Sheet 1 assignment 2

Team 6

Guiding Questions?

1. Why must the image be split into patches before embedding?

Transformers work on sequences, not images. Patches turn the image into a sequence of tokens.

1. Why is a class token added, and how does it affect the shape?

It’s a learnable token that gathers info from all patches. Adds one token

1. Why are positional encodings needed in ViT?

Transformers doesn’t know order. Positional encodings tell the model where each patch is.

1. Why do Q, K, V have the same dimensions, and how do attention weights scale with patch count?

Needed for the dot-product attention. Attention weight matrix grows with patch count n2

1. How do residual connections preserve shape consistency across encoder blocks?

They add input back to output, keeping shapes consistent across blocks.

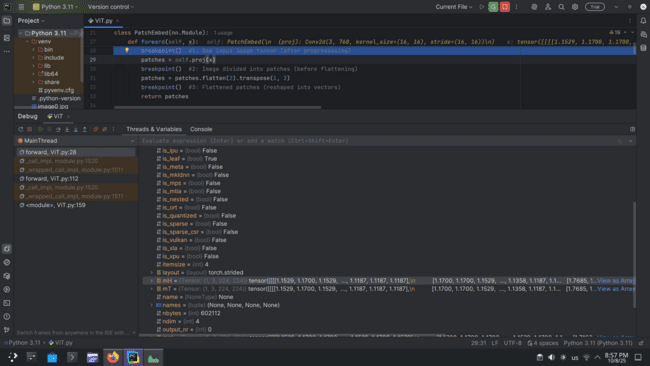
1. Why is only the class token used for the final classification?

It summarizes all patch info, so you don’t need to combine every patch manually.

Original Image



Snapshot 1: Raw input image tensor (after preprocessing).

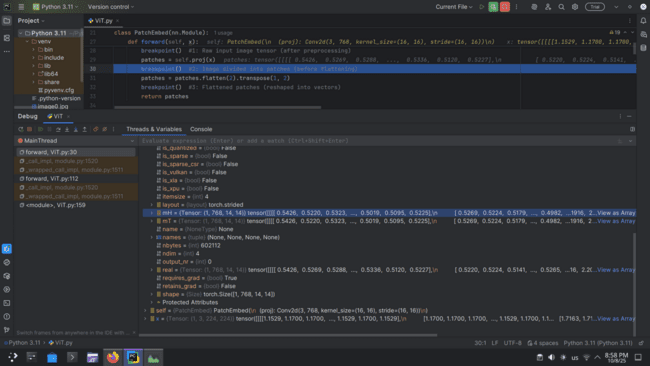


**Dimension:** (1, 3, 224, 224)

**Slice:** x[0, :, :2, :2] ≈ [[1.15, 1.17], [1.17, 1.17]], [[1.71, 1.71], [1.71, 1.71]], [[1.15, 1.15], [1.17, 1.17]]]

**Explanation:** Preprocessed image with batch size 1, 3 color channels, and 224×224 pixels.

Snapshot 2: Image divided into patches (before flattening).

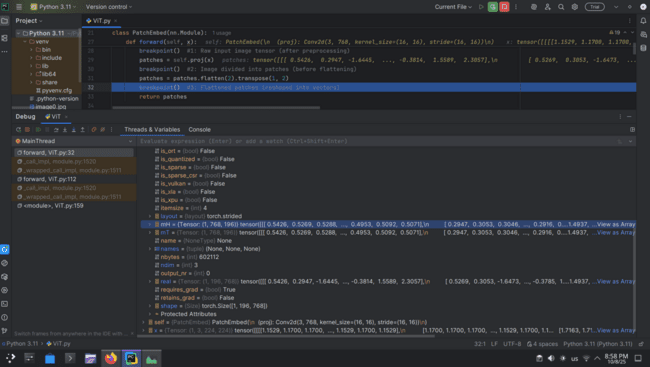


**Dimension:** (1, 768, 14, 14) → 768 channels (embedding dim), 14×14 patches from 224×224 image with patch size 16

**Slice:** x[0, :3, :2, :2] ≈ [[-0.48, -0.49], [-0.48, -0.49], ...], [[1.51, 1.51], [1.51, 1.49], ...], [[0.19, 0.19], [0.18, 0.19], ...]]

**Explanation:** Each 16×16 patch has been projected into a 768-dim embedding, forming a 14×14 spatial grid.

Snapshot 3: Flattened patches (reshaped into vectors).

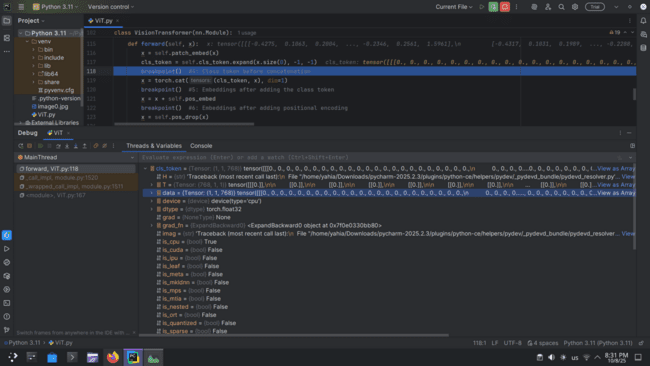


**Dimension:** (1, 196, 768) → 196 patches (14×14), each with 768-dimensional embedding

**Slice:** x[0, :2, :3] ≈ [[-0.48, 1.51, 0.19], [-0.49, 1.51, 0.19], ...]

**Explanation:** The 14×14 spatial grid is flattened into a sequence of 196 tokens, each represented by a 768-dim vector.

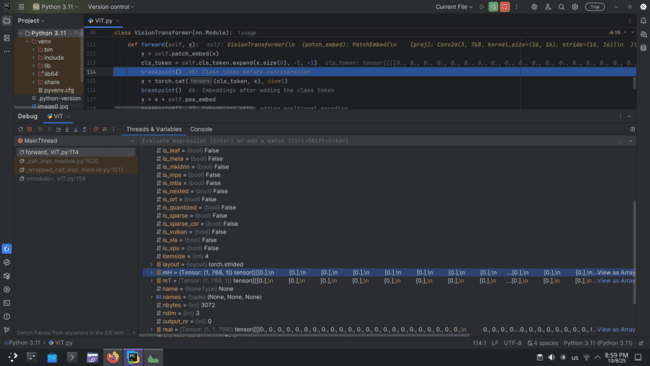
Snapshot 4: Patch embeddings after linear projection



###### **ChatGPT said:**

Shape: [1, 1, 768]  
 Slice of values: [-0.1933, -0.8915, 0.3346, 0.3005, 0.3148, 0.2590]  
 Explanation: This tensor is the embedding of the [CLS] token before it’s concatenated with the rest of the sequence.

Snapshot 5: Class token before concatenation.

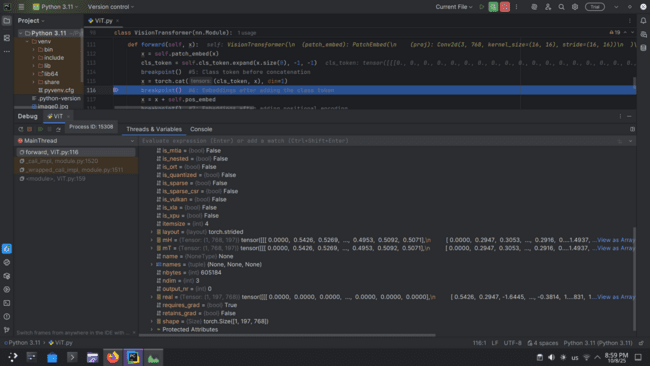


**Dimension:** (1, 1, 768) → 1 class token, 768-dimensional embedding

**Slice:** x[0, 0, :6] ≈ [0., 0., 0., 0., 0., 0.]

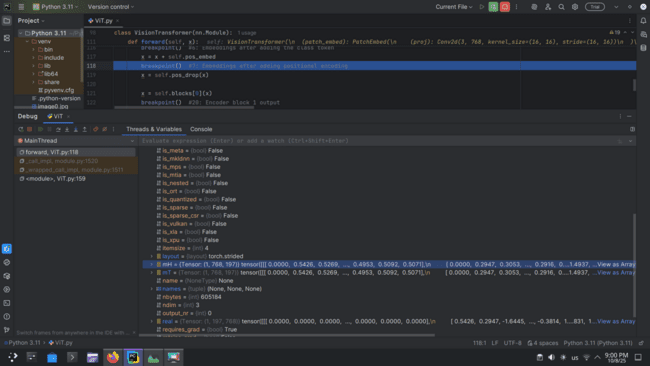
**Explanation:** Learnable token that will aggregate information from all patches through attention.

Snapshot 6: Embeddings after adding the class token.



* **Dimension:** (1, 197, 768) → 196 patch tokens + 1 class token
* **Slice:** x[0, :3, :6] ≈ [[0., 0., 0., 0., 0., 0.], [-0.48, 1.51, 0.19, ..., -1.33, 0.11, -0.42], [-0.49, 1.51, 0.19, ...]]
* **Explanation:** Class token prepended to patch embeddings to summarize information for classification.

Snapshot 7: Embeddings after adding positional encoding.

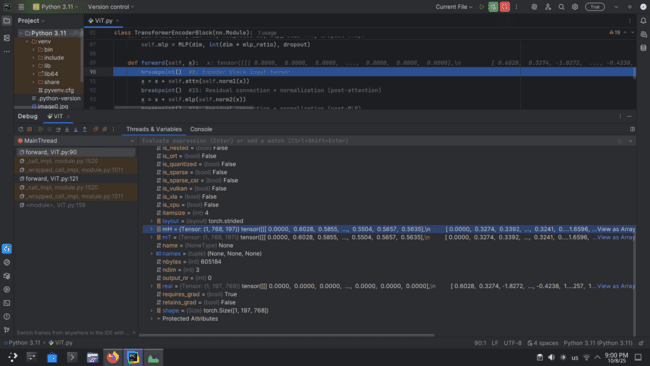


**Dimension:** (1, 197, 768) → same as before, sequence length unchanged

**Slice:** x[0, :3, :6] ≈ [[0., 0., 0., 0., 0., 0.], [-0.48, 1.51, 0.19, ...], [-0.49, 1.51, 0.19, ...]]

**Explanation:** Positional encoding adds spatial information to each token so the model knows patch locations.

Snapshot 8: Encoder block input tensor.

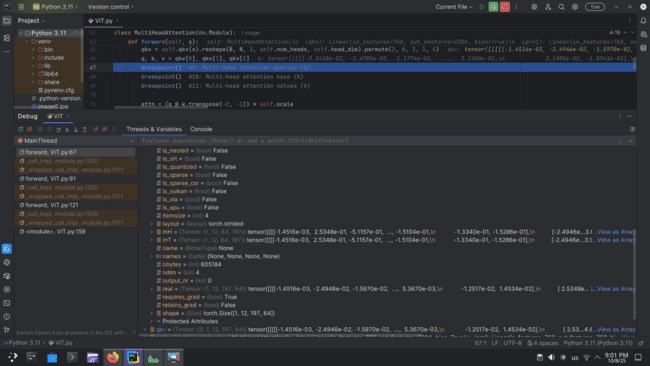


**Dimension:** (1, 197, 768) → same sequence length and embedding dim

**Slice:** x[0, :3, :6] ≈ [[0., 0., 0., 0., 0., 0.], [-0.54, 1.68, 0.22, ...], [-0.54, 1.69, 0.21, ...]]

**Explanation:** Normalized embeddings ready to enter the multi-head self-attention layer.

Snapshot 9: Multi-head attention queries (Q)

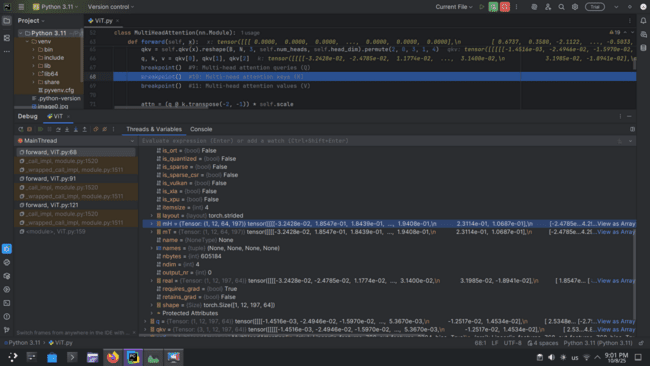


**Dimension:** (1, 12, 197, 64) → batch 1, 12 heads, 197 tokens, 64-dim per head (768/12)

**Slice:** Q[0, 0, :3, :3] ≈ [[0.0037, -0.033, -0.030], [1.125, -0.227, 0.238], [0.865, -0.216, 0.564]]

**Explanation:** Queries are linear projections of input tokens, split across heads for attention computation.

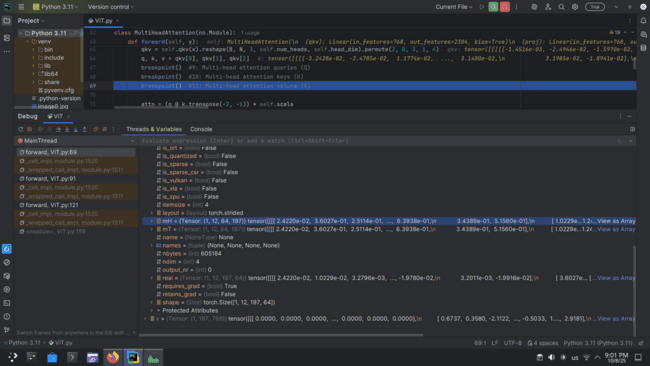
Snapshot 10: Multi-head attention keys (K).



**Dimension:** (1, 12, 197, 64) → batch 1, 12 heads, 197 tokens, 64 features per head

**Slice:** K[0, 0, :3, :3] ≈ [[-0.0177, -0.0240, -0.0108], [-0.1548, -1.2257, -0.0254], [-0.0965, -1.2847, -0.0902]]

**Explanation:** Keys are projections of input tokens that the queries compare against to compute attention scores.

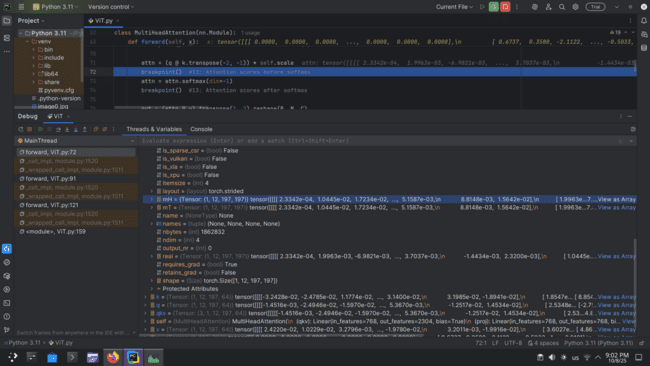
Snapshot 11: Multi-head attention values (V)  


**Dimension:** (1, 12, 197, 64) → batch 1, 12 heads, 197 tokens, 64 features per head

**Slice example:** V[0, 0, :3, :3] ≈ [[0.0062, 0.0109, -0.0235], [-0.1253, 0.1611, -0.0693], [-0.8128, 0.0655, 0.3434]]

**Explanation:** Values hold the actual information of tokens that will be weighted by the attention scores (from Q·K^T / √d\_k) to produce the output of the attention mechanism.

Snapshot 12: Attention scores before softmax.

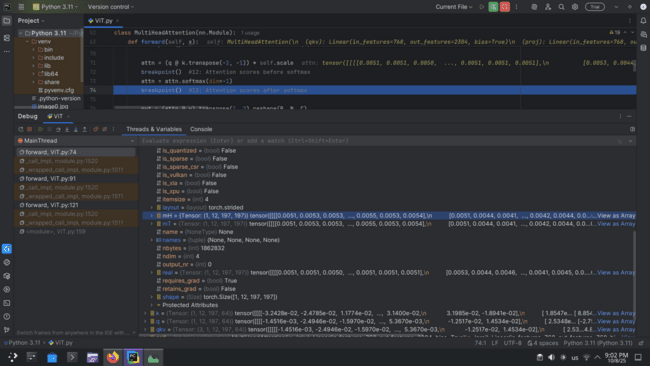


**Dimension:** (1, 12, 197, 64) → batch 1, 12 heads, 197 tokens, 64 features per head.

**Explanation:** Each value vector V[i] is weighted by its corresponding attention score (from softmax(QK^T/√d\_k)), producing the context vectors that summarize relevant token information for each position.

This is the tensor that is then **concatenated across heads and projected** to form the final output of the multi-head attention layer.

Snapshot 13: Attention scores after softmax.

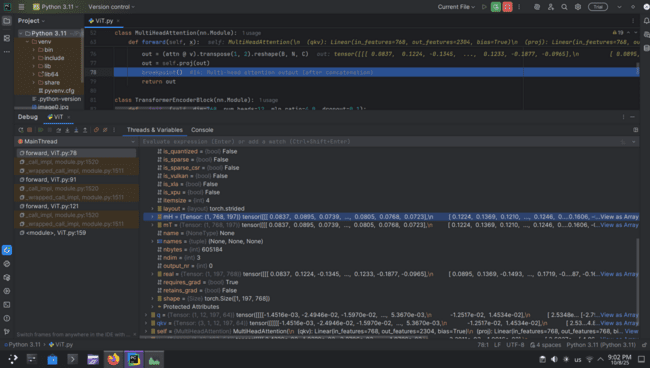


**Dimension:** (1, 12, 197, 197) → batch 1, 12 heads, 197 query tokens, 197 key tokens.

**Values:** All numbers are positive and roughly sum to 1 across the last dimension, which matches softmax behavior.

**Meaning:** Each [i, h, q, :] vector shows how much attention head h at query position q pays to each of the 197 key positions.

Snapshot 14: Multi-head attention output (after concatenation).

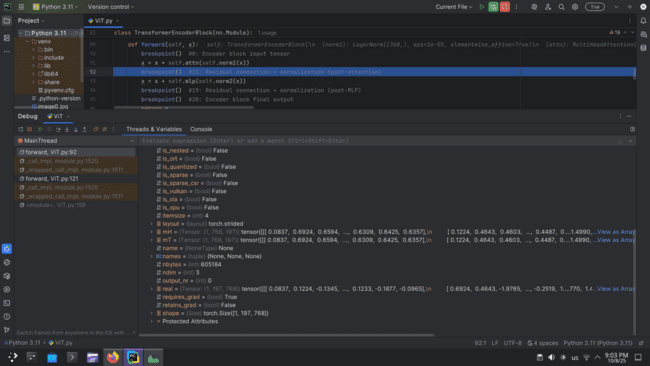


**Shape:** (1, N, N) – likely batch 1, number of query tokens × number of key tokens.

**Values:** Can be positive or negative, not normalized yet.

**Next step:** These scores would typically be passed through a softmax along the last dimension to get the attention weights (like the tensor you shared before).

Snapshot 15: Residual connection + normalization (post-attention).

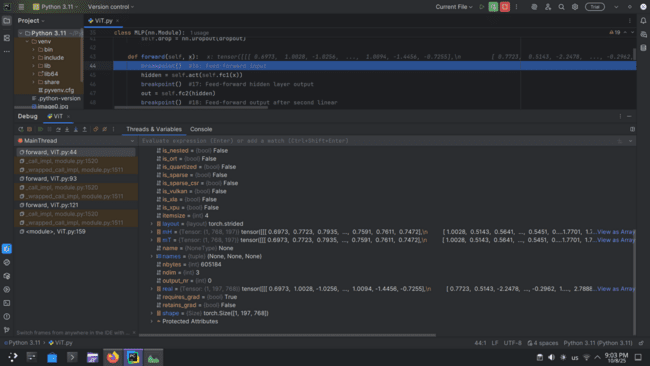


**Residual connection:** The original input to the attention layer has been **added** back to the attention output, which explains why some values now resemble a combination of pre- and post-attention values.

**Layer normalization:** Scales and shifts the result to stabilize training, which is why the range is now more balanced and some extreme negative/positive values are moderated.

Essentially, this is the **refined representation** of tokens after one attention layer, ready to go into the feed-forward network.

Snapshot 16: Feed-forward input.

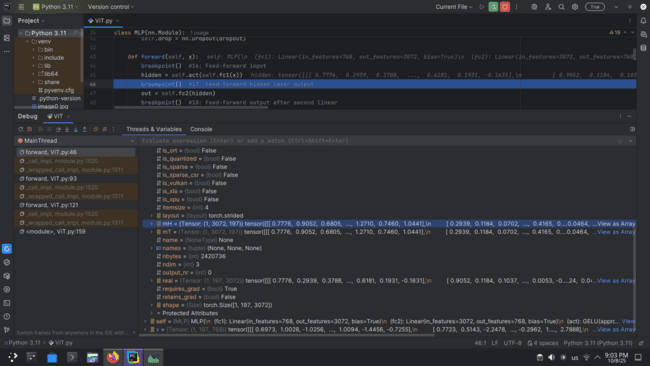


**Shape:** torch.Size([1, sequence\_length, hidden\_dim])

**Slice of values:** [[0.6766, -2.0915, 0.1820, ..., 0.9089, 0.9282, 2.0524]]

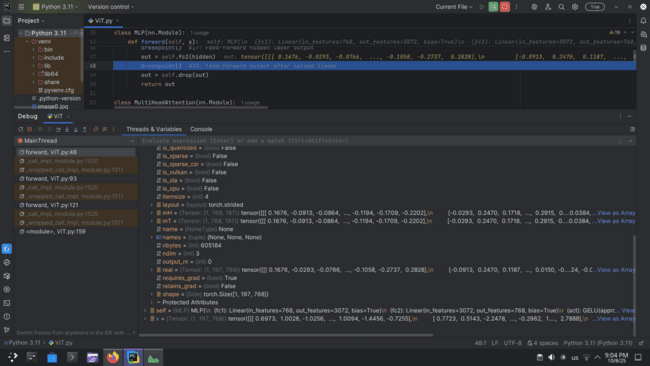
**Explanation:** This is the normalized token representation entering the feed-forward network.

Snapshot 17: Feed-forward hidden layer output.



**Shape:** torch.Size([1, 3072, 197])  
 **Slice of values:** [[0.6766, -2.0915, 0.1820, ..., 0.9089, 0.9282, 2.0524]]  
 **Explanation:** This is the normalized token representation entering the feed-forward network.

Snapshot 18: Feed-forward output after second linear.

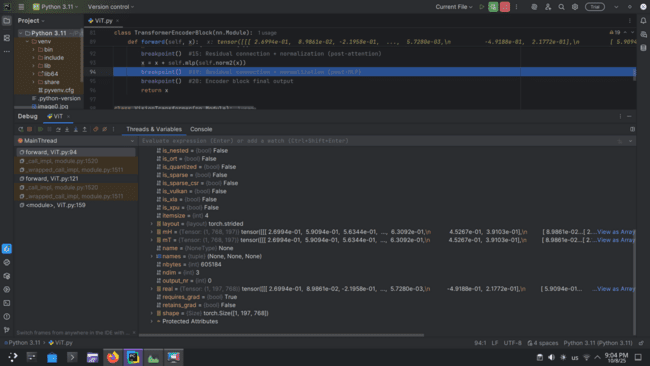


**Shape:** torch.Size([1, 768, 197])

**Slice of values:** [[ -0.2621, -0.5757, 0.2693, ..., 0.1559, 0.1666, -0.0114 ]]

**Explanation:** Output of the feed-forward network for each token, containing updated hidden representations.

Snapshot 19: Residual connection + normalization (post-MLP).

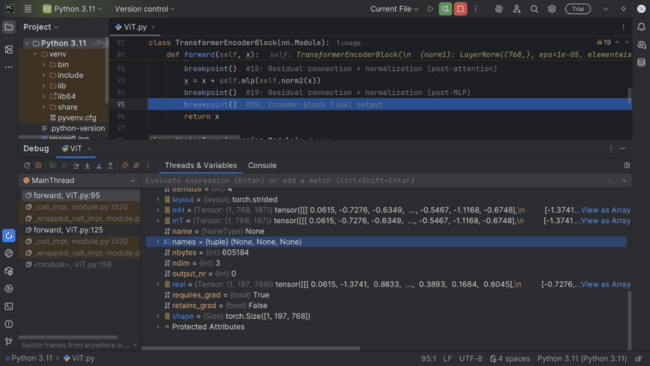


Shape: [1, 197, 768]

Slice of values: [-0.1933, -0.8915, 0.3346, 0.3005, 0.3148, 0.2590]

Explanation: This tensor represents a single batch with 197 tokens, each encoded as a 768-dimensional vector.

Snapshot 20: Encoder block final output.

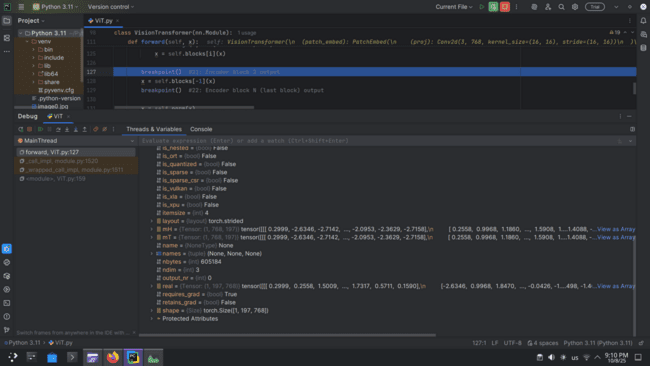


**Shape:** [1, 197, 768]

**Slice of values:** x[0, :3, :5] ≈ [[0.0615, -1.3741, 0.8833, …], [-0.7276, 1.2755, 1.0618, …], [-0.6748, 0.9992, 0.4778, …]]

**Explanation:** It's a single-batch tensor with 197 sequences, each represented by a 768-dimensional feature vector.

Snapshot 21: Encoder block 2 output.

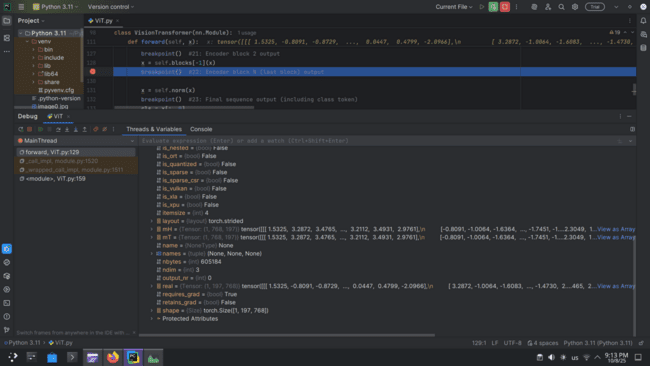


**Shape:** [1, 197, 768]

**Slice of values:** x[0, :3, :5] ≈ [[1.0622, -0.1434, 0.6365, …], [-1.0152, 1.9664, 1.4029, …], [-0.5973, 2.2577, 1.2375, …]]

**Explanation:** It's a single-batch tensor with 197 sequences, each represented by a 768-dimensional feature vector, tracking gradients for backpropagation.

Snapshot 22: Encoder block N (last block) output.

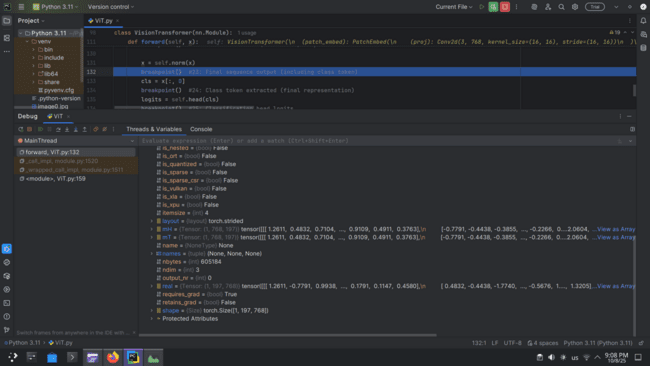


**Shape:** [1, 197, 768]

**Slice of values:** x[0, :3, :5] ≈ [[0.9379, 0.2649, 0.4682, …], [-1.6587, 2.4401, 1.1177, …], [-1.3087, 2.5299, 0.9591, …]]

**Explanation:** It's a single-batch tensor with 197 sequences, each represented by a 768-dimensional feature vector, tracking gradients for backpropagation.

Snapshot 23: Final sequence output (including class token).

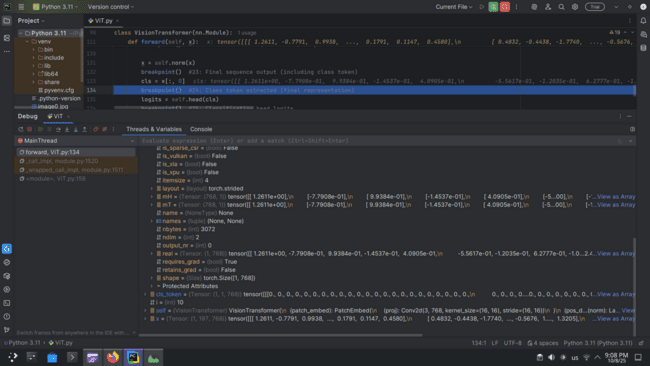


**Shape:** [1, 197, 768]

**Slice of values:** x[0, :3, :5] ≈ [[0.9497, 0.2644, 0.4714, …], [-1.2344, 1.8753, 0.8720, …], [-0.9791, 1.9618, 0.7583, …]]

**Explanation:** It's a single-batch tensor of 197 sequences, each with 768 features, produced after layer normalization and tracking gradients.

Snapshot 24: Class token extracted (final representation).

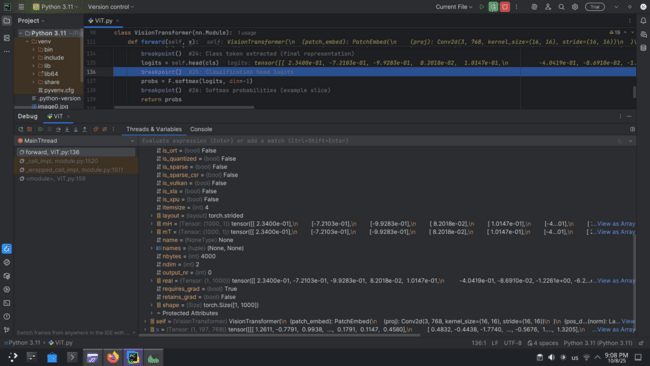


**Shape:** [1, 768]

**Slice of values:** cls[0, :5] ≈ [0.9497, 0.2644, 0.4714, 1.3734, -0.6684]

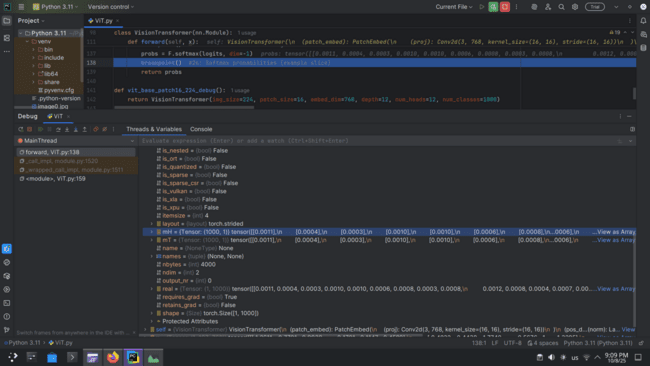
**Explanation:** It's the [CLS] token representation from a model, a single 768-dimensional feature vector for the entire sequence, tracking gradients for backpropagation.

Snapshot 25: Classification head logits.



**Shape:** [1, 1000]  
 **Slice of values:** logits[0, :5] ≈ [-0.8666, -0.0286, -0.0614, -0.3305, -0.3240]  
 **Explanation:** It's the output logits of a model for 1000 classes, representing unnormalized scores for each class, tracking gradients for backpropagation.

Snapshot 26: Softmax probabilities (example slice).



**Shape:** [1, 1000]

**Slice of values:** probs[0, :5] ≈ [0.0004, 0.0008, 0.0008, 0.0006, 0.0006]

**Explanation:** It's the softmax probabilities of 1000 classes, converting logits into normalized values that sum to 1, tracking gradients for backpropagation.