**HCMUS - VNUHCM / FIT /**

**Computer Vision & Cognitive Cybernetics Department**

**Digital Image and Video Processing Application**

**Student ID:** 21127690

**Student name:** Ngo Nguyen Thanh Thanh

**Report: Vision Transformer**

**I. Evaluation summary:**

|  |  |  |
| --- | --- | --- |
| **Task** | **Completion** | **Notes** |
| **ViT Model Implementation** | 100% | Includes all required modules |
| **Training on CIFAR-10** | 100% | Completed with 2 variants |
| **Accuracy & Loss Visualization** | 100% | Plots generated |
| **Report Documentation** | 100% | Includes all required sections |
| **Unit Testing (tests folder)** | 100% | Covered all core modules |

1. **List of features and file structure:**

**Project Directory Tree**

**A screenshot of a computer program

Description automatically generated**

The project is organized into modular components, ensuring separation of concerns and scalability.

* **configs/**:  
  Contains configuration files, such as experiment\_config.py, for managing model and training settings.
* **data/**:  
  (Expected to) handle data loading, preprocessing, or dataset management.
* **evaluation/**:  
  For evaluation scripts to test model performance (e.g., accuracy, confusion matrix).
* **experiments/**:  
  Used to manage different training runs or experimental setups.
* **logs/**:  
  Stores output logs like training loss and accuracy in .csv or .json format.
* **models/**:  
  Contains model definition files, such as ViT architecture components.
* **tests/**:  
  Used for unit tests or validation scripts to ensure code correctness.
* **training/**:  
  Includes training pipeline logic, loss computation, and optimizer settings.
* **utils/**:  
  Utility functions for data handling (data\_utils.py) and visualization (visualize.py).
* **main.py**:  
  Entry point for running the whole training or testing pipeline.
* **README.md**:  
  Provides project overview and usage instructions.
* **requirements.txt**:  
  Lists Python package dependencies for easy environment setup.

**Functions and methods used:**

**List of Functions**:

* Attention(embed\_dim, heads): Computes multi-head self-attention.
* TransformerBlock(embed\_dim, mlp\_dim, heads): Implements a single transformer layer.
* Transformer(embed\_dim, mlp\_dim, layers, heads): Stacks multiple transformer layers.
* ClassificationHead(embed\_dim, classes): Predicts class probabilities.
* VisionTransformer(params): Full ViT model.
* train(model, trainloader, num\_epochs, ...): Trains the model.
* evaluate(model, testloader, device): Computes accuracy.
* get\_dataloaders(): Loads CIFAR-10 data.
* plot\_loss(csv\_file, output\_path): Plots training loss.
* plot\_accuracy\_comparison(json\_files, output\_path): Plots accuracy comparison.

**Function List with Screenshots**

**Attention**

**A screen shot of a computer program

Description automatically generated**

**ClassificationHead**

**A screen shot of a computer program

Description automatically generated**

**TransformerBlock**

**A screen shot of a computer program

Description automatically generated**

**Transformer**

**A screen shot of a computer program

Description automatically generated**

**Vision Tranformer**

**A screen shot of a computer program

Description automatically generated**

**III. Architecture model and Evaluation**

**1. Model Architecture Summary**

**1. Overview of Vision Transformer (ViT)**

The Vision Transformer (ViT) applies Transformer architecture to image classification. It replaces convolutional layers with a sequence-based processing of image patches. ViT includes:

* **Patch Embedding**: Splits input image into fixed-size patches, flattens and linearly projects them to embeddings.
* **Transformer Encoder**: A stack of blocks, each with Multi-Head Self-Attention (MHSA), MLP, LayerNorm, and residual connections.
* **Classification Head**: Uses a special [CLS] token whose final representation is passed through an MLP to predict class labels

**2. Implementation Highlights**

* **Patch Embedding**:  
  Each image is divided into patches (e.g., 4×4), flattened, then projected into a fixed-size vector (embedding dimension).
* **Positional Encoding**:  
  Learnable position vectors are added to preserve spatial order, including a learnable [CLS] token.
* **TransformerBlock**:  
  Each block contains:
  + Pre-LayerNorm → Multi-Head Attention → residual
  + Pre-LayerNorm → Feedforward MLP → residual
* **Transformer Encoder**:  
  Stacks multiple TransformerBlocks (e.g., 6–12 layers) for deep feature learning.
* **Classification Head**:  
  A two-layer MLP applied to the [CLS] output to produce final logits.

**3. Diagram**

**A diagram of a transformer

Description automatically generated**

**Hyperparameter Settings**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment** | **Patch Size** | **Embed Dim** | **MLP Dim** | **Heads** | **Layers** | **Dropout** | **Epochs** | **Top-1 Accuracy (%)** |
| Exp\_1 | 4 | 256 | 512 | 8 | 6 | 0.1 | 20 | 10.0 |
| Exp\_2 | 8 | 128 | 256 | 16 | 12 | 0.1 | 20 | 10.0 |

**2. Visualization**

**2.1 Training Loss Over Epochs**

**Experience 1:**

A graph with a line

Description automatically generated

**Eperience 2:**

A graph with a line

Description automatically generated

**2.2 Accuracy Comparison**

A blue squares with white text

Description automatically generated

**3. Evaluation**

* **Dataset**: CIFAR-10 (50,000 train / 10,000 test)
* **Metric**: Top-1 Accuracy
* **Optimizer**: Adam
* **Loss**: CrossEntropyLoss
* **Epochs**: 20

**Observations:**

All two experiments—regardless of the configuration—report a **Top-1 Accuracy of only 10.0%**, which is **equivalent to random guessing** on a dataset with 10 classes (e.g., CIFAR-10).

**Observations:**

* **Experiment 1**  (Patch Size 4, Embed Dim 256, MLP 512, Heads 8, Layers 6): both gave exactly 10.0% accuracy.
* **Experiment 2** (Patch Size 8, Embed Dim 128, MLP 256, Heads 16, Layers 12): also yielded 10.0% accuracy.

**Interpretation:**

* The model has **not learned** to generalize; it performs no better than random.
* This strongly suggests an issue in:
  + Training loop (e.g., learning rate too low/high)
  + Model not updating weights (e.g., optimizer setup)
  + Dataset loading or label mismatch
  + Incorrect forward pass or frozen layers
  + Loss function misbehavior (e.g., not being minimized)

**VI. Implementation details**

**Function & Class Implementation Summary with Usage Explanation**

**1. MultiheadAttention – models/attention.py**

**Attention – models/attention.py**

**Purpose:** Multi-head self-attention layer for sequence modeling.  
**Usage:**

attn = Attention(embed\_dim=256, heads=8)

output = attn(input\_tensor) # input\_tensor: [batch\_size, seq\_len, embed\_dim]

**Logic:**  
• Project input into Q, K, V using linear layers  
• Reshape and split into multiple heads  
• Compute scaled dot-product attention  
• Apply dropout and merge heads  
• Project back to original embedding dimension

**2. TransformerBlock – models/transformer\_block.py**

**Purpose**: A complete encoder block that integrates attention, feed-forward layers, and normalization.

**Usage**:

block = TransformerBlock(embed\_dim=256, mlp\_dim=512, heads=8)

output = block(x)

**Flow**:

Input → LayerNorm → Attention → Residual Add → LayerNorm → FeedForward → Residual Add

**Implementation Notes**:

* FeedForward: 2 linear layers with ReLU/GELU
* Dropout is applied after attention and MLP
* Maintains original sequence length and dimension

**3. Transformer – models/transformer.py**

**Purpose**: Stacks multiple TransformerBlocks sequentially.

**Usage**:

encoder = Transformer(embed\_dim=256, layers=6, heads=8)

output = encoder(x)

**Behavior**:

* For each layer:
* x = TransformerBlock\_i(x)
* Maintains dimension: (batch, seq\_len, embed\_dim)
* Adds capacity and abstraction with each layer.

**4. ClassificationHead – models/classification\_head.py**

**Purpose**: Projects [CLS] token to class logits for classification.

**Usage**:

head = ClassificationHead(embed\_dim=256, classes=10)

logits = head(cls\_token) # (batch, classes)

**Steps**:

* Linear → GELU → Dropout → Linear → Softmax (optional during inference)

**5. VisionTransformer – models/vision\_transformer.py**

**Purpose**: Full ViT model, includes patch embedding, transformer encoder, and classification head.

**Usage**:

model = VisionTransformer(

inp\_channels=3, patch\_size=4, max\_len=100,

heads=8, classes=10, layers=6, embed\_dim=256, mlp\_dim=512, dropout=0.1

)

logits, hidden\_states = model(images)

**Pipeline**:

Image → Patch Embedding → Positional Encoding → Transformer Encoder → Classification Head

**Design Choices**:

* Learnable positional embeddings
* Prepend [CLS] token
* Outputs logits + intermediate token sequence

**6. train\_model() – training/train.py**

**Purpose**: Main training loop.

**Usage**:

train\_model(config)

**Key Operations**:

* Load CIFAR-10 dataset
* Instantiate model and optimizer
* Train for N epochs, save logs to .csv/.json
* Use accuracy and loss as metrics
* Log best-performing model config

**7. plot\_loss() – utils/visualize.py**

**Purpose**: Visualize training loss across epochs.

**Usage**:

plot\_loss("logs/experiment\_xx.csv", "loss\_plot.png")

**Workflow**:

* Read CSV
* Filter valid epoch rows
* Plot loss vs. epoch
* Save as PNG

**8. plot\_accuracy\_comparison() – utils/visualize.py**

**Purpose**: Compare multiple experiment accuracies.

**Usage**:

plot\_accuracy\_comparison(["logs/exp1.json", "logs/exp2.json"])

**Logic**:

* Read accuracy and hyperparams from .json
* Draw bar plot with experiment labels (patch size, layers)
* Useful for side-by-side performance review