

AI VIET NAM – AI COURSE 2025

Project: Nhận diện hành động trong video (Đề Olympic AI 2025)

Nguyễn Phúc Thịnh
Nguyễn Quốc Thái

Hồ Quang Hiển
Đinh Quang Vinh

I. Bối cảnh bài toán

Phân loại video là một trong những bài toán nền tảng và quan trọng nhất trong lĩnh vực thị giác máy tính. Khác với phân loại hình ảnh tĩnh, dữ liệu video chứa đựng thêm chiều thời gian, nơi các hành động và sự kiện diễn ra liên tục qua một chuỗi các khung hình. Điều này đặt ra thách thức lớn cho các mô hình học máy, yêu cầu không chỉ trích xuất đặc trưng không gian từ từng khung hình mà còn phải mô hình hóa được sự phụ thuộc và thay đổi theo thời gian. Thông thường, với task này mỗi video sẽ chỉ có một label duy nhất.

Trong những năm gần đây, sự bùng nổ của Deep Learning với các kiến trúc như 3D-CNN (Convolutional Neural Networks) hay Video Transformers đã mang lại những bước tiến vượt bậc. Các ứng dụng của phân loại video trải rộng trên nhiều lĩnh vực thực tế như: giám sát an ninh thông minh, phân tích hành vi trong thể thao, tương tác người-máy, và hỗ trợ chẩn đoán y tế.



Hình 1: Minh họa bài toán nhận diện ngôn ngữ ký hiệu tiếng Việt.

Mục tiêu chính của bài toán là xây dựng một hàm ánh xạ F , nhận đầu vào là một đoạn video $X = \{I_1, I_2, \dots, I_T\}$ bao gồm T khung hình, và đưa ra dự đoán nhãn y thuộc tập hợp các lớp hành động C đã được định nghĩa trước. Trong khuôn khổ bài viết này, chúng ta sẽ tập trung vào việc áp dụng các kỹ thuật tiên tiến để giải quyết bài toán phân loại hành động trên tập dữ liệu đơn giản gồm 51 lớp.

Mục lục

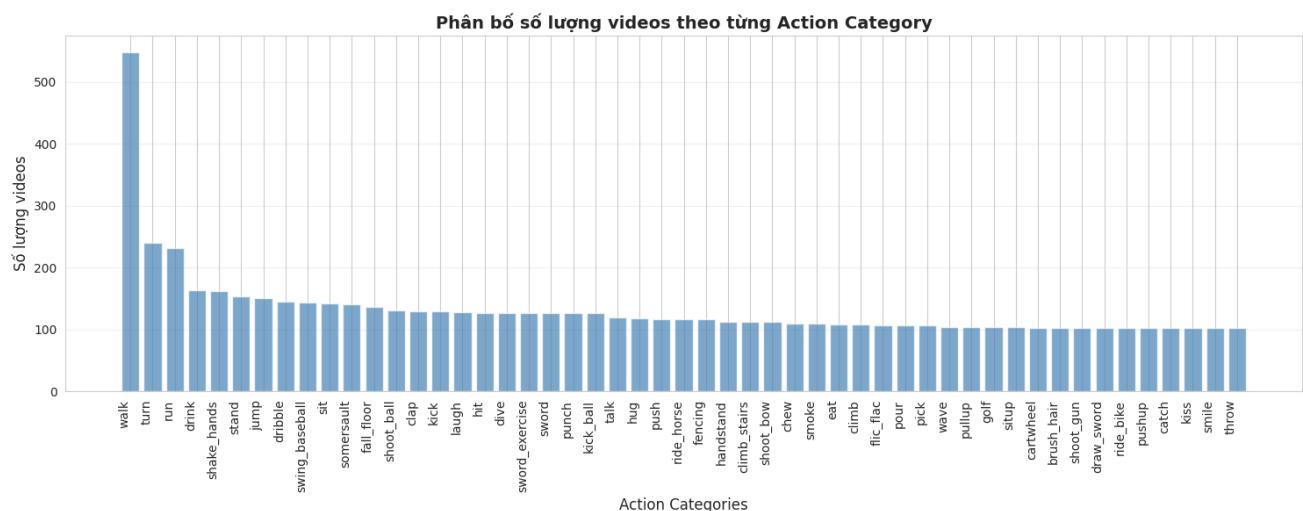
I.	Bối cảnh bài toán	1
II.	Giới thiệu bộ dữ liệu HMDB51	3
II.1.	EDA	3
III.	Kiến trúc mô hình LS-ViT	6
III.1.	Tại sao chọn LS-ViT?	6
IV.	Cấu hình và các module cơ bản	7
IV.1.	PatchEmbed	8
IV.2.	Cơ chế Regularization: DropPath	9
IV.3.	Cơ chế Multi-Head Self-Attention (MSA)	10
V.	Các module đặc trưng của LS-ViT	11
V.1.	SMIFModule (Spatial-Motion Interaction Fusion)	11
V.2.	LMIModule (Long-term Motion Interaction)	13
VI.	Kiến trúc Backbone và Head	14
VII.	Xử lý dữ liệu	17
VIII.	Huấn luyện và đánh giá	21
IX.	Câu hỏi trắc nghiệm	23
	Phụ lục	26

II. Giới thiệu bộ dữ liệu HMDB51

Để xây dựng hệ thống nhận diện hành động, dữ liệu đóng vai trò cốt lõi. Trong bài học này, ta sử dụng bộ dữ liệu HMDB51, một bộ dữ liệu được thu thập từ [Serre Lab](#) tại trường đại học Brown. Bộ dữ liệu bao gồm 51 lớp hành động khác nhau, từ các cử động khuôn mặt đơn giản (cười, nhai) đến các hành động cơ thể phức tạp (nhảy, đấu kiếm). Các video được thu thập từ nhiều nguồn như phim ảnh và video công cộng, tạo ra sự đa dạng lớn về góc quay và điều kiện ánh sáng. Thông tin chi tiết có thể tham khảo tại [HuggingFace](#).

II.1. EDA

Trước khi đưa dữ liệu vào mô hình, chúng ta sẽ thực hiện việc kiểm tra và khám phá các dữ liệu hiện có. Các bạn có thể chạy file `hmdb51_analysis.ipynb` để có thể làm theo. Đầu tiên chúng ta sẽ trực quan hóa số lượng video cho mỗi một lớp dữ liệu.



Hình 2: Biểu đồ phân phối số lượng video clip cho từng lớp hành động trong tập huấn luyện.

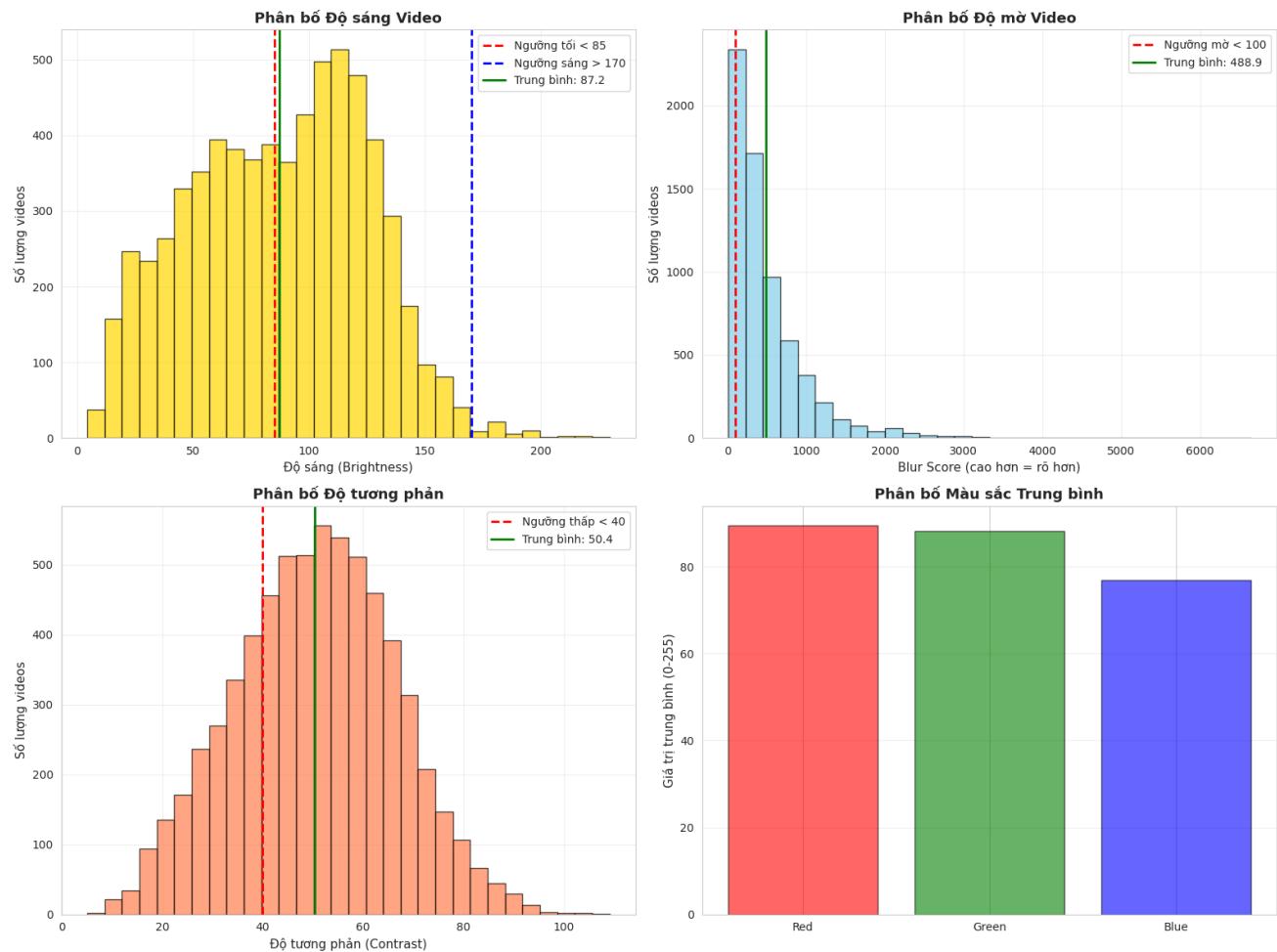
Chúng ta có thể thấy rằng dữ liệu bị imbalance khá nhiều, điều này yêu cầu lúc chúng ta huấn luyện mô hình cần phải chú ý chọn cơ chế tăng cường dữ liệu, hàm loss và cơ chế training phù hợp. Việc sử dụng phương pháp tăng cường dữ liệu nào sẽ được quyết định khi chúng ta thấy các vấn đề của video trong phần tiếp theo.

Để hiểu rõ hơn về chất lượng dữ liệu đầu vào, chúng ta tiến hành phân tích các đặc trưng thị giác cơ bản của video bao gồm độ sáng (brightness), độ mờ (blur), độ tương phản (contrast) và phân bố màu sắc. Do dữ liệu video được load dưới dạng đối tượng `torchcodec.decoders.VideoDecoder` (thay vì file path hay numpy array thông thường), quá trình trích xuất frame cần được xử lý đặc biệt để tối ưu hóa hiệu năng và bộ nhớ.

Đoạn mã dưới đây minh họa cách chúng ta decode ngẫu nhiên các frame từ `VideoDecoder` và tính toán các chỉ số thống kê. Việc sử dụng thư viện `torchcodec` giúp chúng ta có thể tận dụng GPU để đọc video được nhanh hơn, nhất là với các bài toán có nhiều video như dataset này.

Hàm đọc video từ VideoDecoder

```
1 def decode_video_frames(video_decoder, max_frames=16):
2     metadata = video_decoder.metadata
3     total_frames = metadata.num_frames
4
5     # Chọn các chỉ số frame (indices) cách đều nhau
6     num_frames_to_extract = min(max_frames, total_frames)
7     if total_frames > 1:
8         indices = [
9             int(i * (total_frames - 1) / (num_frames_to_extract - 1))
10            for i in range(num_frames_to_extract)
11        ]
12     else:
13         indices = [0]
14
15     frames = []
16     for idx in indices:
17         if idx >= total_frames: break
18         # Sử dụng API của torchcodec để lấy frame tại index cụ thể
19         frame_batch = video_decoder.get_frames_at(indices=[idx])
20
21         # Chuyển đổi Tensor (C, H, W) sang Numpy (H, W, C) cho OpenCV xử lý
22         frame_tensor = frame_batch.data[0]
23         frame_np = frame_tensor.permute(1, 2, 0).cpu().numpy()
24
25         # Chuẩn hóa về range [0, 255] uint8
26         if frame_np.max() <= 1.0:
27             frame_np = (frame_np * 255).astype(np.uint8)
28         else:
29             frame_np = frame_np.astype(np.uint8)
30
31         frames.append(frame_np)
32
33 return frames
```

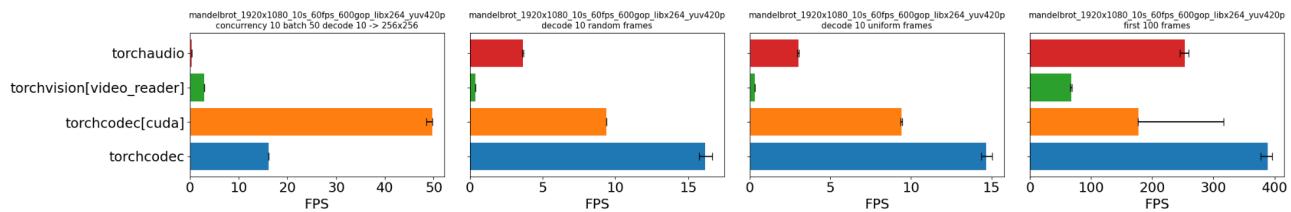


Hình 3: Các phân tích về tính chất của video trên nhiều thông số khác nhau.

Chúng ta có các biến đổi có thể thực hiện dựa trên kết quả phân tích định lượng như sau:

- Biến đổi hình học và tỷ lệ: Để mô phỏng sự thay đổi về khoảng cách giữa người thực hiện và camera, quá trình huấn luyện áp dụng cắt ngẫu nhiên (random crop) với tỷ lệ khung hình từ 0.8 đến 1.0. Ngoài ra, kỹ thuật lật ngang (horizontal flip) với xác suất 0.5 được sử dụng để tăng tính đa dạng về hướng của hành động.
- Biến đổi quang học: Nhằm giúp mô hình thích nghi với các điều kiện ánh sáng khác nhau mà không làm sai lệch thông tin gốc, các thuộc tính như độ sáng, độ tương phản và độ bão hòa được điều chỉnh ngẫu nhiên. Các thay đổi này được áp dụng độc lập với xác suất thấp (0.3) và cường độ nhẹ (dao động trong khoảng $\pm 10\%$) để đảm bảo tính toàn vẹn của dữ liệu video vốn có độ sáng thấp.
- Chuẩn hóa dữ liệu: Đầu vào được chuẩn hóa bằng cách trừ đi giá trị trung bình (mean) và chia cho độ lệch chuẩn (std) đều là 0.5 cho cả ba kênh màu, đưa giá trị pixel về khoảng $[-1, 1]$ giúp quá trình hội tụ của mô hình ổn định hơn.

Kết quả của quá trình trực quan hóa giúp ta xác nhận rằng các bước VideoTransform (như resize, crop) hoạt động đúng như mong đợi.



Hình 4: Tốc độ đọc của torchcodec trên CUDA với các đọc truyền thống.



Hình 5: Minh họa các khung hình được trích xuất từ một video mẫu sau khi đã qua bước tiền xử lý.

III. Kiến trúc mô hình LS-ViT

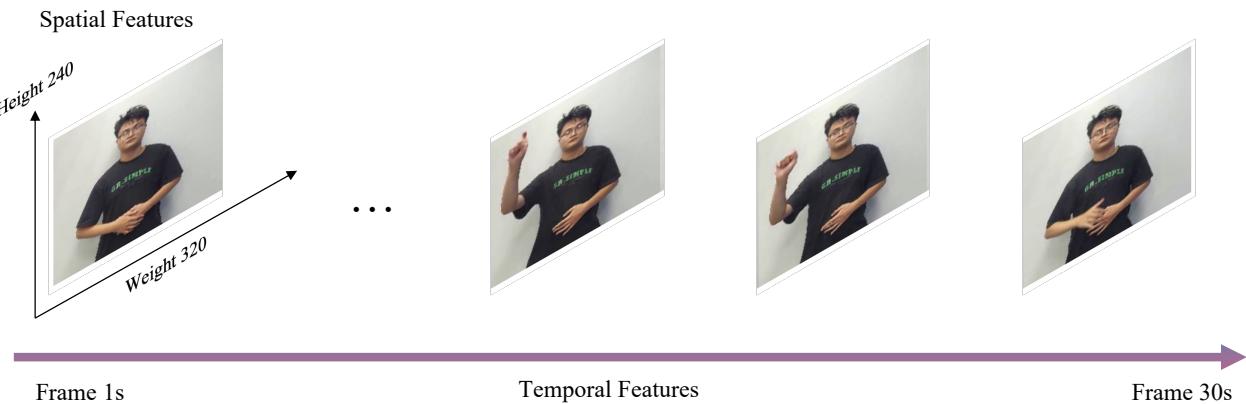
Sau khi đã nắm vững dữ liệu, ta chuyển sang thiết kế mô hình. Ta sử dụng kiến trúc LS-ViT (Long-Short Term Video Transformer). Đây là một bước tiến so với các mô hình 2D CNN truyền thống vốn chỉ xử lý từng khung hình riêng lẻ.

III.1. Tại sao chọn LS-ViT?

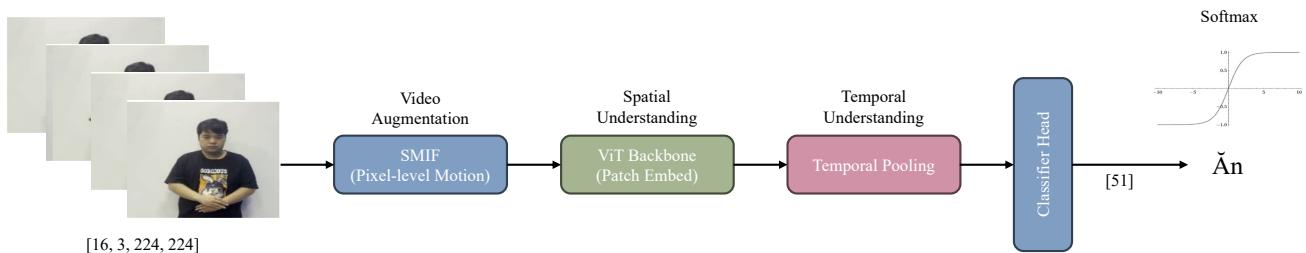
Thách thức lớn nhất của nhận diện hành động là sự phụ thuộc vào thời gian. Một hành động như “ngồi xuống” và “đứng lên” có thể trông giống hệt nhau nếu chỉ nhìn vào một khung hình tĩnh, nhưng thứ tự thời gian lại hoàn toàn ngược lại.

LS-ViT giải quyết vấn đề này bằng cách kết hợp hai luồng thông tin:

- Spatial (Không gian): Xử lý chi tiết hình ảnh trong từng khung hình (tương tự ViT gốc).
- Temporal (Thời gian): Liên kết thông tin giữa các khung hình để nắm bắt chuyển động.



Hình 6: Vấn đề không gian và thời gian trong bài toán phân loại video.



Hình 7: Sơ đồ tổng quan kiến trúc LS-ViT với các module xử lý không gian và thời gian đan xen.

Cụ thể, sức mạnh của LS-ViT đến từ việc tích hợp các thành phần chuyên biệt trong kiến trúc của nó:

- SMIF (Spatial-Motion Interaction Fusion):** Khác với các ViT thông thường, LS-ViT sử dụng module này để tính toán sự sai biệt giữa các khung hình (frame differencing) ngay tại đầu vào, giúp mô hình tập trung vào các vùng chuyển động (motion regions) trước khi đi vào xử lý sâu.
- LSViTBlock:** Thay thế khối Transformer tiêu chuẩn bằng kiến trúc lai, kết hợp **Spatial Attention** để hiểu ngữ cảnh tĩnh và các lớp trộn thông tin thời gian (Temporal Mixing) để mô hình hóa chuỗi hành động, đảm bảo sự cân bằng giữa độ chính xác và chi phí tính toán.

Chi tiết lập trình của các lớp **SMIFModule** và **LSViTBlock** sẽ được phân tích sâu ở phần hiện thực hóa.

IV. Cấu hình và các module cơ bản

Trước khi xây dựng mô hình, ta cần định nghĩa các tham số cấu hình và các khối cơ bản của Vision Transformer (ViT). Class **ViTConfig** đóng vai trò là nơi lưu trữ trung tâm cho các siêu tham số (hyperparameters) như kích thước ảnh, kích thước patch, số lượng layer, và số lượng head trong multi-head attention.

Cấu hình ViT

```

1 @dataclass
2 class ViTConfig:
3     image_size: int = 224
4     patch_size: int = 16
5     in_chans: int = 3
6     embed_dim: int = 768
7     depth: int = 12
8     num_heads: int = 12
9     mlp_ratio: float = 4.0
10    drop_rate: float = 0.1
11    # ... (các tham số khác)

```

IV.1. PatchEmbed

Module PatchEmbed chịu trách nhiệm chia ảnh đầu vào thành các mảnh nhỏ (patches) và chiếu chúng vào không gian vector (embedding). Đây là bước đầu tiên để biến đổi dữ liệu hình ảnh thành dạng chuỗi (sequence) mà Transformer có thể xử lý.

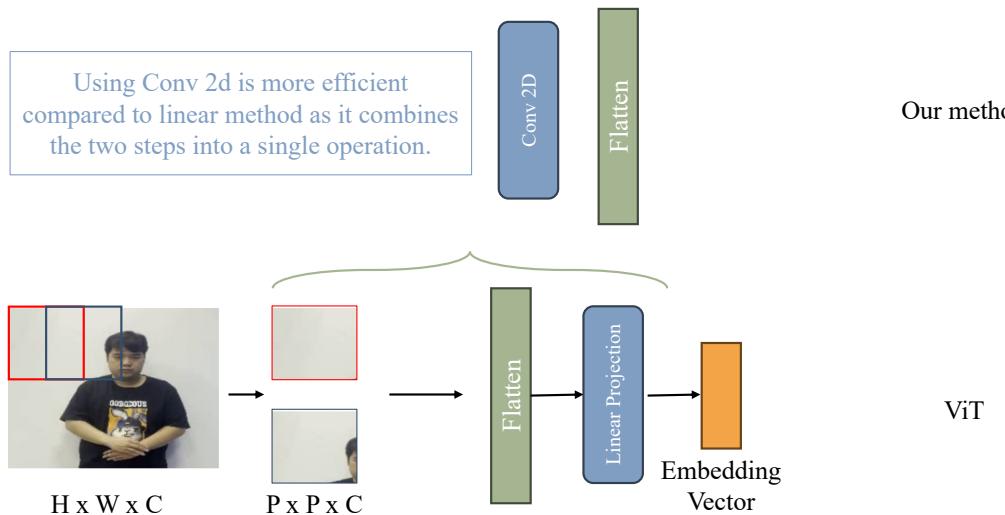
PatchEmbed Module - Code Exercise

```

1 class PatchEmbed(nn.Module):
2     def __init__(self, config: ViTConfig):
3         super().__init__()
4         self.image_size = config.image_size
5         self.patch_size = config.patch_size
6         self.num_patches = (config.image_size // config.patch_size) ** 2
7         self.proj = nn.Conv2d(
8             config.in_chans,
9             config.embed_dim,
10            kernel_size=config.patch_size,
11            stride=config.patch_size,
12        )
13
14     def forward(self, x: torch.Tensor) -> torch.Tensor:
15         x = ***Your code here***
16         x = ***Your code here***
17         return x

```

Trong bước này, bạn cần hoàn thiện hàm `forward` để biến ảnh đầu vào thành một chuỗi các biểu diễn theo *từng patch*. Cụ thể, đầu tiên cần tạo ra đặc trưng cho mỗi patch sao cho mỗi patch được ánh xạ thành một vector có kích thước bằng `embed_dim`. Sau đó, cần sắp xếp lại tensor kết quả để chuyển từ dạng đặc trưng không gian 2D sang dạng chuỗi token phù hợp với Transformer, trong đó mỗi token tương ứng với một patch của ảnh. Kết quả trả về là tensor có dạng (B, N, D) , với B là batch size, N là số patch, và D là kích thước embedding.



Hình 8: Quá trình chia ảnh thành các patch và chuyển đổi thành vector embedding. Sử dụng lớp Conv chỉ là trick chứ không thay đổi ý tưởng patch embedding gốc.

IV.2. Cơ chế Regularization: DropPath

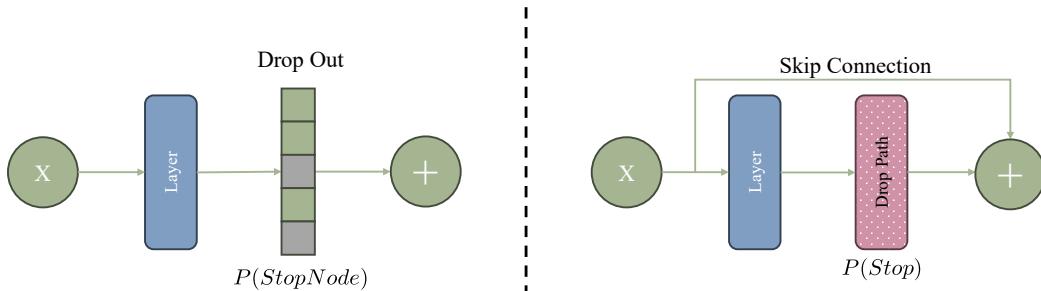
Trong các mạng nơ-ron sâu như Vision Transformer, việc huấn luyện rất dễ gặp hiện tượng overfitting. Để giải quyết vấn đề này, module DropPath (hay còn gọi là Stochastic Depth) được sử dụng. Khác với Dropout thông thường (ngắt ngẫu nhiên các nơ-ron), DropPath ngắt ngẫu nhiên toàn bộ một luồng dữ liệu (residual path) của một mẫu trong batch trong quá trình huấn luyện, giúp mạng học được các đặc trưng đa dạng và mạnh mẽ hơn.

DropPath Module

```

1 class DropPath(nn.Module):
2     """Drop paths (Stochastic Depth) per sample when applied in main path of residual
3         blocks."""
4
5     def __init__(self, drop_prob: float = 0.0):
6         super().__init__()
7         self.drop_prob = drop_prob
8
9     def forward(self, x: torch.Tensor) -> torch.Tensor:
10        if self.drop_prob == 0. or not self.training:
11            return x
12
13        keep_prob = 1 - self.drop_prob
14        # Tạo shape mask (batch_size, 1, 1, ...) để broadcast
15        shape = (x.shape[0],) + (1,) * (x.ndim - 1)
16
17        # Tao mask ngẫu nhiên Bernoulli
18        random_tensor = keep_prob + torch.rand(shape, dtype=x.dtype, device=x.device)
19        random_tensor.floor_() # Binarize
20
21        # Scale output để giữ nguyên kỳ vọng giá trị
22        return x.div(keep_prob) * random_tensor

```



Hình 9: Minh họa sự khác nhau giữa DropOut và DropPath. DropOut có thể áp dụng với tất cả loại mạng, trong khi đó DropPath chỉ có thể áp dụng với mạng có Residual Connection.

IV.3. Cơ chế Multi-Head Self-Attention (MSA)

Trái tim của Transformer là cơ chế Attention. Module này cho phép mỗi patch trong hình ảnh “nhìn” và tổng hợp thông tin từ tất cả các patch khác (tổng cục). Chúng ta sử dụng cơ chế Attention tiêu chuẩn với việc tính toán song song nhiều đầu (Multi-Head) để mô hình có thể tập trung vào các không gian biểu diễn khác nhau của hình ảnh. Điểm khác biệt của mạng được lập trình dưới đây là sự gọn nhẹ vì chúng ta sử dụng một ma trận duy nhất cho cả QKV. Việc này không giảm quá độ chính xác của mô hình nhưng vẫn mô phỏng được quá trình tính toán của Attention.

Công thức tính toán Attention được mô tả như sau:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (1)$$

Attention Module - Code Exercise

```

1 class Attention(nn.Module):
2     def __init__(self, dim: int, num_heads: int, qkv_bias: bool, attn_drop: float,
3                  proj_drop: float):
4         super().__init__()
5         self.num_heads = num_heads
6         head_dim = dim // num_heads
7         self.scale = head_dim ** -0.5
8         self.qkv = ***Your code here***
9         self.attn_drop = nn.Dropout(attn_drop)
10        self.proj = ***Your code here***
11        self.proj_drop = nn.Dropout(proj_drop)
12
13    def forward(self, x: torch.Tensor) -> torch.Tensor:
14        B, N, C = x.shape
15        qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, C // self.num_heads)
16        qkv = qkv.permute(2, 0, 3, 1, 4)
17        q, k, v = ***Your code here***
18        attn = ***Your code here***
19        attn = attn.softmax(dim=-1)
20        attn = self.attn_drop(attn)
21        x = ***Your code here***
22        x = x.transpose(1, 2).reshape(B, N, C)

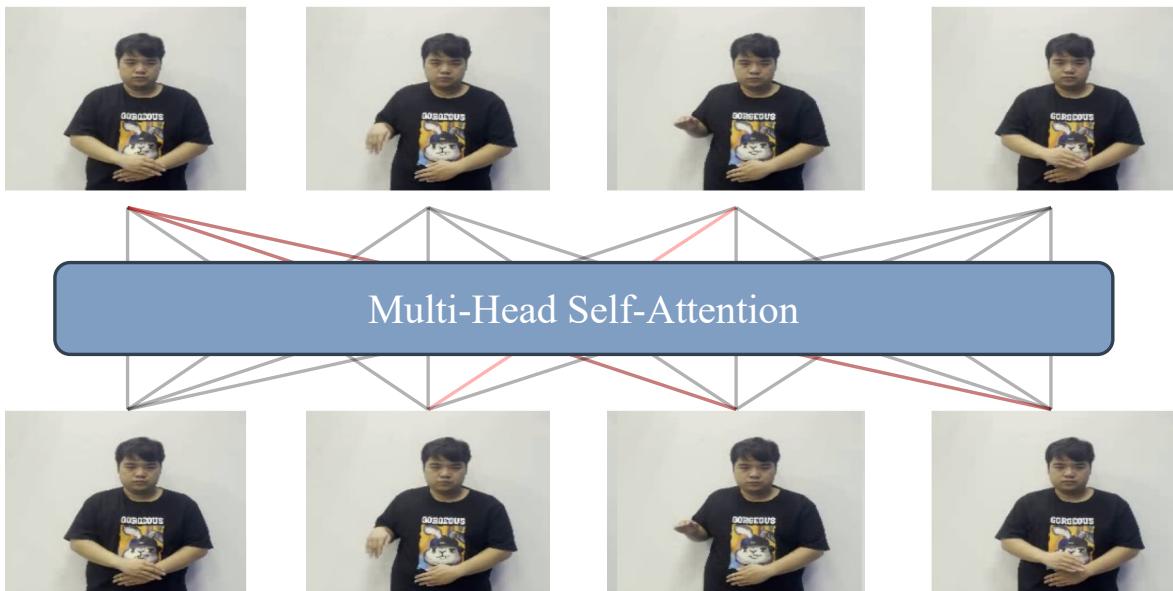
```

```

22     x = ***Your code here***
23     x = self.proj_drop(x)
24     return x

```

Trong phần này, bạn cần hoàn thiện lớp Attention để mô hình học cách *tự tập trung* vào các token quan trọng trong chuỗi đầu vào. Trước hết, mỗi token sẽ được ánh xạ thành ba đại diện khác nhau (tương ứng với ba vai trò trong cơ chế chú ý) và được tách thành nhiều *đầu* song song để nắm bắt các kiểu quan hệ đa dạng. Tiếp theo, bạn cần tính mức độ liên quan giữa các token để tạo ra ma trận trọng số chú ý, chuẩn hoá các trọng số này để thể hiện “mức quan tâm” tương đối, rồi dùng chúng để tổng hợp thông tin từ toàn bộ chuỗi thành biểu diễn mới cho từng token. Cuối cùng, các đầu được ghép lại và đưa qua một phép biến đổi đầu ra để thu được tensor cùng kích thước như đầu vào, sẵn sàng chuyển sang các bước tiếp theo của khối Transformer.



Hình 10: Minh họa trong khối Multi-Head Self-Attention các patch của hình ảnh sẽ nhìn thấy lẫn nhau và thấy chính nó.

V. Các module đặc trưng của LS-ViT

Điểm đặc biệt của LS-ViT nằm ở hai module chính giúp xử lý thông tin thời gian: **SMIFModule** và **LMIModule**.

V.1. SMIFModule (Spatial-Motion Interaction Fusion)

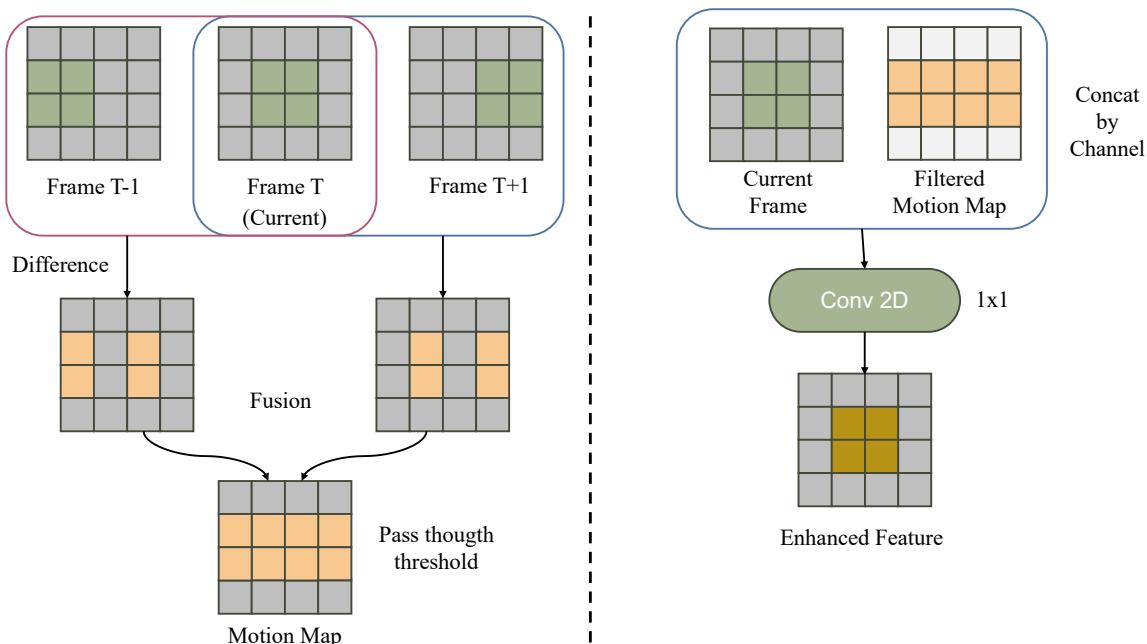
Module này được thiết kế để nắm bắt các chuyển động ngắn hạn (short-term motion) giữa các khung hình lân cận. Ta tính toán sự khác biệt (difference) giữa khung hình hiện tại và các khung hình trước/sau nó để tạo ra bản đồ chuyển động (motion map).

Module SMIF

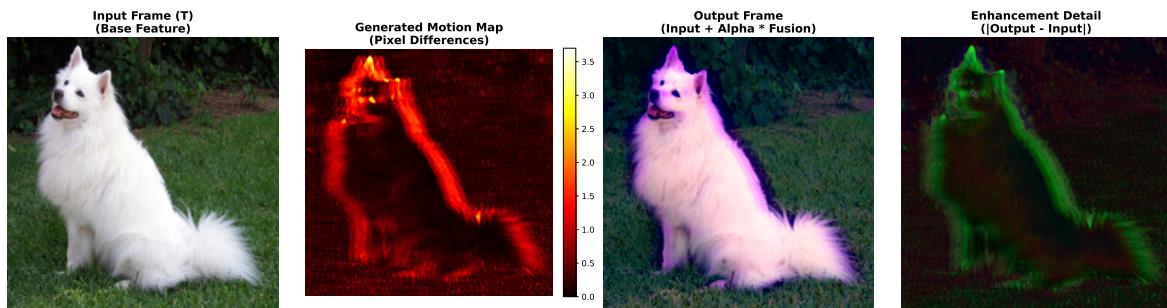
```

1 class SMIFModule(nn.Module):
2     def __init__(self, channels: int, window_size: int = 5, ...):
3         # ... khởi tạo ...
4
5     def forward(self, video: torch.Tensor) -> torch.Tensor:
6         # Tính toán motion map từ sự khác biệt giữa các frame
7         for offset in range(1, self.window_size + 1):
8             diff_f = next_frames - video
9             diff_b = video - prev_frames
10            motion_accum = motion_accum + diff_f.abs() + diff_b.abs()
11
12        # Kết hợp motion map vào đặc trưng gốc
13        fused = torch.cat([base, motion_flat], dim=1)
14        out = base + self.alpha.tanh() * fused
15
16        return out

```



Hình 11: Sơ đồ hoạt động của SMIFModule trong việc trích xuất đặc trưng chuyển động cục bộ. Chúng ta sử dụng Kernel 1x1 để có thể chuyển tensor về dạng ban đầu để cho việc Skip Connection.



Hình 12: Minh họa hình ảnh trước và sau khi được tăng cường.

V.2. LMIModule (Long-term Motion Interaction)

Trong khi SMIF xử lý cục bộ, LMIModule giúp mô hình nắm bắt sự phụ thuộc dài hạn (long-term) giữa các token trong chuỗi thời gian. Nó sử dụng cơ chế attention để tăng cường các đặc trưng quan trọng dựa trên thông tin chuyển động toàn cục.

Module LMI

```

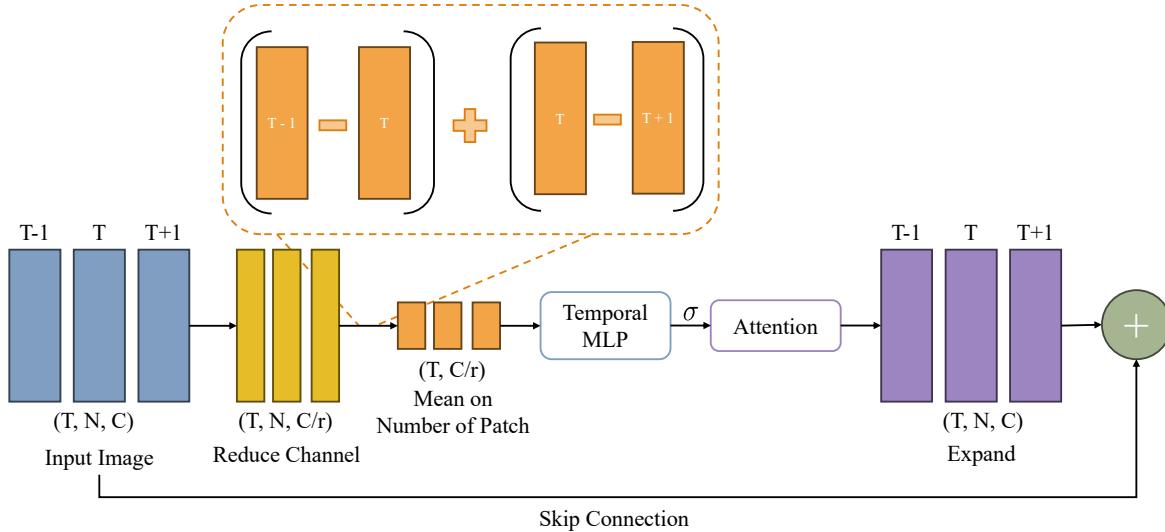
1 class LMIModule(nn.Module):
2     def __init__(self, dim: int, reduction: int = 4, delta: float = 0.1):
3         super().__init__()
4         reduced_dim = max(1, dim // reduction)
5         self.reduce = nn.Linear(dim, reduced_dim)
6         self.expand = nn.Linear(reduced_dim, dim)
7         self.temporal_mlp = nn.Sequential(
8             nn.LayerNorm(reduced_dim),
9             nn.Linear(reduced_dim, reduced_dim),
10            nn.GELU(),
11            nn.Linear(reduced_dim, reduced_dim),
12        )
13         self.delta = nn.Parameter(torch.tensor(delta))
14
15     def forward(self, x: torch.Tensor) -> torch.Tensor:
16         B, T, N, C = x.shape
17         reduced = self.reduce(x)
18
19         if T > 1:
20             diff_f = reduced[:, 1:] - reduced[:, :-1]
21             diff_f = torch.cat([diff_f, diff_f[:, -1:]], dim=1)
22             diff_b = reduced[:, :-1] - reduced[:, 1:]
23             diff_b = torch.cat([diff_b[:, :1], diff_b], dim=1)
24         else:
25             diff_f = torch.zeros_like(reduced)
26             diff_b = torch.zeros_like(reduced)
27
28         motion = (diff_f.abs() + diff_b.abs()).mean(dim=2)
29         motion = self.temporal_mlp(motion)
30
31         attn = torch.sigmoid(motion).unsqueeze(2)
32         attn = self.expand(attn)

```

```

33     attn = attn.expand(-1, -1, N, -1)
34     enhanced = x * attn
35     return x + self.delta.tanh() * enhanced

```



Hình 13: Cơ chế Long-term Motion Interaction giúp liên kết thông tin qua nhiều khung hình.

VI. Kiến trúc Backbone và Head

Mô hình được xây dựng bằng cách xếp chồng các khối LSViTBlock. Mỗi khối bao gồm lớp Attention chuẩn, lớp MLP, và được tích hợp thêm LMIModule để xử lý thông tin video.

LSViT Block

```

1 class LSViTBlock(nn.Module):
2     def __init__(self, dim: int, num_heads: int, mlp_ratio: float, drop_rate: float,
3                  attn_drop: float, drop_path: float):
4         super().__init__()
5         self.norm1 = nn.LayerNorm(dim)
6         self.attn = Attention(dim, num_heads, True, attn_drop, drop_rate)
7         self.drop_path1 = DropPath(drop_path)
8         self.norm2 = nn.LayerNorm(dim)
9         self.mlp = Mlp(dim, mlp_ratio, drop_rate)
10        self.drop_path2 = DropPath(drop_path)
11        self.lmim = LMIModule(dim)
12
13    def forward(self, x: torch.Tensor, B: int, T: int) -> torch.Tensor:
14        x = x + self.drop_path1(self.attn(self.norm1(x)))
15        x = x + self.drop_path2(self.mlp(self.norm2(x)))
16        BT, Np1, C = x.shape
17        assert BT == B * T
18        x = x.view(B, T, Np1, C)
# Áp dụng LMI sau các lớp transformer chuẩn

```

```

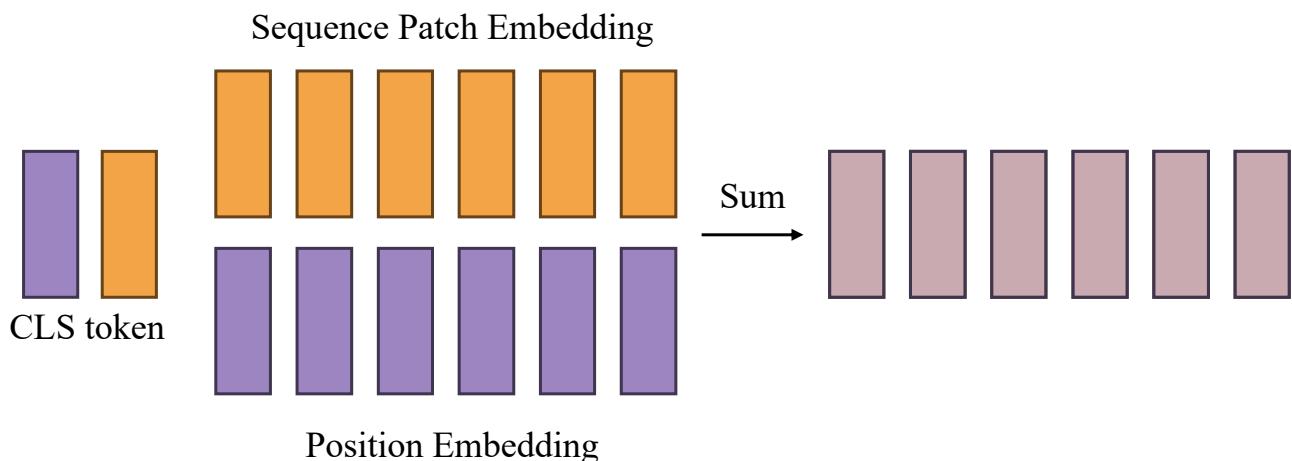
19     x = self.lmim(x)
20     x = x.view(B * T, Np1, C)
21     return x

```

Sau khi có LSViTBlock, ta sẽ tạo ra Class LSViTBackbone đóng vai trò là “xương sống” của toàn bộ mô hình. Nhiệm vụ của nó là kết nối các thành phần rời rạc thành một luồng xử lý thống nhất như sau.

1. Patch Embedding: Chuyển đổi video đầu vào thành chuỗi các vector.
2. CLS Token & Position Embedding: Thêm một token đặc biệt (`cls_token`) để học biểu diễn toàn cục của từng khung hình và cộng thông tin vị trí vào chuỗi.
3. Transformer Blocks: Đẩy dữ liệu qua chuỗi các khối LSViTBlock để trích xuất đặc trưng không gian và thời gian.

Đặc biệt, hàm `_interpolate_pos_encoding` cho phép mô hình linh hoạt xử lý các kích thước ảnh đầu vào khác nhau bằng cách nội suy (interpolate) bộ mã hóa vị trí đã được huấn luyện trước.



Hình 14: Quá trình thêm CLS token và Position Embedding vào chuỗi Patch Embedding.

LSViT Backbone

```

1 class LSViTBackbone(nn.Module):
2     def __init__(self, config: ViTConfig):
3         super().__init__()
4         self.config = config
5         self.patch_embed = PatchEmbed(config)
6         num_patches = self.patch_embed.num_patches
7
8         # Token phân loại (CLS) và mã hóa vị trí (Positional Embedding)
9         self.cls_token = nn.Parameter(torch.zeros(1, 1, config.embed_dim))
10        self.pos_embed = nn.Parameter(torch.zeros(1, num_patches + 1, config.embed_dim))
11        self.pos_drop = nn.Dropout(config.drop_rate)

```

```

12
13     # Tỷ lệ DropPath tăng dần theo độ sâu (Stochastic Depth)
14     dpr = torch.linspace(0, config.drop_path_rate, steps=config.depth).tolist()
15
16     self.blocks = nn.ModuleList(
17         [
18             LSViTBlock(
19                 dim=config.embed_dim,
20                 num_heads=config.num_heads,
21                 mlp_ratio=config.mlp_ratio,
22                 drop_rate=config.drop_rate,
23                 attn_drop=config.attn_drop_rate,
24                 drop_path=dpr[i],
25             )
26             for i in range(config.depth)
27         ]
28     )
29
30     self.norm = nn.LayerNorm(config.embed_dim)
31
32     # Khởi tạo trọng số
33     nn.init.trunc_normal_(self.cls_token, std=0.02)
34     nn.init.trunc_normal_(self.pos_embed, std=0.02)
35
36     def _interpolate_pos_encoding(self, x: torch.Tensor) -> torch.Tensor:
37         """Nội suy Positional Embedding nếu kích thước ảnh thay đổi."""
38         B, N, C = x.shape
39         num_patches = N - 1
40         if num_patches == self.patch_embed.num_patches:
41             return self.pos_embed
42
43         cls_pos = self.pos_embed[:, :1]
44         patch_pos = self.pos_embed[:, 1:]
45
46         dim = patch_pos.shape[-1]
47         gs_old = int(math.sqrt(patch_pos.shape[1]))
48         gs_new = int(math.sqrt(num_patches))
49
50         patch_pos = patch_pos.reshape(1, gs_old, gs_old, dim).permute(0, 3, 1, 2)
51         patch_pos = F.interpolate(patch_pos, size=(gs_new, gs_new), mode="bicubic",
52                                   align_corners=False)
53         patch_pos = patch_pos.permute(0, 2, 3, 1).reshape(1, gs_new * gs_new, dim)
54
55         return torch.cat([cls_pos, patch_pos], dim=1)
56
57     def forward(self, video: torch.Tensor) -> torch.Tensor:
58         B, T, C, H, W = video.shape
59         # Gộp Batch và Time để xử lý song song như ảnh tĩnh
60         x = video.reshape(B * T, C, H, W)
61         x = self.patch_embed(x)
62
63         # Thêm CLS token vào đầu chuỗi
64         cls_tokens = self.cls_token.expand(x.shape[0], -1, -1)
65         x = torch.cat((cls_tokens, x), dim=1)

```

```

65     # Công mã hóa vị trí
66     pos_embed = self._interpolate_pos_encoding(x)
67     x = x + pos_embed
68     x = self.pos_drop(x)
69
70     # Đi qua các khối LSViTBlock
71     for block in self.blocks:
72         x = block(x, B, T)
73
74     x = self.norm(x)
75
76     # Trả lại kích thước (Batch, Time, Num_tokens, Channels)
77     x = x.view(B, T, x.shape[1], x.shape[2])
78     return x

```

Cuối cùng, class `LSViTForAction` kết hợp Backbone với một lớp `Linear` (Head) để đưa ra dự đoán phân loại hành động.

LSViT Backbone

```

1 class LSViTForAction(nn.Module):
2     def __init__(self, config: ViTConfig, num_classes: int = 51, smif_window: int = 5):
3         super().__init__()
4         self.smif = SMIFModule(config.in_chans, window_size=smif_window)
5         self.backbone = LSViTBackbone(config)
6         self.head = nn.Linear(config.embed_dim, num_classes)
7
8     def forward(self, video: torch.Tensor) -> torch.Tensor:
9         x = self.smif(video)
10        feats = self.backbone(x)
11        cls_tokens = feats[:, :, 0]
12        pooled = cls_tokens.mean(dim=1)
13        logits = self.head(pooled)
14        return logits

```

VII. Xử lý dữ liệu

Dữ liệu video cần được xử lý đặc biệt trước khi đưa vào mô hình do độ dài của các video không đồng nhất. Class `HMDB51Dataset` chịu trách nhiệm quản lý việc tải ảnh, chia tập dữ liệu (train/val/test) và quan trọng nhất là lấy mẫu (sampling). Để đảm bảo mô hình nhận được đầu vào có kích thước cố định (ví dụ: $T = 16$ frames), chúng ta áp dụng phương pháp lấy mẫu dựa trên `torch.linspace` kết hợp với “Temporal Padding”:

- Uniform Sampling:** Sử dụng `linspace` để tạo ra các chỉ số (indices) rải đều trên toàn bộ chiều dài video với một khoảng cách (stride) xác định.
- Temporal Padding:** Đối với các video quá ngắn, không đủ số lượng frame yêu cầu sau khi sampling, hệ thống sẽ tự động lặp lại frame cuối cùng (padding) cho đến khi đủ độ dài.

Cách này ưu việt hơn zero-padding vì nó bảo toàn được ngữ cảnh hình ảnh cuối cùng của hành động.

HMDB51 Dataset và Sampling Logic

```

1  class HMDB51Dataset(Dataset):
2      """Load HMDB51 frame folders with grouped train/val split."""
3      def __init__(self, root: str, split: str, num_frames: int, frame_stride: int,
4                      image_size: int = 224, transform: Optional[VideoTransform] = None, val_ratio: float =
5                          0.1, seed: int = 42):
6          super().__init__()
7          self.root = Path(root)
8          if not self.root.is_dir():
9              raise FileNotFoundError(f"Data root not found: {self.root}")
10
11         # Tự động đọc danh sách class từ tên thư mục
12         self.classes = sorted([d.name for d in self.root.iterdir() if d.is_dir()])
13         if not self.classes:
14             raise RuntimeError(f"No class folders in {self.root}")
15         self.class_to_idx = {name: idx for idx, name in enumerate(self.classes)}
16
17         # Gom nhóm các frames thuộc cùng một video
18         grouped_samples: Dict[Tuple[str, str], List[Tuple[List[Path], int]]] = {}
19         for cls in self.classes:
20             cls_dir = self.root / cls
21             for video_dir in sorted([d for d in cls_dir.iterdir() if d.is_dir()]):
22                 frame_paths = sorted([p for p in video_dir.iterdir() if p.suffix.lower()
23                                       () in [".jpg", ".jpeg", ".png"]])
24                 if not frame_paths:
25                     continue
26                 # Group key dựa trên tên gốc của video (loại bỏ hậu tố _frame_idx)
27                 group_key = (cls, self._base_video_name(video_dir.name))
28                 grouped_samples.setdefault(group_key, []).append((frame_paths, self.
29                                         class_to_idx[cls]))
30
31         if not grouped_samples:
32             raise RuntimeError(f"No frame folders found inside {self.root}")
33
34         # Chia tập Train/Val dựa trên video gốc (tránh data leakage)
35         group_values = list(grouped_samples.values())
36         rng = np.random.RandomState(seed)
37         group_indices = np.arange(len(group_values))
38         rng.shuffle(group_indices)
39
40         split_point = int(len(group_indices) * (1 - val_ratio))
41         if split == "train":
42             selected_groups = group_indices[:split_point]
43         elif split in ("val", "test"):
44             selected_groups = group_indices[split_point:]
45         else:
46             raise ValueError(f"Unknown split: {split}")
47
48         self.samples: List[Tuple[List[Path], int]] = []

```

```

45     for idx in selected_groups:
46         self.samples.extend(group_values[int(idx)])
47
48     if not self.samples:
49         raise RuntimeError("Selected split has no samples; adjust val_ratio.")
50
51     self.split = split
52     self.num_frames = num_frames
53     self.frame_stride = max(1, frame_stride)
54     self.transform = transform or VideoTransform(image_size, is_train=(split == "train"))
55     self.to_tensor = transforms.ToTensor()
56
57     def __len__(self) -> int:
58         return len(self.samples)
59
60     def _select_indices(self, total: int) -> torch.Tensor:
61         """Chọn indices frames với có chẽ Temporal Padding"""
62         if total <= 0:
63             raise ValueError("Video folder has no frames")
64         if total == 1:
65             return torch.zeros(self.num_frames, dtype=torch.long)
66
67         # 1. Uniform Sampling
68         steps = max(self.num_frames * self.frame_stride, self.num_frames)
69         grid = torch.linspace(0, total - 1, steps=steps)
70         idxs = grid[:: self.frame_stride].long()
71
72         # 2. Temporal Padding: Lắp lại frame cuối nếu thiếu
73         if idxs.numel() < self.num_frames:
74             pad = idxs.new_full((self.num_frames - idxs.numel(),), idxs[-1].item())
75             idxs = torch.cat([idxs, pad], dim=0)
76
77         return idxs[: self.num_frames]
78
79     @staticmethod
80     def _base_video_name(name: str) -> str:
81         match = re.match(r"(.+)\_\d+\$", name)
82         return match.group(1) if match else name
83
84     def __getitem__(self, idx: int) -> Tuple[torch.Tensor, int]:
85         frame_paths, label = self.samples[idx]
86         total = len(frame_paths)
87         idxs = self._select_indices(total)
88
89         frames = []
90         for i in idxs:
91             path = frame_paths[int(i.item())]
92             with Image.open(path) as img:
93                 img = img.convert("RGB")
94                 frames.append(self.to_tensor(img))
95
96         video = torch.stack(frames) # (T, C, H, W)
97         video = self.transform(video)

```

```

98     return video, label
99
100 def collate_fn(batch: List[Tuple[torch.Tensor, int]]) -> Tuple[torch.Tensor, torch.
101                                         Tensor]:
102     videos = torch.stack([item[0] for item in batch])
103     labels = torch.tensor([item[1] for item in batch], dtype=torch.long)
104     return videos, labels

```

Ngoài việc sampling, VideoTransform (được gọi bên trong `__getitem__`) sẽ áp dụng các kỹ thuật tăng cường dữ liệu như đã nhắc ở phần II.1. để giúp mô hình học tốt hơn và tránh overfitting.

HMDB51 Dataset Transform

```

1 class VideoTransform:
2     def __init__(self, image_size: int, is_train: bool = True):
3         self.image_size = image_size
4         self.is_train = is_train
5         self.mean = [0.5, 0.5, 0.5]
6         self.std = [0.5, 0.5, 0.5]
7
8     def __call__(self, frames: torch.Tensor) -> torch.Tensor:
9         # frames: [T, C, H, W]
10
11         if self.is_train:
12             # Random resized crop (scale 0.8-1.0)
13             h, w = frames.shape[-2:]
14             scale = random.uniform(0.8, 1.0)
15             new_h, new_w = int(h * scale), int(w * scale)
16             frames = TF.resize(frames, [new_h, new_w], interpolation=InterpolationMode.
17                                 BILINEAR)
18
19             # Random crop to target size
20             i = random.randint(0, max(0, new_h - self.image_size))
21             j = random.randint(0, max(0, new_w - self.image_size))
22             frames = TF.crop(frames, i, j, min(self.image_size, new_h), min(self.
23                               image_size, new_w))
24             frames = TF.resize(frames, [self.image_size, self.image_size],
25                               interpolation=InterpolationMode.BILINEAR)
26
27             # Horizontal flip
28             if random.random() < 0.5:
29                 frames = TF.hflip(frames)
30
31             # Color jitter (brightness, contrast, saturation) - nhẽ
32             if random.random() < 0.3:
33                 brightness_factor = random.uniform(0.9, 1.1)
34                 frames = TF.adjust_brightness(frames, brightness_factor)
35
36             if random.random() < 0.3:
37                 contrast_factor = random.uniform(0.9, 1.1)
38                 frames = TF.adjust_contrast(frames, contrast_factor)

```

```

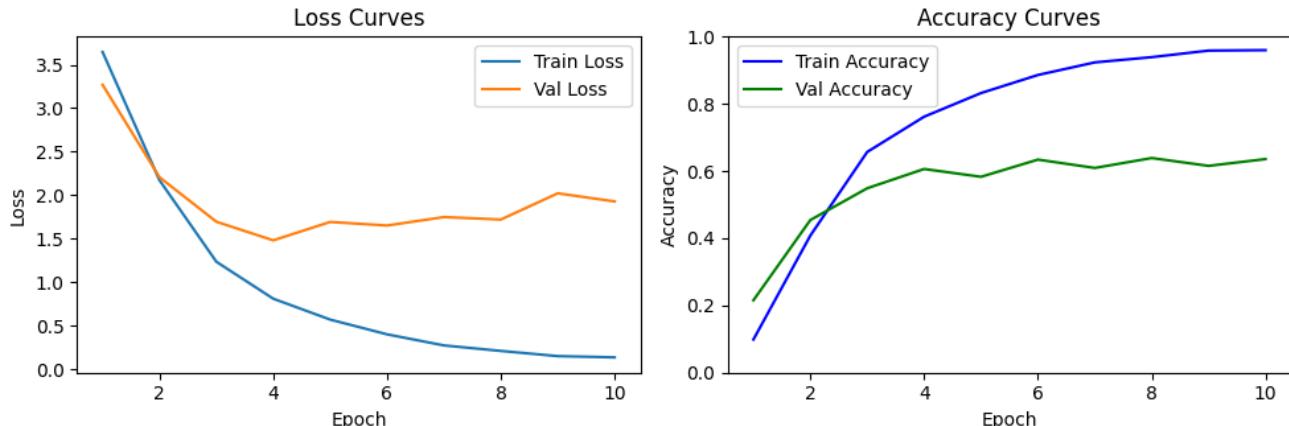
37     if random.random() < 0.3:
38         saturation_factor = random.uniform(0.9, 1.1)
39         frames = TF.adjust_saturation(frames, saturation_factor)
40     else:
41         # Val/test: center crop
42         frames = TF.resize(frames, [self.image_size, self.image_size],
43                             interpolation=InterpolationMode.BILINEAR)
44
45         # Normalize
46         normalized = [TF.normalize(frame, self.mean, self.std) for frame in frames]
47         return torch.stack(normalized)

```

VIII. Huấn luyện và đánh giá

Quá trình huấn luyện sử dụng vòng lặp của PyTorch với AdamW optimizer và CrossEntropyLoss. Ta cũng sử dụng GradScaler để hỗ trợ huấn luyện với độ chính xác hỗn hợp (mixed precision), giúp tăng tốc độ và giảm bộ nhớ.

Sau khi huấn luyện, ta trực quan hóa kết quả bằng biểu đồ Loss và Accuracy, đồng thời hiển thị dự đoán trên các video mẫu từ tập kiểm thử.



Hình 15: Biểu đồ minh họa sự thay đổi của hàm mất mát và độ chính xác qua các epoch.



Hình 16: Kết quả dự đoán thực tế của mô hình trên một video clip mẫu.

IX. Câu hỏi trắc nghiệm

1. (Lý thuyết – Video Classification) Điểm khác biệt cốt lõi giữa bài toán phân loại video và phân loại ảnh tĩnh trong nội dung tài liệu là gì?
 - (a) Video chỉ cần trích xuất đặc trưng không gian, không cần xét thời gian.
 - (b) Video có thêm chiều thời gian, yêu cầu mô hình vừa học đặc trưng không gian theo từng frame vừa mô hình hoá phụ thuộc theo thời gian.
 - (c) Video luôn có nhiều nhãn cho mỗi clip nên khó hơn ảnh.
 - (d) Video không cần tiền xử lý vì dữ liệu đã đồng nhất độ dài.
2. (Lý thuyết – HMDB51) Nhận định nào sau đây mô tả đúng về bộ dữ liệu HMDB51 theo tài liệu?
 - (a) Gồm 51 lớp hành động, video được thu thập từ nhiều nguồn như phim ảnh và video công cộng.
 - (b) Chỉ gồm các hành động khuôn mặt, không có hành động toàn thân.
 - (c) Tất cả video được quay trong điều kiện ánh sáng và góc nhìn giống nhau để giảm nhiễu.
 - (d) Chỉ dùng cho bài toán phát hiện đối tượng, không dùng cho nhận diện hành động.
3. (Lý thuyết – EDA & Augmentation) Khi dữ liệu bị *imbalance* mạnh, hệ quả trực tiếp trong huấn luyện được nhấn mạnh trong tài liệu là gì?
 - (a) Mô hình chắc chắn không thể hội tụ dù thay đổi hyperparameter.
 - (b) Cần chú ý lựa chọn tăng cường dữ liệu, hàm loss và cơ chế training phù hợp.
 - (c) Chỉ cần tăng batch size là đủ để cân bằng dữ liệu.
 - (d) Imbalance ảnh hưởng nếu dùng Transformer.
4. (Lý thuyết – LS-ViT) Theo tài liệu, LS-ViT giải quyết thách thức thời gian bằng cách kết hợp hai luồng thông tin nào?
 - (a) Âm thanh và hình ảnh.
 - (b) Không gian (spatial) và thời gian (temporal).
 - (c) Cạnh (edge) và màu sắc (color).
 - (d) Đặc trưng thủ công và đặc trưng học sâu.
5. (Lý thuyết – Regularization) DropPath (Stochastic Depth) trong tài liệu khác Dropout ở điểm chính nào?
 - (a) DropPath ngắt ngẫu nhiên toàn bộ một nhánh residual theo mẫu trong batch trong quá trình huấn luyện.
 - (b) DropPath chỉ dùng để tăng số tham số của mô hình.

- (c) DropPath luôn làm thay đổi kích thước tensor đầu ra.
- (d) DropPath chỉ áp dụng được khi mô hình không có residual connection.
6. (Tính toán – PatchEmbed) Cho ảnh đầu vào có kích thước (B, C, H, W) với $H = W = 224$, patch size $P = 16$, số kênh embedding $D = \text{embed_dim} = 768$. Số lượng patch được tính bởi:

$$N = \left(\frac{H}{P}\right) \left(\frac{W}{P}\right).$$

Giá trị N là:

- (a) 196
- (b) 224
- (c) 256
- (d) 784
7. (Tính toán – Shape sau PatchEmbed) Với cấu hình ở câu trên, đầu ra của PatchEmbed sau khi chuyển sang dạng chuỗi token có dạng (B, N, D) . Kích thước đúng của tensor đầu ra là:
- (a) $(B, 196, 768)$
- (b) $(B, 768, 196)$
- (c) $(B, 14, 14, 768)$
- (d) $(B, 224, 224, 3)$
8. (Tính toán – MSA: kích thước ma trận attention) Trong Multi-Head Self-Attention, giả sử đầu vào có dạng (B, N, C) , số head là h , khi tách head ta có $d_k = C/h$. Ma trận attention trước softmax thường có dạng:

$$A = \frac{QK^\top}{\sqrt{d_k}}, \quad Q \in \mathbb{R}^{B \times h \times N \times d_k}, \quad K \in \mathbb{R}^{B \times h \times N \times d_k}.$$

Kích thước của A là:

- (a) (B, h, N, N)
- (b) (B, h, N, d_k)
- (c) (B, h, d_k, N)
- (d) (B, N, N, h)
9. (Tính toán – Backbone reshape theo thời gian) Trong LSViTBackbone, video đầu vào có dạng (B, T, C, H, W) và được gộp batch-thời gian để xử lý như ảnh tĩnh theo công thức:

$$(B, T, C, H, W) \rightarrow (B \cdot T, C, H, W).$$

Nếu $B = 4$ và $T = 16$, thì kích thước mới của chiều batch sau khi gộp là:

- (a) 20

- (b) 32
(c) 64
(d) 128
10. (Tính toán – Pooling theo thời gian ở Head) Trong `LSViTForAction`, sau backbone ta lấy `cls_tokens` theo thời gian và thực hiện trung bình theo trực thời gian:

$$\text{cls_tokens} \in \mathbb{R}^{B \times T \times C}, \quad \text{pooled} = \frac{1}{T} \sum_{t=1}^T \text{cls_tokens}_{:,t,:}$$

Kích thước của `pooled` là:

- (a) (B, T, C)
(b) (B, C)
(c) (T, C)
(d) (B, N, C)

Phụ lục

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp [tại đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

🔒 Nhóm Riêng tư · 1,4K thành viên



Hình 17: Hình ảnh group facebook AIO Q&A

4. **Đề xuất phương pháp cải tiến project:** Một số phương pháp đề xuất cải tiến model này:

- **Tiền xử lý và đảm bảo chất lượng dữ liệu**

- Nâng cấp kỹ thuật tăng cường dữ liệu (Data Augmentation) để giải quyết vấn đề mất cân bằng dữ liệu: Thay thế các biến đổi cơ bản (crop, flip, jitter) bằng các phương pháp trộn ảnh hiện đại cho Transformer như **Mixup** hoặc **CutMix**, giúp mô hình học được sự kết hợp phi tuyến và giảm overfitting.
- Cải tiến chiến lược lấy mẫu (Sampling Strategy): Thay vì *Uniform Sampling* kết hợp *Temporal Padding* đơn thuần, hãy thử nghiệm **Dense Sampling** hoặc **TSN-style sampling** (chia video thành các đoạn và lấy ngẫu nhiên 1 frame mỗi đoạn) để nắm bắt ngữ cảnh thời gian đa dạng hơn mà không cần tăng số lượng frame đầu vào quá lớn.

- **Biểu diễn đặc trưng (Feature Engineering) có kiểm soát**

- Chuyển đổi từ 2D Patch Embedding sang **3D Patch Embedding (Tubelet)**: Thay vì dùng Conv2d kernel (16, 16) hiện tại, hãy sử dụng Conv3d với kernel (2, 16, 16) để trích xuất đặc trưng thời gian ngay từ lớp embedding đầu tiên.
- Tận dụng Motion Vector từ codec: Do dự án đã sử dụng thư viện **torchcodec** để decode video, có thể trích xuất trực tiếp *motion vectors* (vector chuyển động) làm đầu vào bổ trợ nhẹ nhàng cho module SMIF, thay vì chỉ dựa vào tính toán sai biệt frame (frame differencing) thuận tủy.

- **Mô hình hóa và quy chuẩn hóa (Regularization) nâng cao**

- Áp dụng **Transfer Learning** (Học chuyển tiếp): Thay vì khởi tạo mô hình từ đầu với cấu hình tự định nghĩa, hãy khởi tạo trọng số Backbone từ các mô hình đã

huấn luyện trước (Pre-trained) trên ImageNet-21k hoặc Kinetics-400 để cải thiện khả năng hội tụ trên tập dữ liệu nhỏ HMDB51.

- Tối ưu hàm mất mát (Loss Function): Thay thế **CrossEntropyLoss** tiêu chuẩn bằng **Label Smoothing Loss** (với $\epsilon \approx 0.1$). Kỹ thuật này ngăn mô hình trở nên quá tự tin vào các nhãn nhiễu, đặc biệt hiệu quả với các hành động có tính nhập nhằng cao trong HMDB51.

- **Giải thích kết quả mô hình**

- Trực quan hóa **Spatio-Temporal Attention**: Xuất trọng số từ module *LMIModule* hoặc các lớp Attention để tạo bản đồ nhiệt (heatmap) đè lên video, giúp minh bạch hóa việc mô hình đang tập trung vào vùng không gian hay khung hình thời gian nào khi ra quyết định.

- **Đánh giá mô hình**

- Bổ sung chỉ số **Top-k Accuracy** ($k = 5$): Bên cạnh Accuracy tiêu chuẩn, báo cáo thêm Top-5 Accuracy để đánh giá khả năng mô hình nhận diện đúng hành động nằm trong nhóm dự đoán cao nhất.
- Phân tích **Confusion Matrix** (Ma trận nhầm lẫn): Vẽ ma trận nhầm lẫn trên 51 lớp để xác định cụ thể các cặp hành động thường xuyên bị dự đoán sai (ví dụ: *sword* vs *sword_exercise*), từ đó tinh chỉnh lại chiến lược augmentation cho riêng các lớp này.

5. Rubric:

LS-ViT Video Action Recognition - Rubric		
Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Hiểu bản chất bài toán phân loại video: sự khác biệt so với ảnh tĩnh và thách thức về chiều thời gian. - Nắm vững đặc điểm bộ dữ liệu HMDB51 và các vấn đề dữ liệu thực tế (imbalance, điều kiện ánh sáng). - Hiểu các kỹ thuật EDA và tiền xử lý video cơ bản (decode, visualize). 	<ul style="list-style-type: none"> - Phân tích được các thách thức (spatial & temporal) khi xử lý dữ liệu video. - Trả lời đúng các câu hỏi trắc nghiệm lý thuyết về phân loại video, EDA và các kỹ thuật Regularization (DropPath).
2	<ul style="list-style-type: none"> - Nắm vững kiến trúc Vision Transformer (ViT) cơ bản: Patch Embedding, Multi-Head Self-Attention. - Hiểu sâu về module SMIF (Spatial-Motion Interaction Fusion) để xử lý chuyển động ngắn hạn. - Hiểu sâu về module LMI (Long-term Motion Interaction) để nắm bắt phụ thuộc dài hạn. 	<ul style="list-style-type: none"> - Hoàn thiện code (Exercise) cho lớp ‘PatchEmbed’ (chuyển đổi ảnh thành chuỗi token). - Hoàn thiện code cho cơ chế ‘Attention’ (tính toán Q, K, V và ma trận attention). - Cài đặt/Hoàn thiện hàm ‘forward’ cho các module đặc trưng SMIF và LSViTBlock.
3	<ul style="list-style-type: none"> - Hiểu quy trình xây dựng Dataset class: chiến lược Uniform Sampling và Temporal Padding. - Nắm bắt các kỹ thuật tăng cường dữ liệu video (Random crop, Flip, Color jitter). - Hiểu quy trình huấn luyện (Backbone, Head) và đánh giá mô hình phân loại hành động. 	<ul style="list-style-type: none"> - Xây dựng thành công pipeline dữ liệu ‘HMDB51Dataset’ đảm bảo xử lý được video độ dài khác nhau. - Thực hiện huấn luyện, trực quan hóa biểu đồ Loss/Accuracy và phân tích kết quả dự đoán trên video thực tế.

AI VIET NAM – AI COURSE 2025

Hướng dẫn nộp bài trên Kaggle Competition

Nguyễn Quốc Thái Hồ Quang Hiển Đinh Quang Vinh

I. Giới thiệu

Tài liệu này trình bày quy trình nộp bài cho một cuộc thi trên Kaggle theo đúng thứ tự hệ thống chấm điểm của nền tảng. Người tham gia sẽ bắt đầu từ việc mở và sao chép notebook baseline, sau đó chỉnh sửa hoặc chạy notebook để sinh ra tệp `submission.csv`. Tiếp theo, thí sinh có thể nộp bài trực tiếp thông qua notebook hoặc tải thủ công tệp submission lên trang cuộc thi. Tổng quan giao diện khi truy cập cuộc thi trên Kaggle như hình dưới.

AIO-2025: Video Action Classification Challenge

AIO-2025 Video Action Classification Challenge yêu cầu người tham gia xây dựng mô hình phân loại hành động trong video bằng cách xử lý chuỗi



A set of videos $X = \{I_1, I_2, \dots, I_n\}$.

Settings Overview Data Code Models Discussion Leaderboard Rules Team



Your competition is ready to launch! You've completed 10 of 10 tasks to launch your competition.

[Launch Checklist](#)

Overview

AIO-2025: Video Action Classification Challenge tập trung vào việc đánh giá hiệu quả của các mô hình dựa trên kiến trúc Vision Transformer (ViT) trong nhiệm vụ Phân loại Hành động trong Video (Video Action Classification).

Start

Set start date via the [launch checklist](#).



Close

22 days to go



Competition Host

WanHin



Prizes & Awards

Kudos [Edit](#)

Does not award Points or Medals

Participation

0 Entrants

0 Participants

0 Teams

0 Submissions

Hình 1: Giao diện khi truy cập vào cuộc thi trên Kaggle.

Toàn bộ quy trình bao gồm các bước như sau:

- Truy cập link cuộc thi và mở notebook baseline từ tab **Code**;
- Chạy notebook để tạo tệp kết quả;
- Nộp notebook hoặc nộp file submission qua tab **Leaderboard**;
- Xem điểm số cũng như xếp hạng sau khi hệ thống chấm điểm. Phần còn lại của tài liệu sẽ mô tả chi tiết từng bước để đảm bảo người tham gia có thể nộp bài đúng cách và theo dõi kết quả một cách chính xác.

Các mốc thời gian cần chú ý như sau:

- Thời gian bắt đầu mở chính thức cho cuộc thi trên Kaggle là: **00:00 ngày 27/12/2025 đến 23:59 ngày 10/01/2026**
- Thời gian để submit code sau khi cuộc thi kết thúc lên Kaggle là: **00:00 ngày 11/01/2026 đến 23:59 ngày 11/01/2026**

Mục lục

I.	Giới thiệu	1
II.	Link truy cập và mở code baseline	4
III.	Chạy notebook và nộp bài	6
IV.	Nộp file submission và xem kết quả	8
V.	Kết thúc cuộc thi và nộp final code tính điểm	10
	Phụ lục	15

II. Link truy cập và mở code baseline

Đầu tiên chúng ta cần truy cập vào cuộc thi thông qua Competition Link:

Truy cập cuộc thi tại đây

The screenshot shows the AIO-2025 Video Action Classification Challenge competition page. At the top, there is a banner with several small video frames showing people performing actions. Below the banner, the text reads: "AIO-2025 Video Action Classification Challenge yêu cầu người tham gia xây dựng mô hình phân loại hành động trong video bằng cách xử lý chuỗi". The navigation bar includes links for Settings, Overview (highlighted with a red arrow labeled 1), Code, Models, Discussion, Leaderboard, Rules, and Team.

In the center, a message says: "10/10 Your competition is ready to launch! You've completed 10 of 10 tasks to launch your competition." To the right is a "Launch Checklist" button. Below this, the "Notebooks" section is shown. It has a search bar, a "Filters" button, and a "Hotness" dropdown set to "All". The "All" tab is highlighted with a red arrow labeled 2. Under the "All" tab, a notebook titled "Baseline BTC" is listed, showing it was updated 1d ago and has 0 comments. To the right of this notebook is a red arrow labeled 3 pointing to a three-dot menu icon. At the bottom right of the screenshot, there is a small number "4" inside a circle.

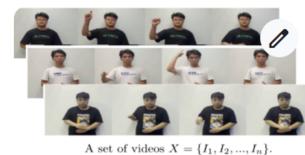
Hình 2: Chọn tab Code và notebook baseline.

Trong bước này, bạn thực hiện:

- Nhấn vào tab **Code** trên trang competition (mũi tên 1).
- Ở phần Notebooks, chọn **All** để hiển thị tất cả notebook (mũi tên 2).
- Tại notebook baseline muốn dùng để nộp bài, nhấn nút **...** ở bên phải để mở menu thao tác (mũi tên 3).

AIO-2025: Video Action Classification Challenge

AIO-2025 Video Action Classification Challenge yêu cầu người tham gia xây dựng mô hình phân loại hành động trong video bằng cách xử lý chuỗi



Settings Overview Data **Code** Models Discussion Leaderboard Rules Team

10/10

Your competition is ready to launch! You've completed 10 of 10 tasks to launch your competition.

Launch Checklist

Notebooks

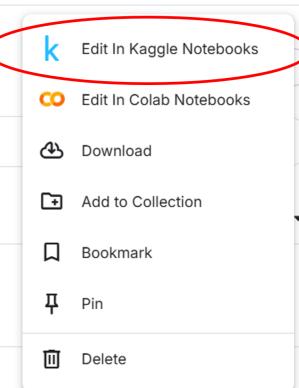
Search notebooks

All Your Work Shared With You Bookmarks



Baseline BTC
Updated 1d ago
0 comments · AIO-2025: Video Action Classification Challenge

Tùy chỉnh baseline

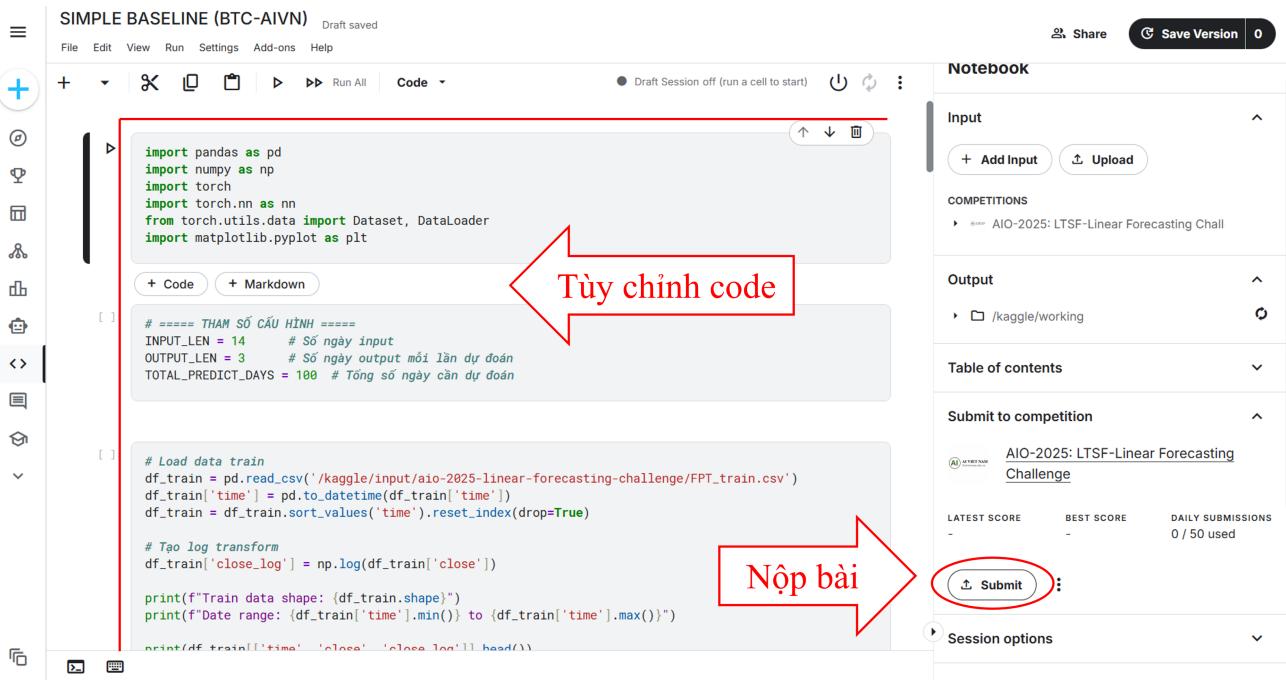


Hình 3: Copy và Edit baseline notebook.

Sau khi menu hiện ra như hình trên, chọn mục **Copy and Edit in Kaggle Notebooks** để mở notebook **SIMPLE BASELINE** trong môi trường Kaggle và chuẩn bị chạy notebook để sinh file **submission.csv**.

Nếu bạn muốn tạo một notebook mới từ đầu thì nhấp vào ô "New Notebook"

III. Chạy notebook và nộp bài



Hình 1: Tuỳ chỉnh code và nhấn Submit để nộp bài

Trong bước này, bạn thực hiện:

- Tuỳ chỉnh (hoặc chạy thử) phần code trong notebook sao cho sinh ra file `submission.csv`, file được tạo nằm trong folder output và Kaggle sẽ lấy file để tính điểm.
- Ở panel **Submit to competition** bên phải nhấn nút **Submit** để gửi notebook lên hệ thống chấm điểm, hệ thống sẽ chạy lại tất cả cell và tính điểm tự động dựa vào file `submission.csv` được tạo.

The screenshot shows a Jupyter Notebook interface. On the left, there's a sidebar with various icons. The main area contains a code cell with Python code for data loading and processing. Below the code cell, a message box says "Chấm điểm xong" (Graded). To the right of the code cell, there's a "Notebook" panel with sections for "Input" and "Output". Under "Output", it shows a competition submission status: "AIO-2025: LTSF-Linear Forecasting Chall... just now Competition submission Complete". Further down, under "Submit to competition", it shows the "AIO-2025: LTSF-Linear Forecasting Challenge" page with "LATEST SCORE 1922.4857 V1", "BEST SCORE 1922.4857 V1", and "DAILY SUBMISSIONS 1 / 50 used". A large red arrow points from the "Chấm điểm xong" message to the competition submission status, and another red arrow points from the competition submission status to the "Kết quả" (Results) section.

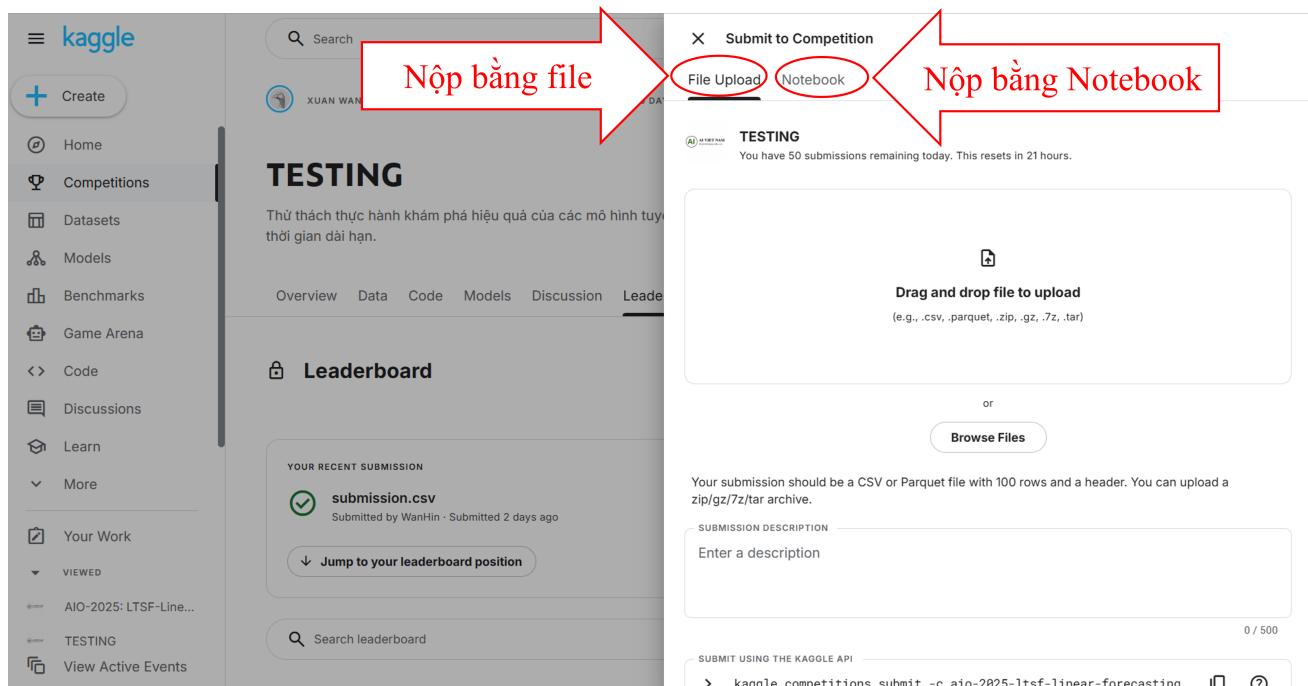
Hình 2: Notebook được chấm điểm và hiển thị kết quả

Sau khi Kaggle chạy xong notebook:

- Ở góc bên trái phía dưới xuất hiện mục **Competition submission** với trạng thái **Complete**.
- Trong phần **Submit to competition** sẽ hiện **Latest score**, **Best score** và số lần nộp trong ngày, cho biết bài nộp đã được chấm thành công.

IV. Nộp file submission và xem kết quả

Ngoài cách nộp bài ở Bước 2 ta vẫn còn cách khác để nộp bài.



Hình 3: Nhấn nút Submit Prediction trên tab Leaderboard

Bạn cần thực hiện các bước:

- Trên trang competition, chuyển sang tab **Leaderboard**.
- Nhấn nút **Submit Prediction** ở góc trên bên phải để mở cửa sổ nộp bài.

The screenshot shows the Kaggle interface for the AIO-2025: LTSF-Linear Forecasting Challenge. The 'Leaderboard' tab is highlighted with a red circle. On the left sidebar, there is a red box around the 'Kết quả' (Results) section. The main area displays a recent submission named 'submission.csv' with a score of 1922.4857. The leaderboard table shows one entry from 'WanHin' with a score of 1922.4857. A red box highlights the first row of the leaderboard table.

#	Team	Members	Score	Entries	Last	Join
1	WanHin		1922.4857	1	18m	

Hình 4: Chọn cách nộp: File Upload hoặc Notebook

Tại cửa sổ Submit to Competition:

- Chọn tab **File Upload** nếu bạn đã có sẵn file **submission.csv** và muốn upload trực tiếp.
- Hoặc chọn tab **Notebook** nếu muốn nộp thông qua một notebook đã chạy trên Kaggle.
- Với cách nộp file, kéo thả hoặc chọn file **submission.csv** rồi xác nhận nộp bài.

The screenshot shows the Kaggle interface for the XUAN WAN - COMMUNITY PREDICTION COMPETITION - PRIVATE - 20 DAYS TO GO. The 'Leaderboard' tab is highlighted with a red circle. On the left sidebar, there is a red box around the 'TESTING' section. The main area displays a recent submission named 'submission.csv' with a score of 350.76. A red box highlights the 'Nộp bài' (Upload) button. To the right, there is a chart with annotations 'Quá khứ' (Past) and 'Tương lai' (Future).

Hình 5: Kết quả điểm số và thứ hạng trên leaderboard

Sau khi hệ thống chấm điểm xong, trang **Leaderboard** sẽ hiển thị:

- Tên file submission mới nhất.
- Điểm số tương ứng (Public score).
- Vị trí của bạn trên bảng xếp hạng.

V. Kết thúc cuộc thi và nộp final code tính điểm

Vào tab **Code** trong trang cuộc thi, chọn **Your Work**, sau đó nhấn vào notebook bạn muốn dùng làm bản **final** để nộp chấm điểm.

The screenshot shows the competition interface with the 'Code' tab selected. In the 'Your Work' section, there is a notebook titled 'Baseline BTC' with a red arrow pointing to it labeled 'Chọn final notebook'.

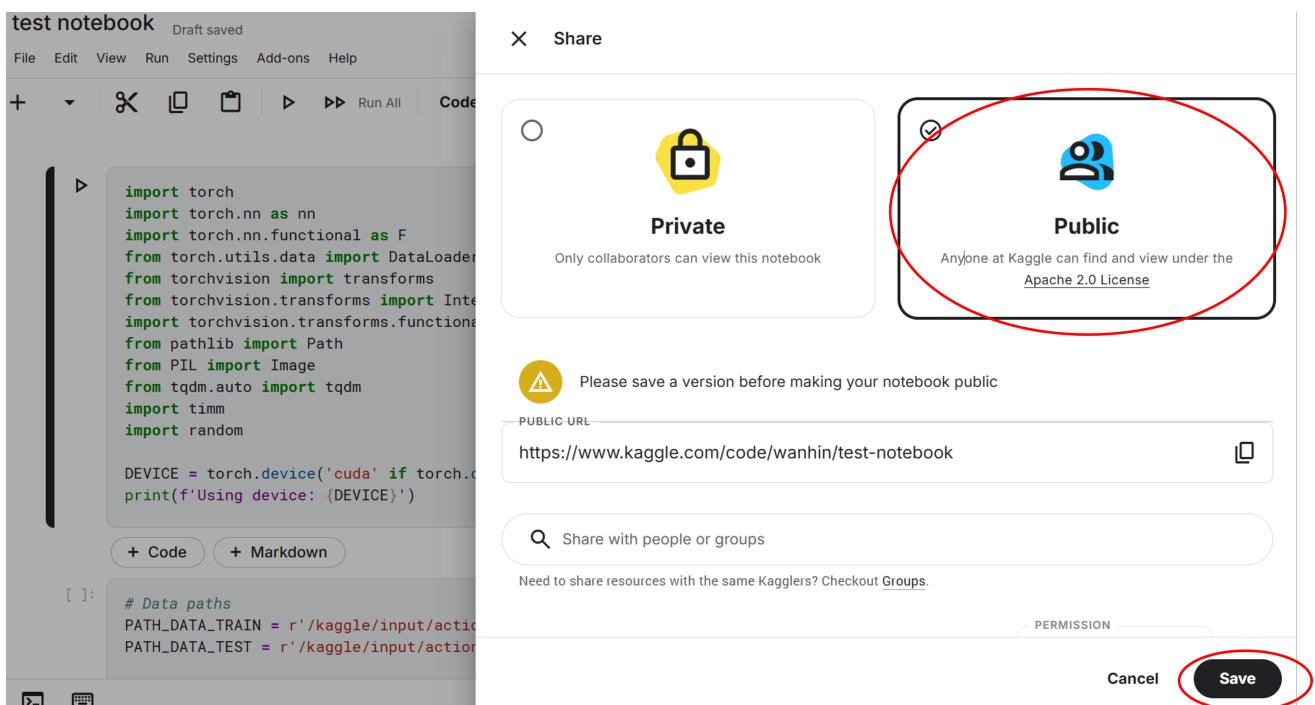
Hình 4: Vào Code → Your Work và chọn notebook final

Trong notebook đã chọn, nhấn **Share** để mở phần chia sẻ và bắt đầu thiết lập quyền truy cập cho notebook.



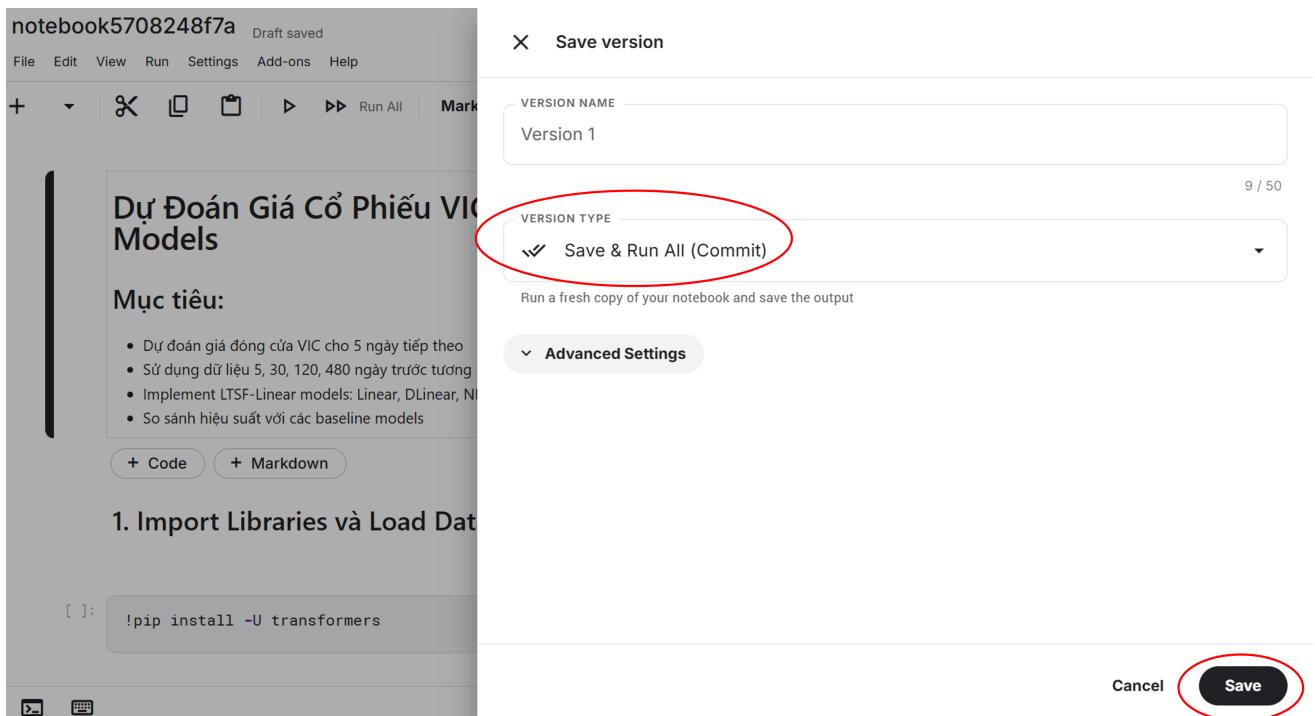
Hình 5: Mở Share để tiến hành công khai notebook

Chọn chế độ **Public** và nhấn **Save** để notebook có thể được hệ thống cuộc thi truy cập khi bạn nộp bài.



Hình 6: Chọn Public và Save

Tiếp theo, nếu bạn chưa chạy notebook này thì bạn cần chọn **Save & Run All (Commit)** để đảm bảo Kaggle hiểu thông tin log run notebook của bạn.



Hình 7: Save và run notebook vừa public để Kaggle hiểu thông tin sau khi chạy xong

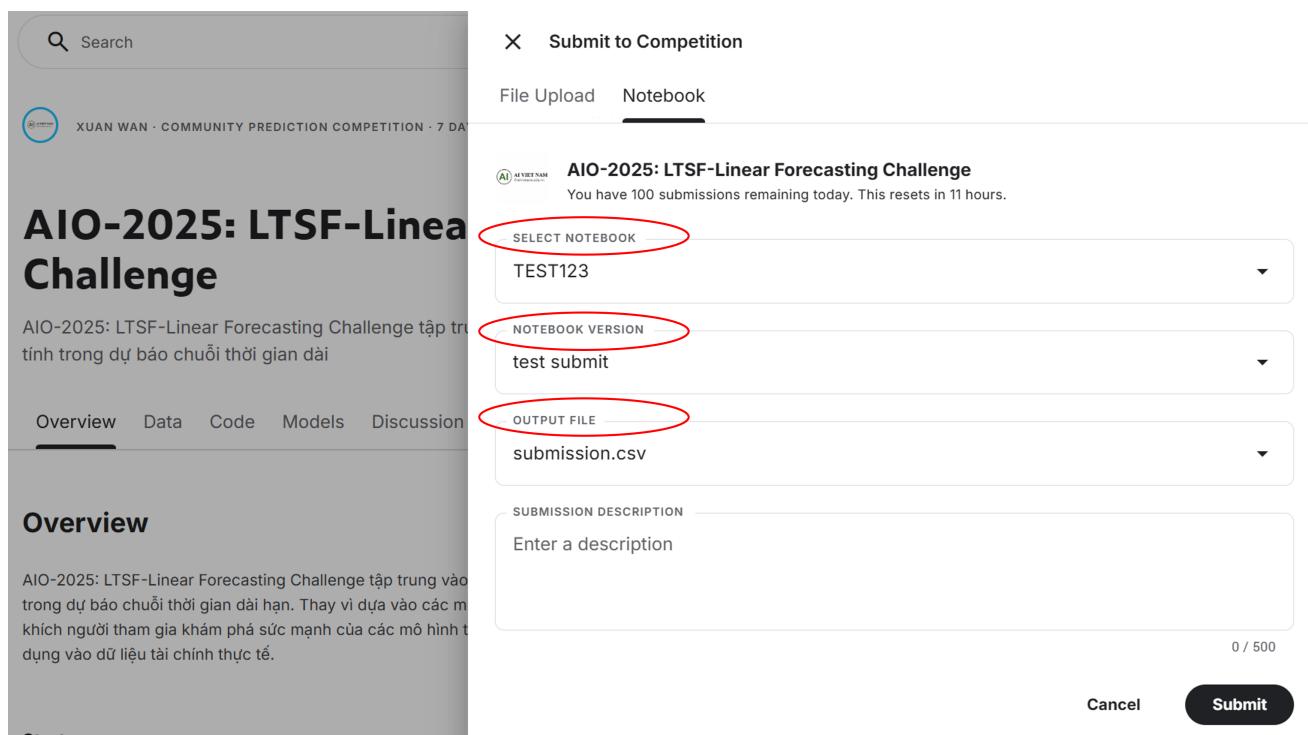
Về lại trang cuộc thi, nhấn **Late Submission** để bắt đầu quy trình nộp bài tính điểm sau khi notebook đã chạy xong.

The screenshot shows the AIO-2025 challenge page. At the top, there's a user profile for 'XUAN WAN - COMMUNITY PREDICTION COMPETITION - 7 DAYS AGO'. Below it, the challenge title is 'AIO-2025: LTSF-Linear Forecasting Challenge'. A descriptive text below the title says 'AIO-2025: LTSF-Linear Forecasting Challenge tập trung vào việc đánh giá hiệu quả của các mô hình tuyến tính trong dự báo chuỗi thời gian dài'. A navigation bar at the bottom includes 'Overview', 'Data', 'Code', 'Models', 'Discussion', 'Leaderboard', 'Rules', 'Team', 'Submissions', and 'Settings', with 'Overview' being underlined. To the right of the chart, there are sections for 'Competition Host' (Xuan Wan), 'Prizes & Awards' (Kudos Edit, Does not award Points or Medals), and a logo for 'AI VIET NAM'.

Hình 8: Vào lại trang cuộc thi và chọn Late Submission

Trong phần nộp bài, chọn **Loại nộp là Notebook**, sau đó chọn đúng **Notebook** vừa chạy xong, chọn đúng **Version** tương ứng, chọn đúng **File đầu ra để chấm điểm**, rồi nhấn **Submit**. Khi hệ thống xử lý xong, submission sẽ được chấm và trả về điểm.

Lưu ý: Cần đặt tên notebook đúng theo **tên nhóm** để ban tổ chức ghi nhận điểm chính xác.



Hình 9: Chọn Notebook, Version, Output File và Submit

Sau khi nộp, quay lại tab **Code** và vào mục **All** để kiểm tra notebook của bạn đã thực sự được share và đã hiện **Score** hay chưa. Đây là bước xác nhận cuối cùng để đảm bảo hệ thống đã chấm điểm thành công.

Sau đó, TA quản lý sẽ vào từng notebook đã share để đối soát và ghi nhận điểm của các nhóm.

AIO-2025: LTSF-Linear Forecasting Challenge

Late Submission ...

Overview Data **Code** Models Discussion Leaderboard Rules Team Submissions Settings

Notebooks

+ New Notebook

Search notebooks Filters

All Your Work Shared With You Bookmarks Hotness ▾

Pinned notebooks

 SIMPLE BASELINE (BTC-AIVN)
Updated 1mo ago
Score: 422.4133 · 0 comments · AIO-2025: LTSF-Linear Forecasting Challenge

Unpinned notebooks

 time_series
Updated 11d ago
Score: 36.6791 · 0 comments

Đã chấm điểm

Hình 10: Kiểm tra Score trong tab Code → All

Phụ lục

1. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

🔒 Nhóm Riêng tư · 1,4K thành viên



Hình 11: Hình ảnh group facebook AIO Q&A