

## Temat projektu

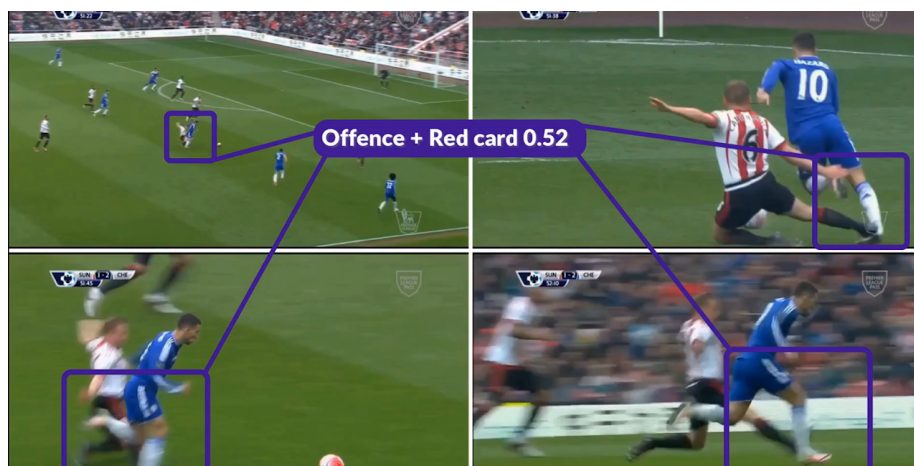
ZZSN 24L KD - temat nr 5 Wielozadaniowe rozpoznawanie faulów przy użyciu modelu wykorzystującego wiele widoków

## Członkowie zespołu

- Adam Górski, 304054
- Miłosz Łopatto, 305898

## Opis zadania

Zadanie jest częścią konkursu łączącego piłkę nożną i wizję komputerową [Soccer-net: Multi-View Foul Recognition 2024](#).



Problem dotyczy wieloetykietowej klasyfikacji przewinień z meczów piłki nożnej. Dla każdej akcji widzianej z wielu perspektyw należy przypisać dwie etykiety:

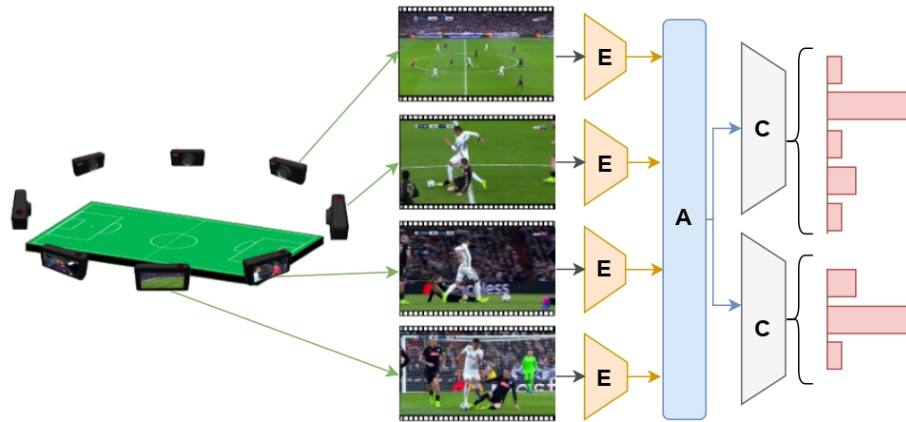
- pierwsza etykieta określa, czy wystąpił faul wraz z odpowiadającym mu stopniem powagi:
  - *No Offence*
  - *Offence + No Card*
  - *Offence + Yellow Card*
  - *Offence + Red Card*
- druga etykieta identyfikuje typ akcji:
  - *Standing Tackle*
  - *Tackle*
  - *Holding*
  - *Pushing*
  - *Challenge*
  - *Dive/Simulation*
  - *High Leg*

– *Elbowing*

## Ustalony cel

W trakcie omawiania koncepcji projektu ustalone zostało, że celem jest pobicie baseline'u.

## Opis architektury modelu



### Pierwsza część architektury - enkoder

Pierwszą częścią architektury jest model wyciągający cechy z klipów wideo. Domyślnie wykorzystywane są wcześniej wytrenowane modele wideo z biblioteki torchvision - takie jak r3d\_18, s3d, mc3\_18, r2plus1d\_18 i mvit\_v2\_s.

### Druga część architektury - agregator

Kolejną warstwą architektury jest agregator, który łączy ze sobą wyniki z kilku wcześniej wspomnianych enkoderów.

### Trzecia część architektury - głowica do klasyfikacji wielozadaniowej

Ostatnia część architektury zwraca prawdopodobieństwa poszczególnych klas dla każdego z zadań.

## Przeprowadzone eksperymenty

### Wykorzystane technologie

Rozwiązanie zostało zaimplementowane przy użyciu języka Python 3.10 z użyciem bibliotek PyTorch oraz PyTorch Lightning. Ta biblioteka pozwoliła nam na

znaczące oczyszczenie kodu oraz jego strukturyzację z użyciem metod programowania obiektowego. Dodatkowo PyTorch Lightning dobrze integruje się ze Slurm-em oraz Weights and Biases, które wykorzystaliśmy do śledzenia eksperymentów.

## Przetwarzanie danych

**Liczba klatek na sekundę:** Zaczęliśmy eksperymentowanie z liczbą klatek na sekundę (fps) oraz klatką startową i klatką końcową. Środek nagrań utrzymaliśmy na 75 klatce, ponieważ wtedy przeważnie występuje foul. Eksperymentowaliśmy natomiast z szerokością okna. Zmiana liczby klatek na sekundę wymusza zmianę szerokości okna. Przykładowo:

- dla 12 klatek na sekundę `start_frame` powinno zostać ustawione na 58, a `end_frame` na 92. Wtedy model przeanalizuje co drugą klatkę (nagrania są w 24 klatkach na sekundę) i środek będzie się znajdował w 75 klatce,
- dla 24 klatek na sekundę `start_frame` powinno zostać ustawione na 63, a `end_frame` na 87. Wtedy model przeanalizuje każdą klatkę, ale okno będzie “węższe”, czyli model ma szansę zobaczyć więcej szczegółów na krótszym fragmencie wideo.

Domyślnie fps było ustawiona na 17. My jednak zostawiliśmy go na 12, ponieważ dawało nam to najlepsze wyniki.

**Augmentacja danych:** Dodatkowo eksperymentowaliśmy także z augmentacją danych. Uzyskiwane przez nas wyniki polepszyło dodanie technik `RandomResizedCrop` oraz `GaussianBlur`. Ostatecznie wyglądało to następująco:

```
transformAug = transforms.Compose([
    transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)),
    transforms.RandomAffine(degrees=(0, 0), translate=(0.2, 0.2), scale=(0.8, 1.2)),
    transforms.RandomPerspective(distortion_scale=0.5, p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.6, saturation=0.6, contrast=0.6),
    transforms.RandomHorizontalFlip(),
    transforms.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 5)),
])
```

## Eksperymenty w ramach enkodera

- ☒ wykorzystanie innych modeli z biblioteki torchvision
- ☒ wykorzystanie enkoderów opartych na transformerach

Ostatecznie najlepszym enkoderem okazał się oparty na transformerze MViTv2. Działał on zdecydowanie lepiej od pozostałych opcji, szczególnie tych opartych na konwolucji.

Prawdopodobnie jedyne co mogło zadziałać lepiej od MViTv2 to Swin3D, który w niektórych benchmarkach dawał niewiele lepsze wyniki. Swin3D także jest oparty na transformerach, lecz powodował problemy z brakiem pamięci.

Table of all available video classification weights

Accuracies are reported on Kinetics-400 using single crops for clip length 16:

Weight	Acc@1	Acc@5	Params	GFLOPS	Recipe
MC3_18_Weights.KINETICS400_V1	63.96	84.13	11.7M	43.34	<a href="#">link</a>
MViT_V1_B_Weights.KINETICS400_V1	78.477	93.582	36.6M	70.6	<a href="#">link</a>
MViT_V2_S_Weights.KINETICS400_V1	80.757	94.665	34.5M	64.22	<a href="#">link</a>
R2Plus1D_18_Weights.KINETICS400_V1	67.463	86.175	31.5M	40.52	<a href="#">link</a>
R3D_18_Weights.KINETICS400_V1	63.2	83.479	33.4M	40.7	<a href="#">link</a>
S3D_Weights.KINETICS400_V1	68.368	88.05	8.3M	17.98	<a href="#">link</a>
Swin3D_B_Weights.KINETICS400_V1	79.427	94.386	88.0M	140.67	<a href="#">link</a>
Swin3D_B_Weights.KINETICS400_IMAGENET22K_V1	81.643	95.574	88.0M	140.67	<a href="#">link</a>
Swin3D_S_Weights.KINETICS400_V1	79.521	94.158	49.8M	82.84	<a href="#">link</a>
Swin3D_T_Weights.KINETICS400_V1	77.715	93.519	28.2M	43.88	<a href="#">link</a>

Figure 1: Porównanie wyników uzyskiwanych w benchmarkach przez MViTv2 i Swin3D [5].

Eksperymentowaliśmy również z ViViT (Video Vision Transformer) z niestandardową głowicą klasyfikacyjną, ale nie udało nam się uzyskać wyników lepszych niż bazowe.

## Trenowanie i podział danych

Używaliśmy obiektu PyTorch Learning Trainer, aby znacznie ułatwić zarządzanie treningiem. Podział danych wyglądał następująco:

- 2915 akcji w zbiorze treningowym
- 410 akcji w zbiorze walidacyjnym
- 300 akcji w zbiorze testowym
- 272 akcje w zbiorze challenge

Wykorzystywaliśmy również różne callbacki, takie jak zapisywanie modelu na krok i wczesne zatrzymanie.

Do uzyskania dobrych wyników niezbędne było także wykorzystanie ważonej funkcji straty.

## Dalsze możliwości rozwoju

Ciekawym pomysłem byłoby przetestowanie architektury Hiera [4].

## Hyperparametry dla najlepszego modelu

```
{
  "pre_model": "mvit_v2_s",
  "pooling_type": "attention",
  "num_views": 5,
  "fps": 12,
  "start_frame": 58,
  "end_frame": 92,
```

```

"batch_size": 2,
"max_num_worker": 4,
"LR": 0.00005,
"gamma": 0.3,
"step_size": 3,
"weight_decay": 0.001,
"data_aug": true,
"weighted_loss": true
}

```

Logi dla najlepszego modelu: legendary-dream-270

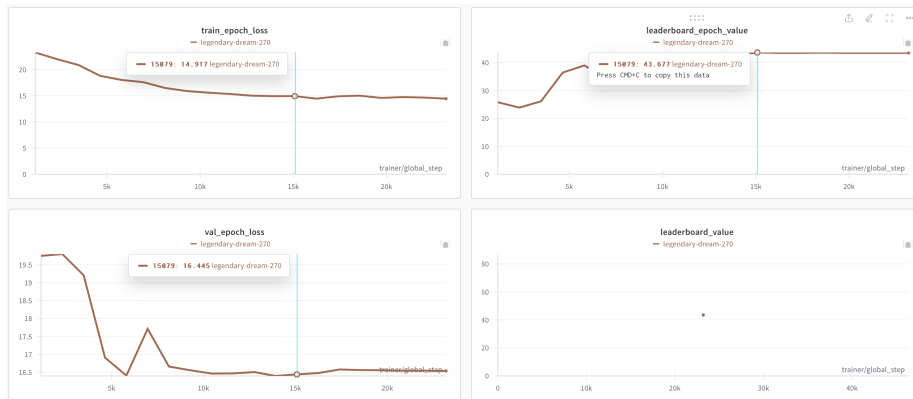


Figure 2: Logi dla ostatecznego modelu. Dokładniej wybrana została wersja modelu z epoki 12 (zaznaczone pionową linią), ponieważ później błąd predykcji na zbiorze walidacyjnym zaczynał rosnąć.

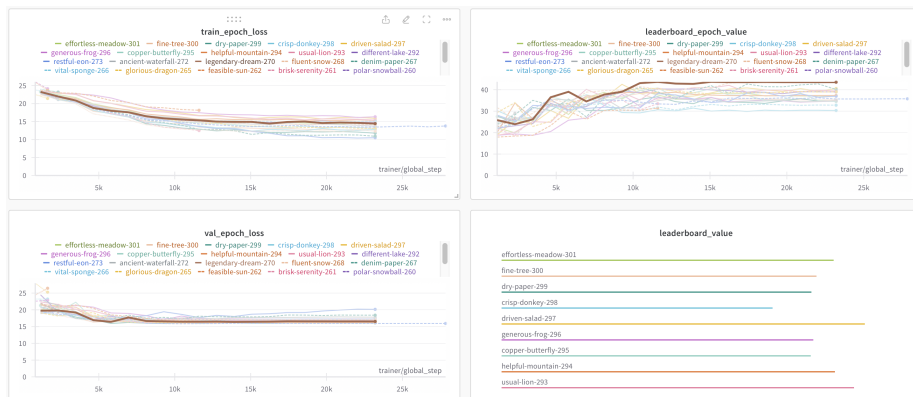


Figure 3: Porównanie logów dla ostatecznego modelu.

## Wyniki eksperymentów

Ostatecznie na zbiorze testowym nasz zespół zdobył trzecie miejsce ze zbalansowaną klasowo celnością dla powagi przewinienia wynoszącą 39.84, zbalansowana celnością dla akcji wynoszącą 45.94 i połączoną metryką 43.68. Dla porównania model bazowy (baseline) miał dla powagi przewinienia 36.25, dla akcji 54.18 i połączoną metrykę 39.60.



Rank	Participant team	Accuracy Offence Severity (↑)	Accuracy Action (↑)	Balanced Accuracy Offence Severity (↑)	Balanced Accuracy Action (↑)	Combined Metric (↑)	Last submission at
1	MLV_SoccerNet (temp)	40.64	36.65	42.24	49.65	45.94	17 days ago
2	ly001	39.44	43.43	43.91	43.87	43.89	2 months ago
3	 PW ZZSN (legendary-dream-270)	40.64	39.84	41.41	45.94	43.68	17 days ago
4	MobiusLabs	32.67	29.48	38.68	46.44	42.56	18 days ago
5	pangzihei	37.85	55.78	37.99	44.14	41.07	17 days ago
6	Host_57937_Team (Baseline) 	36.25	54.18	35.70	43.50	39.60	4 months ago
7	zyz	58.17	58.17	39.42	39.14	39.28	16 days ago
8	xiao_he_shang	45.82	47.01	35.50	40.58	38.04	16 days ago
9	lularsenal	41.04	51.79	34.21	40.83	37.52	1 month ago
10	cmri-ai	45.82	23.90	29.92	28.28	29.10	3 months ago
11	t5	34.66	21.12	34.21	13.95	24.08	17 days ago

Figure 4: Ranking

Nasz zespół nazywał się PW ZZSN. Choć nie udało się uzyskać najlepszego wyniku, udało się pobić baseline, a na zbiorze testowym niewiele brakowało nam do drugiego miejsca.

## Wnioski

Ze względu na podobieństwo zadań stosowanie do obu jednego modelu znacznie ułatwiło trening oraz poprawiło odporność modelu. Augmentacja też pozwoliła nam lepiej przewidywać próbki ze zbioru ukrytego - często próbki w zbiorze ukrytym reprezentowały bardzo zbliżone klipy, ale przykładowo z kamery z innym naświetleniem lub pod innym kątem. Dodatkowo ze względu na bardzo dużą liczbę zdjęć (klip video) ważne jest odpowiednie próbkowanie oraz to, żeby dane testowe na których wykonywana jest inferencja były próbkowane w ten sam sposób co treningowe. Przy długim ciągu danych bardzo efektywną strategią

jest także połączenie wyciągania cech danym modelem oraz mechanizm uwagi pomiędzy nimi i dopiero do wyjścia dołączona głowica lub głowice uwagi. Ze względu na przesadne dopasowanie przydatne też okazały się nam mechanizmy regularyzacji takie jak `weight_decay`.

## Bibliografia

1. Held, J., Cioppa, A., Giancola, S., Hamdi, A., Ghanem, B., & Van Droogenbroeck, M. (2023). VARS: Video Assistant Referee System for Automated Soccer Decision Making from Multiple Views.
2. Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., & Schmid, C. (2021). ViViT: A Video Vision Transformer.
3. Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, Christoph Feichtenhofer (2021) MViTv2: Improved Multi-scale Vision Transformers for Classification and Detection.
4. Chaitanya Ryali, Yuan-Ting Hu, Daniel Bolya, Chen Wei, Haoqi Fan, Po-Yao Huang, Vaibhav Aggarwal, Arkabandhu Chowdhury, Omid Poursaeed, Judy Hoffman, Jitendra Malik, Yanghao Li, Christoph Feichtenhofer. 2023. Hiera: A Hierarchical Vision Transformer without the Bells-and-Whistles.
5. Pytorch vision models