detail in the methods and the deliverables section.

The specific Test VI on Image SuperResolution was one of my personal favorites that got me into this project. In this, I implemented multiple super-resolution architectures to upscale the images.

All tests are in TensorFlow and Keras, but I can do the same for this project in PyTorch if needed.

## About Me

I am a final-year computer science undergraduate student from NMIMS university, Mumbai, who is fueled with a passion for research and development in neural networks and computer vision.

Amidst my first year, I spent most of my time developing full-stack deep learning applications using Flutter and Tensorflow Lite and freelanced mobile phone apps for around 17 clients.

During my second year, I was accepted to visit and work as a deep-learning research intern at Max-Planck-Institut für Intelligente Systeme in Stuttgart, Germany.

At Max Planck, I built over 42 physics-informed self-attention transformer neural networks to work as a surrogate model for a finite element mesh simulation to plot human fingertip deformations for static and frequency response situations. Following this, I built a python library to create an interactive 3-D mesh plot for the neural network-generated data and embedded it into a website for public use. A paper for this is in progress and will be published soon along with the tool.

In the course of my third year, I was awarded the Mitacs Globalink Research Internship Grant, for which I spent my summer in Quebec, Canada, building neural networks for temporal sequence data collected from gyroscopic sensors installed into helmets, and implementing feedback loops for improving the safety of users.

Presently, this is the summer of my final year of university and I am looking forward with great positivity to investing my time into working on this project for DeepLens, which has fascinated me and piqued my interest to learn and improve in this domain in many ways. I have thoroughly read the past 3 papers from DeepLens and familiarized myself with the concepts involved in them.

Apart from academics, I invest some of my time into my hobbies (jamming my guitar and representing in karate tournaments) and giving back to the community through social work.

More of my past internships and projects can be found in CV_Yashwardhan_Deshmukh.pdf

## Methods and Project Deliverables - A Detailed Report

In this section, we will discuss the methods in detail that I will implement during the period.

Our main goal is to develop a self-supervised learning transformer model for DeepLense training and inference. We will also refer to and cite previous research in this domain and improve upon it for our custom approach. The following steps will be taken:

### I. Preliminary Overview of Architectures

To create a custom architecture that is optimized for our task, we must analyze and reflect on the insights gained from architectures that have previously been used in this domain. From this, we can understand the strengths and weaknesses of all architectures such that we can unify them to achieve the best from each of them.

| Architecture | Strengths | Weakness |
|---|---|---|
| Equivariant Transformers [4]<br>Exhibit equivariance to certain transformations | - Designed to be invariant.<br>- Generalize well. | - Can sometimes be complex to build and get a perfect architecture. |
| Vision Transformers [5]<br>Image processing through tokenization. | - Scale well with large amounts of data causing better performance on large datasets like ours.<br>- Easy to implement transfer learning which performs exceptionally well. | - Computational cost is high.<br>- Large data is required. |
| Swin Transformers [6]<br>Hierarchical, shifted windows for transformers | - Learn hierarchical representations of images.<br>- Flexible at resolution handling.<br>- Improved efficiency over ViT. | - Complex implementation due to hierarchical structure. |

[4] Tai, K. S., Bailis, P., & Valiant, G. (2019). Equivariant Transformer Networks. In K. Chaudhuri & R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (pp. 6086–6095). Retrieved from http://proceedings.mlr.press/v97/tai19a.html

[5]Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ArXiv. /abs/2010.11929

[6] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., … Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. CoRR, abs/2103.14030. Retrieved from https://arxiv.org/abs/2103.14030

## II. Steps for Developing the Custom Hybrid

Steps I will implement during GSoC to develop our architecture:

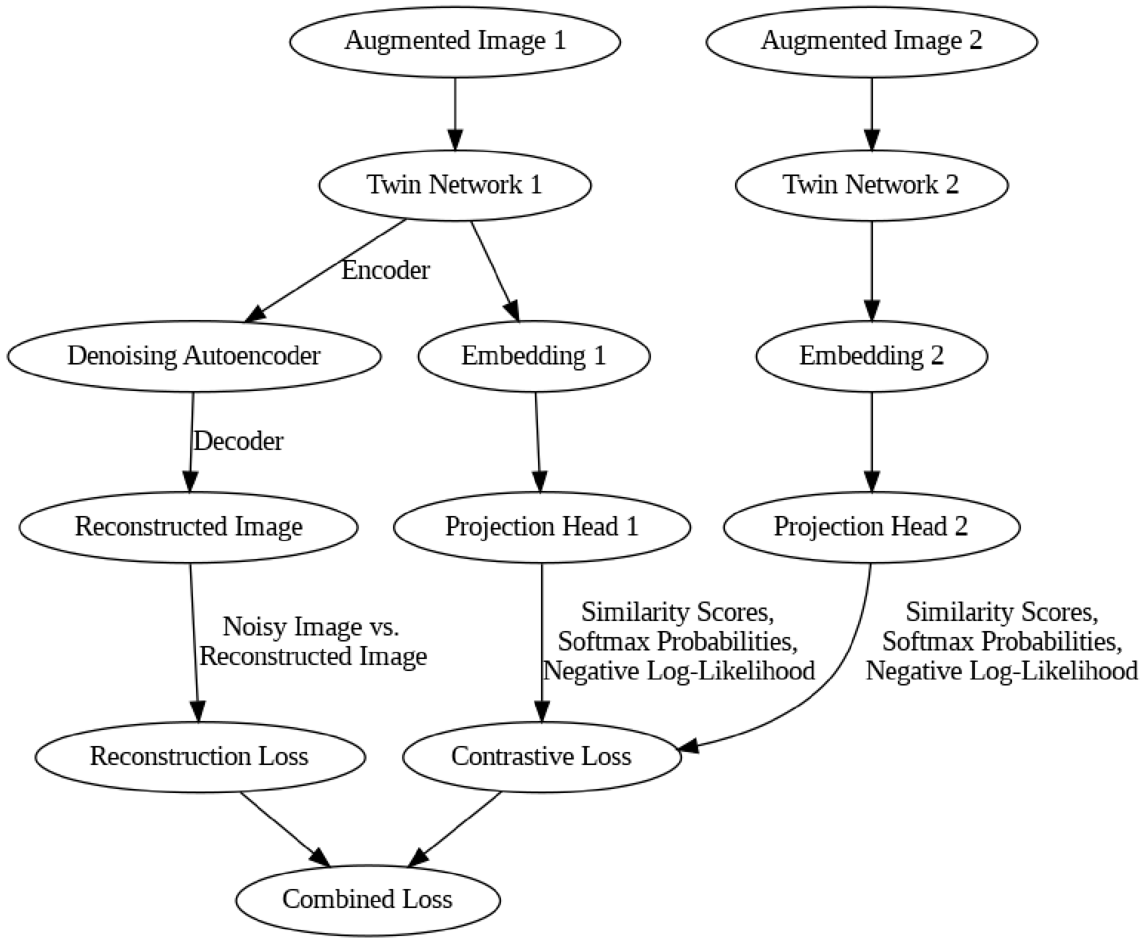

**Fig. 2** Simplified Flowchart version of Fig 1.

- **Data Preprocessing:**
  i. Data Augmentation (Constrastive Learning): I will create two different augmented views of each input image (e.g., random cropping, color jitter, rotation).

  ii. Noise Injection (Denoising Autoencoders): I will add random noise (e.g., Gaussian, salt, and pepper) to input images to create noisy versions.

- **Implementing Twin Networks for Contrastive Learning:**
  The twin network refers to a pair of neural networks that share the same hybrid architecture and weights. However, each network processes a different augmented view of the same input image. The goal is to learn a feature representation that is similar for the two augmented views of the same image while being dissimilar for different images.

  i. Input Layer (Patch Grid): I will write code to convert the input images into a fixed-size grid of non-overlapping patches by resizing and cropping them. Then I will flatten each patch and linearly embed it using a learnable embedding matrix to obtain a sequence of input tokens. Finally, I will add positional encoding to the input tokens to capture spatial data.

ii. Equivariant Layer: An equivariant transformer layer to learn features invariant to specific transformations such as rotation, scaling, and transformation is deployed. This is a good approach for us since substructures can be present in any orientation, of any size, and anywhere, in the data.

In this, we need to design our layer with equivariant constraints and mathematical operations to ensure that when the input undergoes a transformation, the output features undergo a predictable change.

After this, I will design the group convolution layers that incorporate transformation properties, following which, I will incorporate steerable filters into the equivariant layer so that the learned features can be made equivariant to the desired transformations.

iii. Swin Transformer Blocks: Using shifted windows and local self-attention I will construct the swin transformer blocks with hierarchical representational learning. Then I will work on the local attention part by dividing the input tokens into non-overlapping windows and applying self-attention within each window.

iii. ViT Transformer Blocks: Since our focus is dark matter searches, a majority of it will fall in the global context and hence to learn deeper representations, we will implement vision transformer blocks with global self-attention mechanisms to allow the I network to also learn these global contexts and refining features.
In this, we will implement the multi-head self-attention mechanism from the paper[7].

$$Calculated\_Attention([Q],[K],[V]) = \frac{Softmax(([Q]*[K]^{Transpose})}{\sqrt{key\_dimention}} * [V]$$

- Using learned weight matrices for each attention head, we project the input token embeddings into three vectors, query (Q), key (K), and value (V).
- We calculate the importance of tokens relative to each other in the sequence by first measuring the similarity between query (Q) and key (K) by taking their dot product. Then we will divide the dot product by the square root of the key dimension to assist gradient stabilization while the model learns. Finally, we apply softmax to get the attention scores of tokens with respect to each other.
- Once we get the attention scores, we can calculate a weighted sum of the value (V), which will highlight the relevant tokens for our task. This could be the lens in our case.
- Then we can concatenate the weighted value (V) vectors from all attention heads and pass it through a linear layer, to generate our multi-headed self-attention result.

iii. Feature Extraction (Embedding): The last transformer block will generate high-level feature representations (embeddings) from its output. I will apply global average pooling to compute the mean of the token embeddings/use the [CLS] token embedding in classification task

iii. Projection head of Contrastive Learning: I will deploy a small multi-layer perceptron to project the embeddings into a lower dimensional space for use in contrastive learning. This will also include one or more ReLU layers and layer normalization or dropout layers.

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. ArXiv. /abs/1706.03762

- **Denoising Autoencoder:**
  We will implement mechanics to corrupt the input images with random noise to reconstruct the clean image.

  i. Encoder: Use the same above-mentioned twin network (consisting of input, equivariant, Swin, ViT layers) as the encoder.

  ii. Decoder: I will design the decoder network to reconstruct the original image from the noisy embeddings. The process will start with building a fully connected layer to reshape the embeddings back into spatial dimensions of the last Transformer layer output. Then I will use a series of deconvolutional or upsampling layers to increase the spatial dimensions. One approach can be using bilinear upsampling, followed by applying ReLU activations and batch normalization in the decoder layers. Then implement a final dense layer with sigmoid to predict the probabilities.

- **Loss Function:**
  Here I will explain how I compute both the losses for the above two blocks. Then I will combine both losses using a weighted sum as the final loss for our model. This way, the model will learn to generate meaningful representations through contrastive learning and also be robust to noise through denoising autoencoders.

  i. Contrastive Loss: I will compute the NT-Xent Contrastive loss using the projections from both augmented views. Projections of the embeddings will be obtained by passing them through the projection head.

  ii. Calculate Similarity Scores: This will be calculated by me between all pairs of projects in a batch using the cosine similarity metric. To ensure the similarity scores lie between -1 and 1, the projects will be normalized to have a unit length and then their dot product will be taken to calculate the similarity scores that will be stored in a matrix where each entry (i, j) represents the similarity between the i-th and j-th projections in the batch.

  iii. Positive-Negative Sample Split: For each projection, I will use its corresponding projection from the other view as the positive sample, and all other projections as negative samples. Each image in the batch has two augmented views, and their projections serve as positive samples for each other. The projections from the other images in the batch serve as negative samples. This setup encourages the model to learn representations that are similar for augmented views of the same image and dissimilar for different images

  iv. Softmax Computation: For each projection, calculate the softmax probability of its positive sample by dividing the exponentiated similarity score by the sum of exponentiated similarity scores for all negative samples, which will represent how likely the model thinks the positive sample is the correct match among all available samples.

  v. Average Negative Log-Likelihood Calculation: Negate the natural logarithm of the softmax probabilities and compute their average across the batch, such that we can train this model using gradient descent to minimize the contrastive loss.

  vi. Final Loss Function: Combine the contrastive loss and the reconstruction loss using a weighted sum to balance their contributions to the overall training objective

**III. Bayesian Optimization for Hyperparameter Tuning**

Hyperparameters play an extremely vital role in the development of any deep learning model. When it comes to self-supervised tasks, it is extremely important to find a way to get the most optimal hyperparameters for our task such that we don't end up with problems such as vanishing gradients or learning incorrect representations.

Bayesian Optimization is a superior method for tuning hyperparameters. It first constructs a probabilistic model such as a Gaussian Process, to approximate the objective function of the model.

It captures the relationships between the objective function and the model's hyperparameters by modeling their uncertainties. It consists of acquisition functions that search unexplored areas and search near the current best solution.

This makes it use less number of steps as compared to other methods such as grid search and random search.

For our network, I will tune the following parameters:

- Number of Equivariant Global Convolutional layers, its kernel size, and the number of neurons
- Patch size, hidden size, attention heads, dropout rate, and the number of Swin blocks.
- Patch size, hidden size for ViT blocks (cannot change others since imagenet transfer learning)
- Gaussian noise threshold
- Number of Equivariant Global Convolutional layers, its kernel size, and the number of neurons
- Temperature for NT-Xent loss, batch size, learning rate, and epochs.

## Timeline with the Goals

I have divided the workflow into a period of 12 weeks followed by a final submission week, according to the GSoC schedule.

I am assuming the work can be done in TensorFlow and Keras. However, if required by the mentors and the project, I can do it in PyTorch as well.

### ★ Community Bonding Period (May 4 - 28)

- Get to know the project mentors and discuss the project more in-depth with them.
- Get familiar with the work done by previous GSoC contributors and the deep lens pipeline. Find a spot for our superresolution project in the pipeline.
- Explore all the previous research papers in self-supervised algorithms for lensing and papers that may have implemented a strategy similar to ours.
- Partially finalize the architecture we will build such that it can be adjusted later.

★ **Week 1 (May 29 - June 4)** [Data preprocessing and Equivariant Group Convolution]

- Discuss if any changes are necessary for the current week's work with the mentors.
- Implement the data preprocessing steps for data augmentation and noise injection.
- Test and finalize the augmentation types with the mentors.
- Create Equivariant Group Convolution Blocks Class for the architecture.

★ **Week 2 (June 5 - June 11)** [Twin Start, Extract Patches, and Positional Embeddings]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Start building the Twin Architecture and its convolutions.
- Begin Implementing Swin transformers by programming classes to generate patches from images and flatten them to get a linear projection.
- Add positional embeddings to encode the location of each patch.

★ **Week 3 (June 12 - June 18)** [Shifted Window Multi-Head Attention Logic]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Program the class the set up the Swin transformer's shifted window multi-head self-attention mechanism.
- Instantiate and design an outline of a transformer block class such that we can replicate it 'n' number of times to generate more transformer blocks.

★ **Week 4 (June 19 - June 25)** [Swin Transformer Block with MLP]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Continue establishing the Swin transformer block class. Apply layer normalizations and call the multi-head self-attention class implemented in the past week.
- Add dropout and residual connections and more normalizations.
- Build the MLP feed-forward network to learn more complex representations and use the Gaussian Error Linear Unit as activation.

★ **Week 5 (June 26 - July 2)** [Multi-Head Attention Logic for Vision Transformer Block]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Program the class the set up the transformer's multi-head self-attention mechanism, which will include programming the logic to project the embeddings into query, key, and value followed by taking the dot product of query and key and using *tf.matmul(weights, value)* to get the weighted average over the softmax attention scores.
- Instantiate and design an outline of a transformer block class such that we can replicate it 'n' number of times to generate more transformer blocks.
- Extract the Flax keys from the pre-trained imagenet model so we can appropriately name our class layers.

★ **Week 6 (July 3 - July 9)** [Vision Transformer Block with MLP]

- July 10: Start submitting midterm evaluations
- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Continue establishing the transformer block class. Apply layer normalizations and call the multi-head self-attention class implemented in the past week.
- Add dropout and residual connections and more normalizations.
- Build the MLP feed-forward network to learn more complex representations and use the Gaussian Error Linear Unit as activation.

★ **Week 7 (July 10 - July 16)** [Create Correspondence and load Flax Imagenet Pretrained]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Develop a correspondence class that iterates over the transformer blocks and gets the corresponding names for the blocks, pre_logits, embedding layers, dummy_inputs (cls), and encoder_norm layers.
- Get a mapping of the corresponding blocks with the Flax keys extracted in week 3 from the imagenet pretrained model.

★ **Week 8 (July 17 - July 23)** [Denoising Autoencoder]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Configure the denoising autoencoder class,

- Create the encoder and decoder (bilinear upsampling/deconvolution)

★ **Week 9 (July 24 - July 30)** [Embedding, Projections, bug fixes, and Reconstruction loss]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Generate embeddings and projection heads from both the augmented twin networks.
- Bug fixing week: Fix all bugs and document the code that is written so far.
- Implement and finish the programming for twin network 1 and its reconstruction loss.

★ **Week 10 (July 31 - August 6)** [Contrastive Loss and Weighted Sum and Tuning]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Write logic for calculating the NT-Xent Contrastive loss.
- Implement the logic to calculate the weighted sum of contrastive and reconstruction loss. Train the network and discuss the results with the mentors
- Set up hyperparameter tuning using Bayesian Optimization over the network.

★ **Week 11 (August 7 - August 13)** [Finetune and test for regression and classification]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Implement the network to expand the DeepLens pipeline and finetune it with other techniques such as regression and classification.

★ **Week 12 (August 14 - August 20)** [Re-iterate]

- Get feedback about previous weeks' work from the mentors and implement it.
- Discuss if any changes are necessary for the current week's work with the mentors.
- Review the codebase and try to find if any improvements can be proposed and implement them from the conclusions of the exhaustive training.

★ **Final Submission Week (August 21 - August 28)**

- Clean up the code and make sure all bugs are fixed
- Neatly document the entire project including all classes and functions.
- Commit all the work to the mentor's repository or submit it to them as required.

## Benefits to Community

Dark matter halos are structures made of dark matter, which is a theoretical concept that does not absorb, emit or even reflect electromagnetic radiation, it is not visible. However, from strong gravitational lensing data, we can study the distortion and magnification of light emitted by distant background galaxies that pass through a massive foreground object like a dark matter halo.

The gravitational field of the halo appears to distort and bend the light, causing the background galaxy to appear stretched, magnified, or even multiplied. Astronomers analyze these effects on the visible matter to infer the presence and characteristics of the dark matter halo responsible for the lensing effect.

However, a major problem arises due to large amounts of unlabelled data that are generated. Manually labeling it would be exhausting and unsustainable and hence a technique such as self-supervised representational learning can be implemented to learn embeddings and features that are beyond the theoretical scope.

This will be extremely beneficial to the community working together to find dark matter traces in this vast cosmos of ours.

## My Eligibility Towards the Project and Why Me?

- I am eligible to work on the project from GSoC's guidelines.
- I have the necessary knowledge and skillset required to successfully complete the project.
- I have completed and submitted 7 tests required to be eligible to work on the projects.
- I believe in investing my work into open source as it benefits the community as a whole. All of my past projects are available on GitHub for public use.
- On reading past papers and projects from DeepLens, I am impressed by their work to contribute towards unlocking the secrets of the universe. I am passionate to be a part of it.
- I have previously worked in research (at Max Planck) on physics simulations and deep learning methods, which will help me transfer and implement my prior knowledge on this.
- Moreover, I wish to learn from and know more about the work of the mentors (Sir Michael Toomey, Sir Pranath Reddy, Sir Sergei Gleyzer, Sir Emanuele Usai, Sir Saranga Mahanta, and Sir Karthik Sachdev)

## What am I Expecting out of this Project?

To be as honest as possible, I have always been fascinated by the vastness of the universe and want to contribute my knowledge to it by getting the chance to implement this architecture that I have developed and designed through a lot of motivation towards DeepLens. The project is not only a great open-source idea for the science community but also a huge step toward my personal improvement in the domain. I have already invested the past month working on the tests and this proposal which has piqued my interest even more towards completing it.

If possible, I want to also publish a paper with the mentors on this project as the architecture seems to match the current progress in deep learning and will benefit the community as a whole.

To conclude, even if I am not given the opportunity through GSoC, I still desire and want to work with the mentors on this project regardless (it's okay if no funding is available).

I am grateful for the given opportunity and the time the mentors have taken to carefully read the proposals. Thank you very much and I am looking forward to working with you!