AI VIET NAM – AI COURSE 2023

Video Classification Project

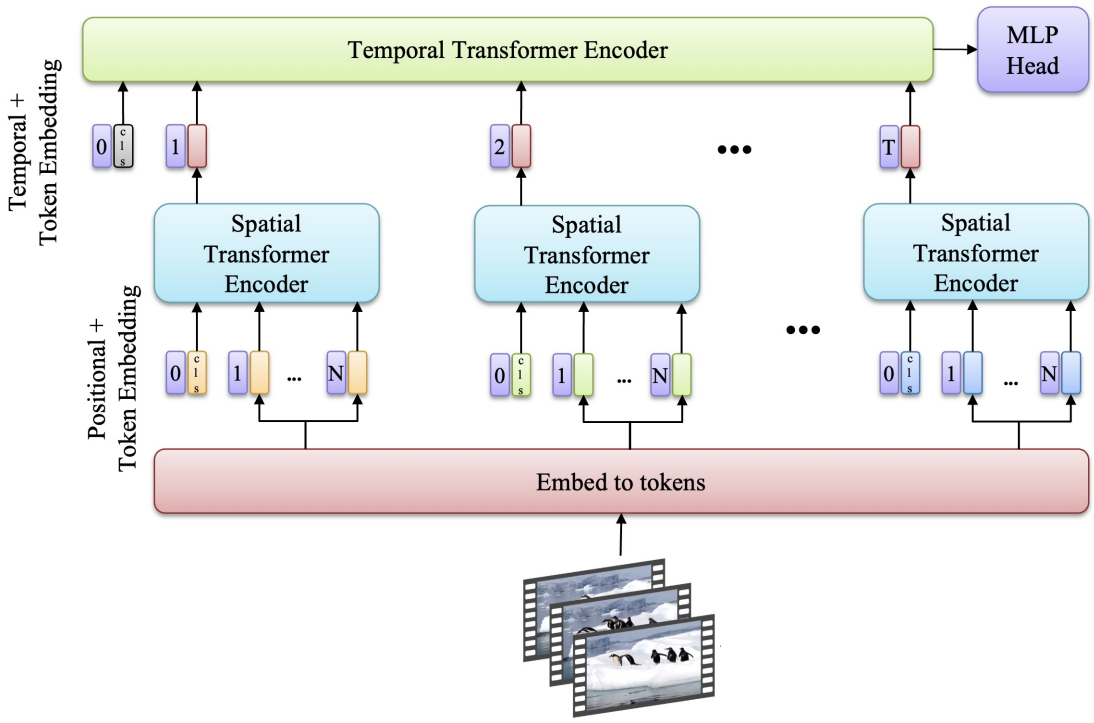
Minh-Duc Bui và Quang-Vinh Dinh

Ngày 2 tháng 5 năm 2024

Phần I: Giới thiệu

Video classification là một bài toán quan trọng trong thị giác máy tính và trí tuệ nhân tạo, liên quan đến việc tự động phân loại và gán nhãn các video dựa trên nội dung.

Các ứng dụng của video classification rất đa dạng, từ an ninh và giám sát, cải thiện các hệ thống gợi ý nội dung trên nền tảng xem video như YouTube hay Netflix, đến phân tích hành vi người dùng và hỗ trợ các phương tiện tự lái. Để đạt được hiệu quả cao, các model phân loại video cần được train trên các bộ dữ liệu lớn, đa dạng và thường phải xử lý các thách thức như biến động của môi trường, sự thay đổi góc quay, và sự đa dạng về hành vi và tương tác trong video.



Hình 1: Minh họa kiến trúc Video Vision Transformer (ViViT) cho bài toán video classification.

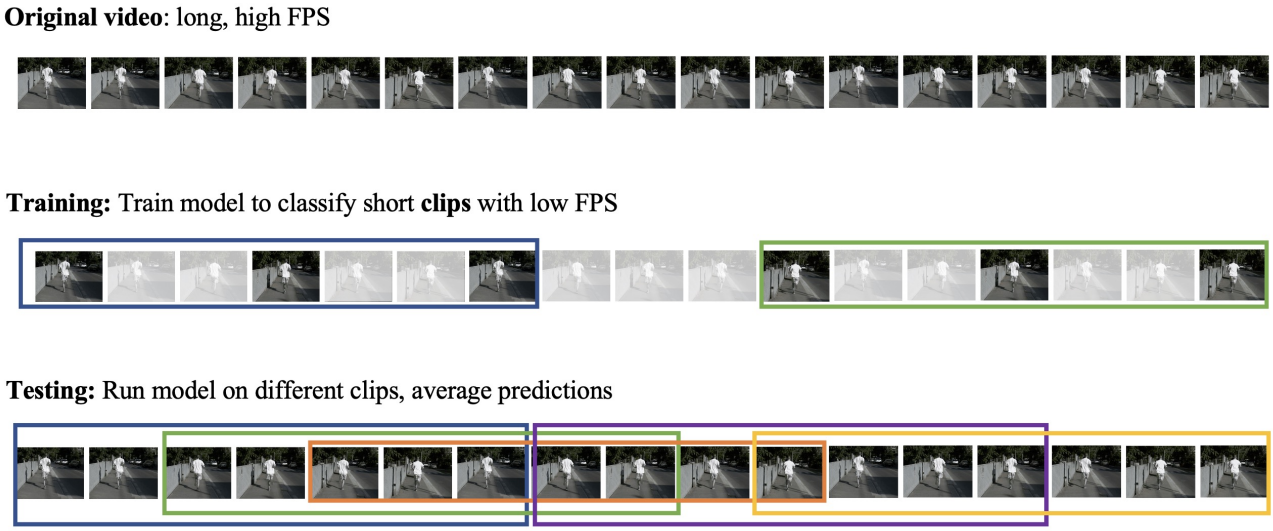
Trong project này, ta sẽ triển khai một số model phân loại video từ cơ bản đến nâng cao.

Phần II: Nội dung

1. **Giới thiệu Video data**

   - **Video data**: Video là chuỗi của các frame theo thời gian, ví dụ một video có `T` frame và mỗi frame có kích thước `(3 x W x H)` thì video sẽ có kích thước `(T x 3 x W x H)` (tensor 4 chiều).
   - **Đặc điểm**: Video có kích thước rất lớn, ví dụ một video thông thường có khoảng 30 frame, khi kích thước mỗi frame là `(3 x 640 x 480)` thì dung lượng của video (khi chưa nén) xấp xỉ `1.5GB/phút`, khi kích thước mỗi frame là `(3 x 1920 x 1080)` thì dung lượng xấp xỉ `10GB/phút`. Video có kích thước rất lớn khi so sánh với các loại data khác như ảnh, âm thanh, hoặc text. Vì thế, trong các bài toán video classification, ta thường dùng video có kích thước nhỏ, `(T x 3 x 112 x 112)` hoặc `(T x 3 x 224 x 224)` cùng với số lượng frame `T` nhỏ (ví dụ `16 FPS`).

     Đối với các trường hợp video có tính thống nhất từ đầu đến cuối, tức label của video sẽ giống nhau dù ta chỉ nhìn vào 1 phần ngắn của video thì ta có thể chia video thành nhiều clip ngắn và train model. Ví dụ ta có 1 video 30s phân loại các hoạt động thể thao, ta có thể chia video thành các clip ngắn (5s) rồi train model. Khi inference thực tế, ta sẽ chia video thành nhiều clip ngắn rồi predict trên các clip ngắn này, kết quả cuối cùng sẽ được tổng hợp lại (tham khảo hình 2).

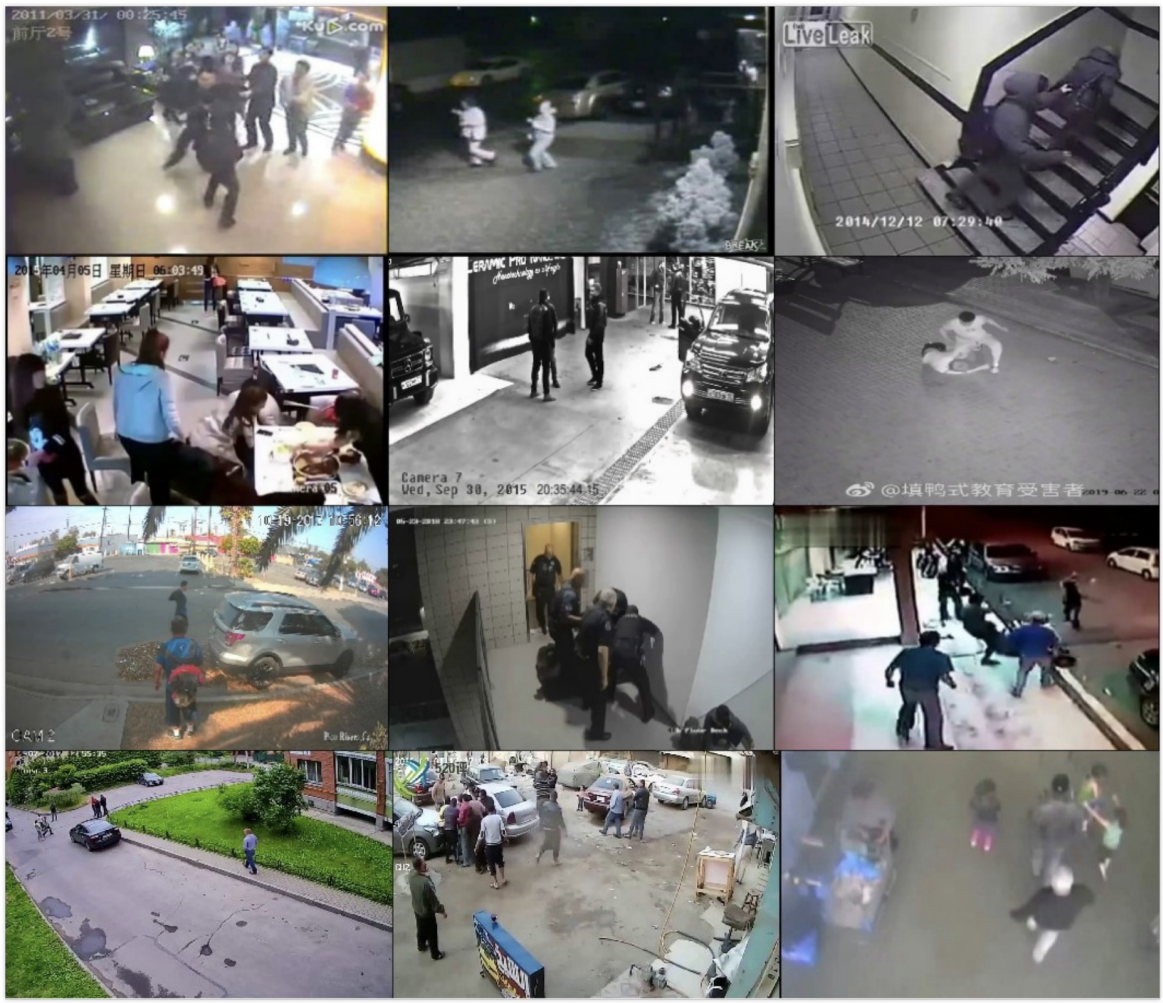


Hình 2: Minh họa cách chia video thành các clip ngắn khi train và cách inference sau khi đã train model.

   - **Thách thức**: các bài toán về video luôn có độ phức tạp rất cao do nhiều tính chất từ video, một số thách thức về video như:
     - Chỉ một phần nhỏ của đối tượng (quyết định label) xuất hiện trong video,
     - Video tại các vị trí đông người,
     - Các hành động chính (quyết định label) chỉ diễn ra trong một khoảng thời gian ngắn trong toàn bộ video,
     - Video có độ phân giải thấp.

2. **Dataset RWF2000 - bài toán violence detection**

RWF2000 là dataset về bài toán violence detection (phát hiện hành vi bạo lực), cụ thể ta sẽ phân loại video dựa vào 2 class fight hoặc non-fight behaviour tương ứng với có hoặc không có hành vi bạo lực trong video. Dataset bao gồm 2000 video, mỗi video dài 5s và được quay ở 30 frame (FPS), tổng cộng ta có 300.000 frame.



Hình 3: Một số sample từ RWF2000.

3. **Cấu trúc data và dataset class**

Data bao gồm 2 folder chính là train và val, trong mỗi folder chính sẽ có 2 subfolder tương ứng với 2 class là Fight and NonFight. Trong mỗi class bao gồm các folder chứa video, mỗi folder chứa 30 frame của video đó.

```
- dataset/rwf-2000
  - train
    - Fight
      - video1
        - frame1.jpg
          ..
        - frame30.jpg
        ...
      - videoN
    - NonFight
  - val
    - Fight
    - NonFight
```

VideoDataset Class:

- `__init__`: Hàm khởi tạo nhận vào `root_dir` (đường dẫn tới thư mục chứa dataset), `phase` để chỉ định tập dữ liệu là `train` hoặc `val`, `transform` để áp dụng các biến đổi cho các frame, và `n_frames` để chỉ định số lượng frame sẽ được lấy từ mỗi video.
- `_load_videos`: Hàm load tất cả các đường dẫn đến frame của các video. Hàm duyệt qua từng thư mục trong `train` hoặc `val`, lấy đường dẫn của từng frame trong mỗi video, sau đó sắp xếp theo thứ tự số. Nếu `n_frames` được chỉ định, hàm sẽ lấy các frame theo uniform distribution bằng cách sử dụng hàm `_uniform_sample`.
- `_uniform_sample`: Hàm chọn ra `n_frames` từ danh sách frames theo uniform distribution.
- `__getitem__`: Trả về data và label tương ứng khi chỉ định index. Các frame được `transform` (nếu có), và stack lại để tạo thành một tensor 4 chiều `(C, T, H, W)`.

```python
class VideoDataset(Dataset):
    def __init__(self, root_dir, phase="train", transform=None, n_frames=None):
        """
        Args:
            root_dir (string): Directory with all the videos (each video as a subdirectory of frames).
            transform (callable, optional): Optional transform to be applied on a sample.
            n_frames (int, optional): Number of frames to sample from each video, uniformly. If None, use all frames.
        """
        self.root_dir = root_dir
        self.transform = transform
```

```
        self.n_frames = n_frames
        self.phase = phase
        self.videos, self.labels = self._load_videos()

    def _load_videos(self):
        videos, labels = [], []
        class_id = 0

        video_folders = os.listdir(os.path.join(self.root_dir, self.phase))

        for folder in video_folders:
            video_paths = os.listdir(os.path.join(self.root_dir, self.phase,
 folder))

            for video_path in video_paths:
                video_folder = os.path.join(
                    self.root_dir, self.phase, folder, video_path
                )
                frames = sorted(
                    (os.path.join(video_folder, f) for f in os.listdir(
video_folder)),
                    key=lambda f: int(
                        "".join(filter(str.isdigit, os.path.basename(f)))
                    ),
                )

                if self.n_frames:
                    frames = self._uniform_sample(frames, self.n_frames)

                videos.append(frames)
                labels.append(class_id)

            class_id += 1

        return videos, labels

    def _uniform_sample(self, frames, n_frames):
        """
        Helper method to uniformly sample n_frames from the frames list.
        """
        stride = max(1, len(frames) // n_frames)
        sampled = [frames[i] for i in range(0, len(frames), stride)]
        return sampled[:n_frames]

    def __len__(self):
        return len(self.videos)

    def __getitem__(self, idx):
        video_frames = self.videos[idx]
        label = self.labels[idx]
        images = []
        for frame_path in video_frames:
            image = Image.open(frame_path).convert("RGB")
            if self.transform:
                image = self.transform(image)
            images.append(image)

        # Stack images along new dimension (sequence length)
        data = torch.stack(images, dim=0)
```

```
68
69          # Rearrange to have the shape (C, T, H, W)
70          data = data.permute(1, 0, 2, 3)
71          return data, label
72
```

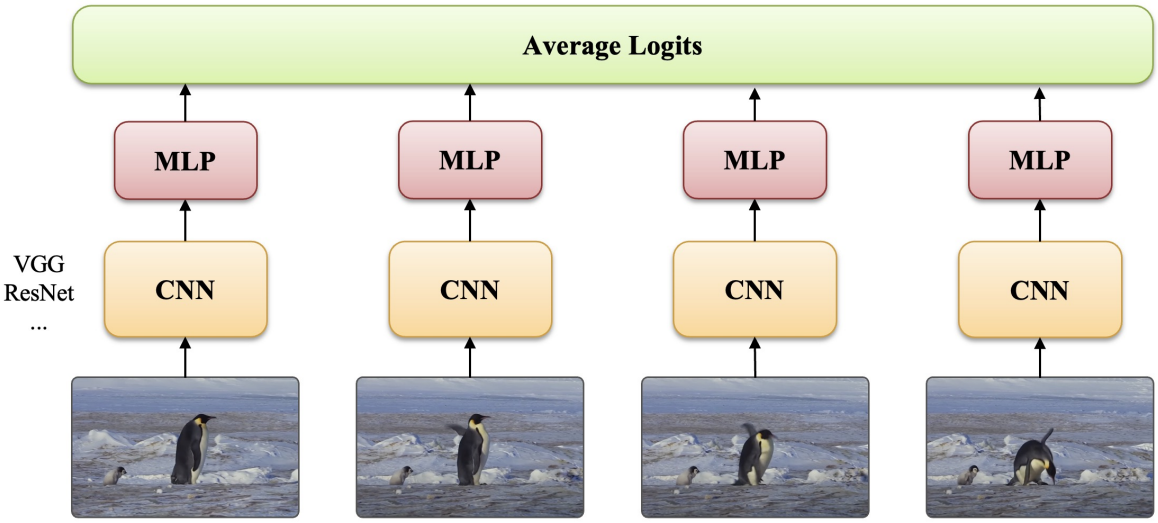
Sau đó ta sẽ tạo dataloader để train model:

```
1  BATCH_SIZE = 16
2  MAX_LEN = 15
3  IMAGE_SIZE = 224
4
5
6  transform = transforms.Compose(
7      [
8          transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
9          transforms.ToTensor(),
10     ]
11 )
12
13 # Load dataset
14 train_dataset = VideoDataset(
15     root_dir="./dataset/rwf-2000", phase="train", transform=transform,
       n_frames=MAX_LEN
16 )
17
18 val_dataset = VideoDataset(
19     root_dir="./dataset/rwf-2000", phase="val", transform=transform,
       n_frames=MAX_LEN
20 )
21
22 # Count number of cpus
23 cpus = os.cpu_count()
24 print(f"Number of cpus: {cpus}")
25
26 # Create data loaders
27 train_loader = DataLoader(
28     train_dataset, batch_size=BATCH_SIZE, num_workers=cpus, shuffle=True
29 )
30 val_loader = DataLoader(
31     val_dataset, batch_size=BATCH_SIZE, num_workers=cpus, shuffle=False
32 )
33
```

4. **Model video classification**

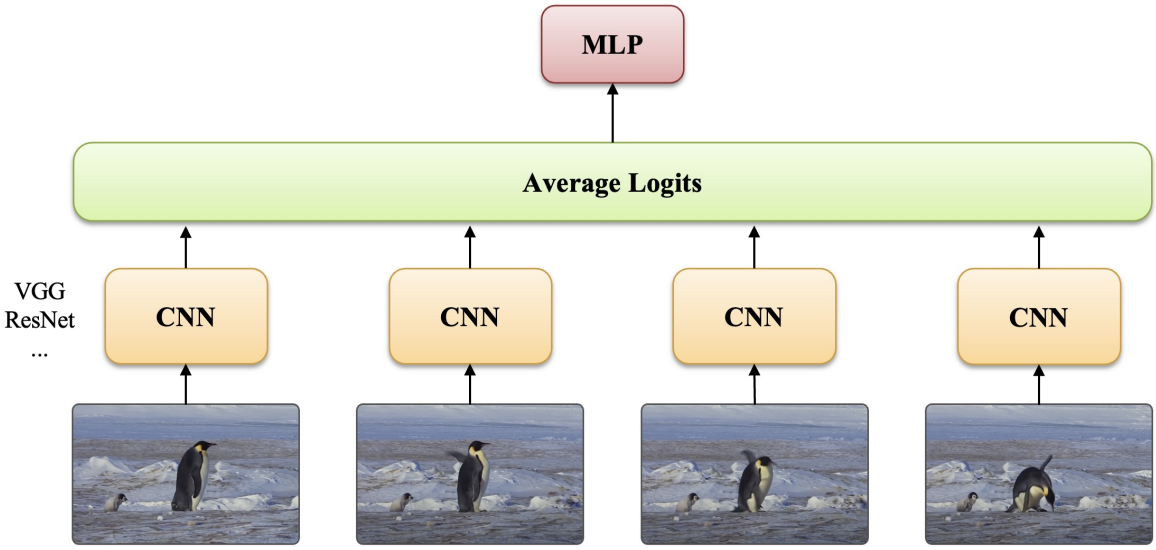
Trong phần này, ta sẽ dùng một số model khác nhau để giải quyết bài toán video classification.

- **Single-frame**: Single-frame hoạt động bằng cách dùng 2D model bất kì (resnet18) để predict trên mỗi frame và tổng hợp kết quả cuối cùng (average). Model single-frame được mô tả như hình sau, lưu ý, ta chỉ cần dùng 1 model (share weight) để predict trên các frame.




```python
class Model(nn.Module):
    def __init__(self, num_classes=2):
        super(Model, self).__init__()
        self.resnet = resnet18(pretrained=True)
        self.resnet.fc = nn.Sequential(nn.Linear(self.resnet.fc.
    in_features, 512))
        self.fc1 = nn.Linear(512, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x_3d):
        # (bs, C, T, H, W) -> (bs, T, C, H, W)
        x_3d = x_3d.permute(0, 2, 1, 3, 4)

        logits = []
        for t in range(x_3d.size(1)):
            out = self.resnet(x_3d[:, t, :, :, :])

            x = self.fc1(out)
            x = F.relu(x)
            x = self.fc2(x)

            logits.append(x)

        # mean pooling
        logits = torch.stack(logits, dim=0)
        logits = torch.mean(logits, dim=0)
        return logits
```

- **Late Fusion**: Late fusion hoạt động bằng cách dùng 2D model (share weight) để biến đổi



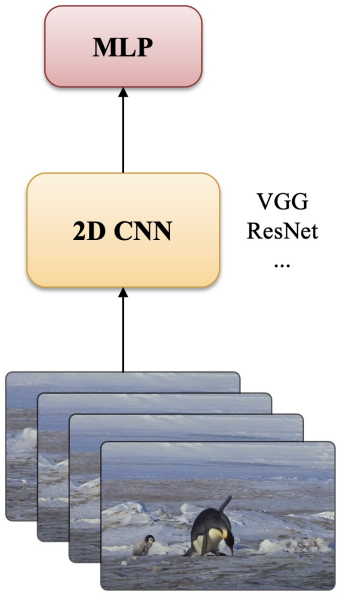
mỗi frame thành feature vector, sau đó ta sẽ tổng hợp các feature vector này và đưa vào MLP head để đưa ra prediction cuối cùng. Model late fusion được mô tả như hình sau, ta có thể concat hoặc tính average của các feature vector sau khi qua CNN, khi concat thì kích thước sẽ rất lớn nên ta sẽ tính average các vector.

để tiết kiệm tài nguyên tính toán và tránh việc kích thước vec

```python
class Model(nn.Module):
    def __init__(self, num_classes=2):
        super(Model, self).__init__()
        self.resnet = resnet18(pretrained=True)
        self.resnet.fc = nn.Sequential(nn.Linear(self.resnet.fc.
    in_features, 512))
        self.fc1 = nn.Linear(512, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x_3d):
        # (bs, C, T, H, W) -> (bs, T, C, H, W)
        x_3d = x_3d.permute(0, 2, 1, 3, 4)

        features = []
        for t in range(x_3d.size(1)):
            out = self.resnet(x_3d[:, t, :, :, :])
            features.append(out)

        # average pooling
        out = torch.mean(torch.stack(features), 0)

        x = self.fc1(out)
        x = F.relu(x)
        x = self.fc2(x)
        return x

```

- **Early Fusion**: Early fusion sẽ kết hợp các input frame để tạo thành tensor có kích thước `(3*T x H x W)` và dùng model 2D (cần thay đổi layer conv đầu tiên để phù hợp với input).

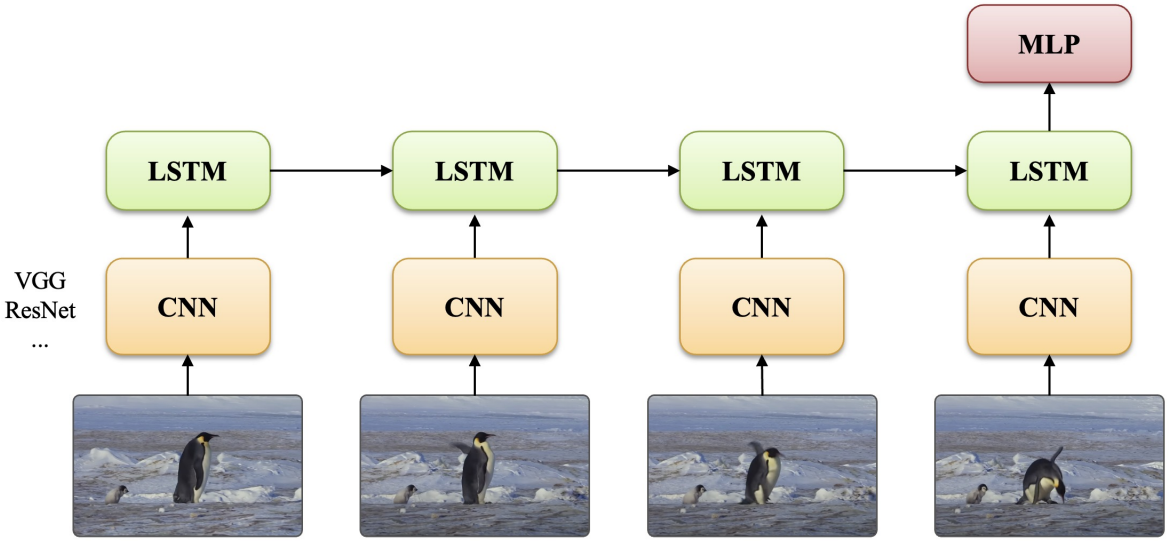
Model early fusion được mô tả như hình sau:




```python
class Model(nn.Module):
    def __init__(self, num_classes=2, num_input_channel=48):
        super(Model, self).__init__()
        self.resnet = resnet18(pretrained=True)
        self.resnet.conv1 = nn.Conv2d(
            num_input_channel, 64, kernel_size=7, stride=2, padding=3, bias=False
        )
        self.resnet.fc = nn.Sequential(nn.Linear(self.resnet.fc.in_features, 512))
        self.fc1 = nn.Linear(512, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x_3d):
        # (bs, C, T, H, W)
        # concatenate all C and T dimensions to make it (bs, C*T, H, W)
        x_3d = x_3d.permute(0, 2, 1, 3, 4).contiguous()
        x_3d = x_3d.view(
            x_3d.size(0), x_3d.size(1) * x_3d.size(2), x_3d.size(3), x_3d.size(4)
        )

        out = self.resnet(x_3d)

        x = self.fc1(out)
        x = F.relu(x)
        x = self.fc2(x)
        return x

```

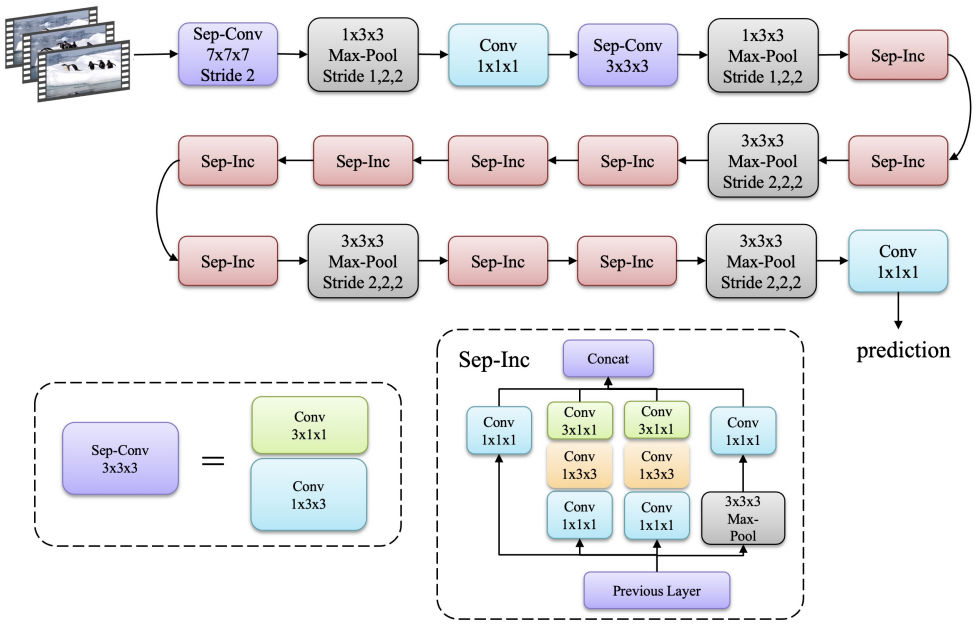
- **CNN-LSTM**: Đối với model CNN-LSTM, ta sẽ dùng 2D model (share weight) để extract

feature từ mỗi frame độc lập và dùng các feature đó để đưa vào LSTM, mỗi frame sẽ tương ứng với mỗi input của model LSTM. Model được mô tả như hình sau:

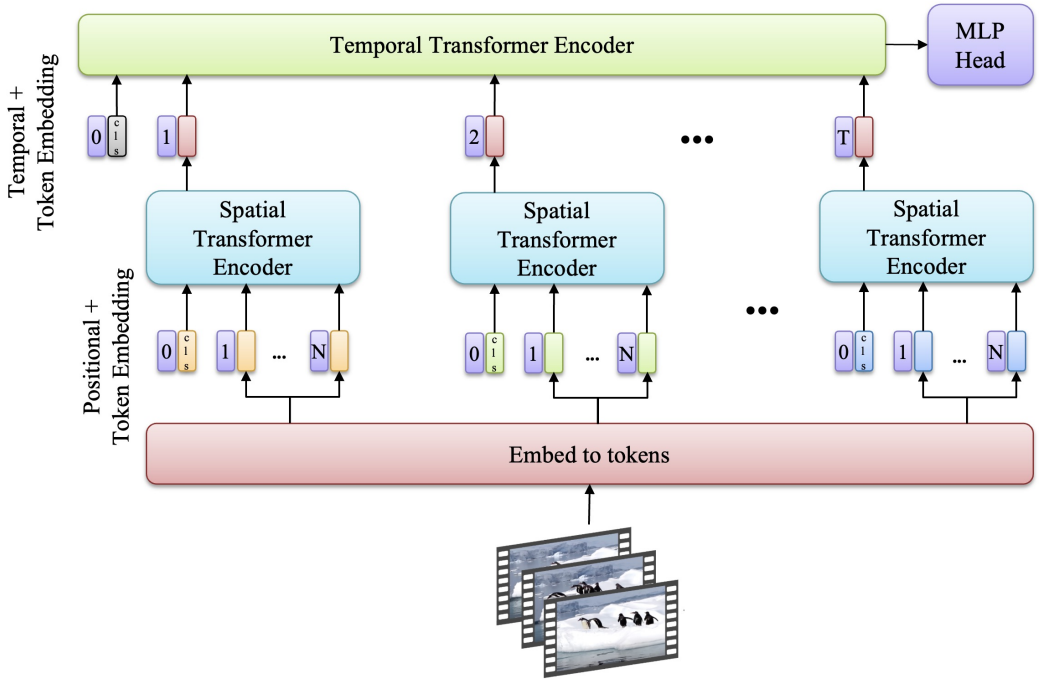



```python
class Model(nn.Module):
    def __init__(self, num_classes=2):
        super(Model, self).__init__()
        self.resnet = resnet18(pretrained=True)
        self.resnet.fc = nn.Sequential(nn.Linear(self.resnet.fc.in_features, 512))
        self.lstm = nn.LSTM(input_size=512, hidden_size=389, num_layers=3)
        self.fc1 = nn.Linear(389, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x_3d):
        # (bs, C, T, H, W) -> (bs, T, C, H, W)
        x_3d = x_3d.permute(0, 2, 1, 3, 4)

        hidden = None
        for t in range(x_3d.size(1)):
            x = self.resnet(x_3d[:, t, :, :, :])
            out, hidden = self.lstm(x.unsqueeze(0), hidden)

        x = self.fc1(out[-1, :, :])
        x = F.relu(x)
        x = self.fc2(x)
        return x

```

- **3D CNN**: Ta cũng có thể sử dụng các model chuyên dành cho data 3D để giải quyết bài toán video classification. Do đoạn code tạo model rất lớn nên sẽ không được bao gồm trong file này (tham khảo tại đây). Sau đây là kiến trúc model S3D mà ta sẽ sử dụng:



- **Video ViT (ViViT)**: Transformer vẫn luôn là backbone được sử dụng rộng rãi trong nhiều các bài toán hiện tại. ViViT là một trong những model đầu tiên sử dụng kiến trúc Vision Transformer (ViT) để áp dụng vào bài toán video classification. Model ViViT được mô tả như hình sau, lưu ý các block Spatial Transformer Encoder là các block share weight.




```python
from transformers import VivitConfig, VivitForVideoClassification

class Model(nn.Module):
    def __init__(self, num_classes=2, image_size=224, num_frames=15):
```

```
        super(Model, self).__init__()
        cfg = VivitConfig()
        cfg.num_classes = num_classes
        cfg.image_size = image_size
        cfg.num_frames = num_frames

        self.vivit = VivitForVideoClassification.from_pretrained(
            "google/vivit-b-16x2-kinetics400",
            config=cfg,
            ignore_mismatched_sizes=True,
        )

    def forward(self, x_3d):
        # (bs, C, T, H, W) -> (bs, T, C, H, W)
        x_3d = x_3d.permute(0, 2, 1, 3, 4)

        out = self.vivit(x_3d)

        return out.logits

```

Phần III: Câu hỏi trắc nghiệm

1. Đâu là một cách xử lý video có kích thước lớn?
   (*a*) Sử dụng toàn bộ frame trong video.
   (*b*) Chia video thành nhiều clip ngắn và train model.
   (*c*) Bỏ qua các frame không quan trọng.
   (*d*) Sử dụng video với tốc độ khung hình cao.

2. RWF2000 là dataset cho bài toán gì?
   (*a*) Phát hiện hành vi bạo lực.
   (*b*) Phân loại động vật.
   (*c*) Nhận diện khuôn mặt.
   (*d*) Dự đoán thời tiết.

3. Đâu là đặc điểm của data trong video classification?
   (*a*) Video là chuỗi của các frame theo không gian.
   (*b*) Mỗi frame thường có kích thước nhỏ.
   (*c*) Video là chuỗi của các frame theo thời gian.
   (*d*) Video không chứa dữ liệu âm thanh.

4. Trong VideoDataset, hàm `_uniform_sample` dùng để làm gì?
   (*a*) Để sắp xếp các frames theo thứ tự số.
   (*b*) Để chuyển đổi các frame sang RGB.
   (*c*) Để lấy mẫu đều các frame từ danh sách.
   (*d*) Để kết nối với GPU cho việc training nhanh hơn.

5. Đâu là kiến trúc sử dụng để chuyển đổi mỗi frame thành feature vector trong late fusion?
   (*a*) CNN
   (*b*) LSTM
   (*c*) MLP
   (*d*) RNN

6. Trong early fusion, các frame được kết hợp như thế nào trước khi đưa vào model?
   (*a*) Tất cả các frames được nén lại thành một frame.
   (*b*) Các frames được giữ nguyên và xử lý độc lập.
   (*c*) Các frames được đưa vào LSTM như là các input độc lập.
   (*d*) Các frame được kết hợp để tạo thành tensor có kích thước (3*T x H x W).

7. Single-frame model hoạt động dựa trên nguyên tắc nào?
   (*a*) Tổng hợp các feature vector từ mỗi frame.
   (*b*) Dùng 2D model để predict trên mỗi frame và tổng hợp kết quả.
   (*c*) Xử lý từng frame với một mạng LSTM.

(*d*) Kết hợp tất cả frames thành một tensor 3D.

8. Trong VideoDataset, `__getitem__` trả về gì?
   (*a*) Chỉ một tensor của các frames.
   (*b*) Một cặp gồm data và label.
   (*c*) Một danh sách các đường dẫn đến frame.
   (*d*) Kết quả của mô hình đã được train.
9. Trong quá trình training, việc xáo trộn (shuffle) dữ liệu trong `train_loader` có mục đích gì?
   (*a*) Giảm dung lượng dữ liệu cần xử lý.
   (*b*) Tăng tốc độ training của model.
   (*c*) Ngăn ngừa model học theo thứ tự dữ liệu, giúp generalization tốt hơn.
   (*d*) Tăng độ chính xác của model trên dữ liệu validation.
10. Đâu là lợi ích của việc sử dụng pre-trained models như trong single-frame model?
    (*a*) Giảm thời gian inference.
    (*b*) Giảm độ chính xác của model.
    (*c*) Giúp model có khả năng transfer learning.
    (*d*) Giảm số lượng layers cần thiết trong mô hình.

- Hết -