Phân đoạn ngữ nghĩa trận đấu bóng đá

1st Nguyễn Sỹ Phong
ID: 20120159
dept. Information and Technology
Ho Chi Minh City University of Science
Ho Chi Minh, Vietnam
20120159@student.hcmus.edu.vn
Phone: 0363997294

2nd Thái Minh Trí
ID: 20120220
dept. Information and Technology
Ho Chi Minh City University of Science
Ho Chi Minh, Vietnam
20120220@student.hcmus.edu.vn
Phone: 0366743666

3rd Phạm Hồng Ánh
ID: 20120252
dept. Information and Technology
Ho Chi Minh City University of Science
Ho Chi Minh, Vietnam
20120252@student.hcmus.edu.vn
Phone: 0379825340

I. Cách chạy code và hướng dẫn sử dụng

Đường dẫn đến thư mục chứa file Video Demo build code, ra file thực thi và chạy file thực thi **thành công trên máy của nhóm**

A. Cấu trúc thư mục mã nguồn

```
Unet
├── build: thư mục build cmake
│   └── Release: Các file thực thi ở trong đây để đảm bảo không xung đột
├── dataset
│   ├── final-dataset: ảnh qua tiền xử lý
│   ├── oversample: ảnh sao chép
│   └── validate: ảnh để thẩm định
│       ├── Team A: ảnh để đánh giá độ chính xác IoU cho lớp
│       │   └── Pred: ảnh dự đoán sau từng epoch khi train
│       ├── Team B
│       └── ...
├── libtorch: thư viện libtorch
├── models: khi train xong, file .pt chứa các tham số của model vừa train sẽ lưu vô đây
│   └── pre-built.txt : Link 2 model Unet và Eff-Unet đã build trước thành công và cho hiệu quả tốt nhất
├── stat: thư mục chứa thông kê lỗi, độ chính xác sau khi train (các file.csv)
├── CmakeLists.txt: cấu hình project với cmake
├── header.h
├── source.cpp
├── TestModel.cpp: cho chương trình TestModel.exe
└── TrainModel.cpp: cho chương trình TrainModel.exe
```

B. Yêu cầu cấu hình để build project

1) Môi trường làm việc: C++ 17, hệ điều hành windows 11.
2) Build code với cmake.
3) Thư viện: opencv phiên bản mới nhất. Link tải thư viện libtorch cần để build project: libtorch cpu 2.2.2, do kích thước file khá lớn không thể nộp trong moodle, mong thầy download về để build code.

C. Cách build project

1) Đầu tiên càn tải thư viện libtorch, dựa trên đường dẫn bên trên, sau đó giải nén và **để thư mục libtorch trong thư mục Unet** (Như cấu trúc thư mục bên trên)
2) Từ thư mục gốc Unet, ta chạy các lệnh sau:

```
cd build
cmake -DOpenCV_DIR=<duong_dan_den_lib_OpenCV> ..
cmake --build . --config Release
```

3) Sau khi build thì xuất hiện 2 file thực thi TrainModel.exe và TestModel.exe trong thư mục build/Release. **Nên để 2 file .exe tại đây và chạy, không nên di chuyên qua thư mục khác**.

D. Cách sử dụng 2 file thực thi.

1) **File thực thi nằm trong thư mục Unet/build/Release**, do build code với cmake, để đảm bảo không xung đột với các đường dẫn, nhóm phải để tại đây
2) Nếu chưa build project ở bước trên, muốn xử dụng các file thực thi có sẵn trong bài nộp, ta cần phải download các file dll về, vì kích thước rất lớn nên nhóm không thể để trong bài nộp, sau đó **để vào thư mục build/Release**
3) Ngoài ra nếu **đường dẫn Opencv không tương thích, phải build lại code** với đường dẫn opencv trên máy tính. Hoặc bất kì lỗi gì xảy ra, nên build lại code thay vì sử dụng file thực thi có sẵn.
4) Trong quá trình chạy file thực thi, Trong trường hợp thầy bị báo lỗi thiếu file mkl.dll nào đó, mong thầy tải trên mạng về và để vô libtorch/lib, để có thể chay chương trình thành công.

TrainModel: là file train model Unet hoặc Eff-Unet với learning rate đầu vào, sau mỗi epoch và lô sẽ in ra thơi gian chạy, tổng cộng 50 epoch và 22 lô mỗi epoch, cách chạy file TrainModel.exe:

```
<duong\_dan\_file\_TrainModel.exe> arg1 arg2
```

1) arg1: tên model (Unet, EffUnet)

2) arg2: learning rate (Vd: 0.001)

TestModel: là file dùng để load model Unet hoặc Eff-Unet và phân đoạn cho ảnh, Đây link đến các file .pt đã build sẵn, thầy có thể download về và sử dụng thay vi train lại, cách chạy file TestModel.exe:

```
<duong\_dan\_file\_TestModel.exe> arg1 arg2 arg3 arg4
```

1) arg1: tên model (Unet, EffUnet)
2) arg2: đường dẫn đến file chứa lưu trọng số sau huấn luyện (.pt)
3) arg3: đường dẫn ảnh đầu vào
4) arg4: đường dẫn thư mục ảnh đầu ra

II. Chuẩn bị dữ liệu

A. Thu thập dữ liệu

Do không có sẵn dữ liệu nên nhóm đã tham khảo trên Kaggle, và do khả năng xử lý laptop cá nhân có giới hạn, không có card đồ hòa rời mạnh để giải quyết bài toán, hơn nữa đầu vào là ảnh của 1 có độ phân giải cao, nhóm ưu tiên chọn những bộ dataset nhỏ vừa phải, mục đích là để cải thiện thời gian huấn luyện và tìm cách điều chỉnh mô hình một cách phù hợp nhất có thể với lượng dữ liệu nhỏ và cho ra hiệu suất tốt nhất. Dữ liệu : https://www.kaggle.com/datasets/sadhliroomyprime/football-semantic-segmentation

Nhóm chọn được một bộ dữ liệu các ảnh trong trận đấu **UEFA Super Cup 2017 | Real Madrid 2-1 Manchester United** với 100 ảnh (kèm nhãn). Bộ này phân ra 11 lớp, tương ứng mỗi lớp sẽ có một màu riêng, các ảnh nhãn phân đoạn dựa trên các màu này. Ảnh dữ liệu là các ảnh không có từ "fuse" và "save" trong tên file. Ảnh nhãn là ảnh có cụm "fuse" trong tên file, ảnh có cụm "save" không dùng đến. Ngoài ra chủ sở hữu không cung cụ thể thông tin màu sắc tương ứng mỗi lớp, nhóm thực hiện code trích các màu và tổng hợp lại thông tin màu:

Lớp	Màu (RGB)
Team B (MU)	255,160,1
Team A (RM)	254,233,3
Goalkeeper B	51,188,255
Goalkeeper A	0, 255, 0
Ground	137,126,126
Advertisement	27,71,151
Audience	111,48,253
Goal Bar	255,0,29
Ball	201,19,223
Referee	238,171,171
Coaches & Officials	0,0,0

Bảng I: Màu của các lớp

B. Xử lý dữ liệu

Ở nhãn bộ ảnh này, màu của 2 thủ môn trong rất giống cầu thủ team tương ứng, cụ thể : team A (254,233,3) và thủ môn team A (255,235,0), team B (255,160,1) và thủ môn team B (255,159,0). Nhóm đã đổi màu 2 thủ môn này để kết quả cho ra quan sát dễ hơn, thủ môn A (0,255,0) và thủ môn B (51,188,255).

Ảnh có độ phần giải 1920x1080, ở chiều dài 1080 = 8*135, do đó 1080 có mũ số 2 tối đa 3 là, khi giảm độ phân giải tới 4 - 5 lần, sẽ bị lẻ và khi tăng độ phân giải bằng tích chập ngược (**Transpose convolution**) thì đầu ra cuối cùng sẽ cho ra độ phân giải chiều dài nhỏ hơn 1080, làm không khớp với độ phân giải của ảnh nhãn. Ta có thể giải quyết bằng cách tính toán trực tiếp thêm padding cho mỗi tầng decoder để giữ nguyên độ phân giải cho đầu ra, hoặc cắt bớt ảnh về chiều dài 1024 = 2^{10}. Nhóm chọn giải giáp cắt ảnh về chiều dài 1024 vì chia đều trái phải mỗi bên 28 pixel, con số nhỏ và rất hợp lý hợp ảnh có độ phân giải cao, không mất các thông tin quan trọng trong ảnh. Ngoài ra đối với các kích thước kernel, cần chọn padding phù hợp để giữ lại đô phân giải.

Ngoài ra, ảnh có độ phân giải 1920x1024, có thể quá lớn không đủ tài nguyên trên máy dẫn đến các hạn chế như ảnh hưởng tốc độ train, crash máy, số lô tại một thời điểm không được nhiều, và phải điều chỉnh thu nhỏ lại mạng UNet dẫn đến độ chính xác không cao. Mhóm giảm độ phân giải xuống 480x256, cho phép chọn số lô dữ liệu huấn luyện 1 lúc nhiều hơn, giúp mạng học nhanh hơn và còn giúp có thể tinh chỉnh số độ sâu nhỏ hơn.

Chuẩn hóa các giá trị pixel trong mẫu input (chia cho 255) có thể giúp mô hình train hiệu quả hơn, tăng khả năng hội tụ nhanh hơn. Các ảnh nhãn khi tải lên để chuyển thành dạng (480x256), mỗi phần là tử (i,j) là chỉ số lớp mà pixel (i,j) thuộc về. Tất cả các ảnh sau khi được xử lý sẽ đưa vào thư mục **final-dataset**. Là dữ liệu cuối cùng để train và đánh giá. Để tải dữ liệu gốc, thầy có thể tham khảo nguồn tụi em đã để trên.

C. Phân bổ của các lớp và các lớp không cân đối

Classes	Instances Count	Concentration Ratio
Team B	210	23%
Team A	192	21%
Advertisement	96	10.5%
Audience	94	10.3%
Ground	86	9.4%
Goal Bar	64	7%
Ball	64	7%
Goalkeeper B	47	5.1%
Goalkeeper A	38	4.2%
Referee	24	2.6%
Coaches & Officials	0	0%

Bảng II: Phân bổ các lớp trong dữ liệu

Trong bộ dữ liệu trên, lớp trọng tài và lớp thủ môn A chiếm tỷ lệ khá thấp, điều này có thể khiến model huấn luyện không hiệu quả,cho ra độ chính xác không cao trên các lớp này. Nhóm sử dụng kỹ thuật oversampling : tạo nhiều bản sao của ảnh, nhằm cũng cố tỷ lệ của 3 lớp trên, đồng thời nên chọn ảnh hạn chế tăng tỷ lệ các lớp khác.

D. Cài đặt và cấu hình U-Net

Input là ảnh có 3 channels tương ưng RGB (đã được chuẩn quá bằng cách chia cho 255).

Contracting path (encoder) và expanding path (decoder), số lượng tầng 2 bên bằng nhau, tương ứng mỗi tầng sẽ có một block mạng con 2 lớp tích chập, và các acitivation liền sau.

Lựa chọn kỹ thuật **Down-sampling, Max-pooling** ở tầng encoder, Max-pooling có thể giúp giảm nhiễu trong dữ liệu, dẫn đến kết quả phân đoạn chính xác hơn. Max-pooling còn giúp trích xuất các đặc trưng nổi trội như các cạnh và góc trong khi loại bỏ thông tin ít hữu ích hơn. Về các kỹ thuật **Up-sampling** có thể sử dụng trong U-Net:

- **Bilinear interpolation**: Kỹ thuật này sử dụng bilinear interpolation để đơn giản hóa quá trình upsampling. Tuy nhiên, bilinear interpolation không học được cách kết hợp thông tin từ các cấp độ độ phân giải khác nhau, dẫn đến kết quả phân đoạn kém chính xác hơn so với tích chập ngược.
- Ở bài này nhóm chọn tích chập ngược **Transpose convolution**: Kỹ thuật này sử dụng tích chập ngược để tăng độ phân giải của dữ liệu. Lợi ích của tích chập ngược là nó cho phép mô hình học được cách kết hợp thông tin từ các cấp độ độ phân giải khác nhau, dẫn đến kết quả phân đoạn chính xác hơn.

Chi tiết tầng mỗi block tầng Encoder, lưu ý mỗi block có 2 lớp tích chập và ReLU:

1) Lớp tích chập với số lượng channels đầu ra tầng đầu tiên là **64**, càng xuống sâu hơn sẽ tăng gấp đôi lượng channels tầng trước, ví dụ 3 tầng thì 64-128-256.
2) **ReLU activation function**. Giúp bổ sung tính phi tuyến tính.
3) **Batch Normalization** : Số lượng đặc trưng bằng số lượng channel từ lớp tích chập trên. bằng cách chuẩn hóa đầu vào, Batch Normalization giúp mạng học hiệu quả hơn, cho phép hội tụ nhanh hơn và tổng quát hóa tốt hơn
4) Lặp lại 3 bước trên.
5) **Downsample với max-pooling (2x2)**, giảm độ phân giải theo tỷ lệ 2.
6) Một Block center : Điểm đến cuối cùng của tầng encoder, là tầng sau nhất của mạng, và sau đó bắt đầu tầng decoder.

Chi tiết tầng mỗi block tầng Decoder, lưu ý mỗi block có 2 lớp tích chập và ReLU, ngoài ra có các skip connection:

1) Ở decoder, khi càng lên thì càng giảm số channels, ví dụ 256-128-64.
2) Skip connection: đầu vào của tầng này lấy đầu ra của tầng bên dưới nó nối liền (concat) tầng tương ứng bên encoder (là tầng bên encoder có cùng độ sâu).
3) 1 lớp tích chập với số lượng channels đầu ra tầng sâu nhất (là tầng liền sau tầng center) bằng với số lượng channels đầu ra của tầng center chia 2, càng lên cao hơn sẽ tăng gấp đôi lượng channels tầng trước,
4) 1 ReLU activation function. Giúp bổ sung tính phi tuyến tính
5) Batch Normalization : Số lượng đặc trưng bằng số lượng channel từ lớp tích chập trên. bằng cách chuẩn hóa đầu vào, Batch Normalization giúp mạng học hiệu quả hơn, cho phép hội tụ nhanh hơn và tổng quát hóa tốt hơn
6) Lặp lại 3 bước trên.
7) **Upsample bằng tích chập ngược (transpose convolution)**, kernel 2x2, stride 2, tăng độ phân giải theo tỷ lệ 2.

Một lớp tích chập cuối ở tầng cao nhất ở decoder, là tầng đầu ra cuối cùng, chuyển số channel về bằng số lượng lớp, mỗi channel tương ứng mỗi lớp cho biết xác suất từng pixel thuộc vào lớp này.

III. Hyperparameter

A. Loss Function

Hàm **Categorical Loss CrossEntropy** được sử dụng cho bài phân đoạn nhiều lớp này. CrossEntropy được thiết kế để hoạt động với các mô hình đưa ra xác suất cho từng lớp. Điều này làm cho nó trở nên lý tưởng cho các bài toán mà mô hình cần dự đoán lớp có khả năng xảy ra nhất từ một tập hợp các khả năng.

Trong bài này, đầu ra của mạng có kích thước (n,C,W,H) (với n là kích thước lô, C là số lớp, W là chiều rộng ảnh và H là chiều cao ảnh) mỗi vị trí tương ứng xác suất pixel đó thuộc về lớp nào, nhãn có kích thước (n,W,H), mỗi vị trí cho biết pixel thuộc lớp nào.

B. Optimizer SGD (Stochastic Gradient Descent)

Lựa chọn kích thước lô vừa phải để đảm bảo tai nguyên máy tính vì dữ liệu ảnh quá lớn nên chỉ có thể sử dụng một lô nhỏ tại một thời điểm, ngoài ra SGD cũng giúp tính khái quát hóa tốt hơn: SGD đôi khi có thể giúp tránh bị mắc kẹt ở cực tiểu địa phương không tốt. Tính chất ngẫu nhiên của các lần update, chỉ sử dụng một tập hợp con dữ liệu tại một thời điểm, có thể giúp mô hình khám phá các vùng khác nhau của không gian tham số và có khả năng dẫn đến hiệu suất khái quát hóa tốt hơn trên dữ liệu chưa nhìn thấy.

Ở bài này nhóm sẽ sử dụng thử nghiệm số lô 4 (do hạn chế về tài nguyên máy tính cá nhân), ngoài ra để cải thiện khi số lô nhỏ sử dụng **Adam optimizer** sử dụng kỹ thuật **momentum** giúp giữ lại các gradient của các lô cũ, kết hợp và tính toán cho gradient hiện tại. Ngoài ra, momentum còn giúp giảm dao động của các bản cập nhật, giúp mô hình ổn định hơn và ít bị mắc kẹt trong các điểm cực tiểu cục bộ.

C. Learning Rate

Learning rate ảnh hưởng quan trọng đến khả năng hội tụ và khái quát hóa của mô hình. Thử nghiệm nhiều learning-rate là một hằng số như 0.01, 0.001, 0.0001.

D. Số lượng kênh

Số lượng kênh trong lớp tích chập tương ứng với số lượng ảnh đặc trưng. Mỗi ảnh đặc trưng có khả năng học và phát hiện một đặc điểm khác nhau trong đầu vào. Tăng số lượng kênh có thể cho phép mô hình tìm hiểu các đặc trưng phức tạp và đa dạng hơn. Tuy nhiên, cũng làm tăng chi

phí tính toán và nguy cơ bị overfit, đặc biệt khi tập dữ liệu nhỏ.

Ở bài nhóm lựa chọn **64 kênh**, là con số chuẩn được của mô hình, còn cũng khá nặng nề khi huấn luyện trên máy tính cá nhân, do đó chỉ có thể dùng 6 lô cùng 1 lúc mặc dù giảm độ phân giải ảnh gấp 16 lần, càng nhiều kênh trên một lượng dữ liệu lớn rất dễ bị phải giảm số lô lại, hơn theo thực nghiệm và các bài đánh giá trên mạng, đây là sự lựa chọn khá ổn cân bằng giữa chi phí tính tính toán và khả năng nắm bắt các đặc trưng phức tạp.

E. Số tầng

Độ sâu của U-Net (tức là số tầng) có thể ảnh hưởng lớn đến khả năng tìm hiểu các pattern phức tạp của ảnh. Độ sâu cho phép U-Net khôi phục các chi tiết bị mất trong quá trình downsampling, dẫn đến kết quả phân đoạn chính xác hơn. Các mạng sâu hơn có thể tìm hiểu các đặc trưng phức tạp và trừu tượng hơn, các layer ban đầu học các đặc trưng đơn giản (như các cạnh hoặc kết cấu) và các layer sau kết hợp chúng thành các đặc trưng cấp cao hơn (như hình dạng hoặc vật thể). Tuy nhiên, các mạng rất sâu có thể khó train hiệu quả và dễ overfit, nó cũng yêu cầu nhiều tài nguyên tính toán hơn. Ở bài này, nhóm sẽ sử dụng 5 tầng.

F. Kích thước kernel

Kích thước kernel lớn hơn cho phép mô hình thu được các cấu trúc không gian lớn hơn, nhưng cũng làm tăng chi phí tính toán và số lượng tham số trong mô hình. Kích thước kernel nhỏ hơn có thể làm cho mô hình nhạy hơn với các chi tiết nhỏ hơn, nhưng có thể có nguy cơ bị overfit. Nhóm chọn kích thước kernel là **3x3** vì đây là một lựa chọn cân bằng các yếu tố kể trên, và trên thực nghiệm đây cũng là lựa chọn phổ biến cho mô hình Unet.

IV. Huấn luyện và đánh giá mô hình

A. Train-Validation Split

Do dữ liệu hạn chế, để đảm bao model train được hiệu quả tốt, chia tập dữ liệu thành 2 tập theo tỷ lệ (85-15) cho training và validation . Sử dụng train dataset để train mô hình và validation dataset để theo dõi hiệu suất và hiệu chỉnh các tham số cho phù hợp.

Validation sẽ chia ra một thư mục cho từng lớp, mỗi thư mục là tập dữ liệu đánh giá dành riêng cho mỗi lớp, dùng định lượng IoU để đánh giá độ chính xác của mô hình cho lớp này, ngoài ra cũng tạo ảnh dự đoán cuối cùng tương ứng mỗi ảnh trong thư mục.

B. Các độ đo dùng để đánh giá mô hình

Sử dụng cross entropy loss làm độ lỗi của model, thể hiện độ lỗi này ở cả 2 training và validation set. Trong bài này, thay vì tính toàn bộ độ lỗi trên toàn bộ dataset 1 lúc (thời gian train không cho phép, vì chạy bình thường đã 8h), thì nhóm lấy trung bình độ lỗi của các lô trong epoch hiện tại. **Do đó phần nào làm validation loss có thể tốt hơn training loss.**

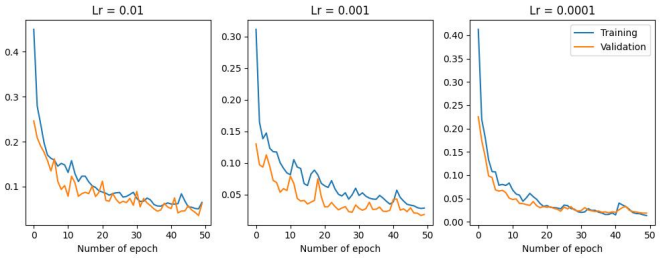
Sử dụng **Intersection over Union (IoU)** làm định lượng độ chính xác chô mõi lớp:

- **True Positive (TP)**: pixel dự đoán ở phân lớp C, ảnh nhãn cũng phân lớp C
- **False Positive (FP)**: pixel dự đoán ở phân lớp C, ảnh nhãn không phân lớp C
- **True Negative (TN)**: pixel không dự đoán ở phân lớp C, ảnh nhãn không phân lớp C
- **False Negative (FN)**: pixel không dự đoán ở phân lớp C, ảnh nhãn phân lớp C

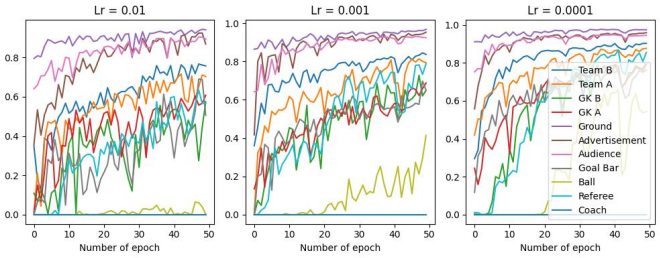
$$IoU_c = \frac{TP_c}{TP_c + FP_c + FN_c} \tag{1}$$

Cụ thể, bây giờ ta chỉ quan tâm một lớp duy nhất, mỗi lớp có thư mục validation riêng. Tương ứng mỗi nhãn và dự đoán từ model (dự đoán đã được chuyển từ dạng xác suất sang dạng lớp có xác suấ cao nhất cho mỗi pixel), ta lấy phần giao của cả 2 chia cho phần hợp của cả 2 (ở đây các pixel mà chứ lớp này thì tương ứng 1, ngược lại 0).

V. Đánh giá và ảnh phân đoạn đầu ra



Hình 1: Error cho từng learning rate



Hình 2: Đo độ chính xác bằng IoU từng learning rate

Như đã đề cập trên, do tập dữ liệu hạn chế, và khi lấy error của mỗi vòng lặp dựa trên trung bình của error từng batch trong vòng lặp khi huấn luyên, nên phần nào đó làm cho validation error nhỏ hơn so với training error.

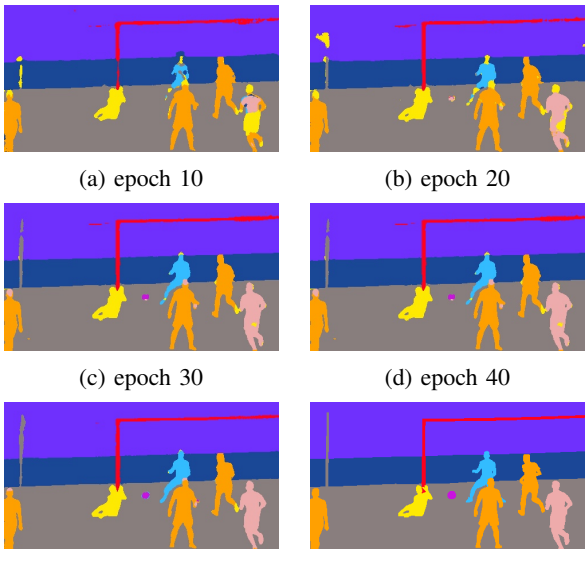
Nhìn vào plot có thể thấy learning rate = 0.0001 cho ra kết quả hội tụ tốt nhất, trường hợp 0.001 và 0.01 cho cho ra kết quả dao động nhiều và hội tụ khó hơn 0.0001, chứng tỏ 2 learning này khá lớn làm cho cho model thường xuyên văng

ra khỏi cực tiểu địa phương. Vậy mô hình tốt nhất ta có thể chọn được là **mô hình với learning rate = 0.0001 cho ra error 0.013**.

Lớp Coach không tính vì trong dữ liệu gốc skhông có lớp này, nhưng họ vẫn để vào (trong bàng phân phối các lớp có ghi rõ). Nên độ chính xác trong plot luôn = 0.

Có thể thấy learning rate = 0.0001 cho ra độ chính xác từng lớp hội tụ về 1 tốt hơn hẳn so với 0.001 và 0.01. Nhìn vào có thể thấy ở learning rate = 0.01 lớp Ball liên tục giao động tạo giá trị 0, tạo ra một hình sin, cứ lên rồi lại xuống 0, mặc dù về sau độ giao động có tăng nhưng vẫn rớt xuống 0, chứng minh 0.01 là một learning rate khá lớn. Với 0.0001 rõ ràng khá ổn giúp lớp Ball ngày càng hội tụ về 1, mặc dù giao động khá lớn, giao động lớn cũng có thể do cách yếu tố đặc trưng khác trông ảnh (như của lớp Audience) cũng nhìn rất giống các đặc trưng trong lớp này.

Hình 3: Ảnh gốc.



(a) epoch 10 (b) epoch 20

(c) epoch 30 (d) epoch 40

(e) epoch 50 (f) Phân đoạn chính xác

Hình 4: Ảnh dự đoán sau từng epoch

Kết quả phân đoạn trong từng ảnh sau mỗi 10,20,30,40,50 epoch ngày càng mang kết quả rất tốt khi error ngày càng hội về 0. Khi tăng epoch lên 100, chắc chắn model càng chính xác hơn nữa.

VI. Mô hình cải thiện : Eff-Unet

A. *Efficient Net*

Efficient Net là kiến trúc mạng tích chập được **Google AI** giới thiệu vào năm 2019. Nó được biết đến với tính hiệu quả và hiệu suất trong nhiều tác vụ khác nhau, bao gồm nhận dạng đối tượng, phân đoạn hình ảnh và thậm chí cả xử lý ngôn ngữ.

Điểm đổi mới quan trọng của EffcientNet là khái niệm mở rộng quy mô phức hợp (compound scaling). Phương pháp này chia tỷ lệ số chiều của mô hình một cách có hệ thống (chiều rộng, chiều sâu và độ phân giải) theo nguyên tắc:

- Chiều rộng: Đề cập đến số lượng kênh trong mỗi lớp của mạng nơron.
- Độ sâu: Liên quan đến tổng số lớp trong mạng.
- Độ phân giải: Liên quan đến việc điều chỉnh kích thước của hình ảnh đầu vào.

Bằng cách cân bằng ba khía cạnh này, EfficientNet đạt được mức độ hiệu quả cực tốt mà không ảnh hưởng đến độ chính xác. Điều này làm cho nó có khả năng thích ứng với nhiều khả năng tính toán và khả năng phần cứng khác nhau.

Mạng **EfficiencyNet-B0** cơ sở dựa trên các khối thặng dư nghẽn đảo ngược (**Inverted bottleneck residual blocks**) của MobileNetV2. Các khối này là một loại mô-đun thặng dư sử dụng đầu vào hẹp, mở rộng nó lên kích thước cao hơn để xử lý và sau đó nén nó trở lại kích thước thấp hơn. Thiết kế này hiệu quả và giúp cải thiện hiệu suất của mạng.

Ngoài các khối thặng dư nghẽn đảo ngược này, EfficiencyNet còn sử dụng các khối nén và kích thích (**Squeeze and Excitation**). Các khối này hiệu chỉnh lại một cách thích ứng hiệu quả của đặc trưng theo từng kênh bằng cách lập mô hình rõ ràng sự phụ thuộc lẫn nhau giữa các kênh.

Với số lượng tham số ít hơn đáng kể, dòng mô hình hoạt động hiệu quả và cũng mang lại kết quả tốt hơn. Điều này đã dẫn đến việc chúng được sử dụng rộng rãi trong nhiều nhiệm vụ học sâu khác nhau.

Các mô hình EfficiencyNet có phạm vi từ B0 đến B7, với mỗi mô hình tiếp theo là phiên bản mở rộng của mô hình trước đó. Ví dụ: Efficient-Net-B0 có 237 lớp, trong khi Efficient-Net-B7 có 813 lớp. **Trong mô hình Eff-Unet này, nhóm sử dụng sử dụng EfficiencyNetB7 làm tầng encoder**.

1) Squeeze and Excitation: Ý tưởng chính của Khối SE là gán cho mỗi kênh của bản đồ đặc trưng một trọng số (Excitation) khác nhau dựa trên mức độ quan trọng của mỗi kênh (Squeeze). SE bao gồm các lớp:

1) **Nén (Squeeze)**: Thao tác này thực hiện Tổng hợp trung bản đồ đặc trưng đầu ra của lớp CNN. Về cơ bản, điều này lấy giá trị trung bình theo chiều không gian (H x W). Kết quả của việc này là một vectơ có hình dạng (C,1,1).
2) Tính toán: Vectơ từ thao tác trước đó được truyền qua hai Lớp được kết nối đầy đủ kế tiếp nhau. Điều này

phục vụ mục đích nắm bắt đầy đủ các phần phụ thuộc theo kênh được tổng hợp từ bản đồ không gian[2]. Kích hoạt ReLU được thực hiện sau lớp FC đầu tiên, trong khi kích hoạt sigmoid được sử dụng sau lớp FC thứ hai.

3) **Kích thích (Excitation)**: Cuối cùng, đầu ra của bước tính toán được sử dụng làm vectơ điều chế trọng số trên mỗi kênh. Nó chỉ đơn giản được nhân với bản đồ đặc trưng đầu vào ban đầu có kích thước (H x W x C). Điều này chia tỷ lệ bản đồ không gian cho từng kênh theo 'tầm quan trọng' của chúng.

2) Mobile Inverted Bottleneck Convolution: **MBConv**, viết tắt của Mobile Inverted Bottleneck Convolution, là một loại lớp tích chập được tối ưu hóa cho thiết bị di động và thiết bị biên, nơi tài nguyên tính toán bị hạn chế. Nó được thiết kế để hoạt động hiệu quả hơn các lớp tích chập truyền thống trong khi vẫn duy trì hoặc thậm chí nâng cao khả năng biểu diễn của mạng.

Cấu trúc của lớp MBConv bao gồm:

1) Giai đoạn mở rộng (Chuyển chập 1x1): Quá trình bắt đầu bằng việc mở rộng số lượng kênh. Điều này đạt được bằng cách sử dụng tích chập 1x1. Hệ số mở rộng, ví dụ: **6 trong MBConv6, có nghĩa là số lượng kênh tăng lên sáu lần**. Ví dụ: nếu đầu vào có 32 kênh thì sau khi mở rộng, nó sẽ có 32 x 6 = 192 kênh. Việc mở rộng này cho phép mạng tạo ra nhiều tính năng phức tạp hơn bằng cách tăng dung lượng kênh.
2) Chuyển đổi theo chiều sâu (3x3): Sau giai đoạn mở rộng, tích chập theo chiều sâu được áp dụng. Trong tích chập chiều sâu, một bộ lọc duy nhất được áp dụng độc lập cho từng kênh đầu vào. Điều này khác với tích chập tiêu chuẩn, trong đó các bộ lọc kết hợp thông tin từ tất cả các kênh đầu vào. Kernel 3x3 được sử dụng cho phép tích chập theo chiều sâu này, có hiệu quả trong việc nắm bắt các mối quan hệ không gian trong mỗi kênh.
3) SENet (Mạng nén và kích thích): Bước này tối ưu hóa hiệu quả của đặc trưng theo kênh, tập trung vào những phần có nhiều thông tin nhất. Nó hiệu chỉnh lại một cách thích ứng các đặc trưng theo kênh bằng cách mô hình hóa rõ ràng sự phụ thuộc lẫn nhau giữa các kênh.
4) Lớp Projection (Chuyển đổi 1x1): Cuối cùng, Lớp này có tích chập 1x1 hợp nhất các đặc trưng, chuẩn bị chúng cho khối tiếp theo. Lớp này làm giảm kích thước của phần kênh nông ra, đưa nó trở lại kích thước thấp hơn.
5) Cộng thặng dư: Sau đó, đầu ra của lớp projecion được thêm vào đầu vào ban đầu của khối (nếu chúng có cùng kích thước), tạo thành kết nối còn lại.

B. Eff-Unet

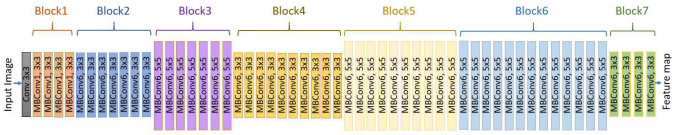
Eff-UNet là kiến trúc mới dành cho phân đoạn theo ngữ nghĩa, được thiết kế đặc biệt cho môi trường phi cấu trúc. Nó kết hợp tính hiệu quả của EfficiencyNet làm bộ mã hóa (encoder) để trích xuất đặc trưng với UNet làm bộ giải mã (decoder) để xây dựng lại bản đồ phân đoạn chi tiết.

Dưới đây là các thành phần và lớp chính trong Eff-UNet:

1) EfficientNet Encoding: **EfficientNet đóng vai trò là bộ mã hóa trong Eff-UNet**. Nó chịu trách nhiệm trích xuất tính năng. EfficientNet sử dụng tỷ lệ kết hợp (mở rộng quy mô chiều rộng, độ sâu và độ phân giải của mạng) để trích xuất thông tin đặc trưng cấp cao.
2) Bộ giải mã UNet: **UNet đóng vai trò là bộ giải mã trong Eff-UNet**. Nó chịu trách nhiệm xây dựng lại bản đồ phân đoạn chi tiết. UNet sử dụng một loạt các tích chập ngược (upsampling) và ghép nối với tầng bên encoder để kết hợp thông tin đối tượng cấp cao với thông tin không gian cấp thấp, rất hữu ích cho việc phân đoạn chính xác.

C. Cài đặt và cấu hình EffUnet

Link bài viết, các cấu hình, hình ảnh, chi tiết từng lớp của mạng bao gồm các tham số đầu vào đầu ra, được nhóm tham khảo tại bài báo này



Hình 5: Tầng encoder với EfficientNetB7, đầu tiên sẽ có một lớp tích chập, sau đó mỗi màu là một block MBConv (tương ứng mỗi tầng trong encoder), mỗi blck sẽ bao gồm một số lớp MBConv.

Chi tiết mỗi lớp Squeeze and Excitation, bao gồm các tham số và các lớp:

1) Tham số đầu vào bao gồm số lượng kênh đầu vào (đầu ra), và tỷ lệ Squeeze để nén (chọn theo tỷ lệ 4).
2) Lớp đầu tiên là Avarage Pooling, dùng để tính trung bình mỗi kênh là cho ra một dãy 1D có kích thước bằng số kênh, mang giá trị trung bình mỗi kênh (Squeeze).
3) Lớp kết nối đầy đủ với lớp trước số nút đầu ra là số kênh / 4, các nút đầu vào là các giá trị được tính bên trên
4) Lớp kết nối đầy đủ với lớp trước, số nút đầu ra là số kênh. để làm trọng số nhân vào mỗi kênh.

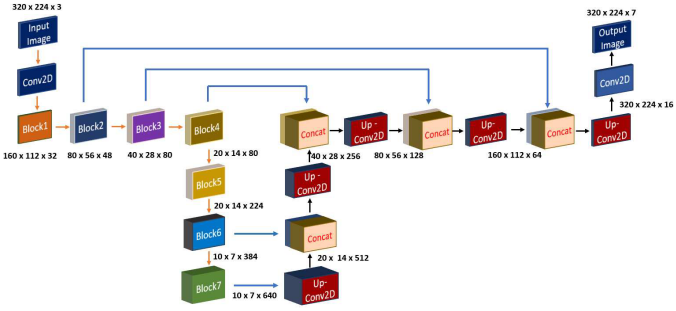
Chi tiết mỗi lớp MBConv, bao gồm các tham số và các lớp:

1) Tham số : Số lượng kênh đầu vào và số lượng kênh đầu ra, Tỉ lệ nhân rộng, stride, kích thước kernel cho từng lớp tích chập.
2) Tích chập 1x1 : nông số lượng kênh. Tiếp theo là batch normalization đẻ model học hiệu quả hơn. Tiếp theo là ReLU activation.
3) Tích chập theo chiều sâu, mỗi channel đầu ra chỉ tích chập với channel đầu vào với độ sâu bằng với nó. Tiếp theo là batch normalization đẻ model học hiệu quả hơn. Tiếp theo là ReLU activation.
4) Lớp Squeeze Excitation, sau đó lấy trọng số đầu ra nhân với đầu vào trước đó.

5) Tích chập 3x3 giảm số lượng channel về kích thước đầu ra. Tiếp theo là batch normalization đẻ model học hiệu quả hơn
6) Nếu kích thước đầu vào ban đầu và đầu ra bằng nhau, lấy tổng thặng dự bằng cách lấy tổng đầu vào ban đầu và đầu ra (Residual Connection).

Tham số	b1	b2	b2	b4	b5	b6	b7
Số channel đầu ra	32	48	80	80	224	384	640
Tỷ lệ nông	1	6	6	6	6	6	6
Stride	1	2	2	2	1	2	1
Kernel	3	3	5	3	5	5	3
Số lượng MBconv	4	7	7	10	10	13	4

Bảng III: Các tham số từng block



Hình 6: Cấu trúc mạng Eff-Unet

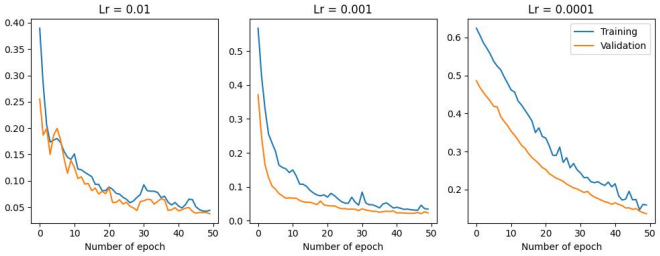
Chi tiết tầng encoder (Tham khảo bảng để xem rõ các tham số bên dưới cho từng block), sử dung kiến trúc EfficientNetB7, mỗi block lần này có nhiều hơn 2 lớp tích chập:

1) Một lớp tích chập 3x3, với stride = 2, giảm độ phân giải ảnh đi 2 lần.
2) Tiếp theo là **7 lớp tích chập Mobile Convolution Block liên tiếp** tương ứng cho mỗi tầng ở encoder các tham số trong từng tầng bao gồm :
   - Số lượng kênh đầu ra, dùng cho số lượng kênh đầu ra cho tất cả các lớp MBConv.
   - Số lượng kênh đầu vào bằng số lượng kênh đầu ra từ tầng trước, sử dụng cho đầu vào lớp MBConv đầu tiên, các lớp MBConv sử dụng số lượng kênh đầu vào bằng số lượng kệnh đầu ra ở trên.
   - Tỉ lệ nhân nông kênh cho tất cả các lớp MBConv
   - Stride cho lớp MBConv đầu tiên, cách lớp MBConv còn lại sử dụng stride = 1
   - Kích thước kernel cho tất cả các lớp MBConv.

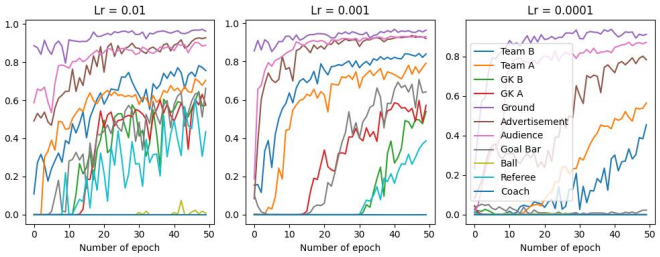
Chi tiết tầng decoder

1) Các block y chang Unet, có điều số kênh đầu ra đã thay đổi : 512, 256, 128, 64, 16 tương ứng các tầng càng lên cao.
2) Ở skip connection, có sự thay đổi, tầng đầu ra 256 nối với block 5 bên encoder, 128 với block3, 64 với block 2, 16 với block 1.
3) Một lớp tích chập với số kênh đầu ra tương ứng số lớp.

D. Đánh giá hiệu suất so với Unet



Hình 7: Error cho từng learning rate



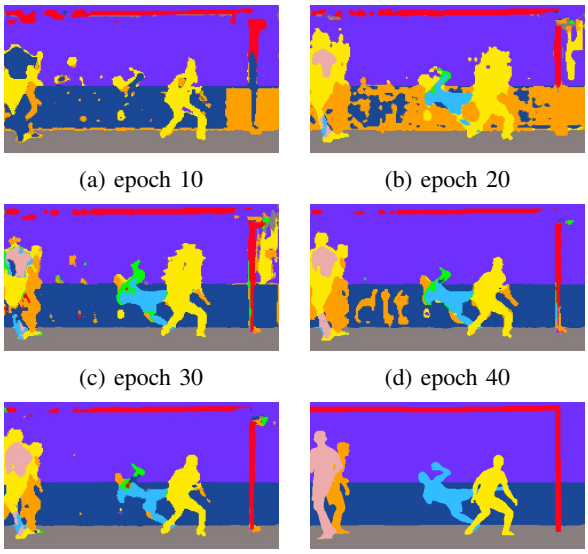
Hình 8: Đo độ chính xác bằng IoU từng learning rate

Có thể thấy khi thử learning 0.01,0.001,0.0001 và 50 epoch, thì learning rate **0.001 cho kết quả hội tụ nhanh nhất với error 0.033**, 0.0001 quá chậm, còn 0.01 vừa cho ra error cuối cùng cao hơn, vừa dao động nhiều và rất lớn trong IoU. Với lr = 0.001 sau 50 epoch, error là 0.033, so với Unet lr = 0.0001 chỉ có 0.013.

Điểm mạnh của Eff-Unet nằm ở độ hiêu quả về mặt tính toán, nhanh hơn hẳn rất nhiều đối với Unet thông thường. **Trung bình một lô Unet mất tới 23s để xử lý xong, trong khi Eff-Unet mất chỉ 6s-7s**, vậy giả dụ ta tăng số epoch lên 4 lần thời gian train cả 2 tương đương nhau, tăng số epoch lên 200 thì error cho Eff-Unet có thể giảm đi rất nhiều không chỉ là theo tỷ lệ 4, vì nó có nhiều cơ hội hơn để tiến về cực tiểu tốt nhất.Hơn nữa, nhờ vậy ta có thể tăng số lượng lô lên, trong bài này nhóm sử dụng lô 4 ảnh để train Eff-Unet.

Khi model cuối cho ra, tốc độ tính toán Eff-Unet nhanh hơn 4 lần so với Unet, với cùng một lượng thời gian train, 2 model cho ra error tương đương, nhưng Eff-Unet là lựa chọn tốt hơn vì nó tính toán nhanh hơn cho các tác vụ phân đoạn.

E. Ảnh phân đoạn đầu ra



(a) epoch 10 (b) epoch 20

(c) epoch 30 (d) epoch 40

(e) epoch 50 (f) Phân đoạn chính xác

Hình 9: Ảnh dự đoán sau từng epoch cho lr = 0.0001

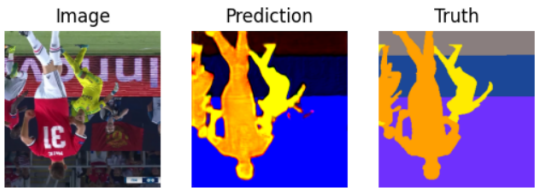
Như đã đề cập trên, lượng error cho ra sau 50 epoch khá cao, 3 lần so với Unet, do đó lớp trọng tài trong ảnh không được phân đoạn hoàn toàn tốt, đặc biệt còn bị lẫn màu của lớp khác, nhưng các lớp còn lại trong ảnh cho ra kết quả phân đoạn khá tốt.

VII. So sánh mô hình cải tiến với các nghiên cứu có liên quan

A. Sử dụng Unet++ trên cùng tập dữ liệu

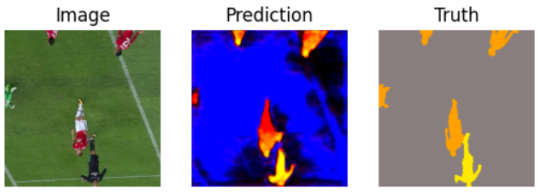
https://www.kaggle.com/code/sivashankarans/football-unet-pytorch-seg-models

Trong bài này, họ sử số lô là 8, learning rate là 0.0003, sử dụng mô hình **Unet++**, và dùng **ResNet34** cho tầng encoder. Tuy nhiên thay vì họ sử dụng 11 kênh đầu ra, thì họ sử dụng 3 đầu ra tượng trưng cho màu ước tính của từng điểm ảnh. Họ dùng **Binary Cross Entropy (BCE) loss** để tính toán lượng lỗi cuối cùng.




```
E: 50 - Loss: 0.5152576565742493, Acc:71.09639323674716%
```

Hình 10: Unet++ sau 50 epoch




```
E: 10 - Loss: 0.526020884513855, Acc:64.75911216858105%
Model Updated
```

Hình 11: Unet++ sau 10 epoch

Do sử dụng đầu ra là 3 kênh màu nên ảnh cho ra 3 kệnh màu không được đúng như ảnh phân đoạn, nhưng nhìn chung lượng màu từng lớp trong ảnh rải dài khá đều, viền cạnh cũng rất tốt. Tuy nhiên ở epoch 50 lớp màu cam vẫn còn nhiều vùng màu vàng, đặc biệt ở 2 cánh tay và đầu.

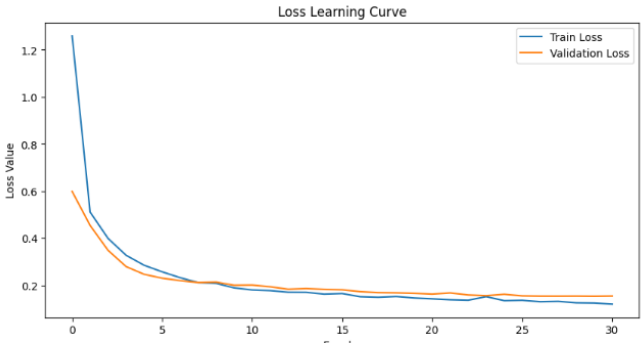
Điều này khả năng là do số kênh đầu ra = 3 của họ chọn, với mô hình Unet của nhóm, sau 50 epoch đã xử lý rất tốt vùng tay và đầu (xem hình 4), khi sử dụng sô lớp làm số kênh đầu ra. Nhìn chung Unet++ chay lâu hơn Unet, vậy với cung thời gian train như Unet, và , Eff-Unet có thể xử lý tương tự (đã đề cập trên phần so sánh với Unet) và tốt hơn mô hình của họ về điểm này.

Tuy nhiên với 10 epoch, họ đã xử lý viền cạnh, biên của các cầu thủ rất tốt, trong khi mô hình Eff-Net của nhóm với epoch 40 vẫn chưa xử lý rõ ràng thông tin này (xem Hình 9).

B. Sử dụng PSPNet trên cùng tập dữ liệu

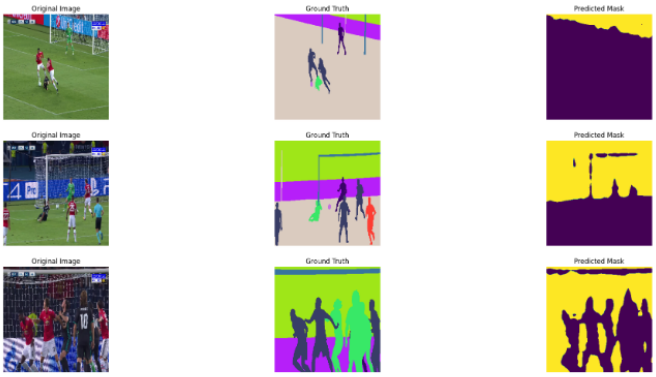
https://www.kaggle.com/code/jainamrajput/ipcv-ii-miniproject-part2-pspnet

Trong bài này, họ sử dụng PSPNet (Pyramid Scene Parsing Network) để phân đoạn trên cùng bộ dữ liệu. Họ sử dụng Cross Entropy Loss và số kênh đầu ra 2 (gom 2 nhóm trong 11 lớp).



Hình 12: PSP error

Nhìn chung error sau 30 epoch của họ cho ra nhìn tốt hơn mô hình Eff-Unet vì họ sử dụng chỉ 2 lớp, và giá trị nhỏ nhất của hành cross-entropy khi chọn 2 và 11 lớp là khác nhau (và không phải lúc nào cũng = 0).



Hình 13: Dự đoán của PsP sau 30 epoch

Sau 30 epoch thì mô hình của họ phân biên cạnh cho các lớp màu nâu (bao gồm cầu thủ) chưa thật sự tốt. Ở 50 epochs cho Eff-Unet, Eff-Unet của nhóm đã cho ra độ chính xác biên cạnh nhìn gần như tương đương với mô hình của họ. PSP được đánh giá huấn luyện khá lâu so với UNet, cùng một lượng thời gian cho 30 epochs sẽ được 120 epochs cho Eff-Unet, rõ ràng đủ nhiều để Eff-Unet của nhóm phân biên cạnh sẽ tốt hơn rất nhiều.