GSoC 2023 Project Proposal

Organization : ML4SCI

Vision Transformers for End-To-End Particle Reconstruction for the CMS Experiment

MENTORS

Emanuele Usai, Ruchi Chudasama, Shravan Chaudhuri, Sergei Gleyzer, Eric Reinhardt, Samuel Campbell

AUTHOR

Abhimanyu Borthakur

Contents

1. Contributor Information

- **Name**: Abhimanyu Borthakur
- **E-Mail**: abhimanyuborthakur@hotmail.com, abhimanyuborthakur@gmail.com
- **GitHub**: abhimanyu911
- **Contact**: +91-9007063439
- **University**: Manipal Institute of Technology, MAHE, Karnataka, India (2018-22)
- **Major**: BTech Electronics and Communication
- **Minor**: Data Science
- **CGPA** (overall): 9.24/10
- **CGPA** (minor): 10/10
- **Achievements**: Ranked 1st out of 227 students in the 7th semester with a GPA of 9.8/10 and 11th overall.
- **CV:** Click here

2. Introduction

The Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) is a ground-breaking international scientific collaboration that aims to explore the fundamental nature of matter and the universe. Located at the European Organization for Nuclear Research (CERN), the CMS experiment has made significant contributions to our understanding of particle physics, such as the discovery of the Higgs boson in 2012, which confirmed the existence of the Higgs field and completed the standard model of particle physics. [1]

Particle identification plays a crucial role in the analysis and interpretation of the vast amount of data generated by the LHC collisions. Accurate identification of particles helps researchers disentangle signal events from background noise, enabling the discovery of new particles and phenomena. The ability to distinguish between different types of particles with high precision is essential for understanding the underlying physics processes and testing theoretical predictions.

With the advent of machine learning and deep learning techniques, neural networks have emerged as a powerful tool for tackling complex pattern recognition tasks, such as particle identification in high-energy physics experiments. Convolutional Neural Networks (CNNs) [2] and Vision Transformers (ViTs) [3] are two prominent neural network architectures that have shown promising results in various computer vision tasks.

CNNs employ a hierarchical structure with convolutional layers that can automatically learn local features and spatial hierarchies, making them well-suited for analysing images and other grid-like data. On the other hand, ViTs divide input images into patches and process them using the Transformer architecture, which relies on self-attention mechanisms [4] to capture long-range dependencies and interactions between patches.

In the context of the CMS experiment, both CNNs and ViTs can be used to analyse detector images and identify particles by learning to recognize their unique signatures. However, the learning processes of these two architectures are fundamentally different, with CNNs leveraging inductive biases and local feature extraction while ViTs focus on global self-attention mechanisms. Understanding these differences and comparing their performance can provide valuable insights into the strengths and limitations of each approach, guiding future research and development of neural network architectures for particle identification and beyond.

In the following sections, we will delve deeper into the workings of CNNs and ViTs, discuss their architectural nuances, and explore their learning processes, shedding light on the intricate dynamics that govern their ability to identify particles in the CMS experiment.

3. Technical details

3.1 Convolutional Neural Networks (CNNs)

CNNs are designed to exploit the spatial structure of input data, such as images, through a series of convolutional layers followed by non-linear activation functions. A typical convolution operation can be represented as:

$$Y(i, j) = \sum \sum X(i + m, j + n) * K(m, n)$$

Here, $Y(i, j)$ is the output feature map, $X(i + m, j + n)$ is the input feature map, and $K(m, n)$ is the kernel or filter, whose parameters are learned during the training process. The summations are performed over the kernel dimensions, m and n.

CNNs learn hierarchical representations by stacking multiple convolutional layers. This enables them to capture local features in the early layers and progressively learn higher-level abstractions in deeper layers. The inductive biases in CNNs, such as local connectivity and weight sharing, make them well-suited for grid-like data and contribute to their sample efficiency. [5]

Inductive biases are the set of assumptions that a machine learning model makes about the underlying structure of the data it learns from. These biases guide the learning process and can influence the model's ability to generalize to unseen data. In the context of Convolutional Neural Networks (CNNs), inductive biases are hardcoded into the architecture through the use of specific design choices and constraints that promote the learning of spatially invariant features and local patterns in the input data.

Here are some key inductive biases in CNNs:

Local connectivity: CNNs employ convolutional layers, where each neuron is connected to a small local region of the input (called the receptive field). This local connectivity enforces the assumption that spatially close input features are more relevant to each other than distant ones. This design choice helps CNNs learn local patterns and spatial hierarchies in the data.

Parameter sharing: In a convolutional layer, the same set of weights (also known as a kernel or filter) is applied across the entire input, effectively sharing parameters between different spatial locations. This shared structure enforces the assumption that the same features are useful for detecting patterns across different parts of the input, promoting translational invariance and reducing the number of learnable parameters.

Pooling: CNNs often include pooling layers (e.g., max-pooling or average-pooling) that aggregate information from spatially adjacent regions of the input. These layers introduce the assumption that the precise spatial arrangement of features is less important than their presence in a local neighbourhood. Pooling layers help the network become robust to small changes in the input and reduce the spatial dimensions of the data, further reducing the number of learnable parameters.

Hierarchical feature learning: CNNs are typically composed of multiple layers of convolutional and pooling layers, which allows the network to learn a hierarchical representation of the input data. Lower layers learn simple, local features (e.g., edges, corners), while deeper layers combine these low-level features to form more complex, higher-level representations (e.g., shapes, textures). This hierarchical structure reflects the assumption that meaningful features in the data are organized in a compositional manner.

A typical CNN architecture is shown below:

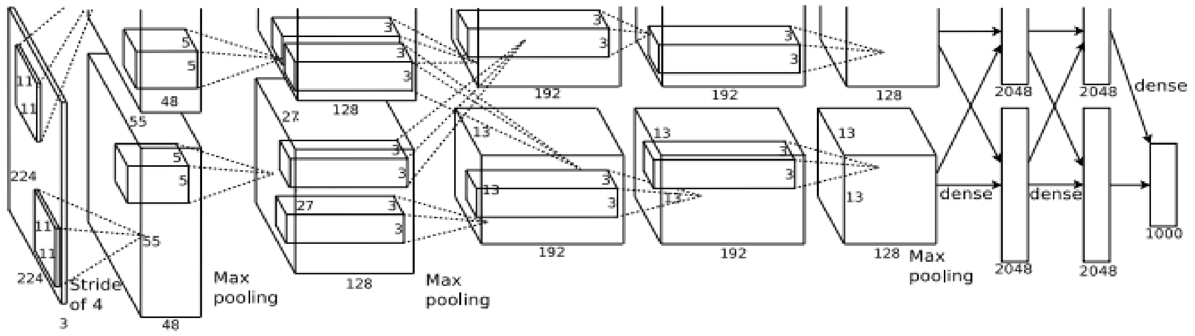

Figure 1: AlexNet CNN [6]

These inductive biases are hardcoded into the architecture of CNNs, making them particularly well-suited for learning from grid-like data, such as images or spatiotemporal data, where spatial relationships and local patterns are important. By incorporating these biases, CNNs can effectively learn relevant features from the data and generalize well to unseen examples.

3.2 Vision Transformers (ViTs)

To understand the ViT, let us break down the key components of the architecture:

Input preparation: The input image is divided into non-overlapping fixed-size patches (e.g., 16x16 pixels). Each patch is flattened and linearly embedded into a fixed-dimensional vector:

$$Z_i = \text{MLP}\ (X_i)$$

Here, Z_i is the embedded vector for the i-th patch, X_i is the i-th input patch, and MLP is a multi-layer perceptron. The vector **Z** can be thought of as a collection of tokens. In addition, a learnable position embedding is added to each patch embedding to incorporate positional information, although, sinusoids with varying frequencies as fixed positional encodings have also been used. [7]

These tokens are then fed into the Transformer encoder. During the forward pass, the "class" token is prepended to the sequence of patch embeddings and processed together with them through the multi-head self-attention layers. Throughout the Transformer layers, the self-attention mechanism allows the "class" token to interact with all other tokens, capturing global context and long-range dependencies across the input patches.

Using the "class" token in this manner has a few advantages:

Global context: The "class" token can aggregate information from all the input patches, providing a global summary that is useful for tasks like image classification.

Reduced complexity: The model's output is condensed into a single token, simplifying the classification task and reducing the number of parameters in the final linear layer.

Thus, the final hidden state corresponding to the first patch (the "class" token) is used for the classification task.

The embedding process is illustrated below: -

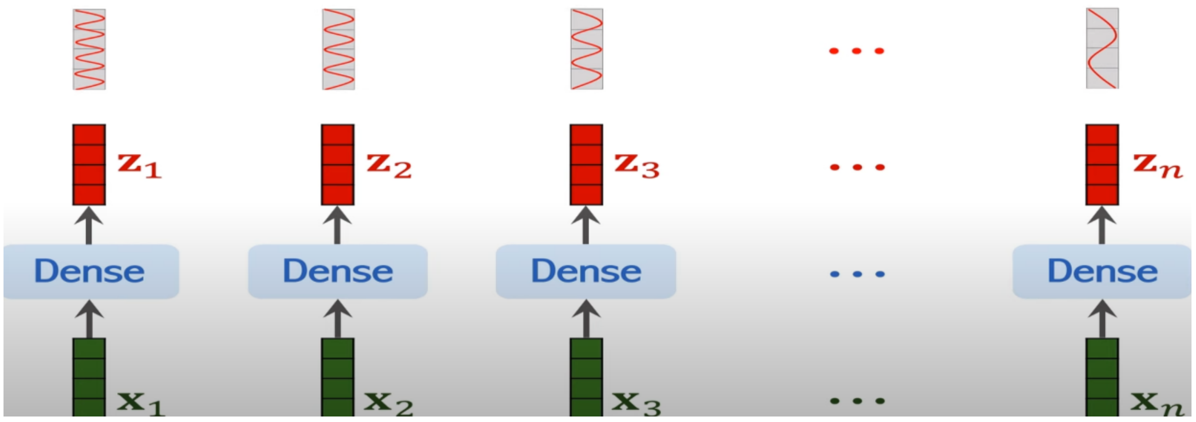

Figure 2: Input preparation and embedding [8]

Self-attention: The self-attention mechanism in ViTs is built upon the Transformer architecture. The core component is the scaled dot-product attention, which can be computed as:

$$C = \text{SoftMax}\ (QK^{\wedge}T\ /\ \sqrt{d}) * V$$

Here, Q (query), K (key), and V (value) are matrices derived from the input patch embeddings, as follows:

$$Q = W^Q \text{ x } \mathbf{Z}$$

$$K = W^K \text{ x } \mathbf{Z}$$

$$V = W^V \text{ x } \mathbf{Z}$$

Here, W^Q, W^K and W^V are learned during the training process. The SoftMax operation is applied along the key dimension, resulting in a probability distribution that indicates the relevance of each input patch to every other patch. The output is a weighted sum of the value vectors, with weights determined by the SoftMax probabilities.

The dimension d is the dimension of the queries and keys, and it is used to scale the dot product to prevent large values from dominating the SoftMax operation.

Multi-head attention: To allow the model to focus on different aspects of the input, ViTs employ multi-head attention. This involves computing multiple attention functions in parallel, with each attention function having its own set of learnable weights. The outputs of the attention heads are concatenated and linearly transformed:

$$C' = \text{Concat}\ (C^0, ..., C^i, \ldots, C^{h-1}) \text{ x } W^O$$

Here, C^i is the output of the i-th attention head, h is the number of heads, and W^O is a learnable matrix for the output linear transformation. As stated before, only C^0 is used for the classification task.

Layer Normalisation and Residual Connections: Layer normalization and residual connections are essential components of the Transformer architecture that help improve training stability and facilitate deeper models. Layer normalization is a technique for normalizing the activations of neurons in a network to improve training stability and convergence. It is applied independently to each input feature by computing the mean and variance of the feature values and then normalizing them.

Residual connections, also known as skip connections or shortcut connections, are a technique to alleviate the vanishing gradient problem in deep neural networks. In the Transformer architecture, residual connections are used to connect the input of each sub-layer (e.g., multi-head self-attention) to its output:

$$y = x + F(x)$$

where x is the input to the sub-layer, F(x) is the output of the sub-layer, and y is the final output after the residual connection.

The forward propagation is summarised in the following figure:

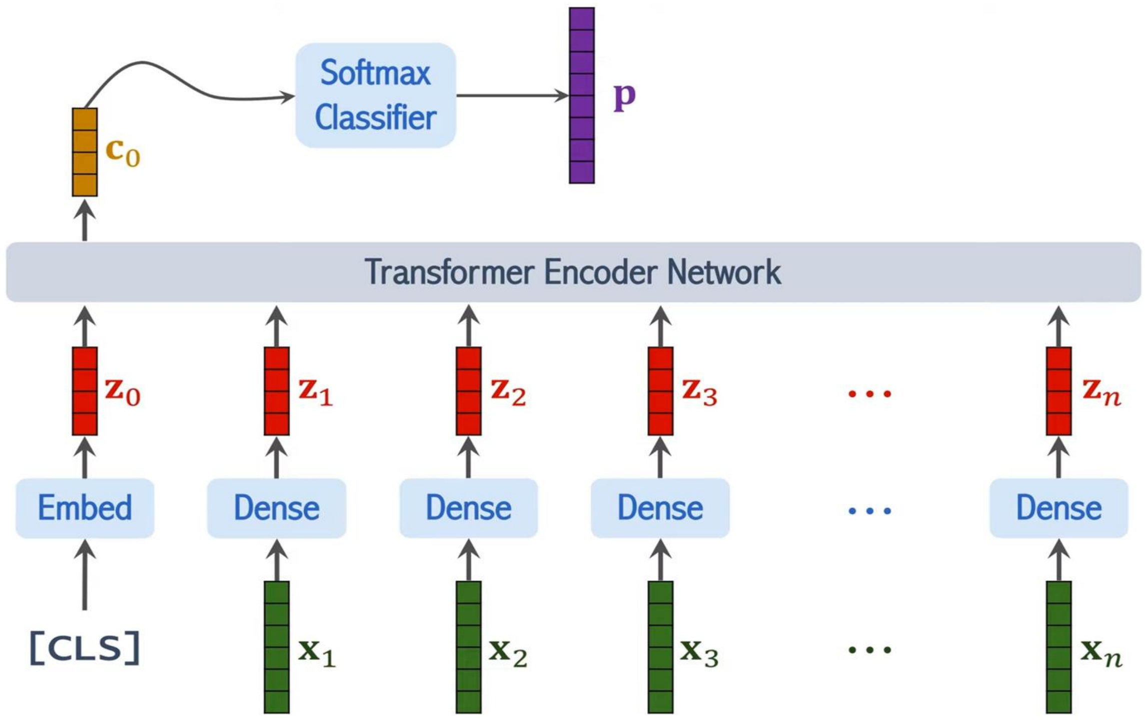

Figure 3: Propagation through a transformer [8]

A high-level overview of ViTs is presented below: -

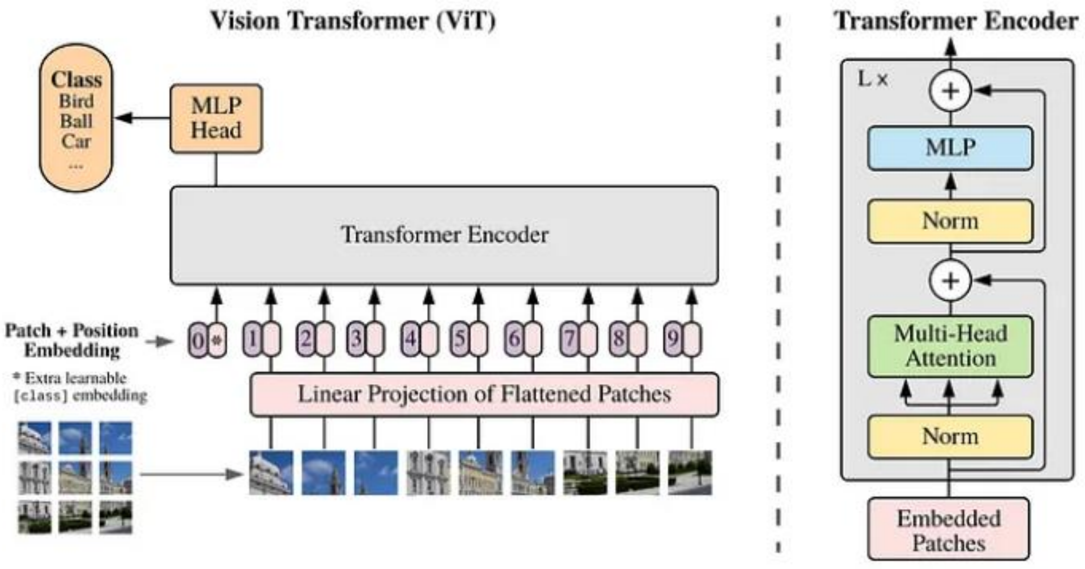

Figure 4: The ViT [9]

3.3 CNNs vs ViTs

The absence of explicit inductive biases in ViTs can make them less effective in capturing local features compared to CNNs. However, their ability to model global context can be advantageous in certain tasks, particularly when large training datasets are available.

To summarize, CNNs have an advantage where you might not have much data to train on or you might not be able to perform transfer learning, such as cases where:

a) there are not enough samples for the ViT to capture strong representations using global self-attention, and

b) there is a lack of similarity between the tasks of publicly available pre-trained ViT weights and the task of particle identification making it unsuitable for fine-tuning.

In situations where you can expect transfer learning to work well, ViTs can be a good choice, otherwise the powerful and useful inductive bias of the convolution operation and this is useful in limited data setting.

This highlights the importance of carefully considering the choice of model architecture and its suitability for the specific problem and dataset. It warrants the need to explore architectural modifications, data augmentation techniques, hyperparameter optimization or pretraining strategies to improve the performance of the ViT.

Although, one area where ViTs do outperform CNNs is in their robustness and generalizability in the presence of perturbations like Gaussian noise. This is studied in-depth in the evaluation tasks which are described in the following sub-section.

3.4 Evaluation Task details

The code repository containing all the requisite tasks can be found here

3.4.1 Task 1: Electron-Photon Classification

Task objective: Design a deep learning model to classify 32x32x2 tensors as either electrons or photons, with a minimum ROC-AUC score of 0.80

Task results: Successfully designed a CNN with 30.5K parameters and achieved an ROC-AUC score 0.81 on the tests set using TensorFlow and translated the same model to PyTorch. Detailed results are listed as follows: -

Table 1: Task 1 scores on different splits

DATA SPLIT	ROC AUC SCORE
Training	0.812
Validation	0.815
Test	0.81

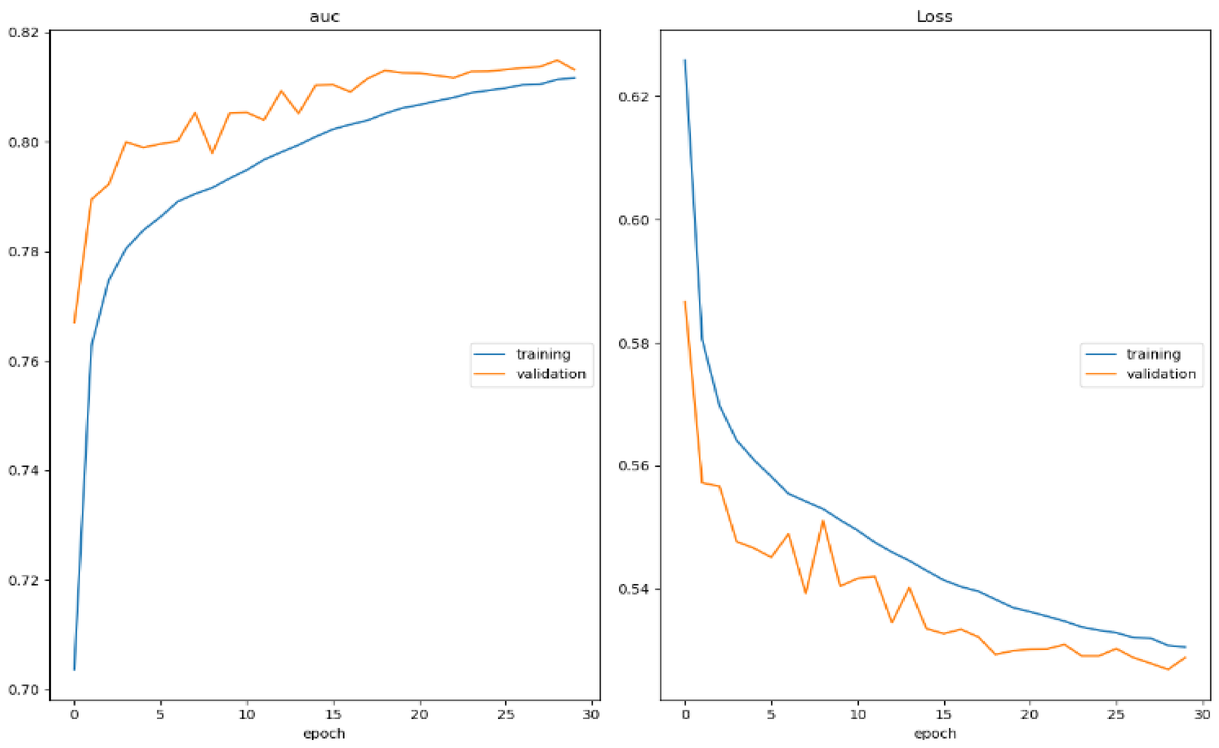

Figure 10: Learning curves for task 1

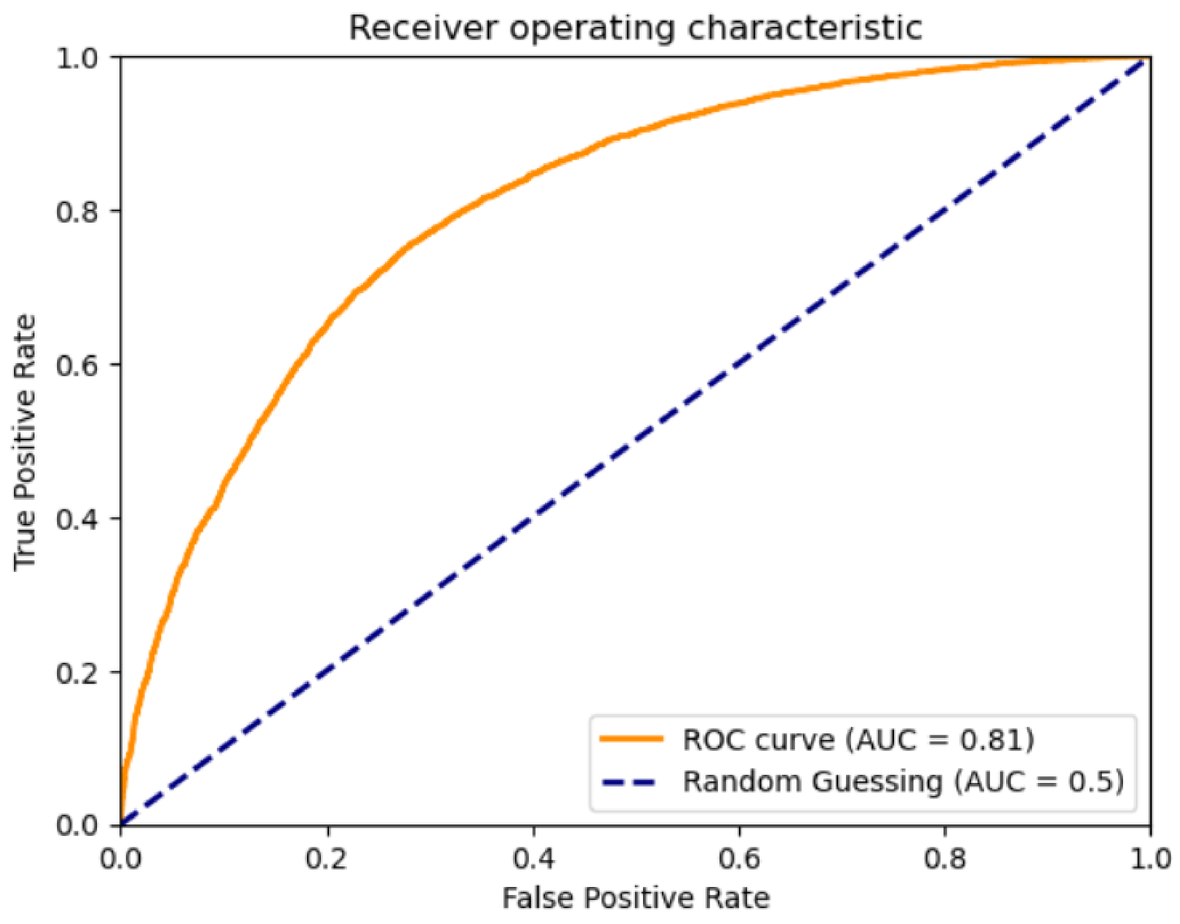

Figure 11: ROC curve for task 1

Solution highlights:

- Prioritised both efficiency and performance by designing a small model with limited number of parameters and yet, achieved the required criteria of ROC AUC score
- Incorporated software engineering principles such as code separation, clean code, modularisation, model checkpointing and logging in the experiment
- Did not overfit on the test or validation test set as the advantageous Dropout layers were leveraged.
- Set a random state parameter to 42 such that the experiment produces reproducible splits on every iteration to provide consistent results in tasks 1 and 3 and avoids any data leakage.

3.4.2 Task 2: Quark Gluon Classification

Task objective: Design a deep learning model to classify 125x125x3 images as either quarks or gluons.

Task results: Successfully designed a relatively small CNN with <100K parameters and achieved an ROC AUC score of 0.77 on the test set. Detailed results are as follows:

Table 2: Task 2 scores on different splits

DATA SPLIT	ROC AUC SCORE
Training	0.762
Validation	0.78
Test	0.77

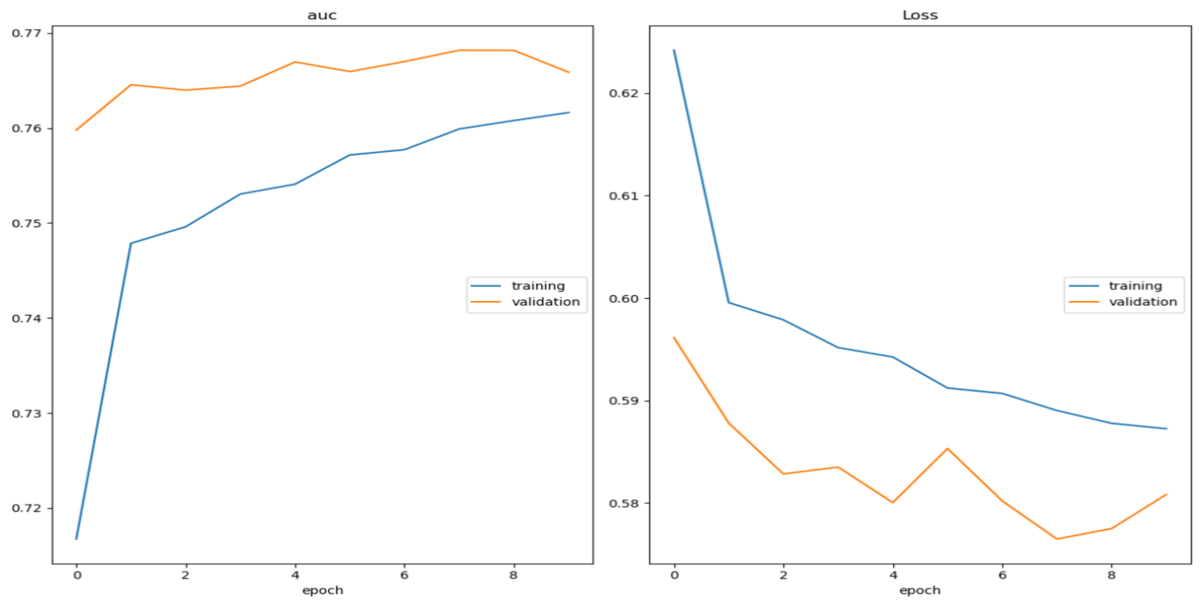

Figure 12: Learning curves for task 2

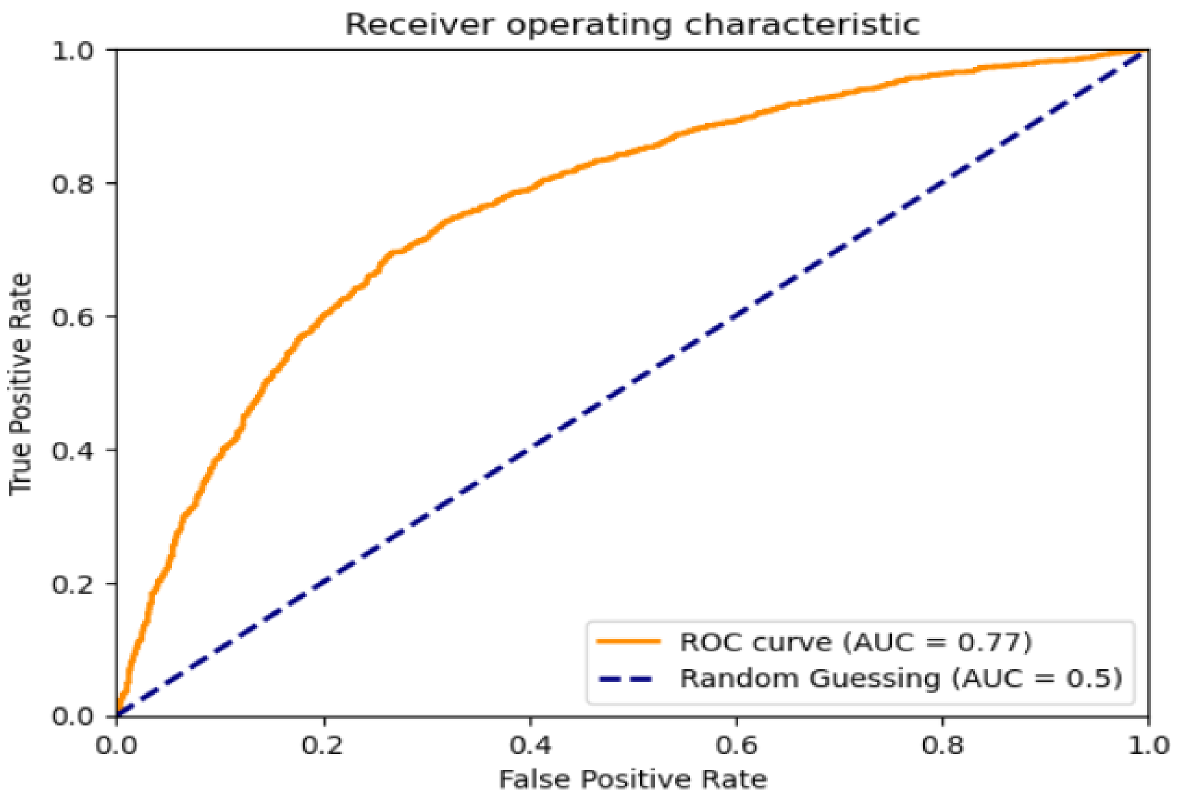

Figure 13: ROC curve for task 2

Task highlights:

- Was able to use all 3 parquet files in the experiment by compressing the images to 33x33x1 using the insights from this workshop and storing them in 3 hdf5 files – one for each of the training, validation and test splits. Pyarrow was used to iterate over the files and stream the ('X_jets', 'y') pairs from the disk.
- I/O was optimized by separating compute and storage – a custom stream-from-disk pipeline was defined for the hdf5 files in order to stream (image, label) pairs in batches from the disk to the GPU, where they were fed into the model. This is done to reduce the training time and memory footprint of our model, while also giving us the opportunity to increase the throughput of our model by using a larger batch size. This is because the images are now smaller and thus can be processed faster by the GPU. It also makes our pipeline a potentially effective one when it comes to deploying our model on a real-time application.
- Incorporated software engineering principles such as code separation, clean code, modularisation, model checkpointing and logging in the experiment
- Did not overfit on the test or validation test set as the advantageous Dropout layers were leveraged.
- Was able to achieve satisfactory scores with a relatively small model.

3.4.3 Specific Task: ViT for Electron-Photon Classification

Task objective: Train a transformer model in order to perform classification on the same dataset as task 1 and try to achieve a level of performance close to that of the CNN in task 1.

Task results: Successfully trained a Vision Transformer to classify the data as either Electrons or Photons with a ROC-AUC score of 0.76 on the test set.

Detailed results are as follows:

Table 3: Task 3 scores

DATA SPLIT	ROC AUC SCORE
Training	0.77
Validation	0.78
Test	0.76

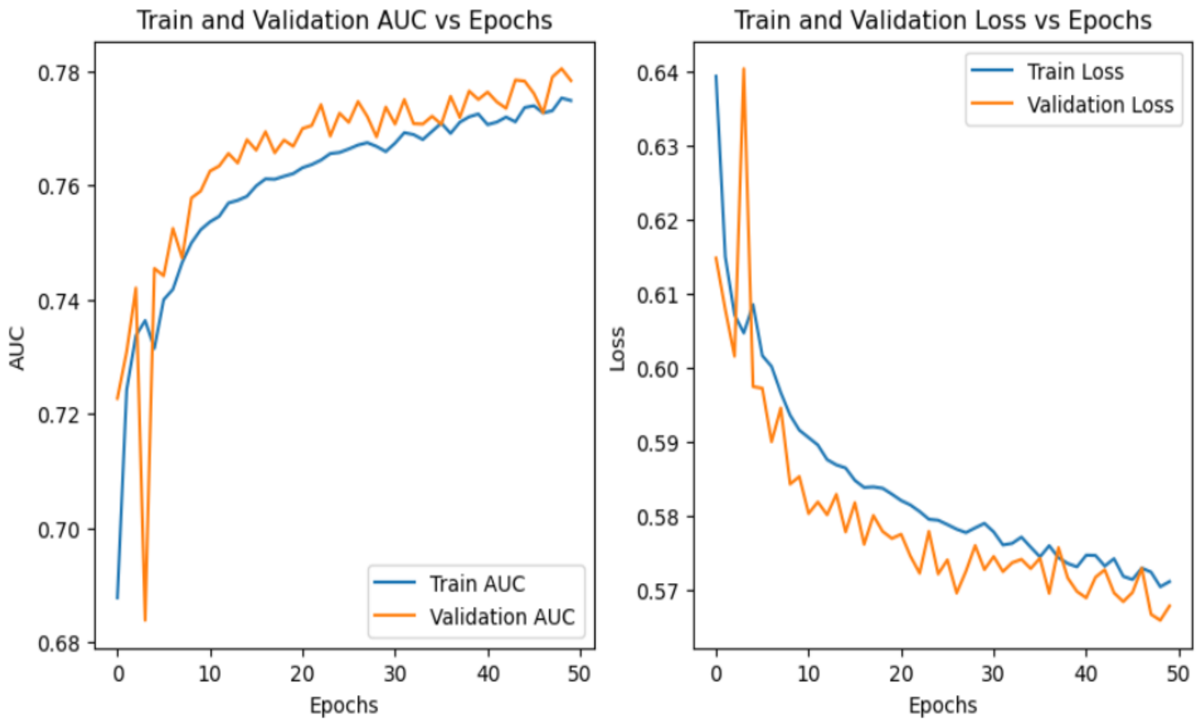

Figure 14: Learning curves for task 3

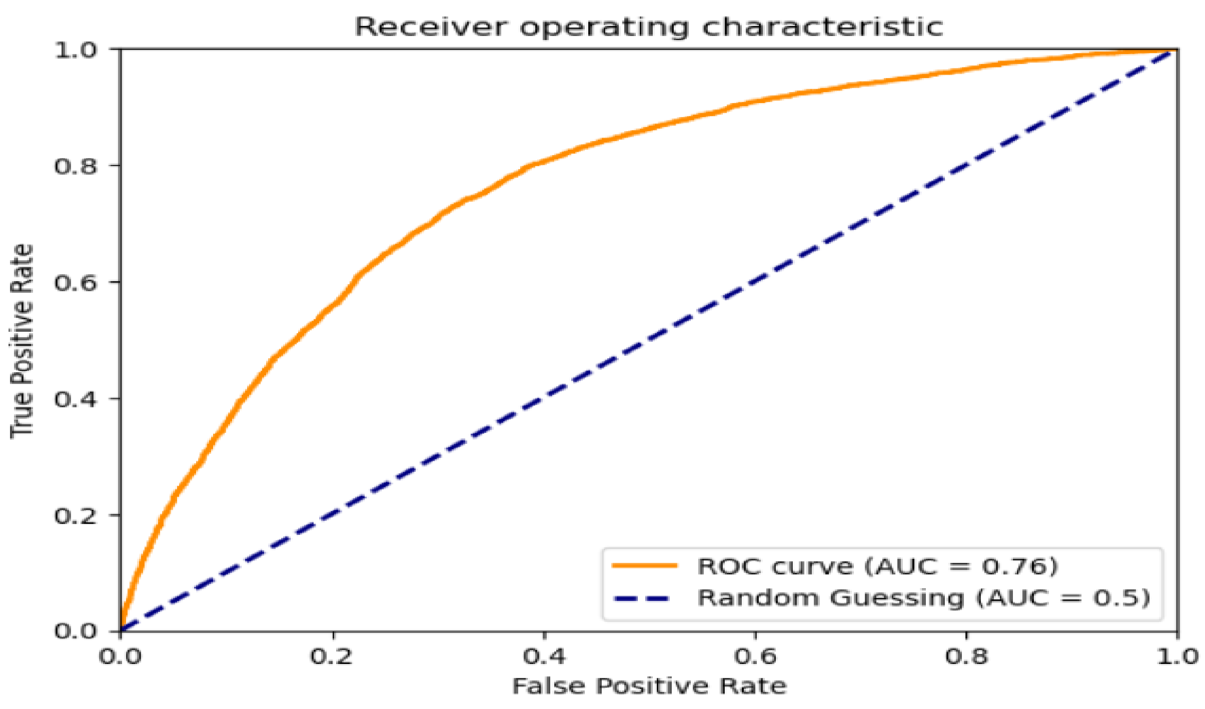

Figure 15: ROC curve for task 3

Discussion:

The CNN in task 1 performs better here, due to the ability of the powerful and useful inductive bias to learn local patterns and hierarchical features in this limited data setting. Presumably, there are not enough samples for the ViT to learn strong representations using the global self-attention mechanism when trained from scratch. The lack of explicit/hardcoded inductive bias in ViTs puts them at a disadvantage in such situations.

Although, the ViT outperforms the CNN when it comes to robustness and generalizability in the presence of perturbations such as Gaussian noise:

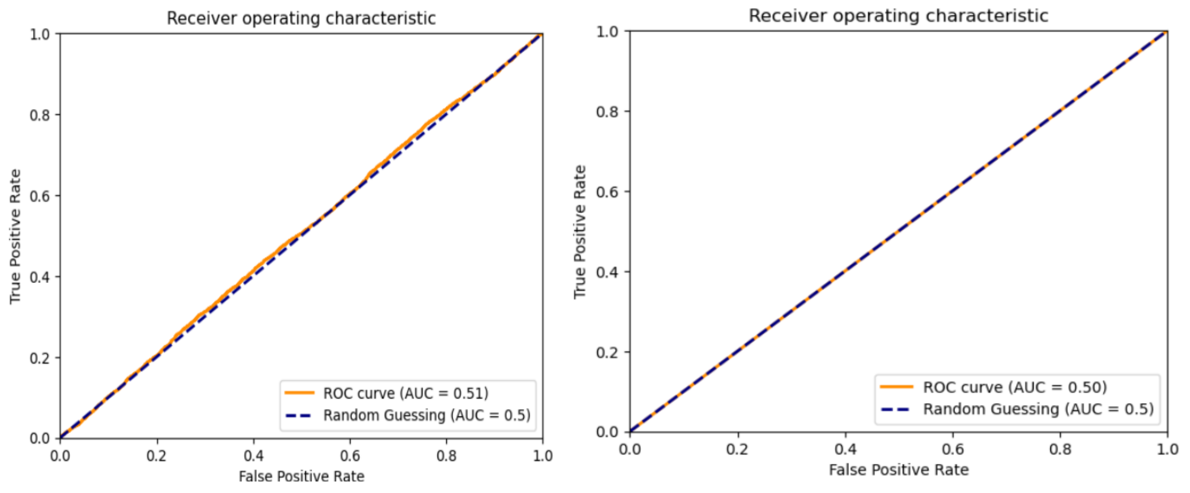

Figure 16: ViT(left) scores higher than CNN from task 1(right) when asked to make predictions on a noisy test set(Normally distributed with mean = 0 and standard deviation = 0.1)

4. Proposed Deliverables

Overarching objective: Develop a transformer based model for classifying high energy particles, achieve competetive results using said model and test and benchmark the same on a CPU and a GPU.

4.1 Devliverables/Methodology

1) Exploratory data analysis: Before any model development, it is essential to perform an exploratory data analysis across the entire dataset as it is very likely that our data will be composed of different file types, file sizes and variable image sizes and label distributions. Gaining insights across the dataset about these variations will help inform the structure of our data ingestion pipeline in deliverable #2 and also about whether our dataset is balanced or not

2) ETL Data Pipeline: ETL stands for Extract-Transform-Load. In the presence of variable file types and image sizes, it is important that we create a unified framework for ingesting the data from files on the disk, transforming it and then feeding it to the model that will reside on our GPU. It is important that our model talks to a data loader that streams the data from the storage disk instead of the entire data being loaded into the memory as the latter can cause OutOfMemory exceptions and impact the throughput of our model by forcing us to choose a lower batch size because of the memory footprint. We will have to explore preprocessing/augmentation techniques as a part of the 'Transform' as well.

3) Model development lifecyle:

Selection: We can choose 3-4 transformer-based models such as SimpleViT[10], ViT, ScalableViT[11] and Deep ViT[12]. Vit-pytorch library[13] has all the implementations open-sourced.

Training: We train the models on a GPU and observe the learning curves – models with satisfactory performance on the validation data will be fast-tracked for benchmarking on the test set and the others will be discarded. Using the pytorch[14] framework for training and Weights and Biases[15] for logging and experiment tracking will help us expedite the process of iterating over our experiments.

Benchmarking on test set: Benchmark the qualified models on the test set using torch-profiler[16] on both CPU and GPU and plot FLOPs vs AUC graph for the models in both cases in order to find the sweet spot between performance and speed.

Re-training/Ensembling: Incase the results aren't satisfactory/competitive enough, we can re-enter the training task and peform hyperparameter tuning or try ensembling techniques to achieve better accuracy and AUC.

We can also re-train the best model(s) under perturbations and compare their robustness and generalizability udner such conditions with the best CNNs.

Quantization: We may explore model quatization techniques using the torch quanitzation API [17] in order to reduce the memory footprint of the model and make it more compact.

4) Wrapping for integration with the CMS framework: Once the best model is ready and saved, it can be integrated into the CMS inference engine – This also requires refactoring the training, inference code to follow the CMS naming, coding and style rules for packaging.

5) Documentation: Document all the modular aspects of the code in order to enhance readability of and accessibility into our experiments.

4.2 Timeline

1. May 4 - 28

- Community bonding period- getting to know colleagues and mentors
- Finalising the high-level approach with the mentors and getting started with boiler plate code for plots and EDA and finalising the tech stack to expedite the process.
- Getting up to speed with the CMS documentation on GitHub and understand nuances of the same.

2. May 29 – June 10

- Perform exploratory data analysis
- Seek approval for low-level pipeline design for extracting, transforming and loading the data from mentors
- Design data pipeline/loader

3.June 10 – June 25

- Train the candidate models on the dataset
- Present training and validation learning curves to mentors and seek guidance with respect to selecting/rejecting models for the following tasks
- Bench mark selected models on test data using both CPU and GPU and make FLOPs vs AUC and FLOPs vs Accuracy plot in order to inform our choice of the best model.

4.June 25 – July 10

- Collect and document results for the purpose of the midterm evaluation and finish any pending results
- Create a report of the deliverables achieved so far and how they align with the expectations
- Decide whether re-training/ensembling is required for performance improvements in conjunction with mentors

5.July 10 – July 14

- Midterm evaluation period

6.July 15 – August 1

- Retrain best model(s) and CNNs under perturbed a dataset (masked images, noise-added images etc) in order to benchmark robustness and generalizability of both classes of models on the given task
- Discuss the final steps and improvements to be made with the mentors

7. August 1 – 8

- Model quantization
- Refactoring and code cleaning in order to integrate the code with the CMS framework
- Versioning and integration of the code with the CMS framework

8. August 8- August 21

- Documentation of code
- Report writing
- Also serves as a buffer period in case of any unexpected delays
- Closing meet with mentors.

5. Relevant QnA

5.1 Why am I the best fit for the role?

ML4SCI is at the very top of my list of prioritised organizations when it comes to GSOC 2023 – The reason is the harmonious dichotomy between the challenge this project presents and the fact that I possess the best combination of skills required to meet said challenge. The latter is corroborated by past experience as a Machine Learning Engineer in the tech industry, a student researcher in academia and my stellar academic record.

As a researcher at a premier institute like the Indian Institute of Technology, Bombay, I have experience building a custom image processing algorithm on large image datasets in order to automate solar plage segmentations, achieving impressive results, thereby reducing the workload on astronomers and ensuring that their time and energy is better spent on their research. In the same role at IIT Bombay, I was responsible for creating a novel Bayesian Neural Network for galaxy classification, once again coming to the aid of astronomers by automating this task and quantifying any uncertainty in the classification. Both projects lead to co-authored papers with the abstracts being selected at the Astronomical Society of India conference (ASI2023_221 and ASI2023_751). The objectives of these projects not only align with the proposed GSOC project, where we seek to automate particle identification in order to simplify the research pipeline of scientists at CMS, but also demonstrate theoretical and technical competence at understanding and interpreting complex mathematical ideas and abstracting them at an implementation-level in the form of clean code.

As a student researcher at another premier institute, IIT Goa, I worked on a project to implement a research paper from scratch in TensorFlow 2.0, modify the architecture and adapt it for image recovery. This is evidence of the fact that I am proficient at implementing State Of The Art architectures as modular code and using them to drive insights and derive value from data – the cornerstone of any worthwhile machine learning project.

As a Machine Learning Engineer at a B2B tech company, I had the opportunity to take and pass the Google Cloud Professional Machine Learning Engineer exam, which is a certification exam recommended for industry professionals with over 3 years of experience, which I completed in my 7th month in the industry. I also worked on deploying object detection models for the purpose of document intelligence in production. Further, in the evaluation tasks, not only did I meet the listed requirements, but also took the time to include software engineering concepts in my code such as code separation, compute-storage separation, stream from disk data pipelines, data compression, experiment logging and checkpointing, modularisation, clean code and test and compare the robustness of ViTs and CNNs to perturbations in the specific task. The panoply of aforementioned achievements seeks to ratify a great appreciation of fundamental concepts on my part and the ability and passion to go beyond the norm and do more than what is warranted.

This GSOC research project requires a confluence of theoretical and technical competence, interest and passion that my past experience and personal projects coupled with this detailed proposal, will adequately justify.

Thus, it would be a great privilege and honour to be able to work under the expert mentors at ML4SCI and contribute to the scientific community for I, yearn to learn.

5.2 Availability and commitments

- I am available throughout the GSOC period and can easily clock 175 hours in the stipulated time frame. Any extension to the hours can be accommodated easily as well.
- I have no conflicting commitments.
- Meetings can be set up on any platform weekly/every 2 weeks as per the mentors' convenience
- I prefer English as mode of communication for this project.
- I am residing in India: Time zone – GMT +5:30

6. References

[1] Aad, G., Abajyan, T., Abbott, B., Abdallah, J., Abdel Khalek, S., …, & Zucchelli, S. (2012). Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Physics Letters B, 716(1), 1-29.

[2] Keiron O'Shea, Ryan Nash. An Introduction to Convolutional Neural Networks.

[3], [9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

[4], [7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need.

[5] The Inductive Bias of ML models and why you should care about it. (Web reference)

[6] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S. Awwal, Vijayan K. Asari. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches.

[8] Vision Transformer for Image Classification (Web reference)

[10] Gang Li, Di Xu, Xing Cheng, Lingyu Si, Changwen Zheng. SimViT: Exploring a Simple Vision Transformer with sliding windows.

[11] Rui Yang, Hailong Ma, Jie Wu, Yansong Tang, Xuefeng Xiao, Min Zheng, Xiu Li. ScalableViT: Rethinking the Context-oriented Generalization of Vision Transformer.

[12] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, Jiashi Feng. DeepViT: Towards Deeper Vision Transformer.

[13] vit-pytorch library (Web reference)

[14] PyTorch library (Web reference)

[15] Weights and Biases (Web reference)

[16] PyTorch profiler (Web reference)

[17] PyTorch Quantization (Web reference)