|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**6. Briefly explain the role of patch embedding and positional encoding in ViT. You may include parts of your Step 3 model code to support your explanation**

So, patch embedding is kinda like slicing a big pizza into smaller slices. In our code, we chop each 28×28 image into 4×4 squares (that’s 49 little patches), flatten each one into a list of numbers, and then run it through a linear layer to get a 64-element “token.” That way, the Transformer can handle images just like it handles words in a sentence.

Positional encoding is like giving each slice its own “address” so the model knows which slice came from the top-left or bottom-right. Without that, the model wouldn’t know the order — it’d be like reading words out of order and trying to understand a story. We just add a learned vector to each patch token so the model learns spatial location.

**7. Describe which hyperparameters you tuned, why you chose them, and how they affected your final accuracy.**

* **learning rate (LR):** I used 3×10⁻⁴ because it made training smooth. If I cranked it up, the model freaked out and accuracy bounced all over.
* **batch size:** I chose 64 since Colab GPU could handle it without slowing too much. Smaller batches were jittery, bigger batches got slow.
* **patch size:** 4 worked well — 7×7 patches gave enough detail without choking my computer.
* **dim (token size):** 64 felt just right. Smaller (like 32) learned slow; bigger (128) was a bit better but took forever.
* **depth & heads:** I used 6 layers and 4 heads. That combo reached around 84% accuracy. More layers or heads only helped a tiny bit but really slowed training.
* **epochs:** I trained for 30 epochs. By then I hit about 84%. I found that set 25 epochs can not reach to the highest ACC and If I went to 50, it just wiggled up and down (overfitting).

**8. Compare ViT and CNN for image classification: what are the main differences, and when might one be preferred over the other using the provided dataset?**

* **CNNs** use little filters that look at one small neighborhood at a time, and they’re super good when you don’t have tons of data. They’re fast and have built-in “image bias.”
* **ViTs** use self-attention so every patch can talk to every other patch right away. **That’s awesome for catching global patterns** (like overall texture across the whole photo). But they need more data (or good regularization) to not overfit.

On our defect images (about 1,800 total), ViT did great and hit ~84%. If I had only a couple hundred images, I might start with a CNN instead.

**9. Report the final achieved test accuracy; explain whether you reached the >60% baseline — if not, describe what you tried and why it might have failed; if you did, explain how you achieved it.**

I got around **84%** at the best epoch (and about 82–83% at epoch 30). That’s way over the 60% requirement. I hit it by:

1. Choosing a sensible patch size (4×4).
2. Using a token dim of 64, depth 6, heads 4.
3. Adam optimizer with LR=3e-4.
4. Training for around 10–20 epochs to catch the peak before it wiggled too much.

If I hadn’t reached 60%, I’d try early stopping, add a learning-rate scheduler, or throw in more data augmentation.

**10. Based on the paper you referred to (Dosovitskiy et al., An Image is Worth 16x16 Words), please brief explain: You may include parts of your Step 3 model code to support your explanation.**

1. **How does the Vision Transformer process input images from start to finish?**

* Input: 28×28 grayscale image.
* Cut into 4×4 patches → flatten → linear layer → get 49 tokens.
* Stick on a special [CLS] token at the front.
* Add positional embeddings so the model knows slice order.
* Run through 6 layers of Transformer (self-attention + feed-forward).
* Take the [CLS] token’s output, feed it through a small MLP → get class scores.

1. **How are the image patches divided and transformed into input sequences?**

* 28×28 ÷ 4×4 → 7×7 = 49 patches.
* Each patch has size 4×4 = 16 pixels, we project those 16 numbers into a 64-dim vector.

1. **How does the multi-head self-attention mechanism operate within the Transformer encoder?**

* 28×28 ÷ 4×4 → 7×7 = 49 patches.
* Each patch has size 4×4 = 16 pixels, we project those 16 numbers into a 64-dim vector.

1. **How does the model use the [CLS] token (or final output) to produce the final image classification?**

* The [CLS] token’s hidden state after the last layer is like a summary of the whole image.
* We normalize it (LayerNorm) and run it through an MLP head to predict which defect class it is.