

Laboratorium z przedmiotu Komunikacja Człowiek Komputer

Przetwarzanie obrazów - aplikacja

Temat: Wykrywanie kart do gry typu francuskiego i rozpoznawanie układów pokerowych

Prowadzący: dr hab. inż. Piotr Zielniewicz	Autorzy: 145323 145269 145396	Grupa dziekańska: I1.2 Ocena:
---	---	--

Spis treści

1 Wstęp	2
1.1 Skład grupy projektowej	2
1.2 Tematyka projektu	2
1.3 Specyfikacja problemu	2
1.4 Wybór odpowiednich narzędzi	3
2 Pozyskanie danych	4
2.1 Wycięcie kart z próbek wideo	4
2.2 Tworzenie zdjęć testowych	4
3 Trening	5
4 Podsumowanie	6
4.1 Osiągnięte wyniki	6
4.2 Przykład działania	8
4.3 Napotkane trudności	9
4.4 Możliwe ulepszenia	10
4.5 Wnioski	10
5 Źródła	10

1 Wstęp

1.1 Skład grupy projektowej

Grupa I1.2 (Zajęcia czw. 11:45)

- Michał Dzięcielski 145323
- Michał Fredrych 145269
- Dawid Bosy 145396

1.2 Tematyka projektu

Temat naszego projektu to **"Wykrywanie kart do gry typu francuskiego rozpoznawanie układów pokero-wych"**. Celem naszej pracy było wykonanie takiego systemu, który będzie w stanie w czasie rzeczywistym odczytywać karty z przesyłanego do niego wideo, oraz na ich podstawie obliczać odpowiednie kombinacje pokerowe.

Wybraliśmy ten problem, ponieważ uważaemy, że jego rozwiązywanie mogłoby być wykorzystane na szeroką skalę w profesjonalnych grach pokerowych w celu minimalizacji ryzyka oszustwa graczy, jak i do ogólnego upłynnienia przebiegu.

1.3 Specyfikacja problemu

Aby rozwiązać problem tego typu, musielibyśmy utworzyć system potrafiący znaleźć kartę na zdjęciu, a następnie zakwalifikowaniu jej do jednej z 52 klas, odpowiedzialnych za wszystkie możliwe wartości, jakie mogą wystąpić w talii. Po stworzeniu takiego narzędzia, moglibyśmy skupić się na przydzielaniu otrzymanych wartości do zdefiniowanych wcześniej kombinacji.

Postanowiliśmy, że nasz program będzie odczytywał do kilku kart na jednym zdjęciu. Dodatkowo, karty te nie musiały być w pełni widoczne - mogły one być ułożone w tzw. wachlarz, bądź obrócone pod różnymi kątami (rys. poniżej). Jedyne wymaganie dotyczyło tego, aby widoczny był co najmniej jeden róg każdej karty, przedstawiający kolor (jeden z 4) oraz numer (jeden z 13).



Rysunek 1: Przykładowe dozwolone ułożenia kart

1.4 Wybór odpowiednich narzędzi

Z powodu złożoności problemu, zdecydowaliśmy się na model YOLOv3 - głęboką sieć neuronową, wykorzystującą warstwy konwolucyjne. Zaletą tej architektury jest krótki czas inferencji; detekcja obiektów może odbywać się od kilkunastu do kilkudziesięciu razy na sekundę (w zależności od konfiguracji sprzętowej), co umożliwia wykrywanie kart w czasie rzeczywistym, nawet na przeciętnym laptopie.

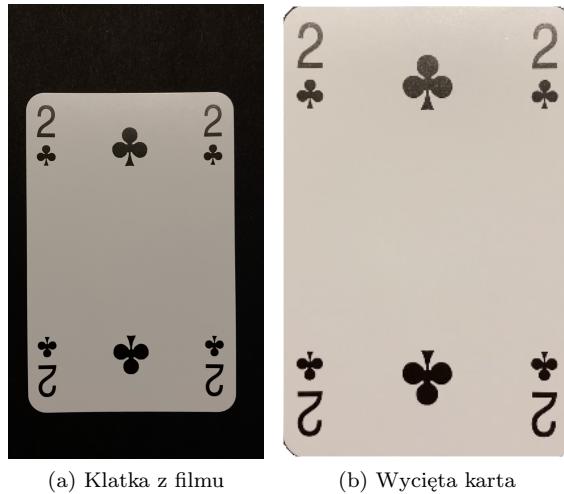
Dodatkowym atutem YOLOv3 jest jego popularność. Sam autor modelu udostępnił gotowy pipeline do treningu oraz testowania, a w internecie z łatwością można znaleźć implementacje modelu w popularnych bibliotekach do uczenia maszynowego, takich jak PyTorch czy TensorFlow.

Postanowiliśmy samodzielnie wytrenować model, na podstawie utworzonych przez nas wcześniej, odpowiednio przygotowanych danych.

2 Pozyskanie danych

2.1 Wycięcie kart z próbek wideo

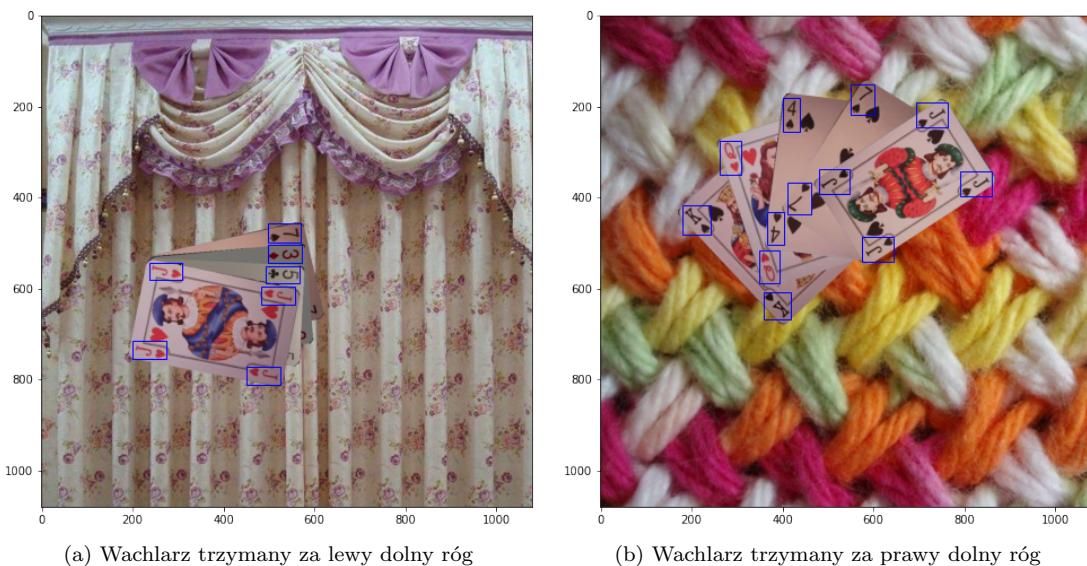
Aby uzyskać odpowiednio duży zbiór danych, zdecydowaliśmy się utworzyć go sami. W tym celu nagraliśmy film trwający od 10 do 15 sekund, przedstawiający jedną kartę z frontu, zmieniając jej oświetlenie. Następnie z jednego filmu, wycinaliśmy około 80 klatek, aby otrzymać z nich próbki kart.



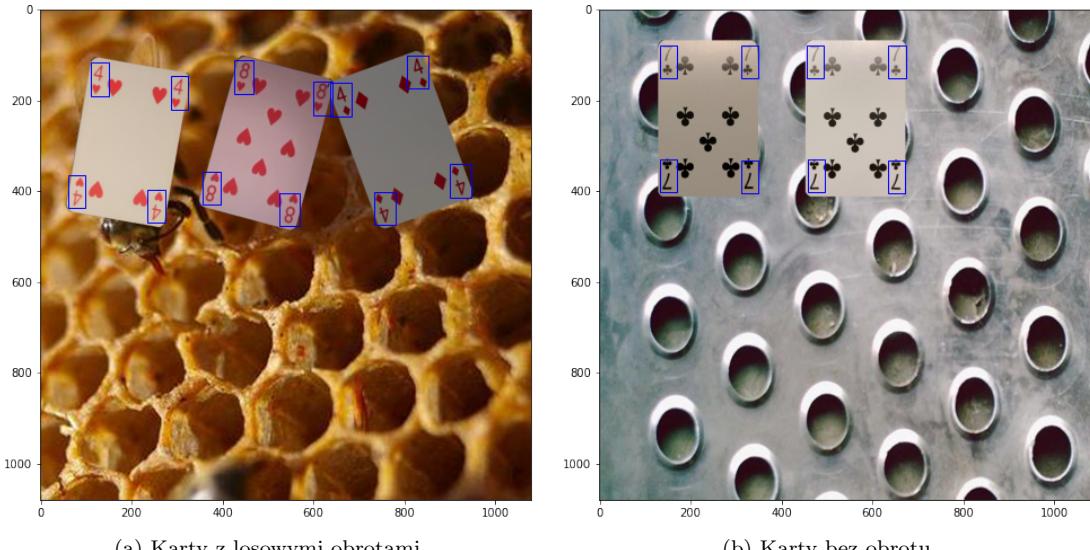
Rysunek 2: Klatka z wideo, oraz ta sama klatka po obróbce

2.2 Tworzenie zdjęć testowych

Po tych operacjach otrzymaliśmy około 4000 próbek pojedyńczych kart. W naszym zamierzeniu program miał odczytywać zbiór kart, dlatego na podstawie tychże próbek zdecydowaliśmy się stworzyć faktyczny zbiór. Biorąc losową ilość kart od 2 do 8, losowe tło z utworzonego wcześniej zbioru przykładowych zdjęć, oraz układając je według 4 ustalonych schematów, udało nam się osiągnąć 50000 unikatowych zdjęć, które mogły nam posłużyć za dane treningowe, walidacyjne i testowe (w proporcjach 70 - 15 - 15).



Rysunek 3: Różne typy generowanych zdjęć (Wachlarze)



Rysunek 4: Różne typy generowanych zdjęć (Rzędy)

3 Trening

Aby wytrenować model skorzystaliśmy z gotowej implementacji modelu YOLOv3 w języku Python oraz biblioteki TensorFlow. Zbiór danych składał się z 50,000 zdjęć w formacie JPG, o rozmiarze 1080×1080 , z czego 40,000 zdjęć przeznaczyliśmy na zbiór treningowy, a 10,000 na zbiór walidacyjny. Adnotacje dla każdej próbki przygotowaliśmy w plikach XML w formacie PASCAL VOC. Model został wytrenowany na karcie graficznej Nvidia Tesla V100, trening trwał około 6 godzin, przez 80 epok.

Parametry treningu:

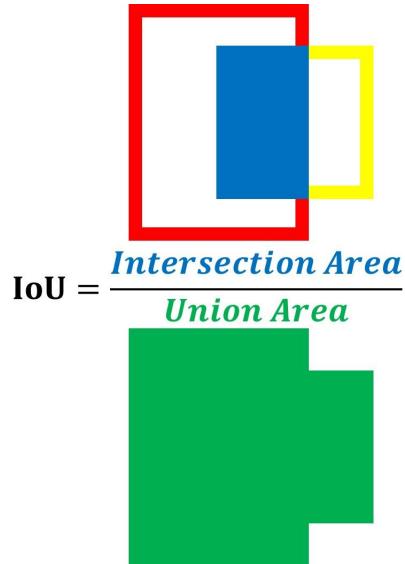
- batch size = 64,
- learning rate = 0.001,
- Model był trenowany na przeskalowanych zdjęciach o rozmiarze 416×416 ,
- Model został zainicjalizowany z wytrenowanymi wcześniej wagami na zbiorze danych COCO (*transfer learning*)

Podczas treningu zastosowaliśmy tak zwany *early stopping* - model automatycznie przestał się trenować, jeżeli po 20 epokach nie nastąpiła poprawa funkcji straty, liczonej na zbiorze walidacyjnym.

4 Podsumowanie

4.1 Osiągnięte wyniki

Wytrenowany model został przetestowany na 10,000 dodatkowo wygenerowanych zdjęciach. Na początku dla każdej z detekcji obliczyliśmy wartość *IoU* (*Intersection Over Union*), czyli metrykę porównującą oryginalnie oznaczony obszar obiektu z nowym obszarem wykrytym przez klasyfikator.



Metryka *IoU* umożliwiła nam podzielenie wykrytych obiektów na trzy klasy:

- *true positive* (poprawnie wykryte obiekty),
- *false negative* (wykryto inny obiekt lub zbyt słaba detekcja)
- *false positive* (brak detekcji obiektu).

Wykryte obiekty przypisaliśmy do odpowiedniej klasy, kierując się poniższym wzorem:

$$\text{class}(IoU) = \begin{cases} \text{true positive} & \text{IoU} \geq \text{Threshold} \\ \text{false negative} & 0 < \text{IoU} < \text{Threshold} \\ \text{false positive} & \text{IoU} = 0 \end{cases}$$

Następnie dla 10 rosnących wartości progu ($\text{Threshold} = 0.2, 0.25, 0.3, \dots, 0.7$) przydzieliśmy jedną z trzech klas dla każdej z detekcji i pogrupowaliśmy wszystkie zdjęcia według klasy karty. Obliczenie liczności zbiorów *true positive*, *false negative* oraz *false positive* umożliwiło nam obliczenie **precyzji** (jaka część wyników zaklasyfikowanych poprawnie jest faktycznie poprawna) oraz **czułości** (jaką część poprawnych wyników wykrył klasyfikator) dla każdej z klasy karty.

$$\text{precision} = \frac{TP}{TP + FP}$$
$$\text{recall} = \frac{TP}{TP + FN}$$

Wartości precyzji i czułości obliczyliśmy dziesięciokrotnie dla wspomnianych powyżej różnych wartości progu, w tabeli 1 umieściliśmy przykładowe wartości dla $\text{Threshold} = 0.5$

Card class	True positives	False positives	False negatives	Precision	Recall
10C	1632	34	459	0.980	0.780
10D	1398	25	717	0.982	0.661
10H	1201	33	742	0.973	0.618
10S	1577	20	573	0.987	0.733
2C	605	42	1397	0.935	0.302
2D	1540	20	564	0.987	0.732
2H	1658	12	373	0.993	0.816
2S	778	36	1294	0.956	0.375
3C	1523	41	668	0.974	0.695
3D	1638	24	435	0.986	0.790
3H	1630	39	364	0.977	0.817
3S	1145	31	1014	0.974	0.530
4C	1520	41	525	0.974	0.743
4D	1097	20	808	0.982	0.576
4H	1620	32	417	0.981	0.795
4S	1650	28	279	0.983	0.855
5C	1266	42	858	0.968	0.596
5D	1564	31	435	0.981	0.782
5H	1796	33	211	0.982	0.895
5S	1482	37	469	0.976	0.760
6C	1640	25	535	0.985	0.754
6D	1755	11	150	0.994	0.921
6H	1790	20	352	0.989	0.836
6S	1488	17	646	0.989	0.697
7C	1051	27	987	0.975	0.516
7D	1483	14	581	0.991	0.719
7H	1690	26	352	0.985	0.828
7S	1078	38	968	0.966	0.527
8C	1465	42	548	0.972	0.728
8D	1669	27	479	0.984	0.777
8H	1428	16	571	0.989	0.714
8S	1733	20	320	0.989	0.844
9C	904	43	1213	0.955	0.427
9D	1668	28	470	0.983	0.780
9H	1332	45	713	0.967	0.651
9S	1220	41	738	0.967	0.623
AC	983	45	1105	0.956	0.471
AD	1921	21	211	0.989	0.901
AH	1239	55	757	0.957	0.621
AS	1191	37	993	0.970	0.545
JC	1575	20	464	0.987	0.772
JD	1118	33	891	0.971	0.556
JH	450	52	1699	0.896	0.209
JS	1494	24	609	0.984	0.710
KC	1409	23	719	0.984	0.662
KD	893	38	1078	0.959	0.453
KH	1461	24	677	0.984	0.683
KS	1634	30	417	0.982	0.797
QC	1464	37	791	0.975	0.649
QD	1882	9	241	0.995	0.886
QH	927	47	1026	0.952	0.475
QS	816	61	1267	0.930	0.392

Tabela 1: Liczności zbiorów oraz obliczone metryki dla Threshold = 0.5

W następnym kroku obliczyliśmy wartości *AP* (*Average Precision*) dla każdej z klas kart, według poniższego wzoru:

$$AP = \sum_{k=0}^{k=n-1} [R(k) - R(k+1)] \cdot P(k)$$

gdzie:

- n - liczba progów (w naszym przypadku 10),
- R - lista z wartościami czułości dla danej klasy,
- P - lista z wartościami precyzji dla danej klasy,
- $R(n) = 0$
- $P(n) = 1$

Na koniec, mając do dyspozycji wartości *AP* dla każdej z klas kart, obliczyliśmy finalną metrykę, określającą ogólną jakość modelu, czyli *mAP* (*mean Average Precision*).

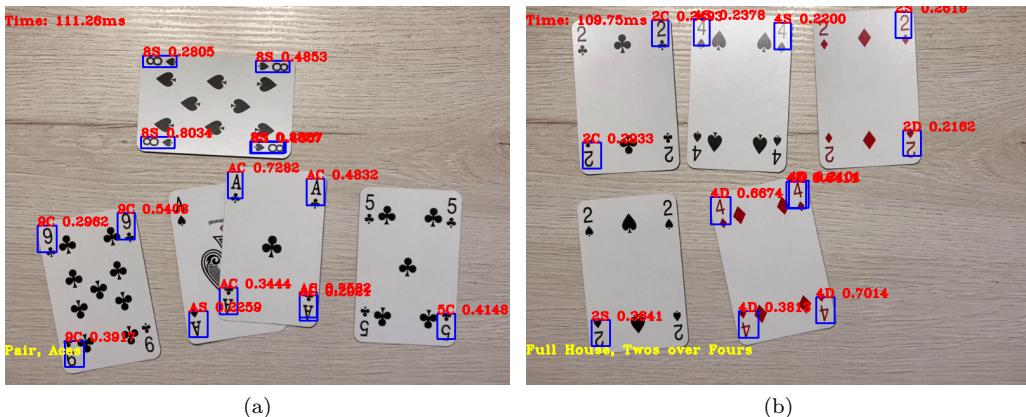
$$mAP = \frac{1}{52} \sum_{k=0}^{k=52} AP_k \approx 0.642$$

Warto również na koniec podkreślić, że rozpoznając karty, model posłużyć może się jednym z czterech rogów. W opisanych metrykach, potraktowaliśmy go bardzo rygorystycznie, uwzględniając brak wykrycia któregokolwiek z widocznych punktów. W praktyce jednak, do poprawnego rozpoznania karty i określenia układu pokerowego, wystarczy wykrycie i poprawne przypisanie chociażby jednego z rogów karty.

Cały proces testowania modelu jest zaprezentowany w załączonym notebooku *test_cards.ipynb*.

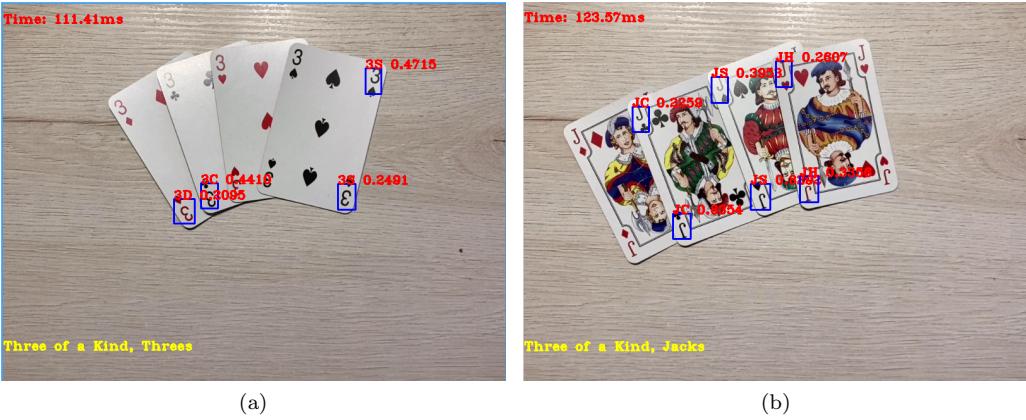
4.2 Przykład działania

Nasz program poddaliśmy nie tylko wnikliwym testom na sztucznie wygenerowanym zbiorze testowym, ale sprawdziliśmy także jak radzi sobie w praktyce. Wyniki przy wykrytych obiektach to pewność danej klasy.



Rysunek 5: Wyniki pozytywne

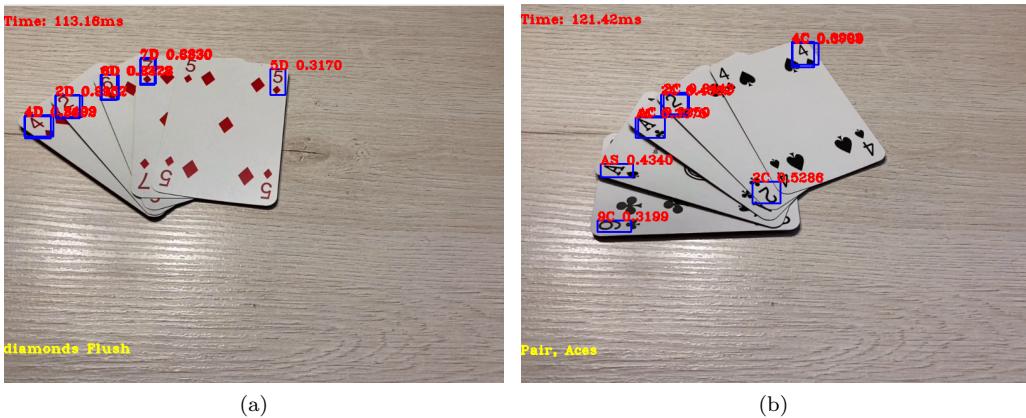
Nasz model nierzadko radzi sobie wprost perfekcyjnie, rozpoznając poprawnie wszystkie karty na klatce wideo, nie generując żadnych fałszywych wyników.



Rysunek 6: Wyniki negatywne

Oczywiście błędy również się zgadzają, choć z punktu widzenia naszego modelu nie są one rażące. Zazwyczaj błędy polegają na tym, że program nie wykryje jednej lub dwóch kart lub błędnie je zinterpretuje. W zdecydowanej większości problemy dotyczą kierów oraz karo.

Niestety, pojedynczy błąd zazwyczaj rzutuje na poprawność rozpoznania ręki pokerowej.



Rysunek 7: Testy "pod kątem"

Pomimo tego że nasz model był trenowany na zdjęciach które odpowiadały widokom "z góry", nasz model zaskakująco dobrze radzi sobie ze zdjęciami "pod kątem". Na Rysunku 7b odczytał błędnie tylko kolor jednej z kart!

4.3 Napotkane trudności

Nasze pierwsze podejście odbyło się na znalezionym w internecie zbiorze danych. Jednak wyniki, które otrzymaliśmy, były dalekie od zadowalających. Skuteczność rozpoznawania była bliska sytuacji, w której karta wybierana byłaby losowo. Stwierdziliśmy wtedy, że należy wielokrotnie rozszerzyć zbiór danych, co opisaliśmy w jednej z powyższych sekcji.

Nawet po zwiększeniu zbioru uczącego do kilkudziesięciu tysięcy próbek, model wciąż stosunkowo często mylił ze sobą kierę oraz pikę co jest zrozumiałe z uwagi na bardzo podobny kształt. Dla mocno różniących się od reszty kolorów - trefla, model rozpoznaje karty praktycznie perfekcyjnie.

4.4 Możliwe ulepszenia

W końcowym etapie prac odkryliśmy, że z powodu wykonywania nagrań kart z podobnej odległości, program najgorzej radzi sobie z kartami bardziej oddalonymi. Z pewnością rozszerzając zbiór treningowy o próbki z różnym rozmiarem kart, zwiększoną zostałaby skuteczność programu.

4.5 Wnioski

Podsumowując, program spełnił początkowe założenia. System z zadowalającym nas prawdopodobieństwem wykrywa wzory kart, a nas określa odpowiednią kombinację. Uważamy, że większość problemów związanych ze skutecznością byłaby możliwa do zniwelowania odpowiednio poszerzając zbiór danych i ponownie trenując sieć.

Praca nad tym zadaniem zdecydowanie poszerzyła naszą wiedzę. Podchodziąc do niego po raz drugi z pewnością zmienilibyśmy kilka rzeczy. Przede wszystkim skupilibyśmy się na przygotowaniu jeszcze liczniejszego i bardziej różnorodnego zbioru danych mając na uwadze: rozmiary kart, oświetlenie, zdjęcia robione "pod kątem" a także użylibyśmy wzorów z więcej niż jednej talii.

5 Źródła

- <https://pjreddie.com/darknet/yolo/>
- <https://github.com/zzh8829/yolov3-tf2>
- <https://blog.paperspace.com/mean-average-precision>
- <https://github.com/hugofloter/Cards-machine-learning>
- <https://www.robots.ox.ac.uk/~vgg/data/dtd/download/dtd-r1.0.1.tar.gz>