# Assignment 2: Debugging Vision Transformers (ViT) with PyCharm and WSL

## Objective

This assignment extends your debugging skills to Vision Transformers (ViTs). You will trace how an input image is patchified, embedded, encoded, and classified. You must inspect tensor shapes at every processing step and provide authentic evidence through debugger snapshots.

The goals are:

* To reinforce your ability to use PyCharm + WSL for debugging deep learning models.
* To ensure you fully understand how a **vanilla Vision Transformer** processes images step by step.
* To enforce authenticity through numbered, labeled snapshots with shapes and values.

## Task Instructions

### 1. Environment Setup

* Use the same PyCharm + WSL setup as in Assignment 1.
* Verify that you can load and debug a ViT model inside PyCharm.

### 2. Transformer Architecture Requirement

* All students must use the **standard vanilla Vision Transformer (ViT)** model.
* Use a reduced but consistent configuration for debugging: e.g., **patch size 16, embedding dimension 768, 12 attention heads, 12 encoder blocks, ImageNet classification head (1000 classes)**.
* This ensures all snapshots follow the same structure while keeping inputs unique.

### 3. Input Image Selection

* Each team must select a **unique input image** (not reused across teams).
* Rules for uniqueness:
  + **Source**: Must come from a public dataset (CIFAR-10, ImageNet sample, COCO, LaSoT), a royalty-free source, or a personal photo.
  + **Content**: Image must clearly show a **single main object** (animal, vehicle, tool, household item, etc.), with minimal background clutter.
  + **Preprocessing**: Convert to RGB, resize to **224 × 224**, normalize with ImageNet mean/std, and format as **(1, 3, 224, 224)** tensor.
* **Proof Requirement**:
  + Include the original image in your report.
  + Snapshot #1 must show the raw preprocessed tensor.
* **Uniqueness Rule**: All teams must **register their chosen image** in the shared spreadsheet to prevent duplicates.

### 4. Debugging Task

* Use PyCharm’s debugger to step through the ViT forward pass.
* Record tensor shapes and representative values at each stage.
* Do not use print statements.
* Each snapshot must include:
  + Snapshot number and name.
  + Shape of the tensor.
  + A slice of actual values.
  + A 1–2 sentence explanation.

### 5. Documentation

* Capture and annotate all required snapshots.
* Follow the numbering exactly.
* Include your original input image and preprocessing details.

## Mandatory Snapshots

### Input and Patchification

1. Raw input image tensor (after preprocessing).
2. Image divided into patches (before flattening).
3. Flattened patches (reshaped into vectors).

### Embedding and Tokens

1. Patch embeddings after linear projection.
2. Class token before concatenation.
3. Embeddings after adding the class token.
4. Embeddings after adding positional encoding.

### Encoder Block (Trace One Block)

1. Encoder block input tensor.
2. Multi-head attention queries (Q).
3. Multi-head attention keys (K).
4. Multi-head attention values (V).
5. Attention scores before softmax.
6. Attention scores after softmax.
7. Multi-head attention output (after concatenation).
8. Residual connection + normalization (post-attention).
9. Feed-forward input.
10. Feed-forward hidden layer output.
11. Feed-forward output after second linear.
12. Residual connection + normalization (post-MLP).
13. Encoder block final output.

### Deeper Encoder Blocks

1. Encoder block 2 output.
2. Encoder block N (last block) output.

### Final Output

1. Final sequence output (including class token).
2. Class token extracted (final representation).
3. Classification head logits.
4. Softmax probabilities (example slice).

## Guiding Questions

* Why must the image be split into patches before embedding?
* Why is a class token added, and how does it affect the shape?
* Why are positional encodings needed in ViT?
* Why do Q, K, V have the same dimensions, and how do attention weights scale with patch count?
* How do residual connections preserve shape consistency across encoder blocks?
* Why is only the class token used for the final classification?

## Submission

Submit a single Word file that includes:

* Introduction about your environment setup.
* The selected input image and preprocessing explanation.
* All 26 numbered snapshots with annotations.
* Answers to the guiding questions.
* A reflection paragraph on how debugging improved your understanding of Vision Transformers.

## Sample Data Format for Students

### Snapshot 1 – Raw Input Image Tensor

Shape: (1, 3, 224, 224)

**Explanation:** Batch of size 1, with 3 channels (RGB), resized to 224×224.

### Snapshot 3 – Flattened Patches

Shape: (1, 196, 768)

**Explanation:** The 224×224 image is split into 16×16 patches → 196 patches. Each is flattened and projected into a 768-d vector.

### Snapshot 7 – Embeddings After Adding Positional Encoding

Shape: (1, 197, 768)

**Explanation:** 196 patch embeddings + 1 class token = 197 tokens. Each has 768 dimensions.

### Snapshot 13 – Attention Scores After Softmax

Shape: (1, 12, 197, 197)

**Explanation:** For 12 heads, each of the 197 tokens attends to all 197 tokens.

### Snapshot 25 – Classification Head Logits

Shape: (1, 1000)

**Explanation:** Final linear projection maps class token to 1000 classes (e.g., ImageNet).

## Grading (100%)

* Environment setup & correctness (20%)
* Debugger usage (20%)
* Shape tracking & accuracy (30%)
* Documentation clarity (20%)
* Reflection & originality (10%)