Mid-Term Project Report: Vision Transformer Implementation

Project: Winter Project 2025-26

Domain: Deep Learning & Computer Vision

Mentee: Mannat Shrivastav (Roll No. 240627)

1. Executive Summary

This report outlines the progress achieved during the first half of the Vision Transformer (ViT) project. The core objective of this phase was not to jump straight into the Transformer architecture, but to build the mathematical and software foundations required to understand it. Over the course of five assignments, I have progressed from building basic data pipelines to implementing Recurrent Neural Networks (RNNs) for sequence generation. This "bottom-up" approach—building algorithms from scratch without relying on high-level libraries—has provided me with a granular understanding of how models process both spatial (image) and temporal (sequence) data.

2. Technical Implementation Roadmap

The coursework was structured into weekly modules, each bridging a gap between basic coding and advanced deep learning.

Week 0: Data Engineering & Python Architecture

Before training models, the focus was on establishing a robust pipeline for data ingestion. The resources covered **Pandas** for dataframe manipulation and **Matplotlib** for exploratory analysis.

Assignment 1 Implementation:

I developed the preprocessing backend for the project. Instead of using standard PyTorch datasets, I implemented a custom object-oriented structure:

- **DataSample Class:** I created a Python class to encapsulate features and labels, ensuring data consistency.
- **Manual Normalization:** I wrote a min_max_norm algorithm from scratch. By scaling input features to the $[0, 1]$ range using the formula $x' = \frac{x - \min}{\max - \min}$, I ensured numerical stability for future training.
- **Batching Logic:** I implemented a generator to slice the dataset into batches of size 50. This is critical for Stochastic Gradient Descent (SGD), as loading the entire dataset into RAM is often infeasible.

Week 1: Neural Networks from First Principles

This week shifted focus to the calculus of Deep Learning. The goal was to demystify the "black box" of neural networks.

Assignment 2 Implementation:

I constructed a Multi-Layer Perceptron (MLP) using only NumPy (no Autograd/PyTorch) to classify MNIST handwritten digits.

- **Backpropagation:** I manually derived the gradients for the Chain Rule. This involved calculating the partial derivatives of the Loss function with respect to weights (dW) and biases (db) and propagating them through Softmax and ReLU activation functions.
- **Optimization:** I implemented the weight update rule hardcoded as `params -= learning_rate * grads`.
- **Result:** The model successfully learned to recognize digits (28×28 pixel inputs), confirming the correctness of my calculus derivation.

Week 2: Theoretical Deep Learning

This phase was analytical, focusing on the hyperparameters that govern model convergence.

Assignment 3 Analysis:

I tackled theoretical problems regarding the stability of training:

- **Regularization:** I analyzed the mechanics of **Dropout** ($p=0.5$). By randomly zeroing out activations during training, we force the network to learn redundant features, which prevents overfitting.
- **Optimizer Comparison:** I compared **Adam** against standard **SGD**. My analysis highlighted how Adam's momentum components (m_t, v_t) allow it to accelerate through flat regions of the loss landscape where SGD typically gets stuck.

Week 3: Convolutional Neural Networks (CNNs)

(Note: Bridging the gap between MLPs and Sequence models)

Before moving to Transformers, I studied the architecture they aim to replace. I analyzed how Convolutional layers utilize learnable kernels to extract spatial hierarchies (edges $\to$ textures $\to$ objects), and how Max Pooling introduces translation invariance, allowing the model to recognize objects regardless of their position in the frame.

Week 4: Sequence Models & RNNs

Since Vision Transformers treat image patches as sequences, understanding sequence modeling was the final prerequisite.

Assignment 5 Implementation: "Dinosaur Island"

I implemented a character-level Recurrent Neural Network (RNN) to generate text.

- **The Task:** The model was trained on a list of dinosaur names to generate new, unique names.
- **Gradient Clipping:** A major challenge I faced was the "exploding gradient" problem inherent in RNNs. I solved this by implementing np.clip() to cap gradients between $[-5, 5]$ during backpropagation.
- **Sampling:** I wrote a function to sample characters based on the probability distribution output by the RNN.
- **Outcome:** The model learned the statistical structure of the language, generating valid-sounding names like "Mangosaurus" and "Toraptor."

3. Key Learnings & Reflection

This mid-term phase has been instrumental in shifting my perspective from simply "using" libraries to "building" them.

1. **Matrices matter:** Implementing the MLP from scratch taught me that efficient Deep Learning is essentially optimized matrix multiplication.
2. **Sequence Dependencies:** The RNN assignment highlighted the difficulty of retaining long-term dependencies, a limitation that the Self-Attention mechanism (our next step) is designed to solve.
3. **Data Discipline:** The Week 0 assignment reinforced that a model is only as good as its input pipeline; without the DataSample class and normalization, the subsequent models would likely fail to converge.