Computational Architectures for Visual Recognition: EEA Winter project Vision Transformer Mid Eval

1. Introduction

This report sums up exactly what I've covered so far, starting from the very basics of Python and data manipulation, moving through the fundamentals of Machine Learning, and finally diving into complex architectures like Convolutional Neural Networks (CNNs) and Sequence Models.[1]

Week 0: Getting My Hands Dirty with Data

I started the project by setting up a robust environment in Google Colab, focusing on the libraries that act as the backbone for everything that followed.[1]

- **Python & Environment:** I brushed up on my Python basics, particularly data structures like lists and dictionaries, which helped me write much more modular and reusable code.[1]
- **Scientific Computing (NumPy & Pandas):** I learned how to use NumPy for high-performance math on multi-dimensional arrays—slicing and vectorization became essential for processing image tensors later on.[1] With Pandas, I practiced cleaning and preprocessing "messy" real-world data.[1]
- **Visualization (Matplotlib):** I used plotting to make sense of the data distributions. Seeing a histogram of features helped me catch anomalies before they could mess up a model.[1]

Assignment 1 Reflection: For my first assignment, I worked on partitioning a dataset into 17 batches. I used tabular previews to verify that my data was segmented correctly and that the feature integrity remained intact before I moved on to the analysis.[1]

Week 1: Bridging the Gap to Intelligent Systems

In Week 1, I moved from simple coding to learning how models actually "think." It was a deep dive into the math behind the magic.[1]

- **Supervised/Unsupervised Learning:** I explored Linear and Logistic Regression for continuous and categorical predictions. I also touched on K-Means Clustering, which was an eye-opener for finding patterns in unlabeled data.[1]

- **Core Dynamics:** I learned that non-linear activations like **ReLU** and **Sigmoid** are what allow a network to learn truly complex patterns.[1] To quantify how well the model was doing, I used **MSE** and **Cross-Entropy** loss functions.[1]
- **Optimization & Regularization:** I studied how backpropagation updates parameters. Using advanced optimizers like **Adam** really sped up the process.[1] I also applied **Dropout** to prevent my models from just "memorizing" the training data.[1]

Assignment 2 Reflection: I built a neural network for the MNIST dataset. It was satisfying to see the model correctly classify handwritten digits by matching its predictions to the ground truth.[1]

Week 2: Deep Dive into Computer Vision

This week, I shifted my focus to images. I quickly realized that standard "dense" networks just aren't efficient enough for visual data.[1]

- **The Convolution Operation:** I learned to use learnable kernels to extract local features like edges and textures. This approach is much better because it respects the spatial structure of an image.[1]
- **Architecture & Hyperparameters:** I experimented with **Stride**, **Padding**, and **Kernel Size** to see how they changed my feature maps. I also used Max Pooling to keep the most important info while reducing the computational load.[1]
- **Stability & Acceleration:** I integrated **Batch Normalization**, which was a game-changer for stabilizing the learning process and making my deep networks train much faster.[1]

Assignment 3 Reflection: By comparing a CNN to a fully connected network, I saw firsthand how much better CNNs are at preserving spatial hierarchies for visual recognition.[1]

Week 3 & 4: Persistence and the Flow of Time

I then moved into the world of sequential data, where the order of information is everything.[1]

- **Recurrent Neural Networks (RNNs):** I started with standard RNNs but quickly learned about their limitations, specifically the "Vanishing Gradient" problem that makes them "forget" long-term dependencies.[1]
- **Memory Cells (LSTM/GRU):** I explored **LSTMs** and **GRUs**, which use clever gating mechanisms to decide what information to keep and what to forget. This solved the memory issues I saw with vanilla RNNs.[1]
- **Generative Modeling:** I looked at how to tokenize text and predict the next item in a sequence, which is the foundation for models that can actually "write".

Assignment 5 Reflection: I built a character-level RNN to generate dinosaur names. After the loss curves stabilized, it started generating names like *Trodolosaurus*, which showed me it had successfully learned the phonetic patterns of the data.[1]

Week 5: The Cutting Edge—Vision Transformers (ViT)

Finally, I explored a huge paradigm shift: moving away from convolutions and toward "attention".

- **Patch-Tokenization:** Instead of looking at an image pixel-by-pixel, ViT splits it into patches and treats them like words in a sentence.
- **Self-Attention Mechanism:** I learned how **Multi-Head Self-Attention (MSA)** allows the model to look at every part of the image at once to capture global relationships—something CNNs struggle to do in their first few layers.
- **Foundation Models:** I analyzed how these concepts scale up to incredible tools like **CLIP** and the **Segment Anything Model (SAM)**, which use ViT encoders to understand almost any visual prompt.

7. Conclusion

Over these past five weeks, I've gone from writing basic Python scripts to building models that can "see" and "write." The journey from simple data handling (Week 0) to complex spatial and temporal modeling (Weeks 2-4) has given me a solid foundation.[1] I'm now ready to dive deeper into the world of Transformers (Week 5), which are fundamentally changing how we approach Computer Vision by combining it with the power of global self-attention.