Deep learning

Vũ Thành Đạt[1], Nguyễn Trần Hải Ninh[2], and Nguyễn Quang Thao[3]

[1]22022620@vnu.edu.vn
[2]22022526@vnu.edu.vn
[3]22022619@vnu.edu.vn

December 2024

Phân loại hình ảnh, video là một trong những bài toán cơ bản trong lĩnh vực thị giác máy tính. Bản báo cáo kỹ thuật này phân tích so sánh độ hiệu quả giữa việc kết hợp mô hình mạng nơ-ron tích chập (CNN) với mô hình mạng nơ-ron hồi quy (RNN) và kết hợp với mô hình Long Short-Term Memory (LSTM) trong phân loại video. Trong đó sử dụng các mô hình CNN đã được tiền huấn luyện như ResNet50, ResNet101 và ResNet152. Các mô hình tiền huấn luyện này dùng để trích xuất các dữ liệu đặc trưng trong hình ảnh, trong khi RNN và LSTM để tăng cường trong việc xử lý dữ liệu tuần tự nhờ khả năng nắm bắt các mối quan hệ với nhau. Nghiên cứu này đánh giá trên tập dữ liệu UCF101 Videos với các mô hình tiền huấn luyện khác nhau và trên các chỉ số khác nhau để xác định sự phù hợp cho bài toán liên quan. Kết quả cho thấy sự khác biệt đáng kể giữa các mô hình kết hợp khác nhau về kết quả và cả hiệu quả tính toán. Những kết luận trong nghiên cứu này cung cấp định hướng hữu ích trong việc lựa chọn mô hình cho các bài toán tương tự.

1 Giới thiệu

Cùng với sự tăng trưởng của lĩnh vực công nghệ thông tin trong những năm gần đây thì dữ liệu cũng được sinh ra với số lượng khổng lồ. Một trong số những loại dữ liệu đó là dữ liệu video. Việc phân loại video, đặc biệt là video ngắn, trở thành một thách thức quan trọng trong nhiều ứng dụng, bao gồm trích lọc nội dung, hệ thống gợi ý, ... Tuy nhiên, tính đa dạng và độ phức tạp trong cấu trúc và ngữ ngữ hình ảnh của video đòi hỏi những phương pháp hiệu quả để xử lý.

Các mô hình như mạng CNN và mạng RNN, đã chứng minh khả năng vượt trội trong xử lý và phân tích dữ liệu video. CNN dựa trên khả năng trích xuất đặc trưng từ hình ảnh trong video, trong khi RNN phù hợp với việc mô tả chuỗi thời gian nhờ cấu trúc tuân tự liên tiếp của dữ liệu video. Khi kết hợp hai kiểu

mô hình này, ta có thể khai thác đồng thời các thông tin hình ảnh liên tục trong video, giúp tăng cường độ chính xác khi phân loại.

Trong nghiên cứu này, nhóm chúng em đề xuất một phương pháp phân loại video ngắn dựa trên kiến trúc CNN-RNN. Phương pháp này sử dụng CNN để trích xuất đặc trưng không gian từ các frame của video, sau đó truyền các đặc trưng này vào RNN nhằm mô hình hóa được mối quan hệ thời gian của các đặc trưng không gian của frames. Chúng em hy vọng với hiệu suất cũng khá tốt của mô hình sẽ góp phần thêm các hướng giải quyết tốt cho việc phân loại video ngắn

2 Mô hình

Trong bài này, chúng em sử dụng mô hình kết hợp CNN-RNN để giải quyết bài toán phân loại video ngắn. Cách kết hợp này sẽ dùng được phần trích xuất đặc trưng của CNN và phần học dữ liệu theo chuỗi của RNN giúp mô hình kết hợp có thế học được cách phân loại video ngắn

2.1 Pretrainned CNN

Có rất nhiều mô hình CNN lớn đã được huấn luyện trên bộ dữ liệu hình ảnh liên quan đến con người và các vật dụng như VGGNet, GoogLeNet, YOLO, ResNet, ... VGGNet có kiến trúc đơn giản với số tầng rất sâu từ 16 đến 19 tầng giúp cho mô hình này có thế học được nhiều đặc trưng của hình ảnh. Tuy nhiên việc có quá nhiều tầng sẽ sinh ra nhiều tham số dẫn đến việc huấn luyện dữ liệu sẽ rất lâu và yêu cầu nhiều tài nguyên tính toán, dữ liệu lớn và dễ bị overfitting nếu dữ liệu nhỏ. GoogLeNet sử dụng 22 tầng với mỗi tầng sử dụng khối inception với các kernel size (3x3, 5x5) của convolution khác nhau trong một tầng giúp học được các đặc trưng phức tạp của hình ảnh, tuy nhiên cấu trúc mạng của nó khá phức tạp và khó tùy chỉnh. YOLO thì sẽ tối ưu hơn cho việc học hình ảnh trong ứng dụng thời gian thực và nó không có hiệu năng tốt trên các ảnh có đối tượng nhỏ hoặc phức tạp. ResNet có rất nhiều tầng (50, 101, 152), tuy nhiên nó sử dụng residual connections để tránh vanishing gradient và tránh overfit, từ đó giúp mô hình có thể học được nhiều đặc trưng của ảnh hơn mà không yêu cầu tài nguyên tính toán quá lớn.

Từ các lý do ở trên và để mô hình phù hợp với bộ dữ liệu chúng em sử dụng nên chúng em sẽ sử dụng mô hình ResNet làm pre-trainned CNN cho mô hình kết hợp CNN-RNN.Trong quá trình cài đặt, chúng em sẽ thay đổi các pre-trainned ResNet50, ResNet101, ResNet152 để so sánh trực quan hiệu suất các mô hình khác nhau.

Resnet dựa trên cấu trúc của VGGNet để thêm các layer và residual connections nhằm mục đích cải thiện được hiệu suất của mô hình nếu việc thêm layer làm tăng hiệu suất thực sự hoặc giữ nguyên hiệu suất của VGGNet dựa trên việc tạo ra residual connections

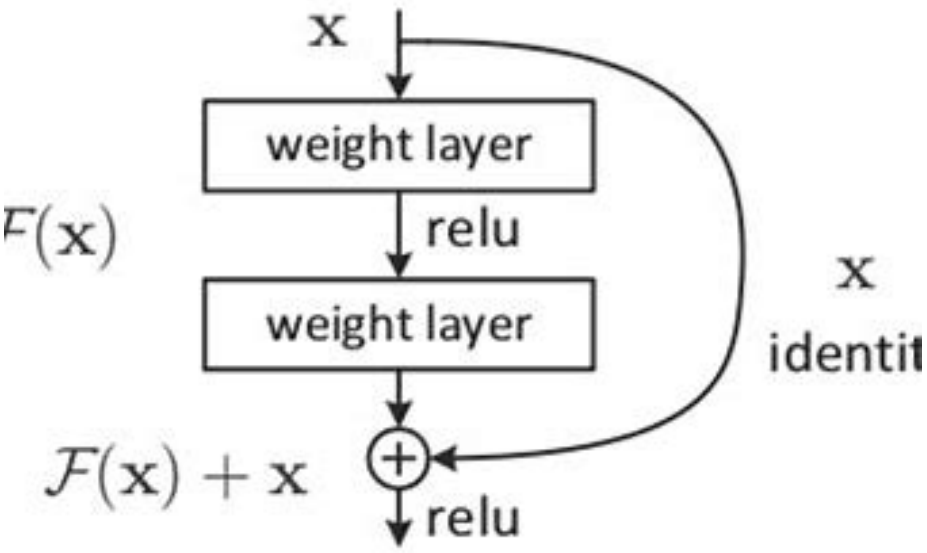


Figure 1: residual block

2.2 RNN

Mô hình RNN được sinh ra để giải quyết các vấn đề của dữ liệu dạng chuỗi như: Mô hình ngôn ngữ và mô hình sinh ngôn ngữ, nhận dạng giọng nói, dịch máy, nhận dạng ảnh, nhận dạng khuôn mặt, phân tích hành động trong video. RNN có 4 dạng chính là: One to One, One to Many, Many to One, Many to Many. Trong bài này chúng em sử dụng dạng cấu trúc Many to One - từ video sinh ra phân loại tương ứng của video.

2.2.1 LSTM

Long Short Term Memory (LSTM) là một biến thể của RNN, nó được thiết kế để giải quyết vấn đề quên dữ liệu ở đầu câu (vanishing gradient) thường gặp trong RNN thông thường. Khi RNN thông thường xử lý dữ liệu chuỗi dài, gradient khi thực hiện backpropagation ở phần đầu chuỗi sẽ bị rất nhỏ và làm cho việc cập nhật trọng số ở phần đầu chuỗi trở nên không đáng kể dẫn đến việc không học được các phụ thuộc. LSTM khắc phục vấn đề này bằng cách tạo ra các cổng như cổng đầu vào i_t, cổng đầu ra o_t, cổng quên f_t và có context C_t được truyền xuyên suốt chuỗi.

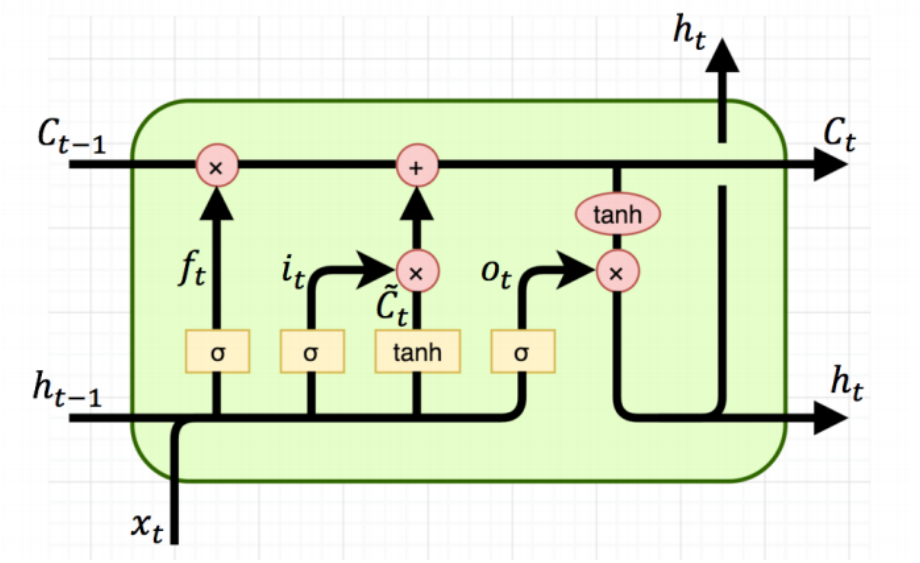


Figure 2: LSTM

3 Thực nghiệm

3.1 Dữ liệu

Dữ liệu được sử dụng trong bài của chúng em là dữ liệu dạng video ngắn có sẵn, được tham khảo trên kaggle **bộ dữ liệu**. Bộ dữ liệu ≈ 450Mb bao gồm 1 folder chứa các file video dùng để train, 1 folder chứa các file video dùng để test và 2 file csv train test tương ứng chứa thông tin tên video và nhãn lớp tương ứng của video. folder video train gồm có 594 video, folder video test gồm có 224 video. Video được lưu dưới dạng file.avi . Nhãn lớp của toàn bộ video trong bộ dữ liệu bao gồm 5 nhãn: Punch, PlayingCello, CricketShot, ShavingBeard, TennisSwing. với số lượng tương ứng ở tập train như sau:

Punch	121
PlayingCello	120
CricketShot	118
ShavingBeard	118
TennisSwing	117

Figure 3: Label train

và tập test như sau:

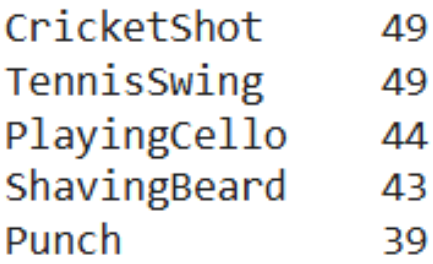


Figure 4: Label test

Bộ dữ liệu train sẽ được sử dụng để thực hiện train và evaluate. Bộ dữ liệu train sẽ được chia theo tỷ lệ 80:20 tương ứng với train và evaluate và chia theo từng label một để tránh trường hợp mất cân bằng dữ liệu ảnh hưởng đến hiệu suất khi huấn luyện mô hình. Bộ dữ liệu test sẽ được sử dụng toàn bộ

3.1.1 Tiền xử lý dữ liệu

Sử dụng thư viện cv2 để đọc dữ liệu video. Đối với mỗi video, sử dụng vòng lặp để đọc từng frame của từng video. Mỗi video sẽ có số lượng frame khác nhau, video có số lượng frame nhiều nhất là hơn 300 frame và ít nhất là dưới 100 frame. Chúng em sẽ lấy 90 video để train và 20 video để test cho mỗi nhãn và trung bình là 100 frame cho mỗi video để phù hợp với tài nguyên phần cứng và cần bằng dữ liệu để đảm bảo hiệu suất phân loại của mô hình . Sau khi đã đọc được các frame của từng video rồi thì sẽ stack các frame của từng video lại với nhau tạo ra một tensor, tensor này của các video khác nhau cũng sẽ khác nhau ở chiều đầu tiên do số lượng frame của từng video là không giống nhau. Bên cạnh đó thì chúng em cũng sử dụng kỹ thuật cắt ảnh để lấy được một frame với chiều rộng và chiều dài bằng nhau

```
def load_video(path, max_frame=100, size=(480,480)):
    capture = cv2.VideoCapture(path)
    all_frames = []
    while True:
        ret, frame = capture.read()
        if len(all_frames) == max_frame:
            break
        if not ret:
            break
        frame = get_center_square(frame)
        frame = cv2.resize(frame,size)
        # ảnh đọc ra bởi cv2 ở dạng BGR, chuyển về RGB
        frame = frame[:, :, [2,1,0]]
        frame = torch.tensor(frame)
        # chuyển về channel, height, width
        frame = frame.permute(2,0,1)
        all_frames.append(frame)
    capture.release()
    return torch.stack(all_frames, dim=0)
    # dim = (100, 3, size[0], size[1])
```

```
    """Get center square of frame"""
    def get_center_square(frame):
        y, x = frame.shape[0:2]
        min_lenght = min(x,y)
        start_x = (x//2) - min_lenght//2
        start_y = (y//2) - min_lenght//2
        frame = frame[start_y : start_y + min_lenght,
                      start_x : start_x + min_lenght]
        return frame
```

3.1.2 Tăng cường dữ liệu

Do tổng số dữ liệu cho việc huấn luyện mô hình mới có 594 video cho train và val (tỷ lệ 80:20) nên chúng em áp dụng kỹ thuật tăng cường dữ liệu cho bộ dữ liệu huấn luyện như sau:

```
    def augmentation_picture(frame):
        transform_a_frame = transforms.Compose([
            # làm mờ ảnh nhẹ với vùng ảnh là 11x11
            transforms.GaussianBlur(kernel_size=11, sigma=1),
            transforms.RandomErasing(scale=(0.01, 0.1), ratio=(1, 1))
            '''scale là phạm vi tỷ lệ (min, max) cho vùng xóa so với
            ảnh ban đầu, ratio là tỉ lệ chiều rộng và cao'''
        ])
        result = transform_a_frame(frame)
        return result

    def augmentation_video(frames):
        new_frames = []
        for frame in frames:
            new_frame = augmentation_picture(frame)
            new_frames.append(new_frame)
        return torch.stack(new_frames,dim=0)
```

Với mỗi video được đọc trong bộ dữ liệu thì sẽ tạo ra một video bao gồm toàn các frame đã được tăng cường và thêm nó vào bộ dữ liệu để train.

3.1.3 Tạo dataset object

Sau khi cho từng video qua tiền xử lý và tăng cường dữ liệu thì được các tensor tương ứng của video bao gồm tất cả frame của từng video. Lúc này chiều của ác tensor đó còn đang khác nhau nên sẽ thêm tất cả các tensor đó vào một list để đưa list đó đi qua `pad_sequence`. `pad_sequence` là một hàm có sẵn của `torch.nn.utils.rnn`, trong trường hợp này thì nó được dùng để chuẩn hóa các tensor đầu vào với độ dài khác nhau thành các tensor có cùng độ dài. Hàm `load_data` trong class dưới đây thể hiện điều đó

```
    class VideoDataset(Dataset):
        def __init__(self, video_paths, img_size=(120,120),
                     all_labels=[], is_train=True):
            self.video_paths = video_paths
            self.img_size = img_size
```

```
        self.all_labels = all_labels
        self.new_labels = []
        self.is_train = is_train
        self.data = self.load_data()

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

    def __getitem__(self, index):
        return self.data[index], self.new_labels[index]

    def load_data(self):
        data = []

        for index, path in tqdm(enumerate(self.video_paths)):
            frames = load_video(path, size=self.img_size)
            self.new_labels.append(self.all_labels[index])
            data.append(frames)

            if self.is_train:
                new_frames = augmentation_video(frames)
                data.append(new_frames)
                self.new_labels.append(self.all_labels[index])
        return pad_sequence(data, batch_first=True, padding_value=0)
```

3.2 Huấn luyện

Trong quá trình huấn luyện, chúng em sẽ thực hiện việc train và evaluate cho mô hình dựa trên tập dữ liệu train và val đã được chia ở trên ở mỗi một epoch để kiểm soát loss và accuracy khi train và evaluate và có hướng xử lý phù hợp với kết quả. Song song với đó thì chúng em sử dụng các kỹ thuật như early stop khi hiệu suất mô hình bắt đầu xấu đi, kỹ thuật giảm learning rate khi loss trên tập dữ liệu val không còn giảm nữa.

3.3 Mô hình RNN thông thường

Giới thiệu tổng quan về mô hình

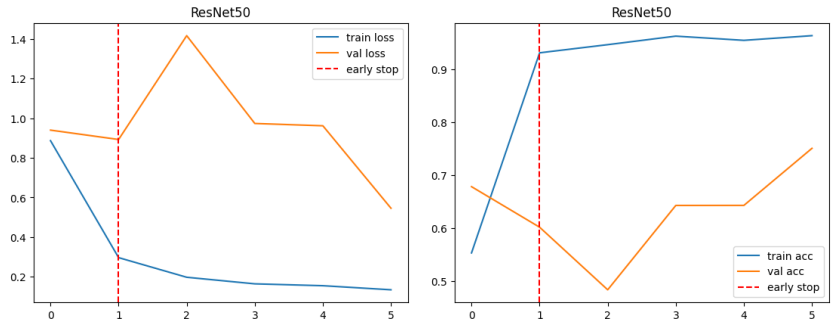
+ Kết quả chính: kết quả tốt nhất của mô hình, so sánh các mô hình

+ Kết quả phụ: phân tích sâu kết quả gồm: training loss, so sánh các kết quả của tuning hyper-parameters (ví dụ so sánh kết quả khi thay đổi learning rate, batch size, thêm bớt layers, ..), các phân tích khác

3.3.1 Kết hợp pretrained với learning rate cố định

Kết quả tốt nhất khi sử dụng với bộ tham số:
- batch size: 32
- optimizer: AdamW
- criterion: CrossEntropyLoss
- learning rate: 1e-4

Kết quả thu được khi train với model ResNet50 cùng với bộ tham số trên:



(a) Loss (b) Accuraccy

Figure 5: RNN + ResNet50

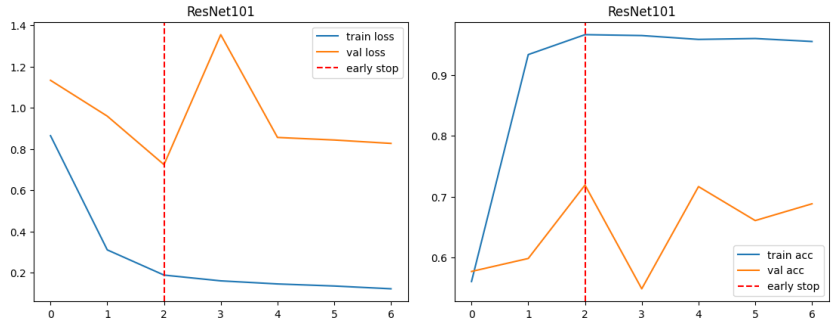
Kết quả thu dược trên tập test:

```
test_resnet50 = Tester(model_resnet50_save, test_dataloader)
test_resnet50.test()
```

```
Acc_test: 0.6339285714285714: 100%|██████████| 7/7 [00:12<00:00,  1.85s/it]
```

Figure 6: RNN + ResNet50 Test Accuraccy

Kết quả thu được khi train với model ResNet101 cùng với bộ tham số trên:



(a) Loss (b) Accuraccy

Figure 7: RNN + ResNet101

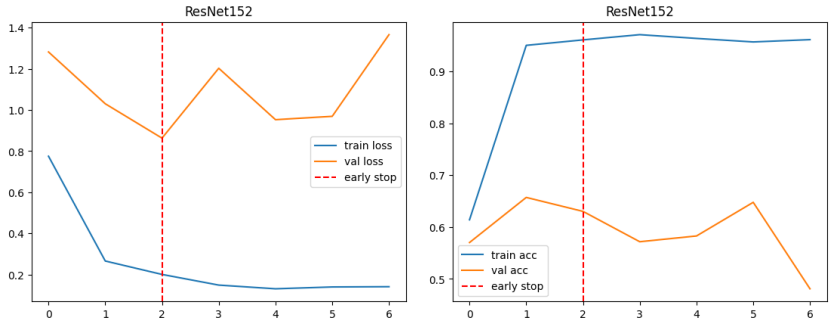
Kết quả thu dược trên tập test:

```
test_resnet101 = Tester(model_resnet101_save, test_dataloader)
test_resnet101.test()
```

```
Acc_test: 0.5803571428571429: 100%|██████████| 7/7 [00:21<00:00,  3.03s/it]
```

Figure 8: RNN + ResNet101 Test Accuraccy

Kết quả thu được khi train với model ResNet152 cùng với bộ tham số trên:



(a) Loss (b) Accuraccy

Figure 9: RNN + ResNet152

Kết quả thu dược trên tập test:

```
test_resnet152 = Tester(model_resnet152_save, test_dataloader)
test_resnet152.test()
```

```
Acc_test: 0.6294642857142857: 100%|██████████| 7/7 [00:29<00:00,  4.26s/it]
```

Figure 10: RNN + ResNet152 Test Accuraccy

Kết quả thu được cho thấy sự kết hợp giữa mô hình RNN và pretrained ResNet50 cho kết quả cao nhất ở trên tập test.

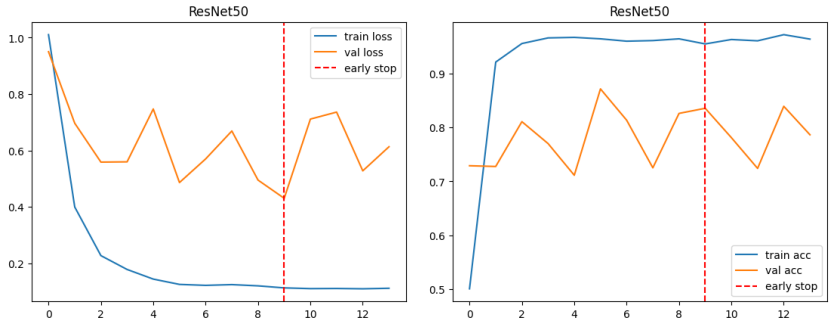
3.3.2 Kết hợp pretrained với learning rate thay đổi

Kết quả khi sử dụng với bộ tham số:
- batch size: 32
- optimizer: AdamW
- criterion: CrossEntropyLoss
- learning rate: 1e-4
- sử dụng kết hợp với ReduceLROnPlateau để giảm learning rate khi evaluate cho mô hình mà loss val khôkng giảm, ReduceLROnPlateau sử dụng bộ tham số như sau:

```
    ReduceLROnPlateau(optimizer, mode='min',
                      factor=0.1, patience=1, verbose=True)
```

trong đó, mode = 'min' là điều kiện metric (ở đây là loss val) không giảm, factor = 0.1 là hệ số giảm của learning rate, patience = 1 là cứ sau mỗi epoch sẽ kiểm tra sự giảm của metric và dùng verbose để thông báo khi có sự thay đổi của learning rate.

Khi train với model pretrained ResNet50 với bộ tham số trên thu được kết quả như sau:



(a) Loss (b) Accuraccy

Figure 11: RNN + ResNet50

Kết quả thu được trên tập test:

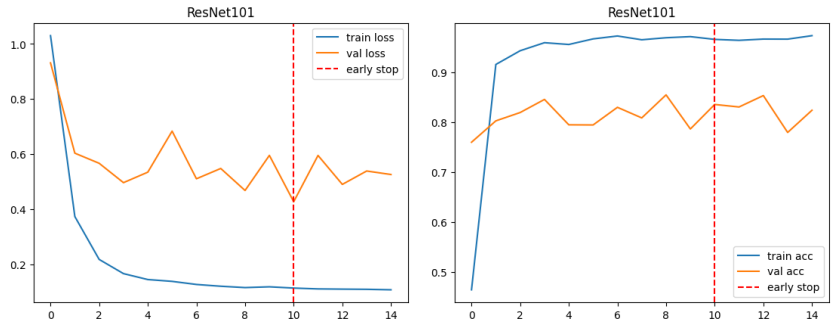
```
[64] test_resnet50 = Tester(model_resnet50_save, test_dataloader)
     test_resnet50.test()

     Acc_test: 0.71875: 100%|██████████| 7/7 [00:21<00:00,  3.12s/it]
```

Figure 12: RNN + ResNet50 Test Accuracy

Như vậy có thể thấy kết quả đạt được khá thấp trên tập test khi sử dụng kết hợp mô hình RNN với pretrained ResNet50.

Tiếp theo là kết quả khi train với model pretrained ResNet101 với bộ tham số như trên thu được kết quả như sau:



(a) Loss (b) Accuraccy

Figure 13: RNN + ResNet101

Kết quả thu được trên tập test:

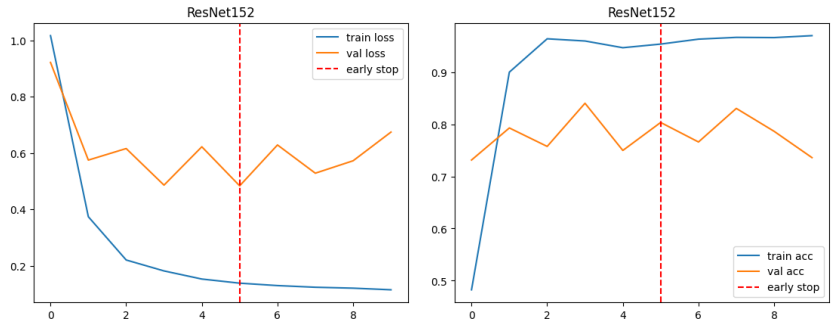
```
[68] test_resnet101 = Tester(model_resnet101_save, test_dataloader)
     test_resnet101.test()

     Acc_test: 0.7723214285714286: 100%|██████████| 7/7 [00:35<00:00,  5.06s/it]
```

Figure 14: RNN + ResNet101 Test Accuraccy

Giống mô hình pretrained ResNet50, kết quả thu được cũng khá thấp, thấp hơn nhiều so với kết quả trên tập train.

Cuối cùng là kết quả thử nghiệm khi train với model pretrained ResNet152 với cùng bộ tham số trên:



(a) Loss (b) Accuraccy

Figure 15: RNN + ResNet152

Kết quả thu được trên tập test:

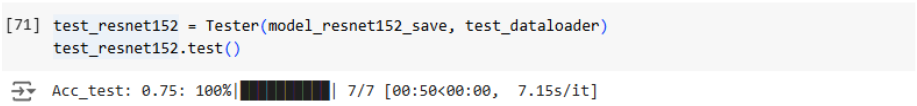


Figure 16: RNN + ResNet152 Test Accuraccy

Kết quả không khác biệt quá nhiều nhưng hầu hết kết quả đều tốt hơn so với train với learning rate cố định.

3.3.3 So sánh kết quả

Mô hình/Tham số	Learning rate cố định	Learning rate thay đổi
ResNet50	0.6339	0.7188
ResNet101	0.5804	0.7723
ResNet152	0.6295	0.7500

Table 1: Bảng so sánh kết quả

3.4 Mô hình dùng LSTM

3.4.1 Mô hình dùng resnet50 làm pretrainned CNN

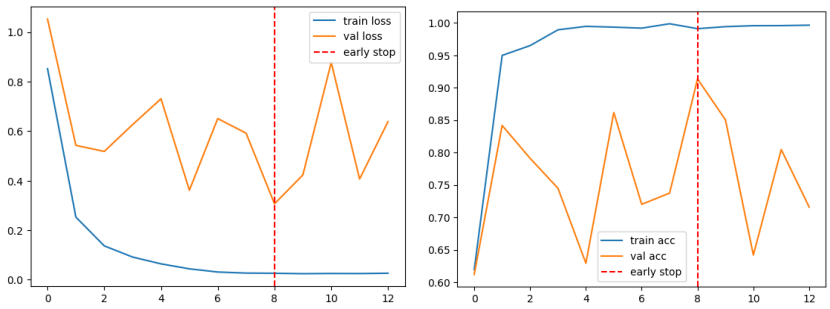
Các tham số dùng cố định: `criterion = CrossEntropyLoss`, `optimizer = AdamW`

Kết quả tốt nhất của mô hình là dùng với bộ tham số như sau: `batch size = 16`, `learning rate = 0.0001` sử dụng kết hợp với `ReduceLROnPlateau` để giảm learning rate khi evaluate cho mô hình mà loss val không giảm. `ReduceLROnPlateau` sử dụng bộ tham số như sau:

```
ReduceLROnPlateau(optimizer, mode='min',
                  factor=0.1, patience=1, verbose=True)
```

trong đó, mode=’min’ là điều kiện metric(ở đây là loss val) không giảm, factor=0.1 là hệ số giảm của learning rate, patience=1 là cứ sau mỗi epoch sẽ kiểm tra sự giảm của metric và dùng verbose để thông báo khi có sự thay đổi của learning rate

Kết quả loss và accuraccy của mô hình sau khi train như sau:



(a) Loss (b) Accuraccy

Figure 17: LSTM + resnet50 với learning rate thay đổi

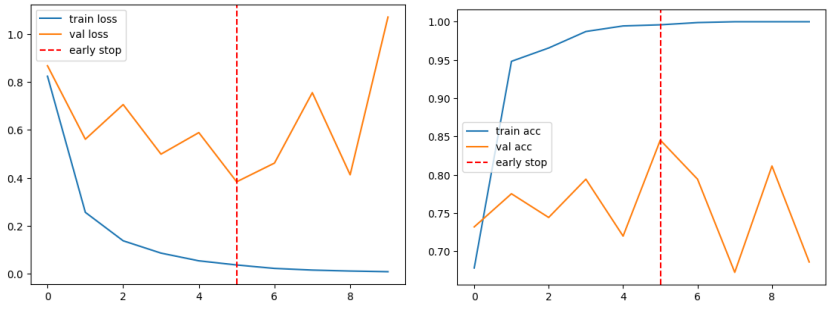
Mô hình sẽ được lưu lại lần cuối cùng ở epoch xảy ra early stop
Và kết quả khi dùng mô hình trên tập test như sau:

```
test1 = Tester(model2_save, test_dataloader)
test1.test()
```

```
Acc_test: 0.8705357142857143: 100%|██████████| 14/14 [00:23<00:00,  1.67s/it]
```

Figure 18: Accuracy test

Khi train mô hình với các tham số giống như trên tuy nhiên không dùng `ReduceLROnPlateau` thì được kết quả như sau:



(a) Loss (b) Accuraccy

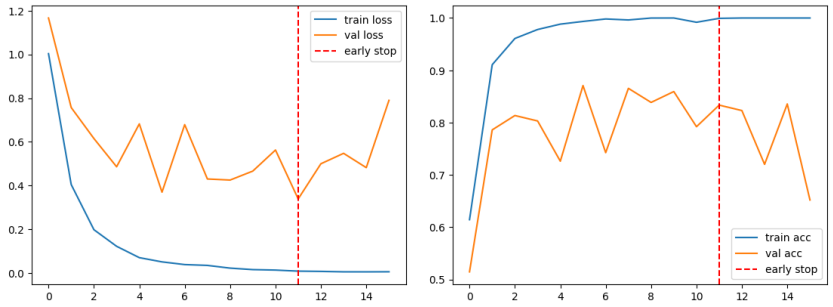
Figure 19: LSTM + resnet50 với learning rate cố định 0.0001

```
test = Tester(model_save, test_dataloader)
test.test()
```

```
Acc_test: 0.7946428571428571: 100%|██████████| 14/14 [00:23<00:00,  1.68s/it]
```

Figure 20: Accuracy test

Khi train mô hình với các tham số giống như trên và giảm learning rate một nửa `learning rate = 0.00005` cùng với các tham số cố định thì được kết quả như sau:



(a) Loss (b) Accuraccy

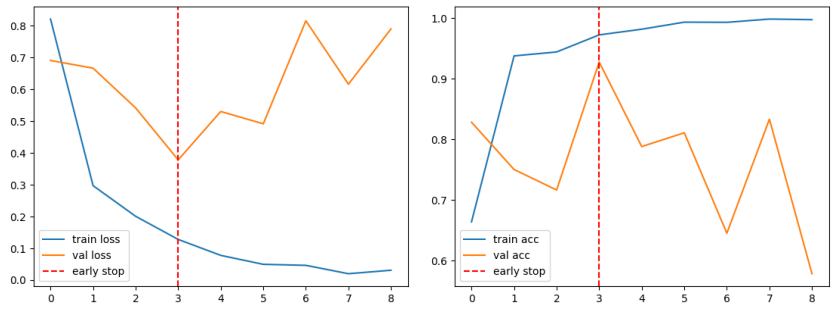
Figure 21: LSTM + resnet50 với learning rate cố định 0.00005

```
test2 = Tester(model2_save, test_dataloader_16)
test2.test()
```

```
Acc_test: 0.7946428571428571: 100%|██████████| 14/14 [00:25<00:00,  1.79s/it]
```

Figure 22: Accuracy test

Khi train mô hình với `batch size=8` và `learning rate = 0.0001` cùng với các tham số cố định đã nêu ở trên thì được kết quả như sau:



(a) Loss (b) Accuraccy

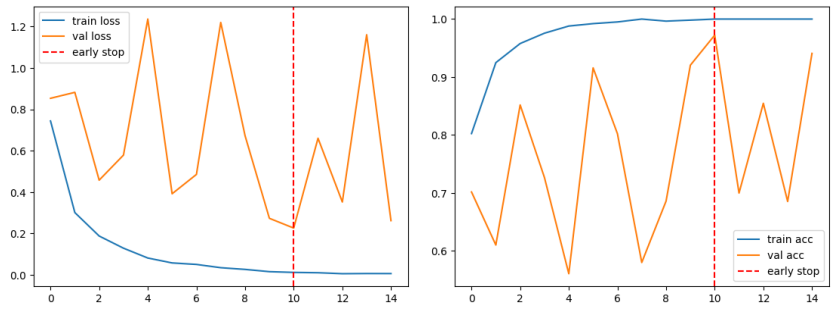
Figure 23: LSTM + resnet50 với learning rate cố định 0.0001 và batch size 8

```
test = Tester(model_save, test_dataloader_8)
test.test()
```

```
Acc_test: 0.6964285714285714: 100%|██████████| 28/28 [00:26<00:00,  1.05it/s]
```

Figure 24: Accuracy test

Khi train mô hình với `batch size=8` và learning rate giảm một nửa `learning rate = 0.00005` cùng với các tham số cố định thì được kết quả như sau:



(a) Loss (b) Accuraccy

Figure 25: LSTM + resnet50 với learning rate cố định 0.00005 và batch size 8

```
tester3 = Tester(model, test_dataloader_8)
tester3.test()
```

```
Acc_test: 0.8616071428571429: 100%|██████████| 28/28 [02:55<00:00,  6.27s/it]
```

Figure 26: Accuracy test

3.4.2 Mô hình dùng resnet101 làm pretrainned CNN

Các tham số dùng cố định: `criterion = CrossEntropyLoss`, `optimizer = AdamW`, `learning rate = 0.0001`

Khi train mô hình với `batch size = 16`

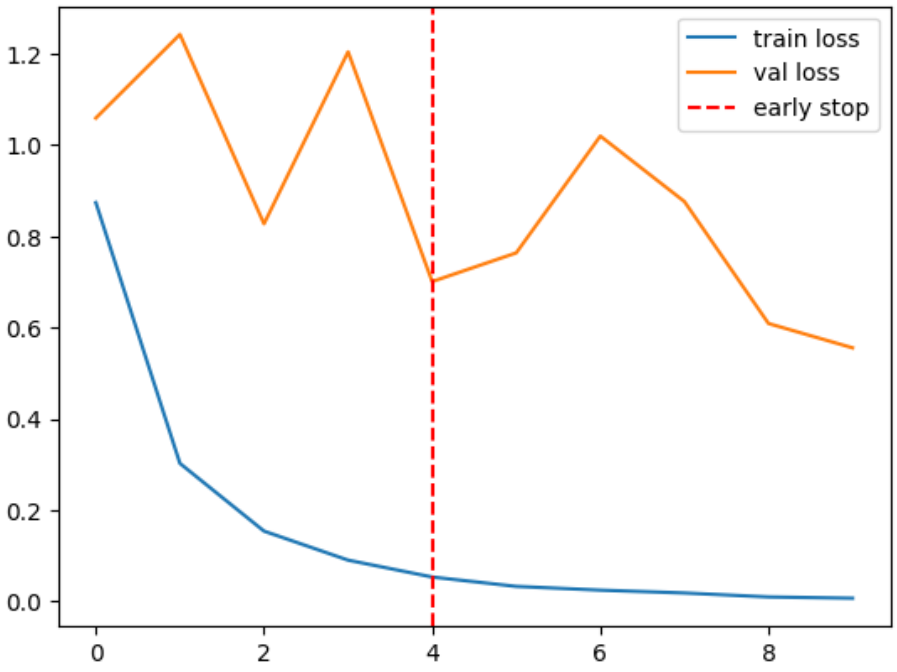


Figure 27: LSTM + resnet101 với learning rate cố định - Loss

```
tester1 = Tester(model1, test_dataloader)
tester1.test()
```

```
Acc_test: 0.6830357142857143: 100%|██████████| 14/14 [00:40<00:00,  2.86s/it]
```

Figure 28: Accuracy test

Khi train mô hình với `batch size = 8`

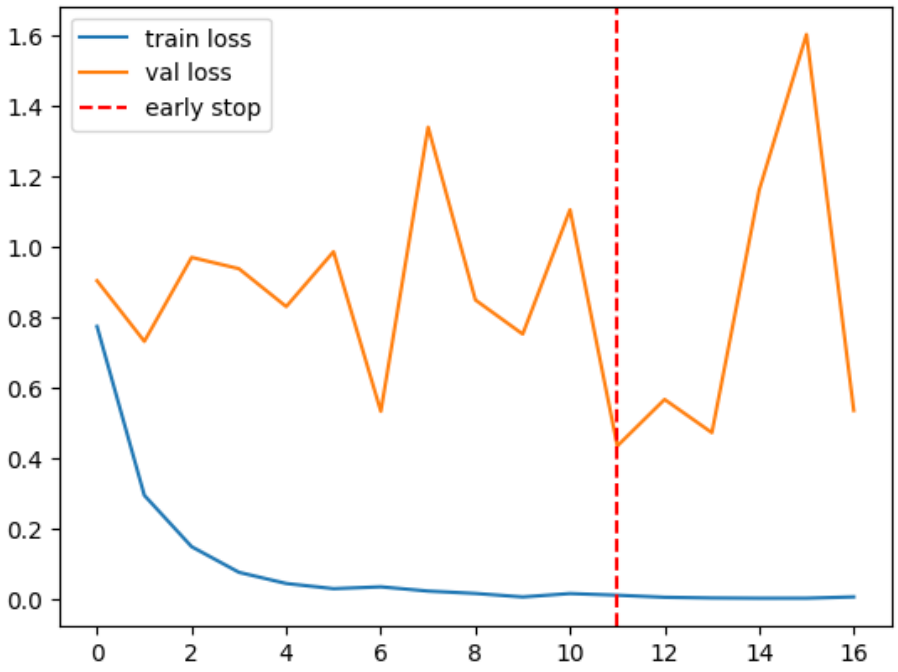


Figure 29: loss

```
tester = Tester(model, test_dataloader)
tester.test()
```

```
Acc_test: 0.8080357142857143: 100%|██████████| 28/28 [00:52<00:00,  1.86s/it]
```

Figure 30: Accuracy test

3.4.3 Mô hình dùng resnet152 làm pretrainned CNN

Các tham số dùng cố định: `criterion = CrossEntropyLoss`, `optimizer = AdamW`, `learning rate = 0.0001`

Khi train mô hình với `batch size = 16`

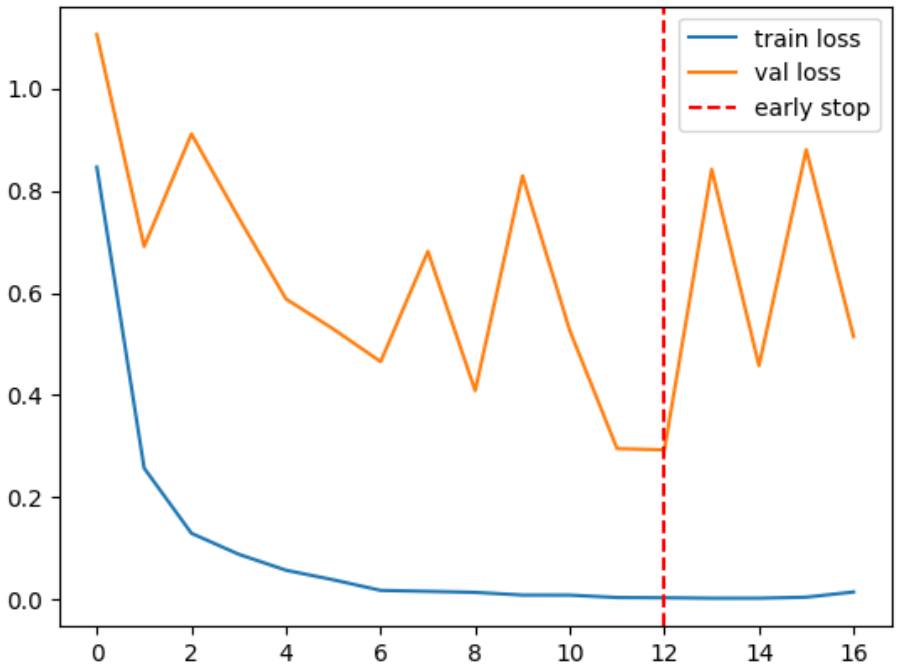


Figure 31: loss

```
tester1 = Tester(model1, test_dataloader_16)
tester1.test()
```

```
Acc_test: 0.7723214285714286: 100%|██████████| 14/14 [00:52<00:00,  3.75s/it]
```

Figure 32: Accuracy test

Khi train mô hình với `batch size = 8`

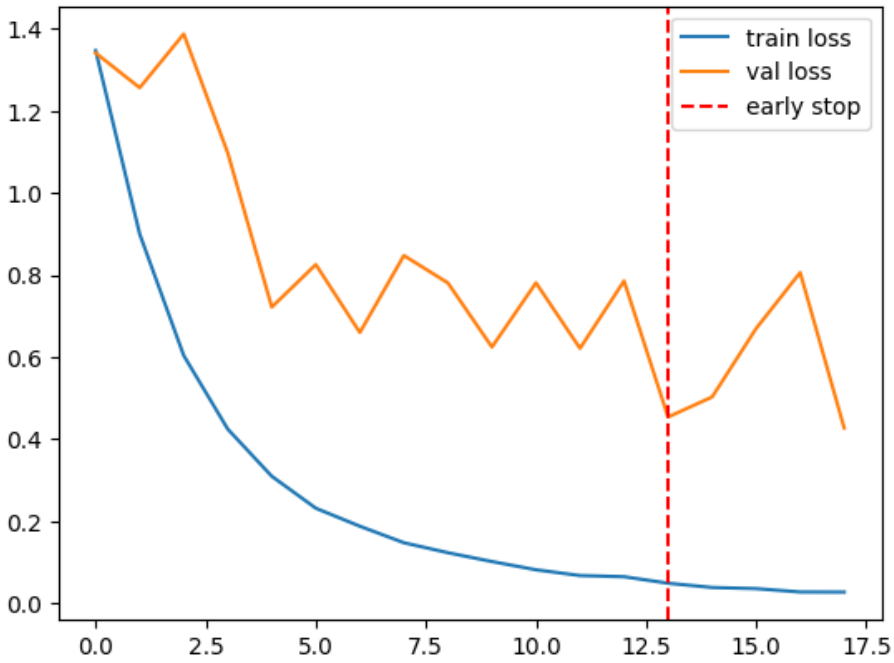


Figure 33: loss

```
tester = Tester(model, test_dataloader_8)
tester.test()
```

```
  0%|          | 0/28 [00:00<?, ?it/s]Acc_test: 0.7544642857142857: 100%|██████████| 28/28 [00:57<00:00,  2.06s/it]
```

Figure 34: Accuracy test

3.4.4 So sánh các mô hình

Sau khi đã train mô hình sử dụng LSTM và các pretrainned CNN khác nhau thì chúng em dựa trên accuracy test của các mô hình để lập ra bảng so sánh như sau:

Các tham số được dùng cố định: criterion=CrossEntropyLoss, optimizer=AdamW, Và phần RNN của mạng tổng, chỉ so sánh dựa trên accuracy test của các mô hình

Tham số/Mô hình	ResNet50	ResNet101	ResNet152
Batch size = 8, lr=0.0001	0.6964	0.808	0.7545
Batch size = 8, lr=0.00005	0.8616	-	-
Batch size = 16, lr=0.0001	0.7946	0.683	0.7723
Batch size = 16, lr=0.0001 + ReduceLROnPlateau	0.8705	-	-
Batch size = 16, lr=0.00005	0.7946	-	-

Table 2: Bảng so sánh kết quả

Ở đây do phần hiệu suất của mô hình dùng pretrainned CNN resnet101 và resnet152 có hiệu suất không tốt bằng resnet50 nên chúng em sẽ chú trọng vào resnet50 để tối ưu.

3.5 Mô hình tốt nhất

Như vậy, mô hình tốt nhất để giải quyết bài toán phân loại video ngắn mà nhóm chúng em cài đặt là dùng Resnet50-LSTM với bộ tham số:
- pretrainned CNN: resnet50
- optimizer: AdamW
- learning rate: 0.0001
- kết hợp với ReduceLROnPlateau
- critetion: CrossEntropyLoss

Với các tham số trên thì mô hình đạt accuracy: 0.8705 trên bộ dữ liệu test

4 Hạn chế và hướng phát triển

Tuy có độ chính xác với bộ dữ liệu test của mô hình tốt nhất là khoảng 87 %, tuy nhiên thì mô hình vẫn còn các điểm hạn chế. Ở phần tiền xử lý dữ liệu, chúng em chỉ thực hiện đọc 100 frame đầu tiên của mỗi video, đây không phải giá trị để có thể đọc hết video mà chỉ đọc được từ 0.5 đến 0.75 số lượng frames của phần lớn video. Chúng em cũng đã thử đọc với tất cả frames của từng video tuy nhiên nó gặp 2 vấn đề chính đó là sự giới hạn của phần cứng chúng em có thể dùng và số chiều của từng tensor đại diện cho tất cả frames của từng video không giống nhau ở các video. Ở vấn đề thứ nhất thì chúng em có thử giải quyết bằng cách không dùng tăng cường dữ liệu tuy nhiên chúng em nhận thấy số lượng dữ liệu nếu không được tăng cường sẽ hơi ít. Ở vấn đề thứ hai thì chúng em giải quyết bằng cách dùng `pad_squence` (em cũng dùng phương pháp này cho mô hình chính để xử lý những video có số frames nhỏ hơn 100), tuy nhiên độ chênh lệch frame của video có số frames lớn nhất (>300) và video có số frames nhỏ nhất(<100) là tương đối lớn và số frames trung bình của các video chỉ khoảng 150 nên khi đưa dữ liệu vào mô hình thì có rất nhiều video có phần frames phía sau giống y hệt nhau nên hiệu suất phân loại của mô hình khá tệ. Chúng em cũng chưa thể trình bày chi tiết về số lượng frames trong bài này do sự giới hạn về phần cứng.

Hơn nữa chúng em cũng chưa thể chạy tất cả các bộ siêu tham số của pretranned resnet101 và 152 như ở resnet50 cũng do sự giới hạn về thời gian của GPU chúng em có thể được sử dụng trên Colab

Từ những hạn chế trên, chúng em có những hướng phát triển để cải thiện hiệu suất của mô hình. Thứ nhất là thay đổi thêm số lượng frames đọc ở mỗi video, thứ hai là chúng em sẽ chạy tất cả các bộ siêu tham số của mô hình dùng resnet50 cho mô hình dùng resnet101 và resnet150, Thứ ba là sử dụng bộ dữ liệu cố số frames trong video dài hơn và đa dạng về nhãn hơn để thay đổi mô hình thành mô hình phân loại video với nhiều chủ đề khác nhau

5 Tổng kết

Như vậy, để giải quyết vấn đề phân loại video ngắn, chúng em đã cài đặt mô hình kết hợp CNN-RNN. Tuy đã được dùng các kỹ thuật để tối ưu mô hình nhưng các kỹ thuật đó vẫn còn một số hạn chế và cần được tinh chỉnh nhiều hơn để có thể làm tăng hiệu suất cũng như có thể xử lý được dữ liệu lớn, phức tạp hơn. Sau khi cài đặt mô hình thì chúng em có thêm kinh nghiệm về xử lý dữ liệu, ghép các mô hình với nhau và sử dụng các kỹ thuật tăng cường cho dữ liệu và tối ưu cho mô hình

[1]

References

[1] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE Access*, 6:1155–1166, 2018.