MID-TERM EVALUATION REPORT
VISION TRANSFORMER PROJECT

SHAURYA VATS
240969

Table of Contents:

1. Introduction

This report covers my progress in the Vision Transformer winter project up to the mid-term evaluation. Vision Transformers are an interesting new approach to computer vision that's quite different from what we've traditionally used. Instead of relying on convolutions like most image recognition systems, they treat images more like text—breaking them into patches and using attention mechanisms to understand relationships between different parts of the image.

The project is structured to start from the ground up. We began with Python programming and basic machine learning concepts, then gradually moved through neural networks, CNNs, and recurrent architectures. Each week built on the previous one, and the assignments helped solidify what we learned in theory. By the time we get to Vision Transformers, we'll have a solid understanding of why they work and how they compare to traditional methods.

2. Week 1: Getting Started with Python and Machine Learning

The first week was all about setting up the right foundation. We started with Python and three main libraries: NumPy for mathematical operations on arrays, Pandas for working with tabular data, and Matplotlib for creating visualizations.

Basic Machine Learning Ideas: We covered supervised learning, where you train a model using examples that already have correct answers. Simple linear models work for straightforward problems, but neural networks can model complicated relationships by stacking multiple layers together.

Important Building Blocks: Activation functions add non-linearity to neural networks. ReLU is the most common one for hidden layers, while Sigmoid and Softmax are used for output layers. Loss functions like Mean Squared Error and Cross-Entropy tell you how wrong your predictions are. Gradient Descent is how the model learns—it calculates which direction to adjust parameters to reduce errors. The learning rate controls how big those adjustments are.

3. Week 2: Understanding Neural Networks

Week two went deeper into how neural networks actually learn through backpropagation. When you feed data into a network, it goes through multiple layers (forward propagation) and produces a prediction. Backpropagation then figures out how

much each parameter contributed to the error and adjusts them accordingly using the chain rule.

Training Considerations: Getting the learning rate right is crucial—too high and training becomes unstable, too low and it takes forever. We learned about preventing overfitting through L1/L2 regularization and Dropout, which randomly "turns off" some neurons during training to force the network to learn more robust features. Understanding matrix dimensions and debugging implementation issues turned out to be just as important as the theory.

4. Week 3: Convolutional Neural Networks

CNNs are what really made deep learning successful for image recognition. Using regular fully connected networks for images is impractical because even small images would need millions of connections. Images have spatial structure that regular networks completely ignore.

How Convolution Works: CNNs use small filters that slide across the image, looking for specific patterns. Because the same filter is reused across the entire image, you need way fewer parameters. This "weight sharing" makes CNNs efficient. Kernel size, stride, and padding are important parameters that control how the filters work.

Additional Techniques: Pooling layers shrink the spatial dimensions by taking maximum or average values in small regions. Batch Normalization normalizes values passing through each layer, which stabilizes training and allows higher learning rates. CNNs learn hierarchically—early layers detect simple features like edges, middle layers combine these into textures, and deep layers recognize object parts and complete objects.

5. Week 4: Working with Sequential Data

Week four shifted focus to data where order matters, like text or time series. Sequential data unfolds over time, and the meaning of each element depends on what came before it. Regular neural networks process inputs independently and can't handle temporal dependencies.

Recurrent Neural Networks: RNNs maintain a "hidden state" that acts like memory. At each time step, the network looks at the current input and previous hidden state, then updates the state and makes a prediction. This carries information forward through the sequence.

Training Difficulties: Training RNNs uses Backpropagation Through Time, which involves unrolling the network across all time steps. Gradients have to flow backward through many steps and can either vanish or explode. Gradient clipping caps the gradient if it gets too big, while truncated BPTT limits how far back you calculate gradients.

6. Week 5: Advanced Recurrent Models

The final week covered more sophisticated architectures that address the limitations of basic RNNs.

Long Short-Term Memory Networks: LSTMs have a "cell state" and three "gates" that control information flow. The forget gate decides what to discard, the input gate decides what to store, and the output gate decides what to use for predictions. This lets LSTMs learn patterns spanning hundreds of time steps.

Gated Recurrent Units: GRUs simplify LSTMs by combining some gates and merging the cell state with the hidden state. Despite being simpler, they often work just as well and train faster.

Advanced Architectures: Deep RNNs stack multiple recurrent layers to increase learning capacity. Bidirectional RNNs process sequences in both directions to get context from past and future. The encoder-decoder framework maps one sequence to another—the encoder compresses the input into a context vector, and the decoder generates the output. This architecture is conceptually similar to how transformers work, which is what Vision Transformers are based on.