Porównanie wydajności złożonych modeli wizyjnych do detekcji dronów

Michał Bober, Antoni Jaszcz

Streszczenie

W naszym projekcie zajęliśmy się problemem detekcji dronów na obrazach/klatkach filmu. W naszych eksperymentach testowaliśmy klasyczne konwolucyjne sieci neuronowe, VGG16 i ResNet50 oraz różniące się wielkością warianty architektur YOLOv5 i YOLOv8. Założenie projektu dotyczyło porównania modeli i wyznaczenia najbardziej optymalnego (pod względem złożoności, wykorzystania zasobów w procesie uczenia i predykcji oraz uzyskanej dokładności) modelu. Wszystkie modele były dotrenowywane przez 10 epok i korzystały z wcześniej wytrenowanych wag na zbiorach ImageNet oraz COCO. Uzyskane wyniki pokazują, że model YOLOv8n jest najlepszym wyborem do wybranego przez nas zadania, osiągając najmniejsze zużycie zasobów pamięciowych i uzsykując dokładność mAP50 wynoszącą 91.48%.

I. Wstęp

Tempo rozwoju technologii i wzrost dostępnej mocy obliczeniowej w ostatnich latach, pozwolił na zastosowanie sztucznych sieci neuronowych (ANN) w problemach dotyczących przetwarzania obrazów. Jednym z nich jest detekcja obiektów, które stanowią kluczowy element inteligentnych systemów stosowanych w medycynie, kontroli ruchem lub monitoringu.

W ostatnich latach przeprowadzony został szereg badań wykorzystujący konwolucyjne sieci neuronowe (CNN) oraz architektury pochodne, które są w stanie w precyzyjny sposób rozpoznać i odszukać obiekt na zdjęciu/klatce filmu. Aby jednak cały system działał sprawnie, niezbędny jest kompromis pomiędzy dwoma przeciwstawnymi wymaganiami: niską złożonością modelu (aby system był wstanie wykonywać predykcje w czasie rzeczywistym), a jego efektywnością (aby były one dostatecznie dobre). Jednym z pierwszych rozwiązań, było wprowadzenie regionalnych sieci neuronowych (RCNN) [1], których zasada działania opiera się na wstępnym generowaniu tzw. regionów zainteresowania (ROI), w celu zminimalizowania ilości obliczeń. Przez długi czas takie podejście stanowiło podstawową architekturę do detekcji obiektów oraz było udoskonalane jako Fast-RCNN [2] i Faster-RCNN [3].

W 2016r., na konferencji CVPR, zaprezentowany został algorytm You Only Look Once (YOLO) [4], który zrewolucjonizował podejście do problemu detekcji obiektów. Model ten przetwarza obraz w całości, dzieląc go na siatkę, w komórkach której przewidywane są granice i prawdopodobieństwa wystąpienia danego obiektu. Dzięki temu, algorytm analizuje obraz tylko raz (stąd nazwa), co pozwala na bardzo szybkie wykrywanie obiektów. Jednocześnie, takie podejście nie skutkuje spadkiem skuteczności całego modelu.

Dzięki nowym architekturom uczenie głębokie jest skutecznie wykorzystywane w szerokim zakresie aplikacji, takich jak analiza obrazów sonarowych [5], segmentacja obrazów z łazików [6] lub śledzenie obiektów latających. Precyzyjne wykrywanie i śledzenie obiektów wymaga modeli, które są zarówno szybkie, jak i dokładne, ponieważ obiekty te mogą poruszać się z dużą prędkością i w zmieniających się warunkach środowiskowych.

W naszym projekcie rozważaliśmy problem detekcji dronów, porównując wydajność popularnych architektur CNN (VGG16 i ResNet50) oraz wariantów sieci YOLO. Pod uwagę została wzięta zarówno złożoność obliczeniowa, pamięciowa i czasowa (w procesie trenowania oraz samej predykcji), jak i jakość uzyskiwanych przewidywań modelów.

II. Metodologia

A. Konwolucyjne sieci neuronowe

Konwolucyjne sieci neuronowe (CNN) opierają się na tym, jak ludzkie oko postrzega otaczający świat. Zamiast zwracać uwagę na każdy pojedynczy piksel, możemy wyodrębnić informacje z tak zwanego pola percepcji. Każda warstwa konwolucyjna posiada zestaw filtrów (macierz wag), o wymiarach odpowiadającym wybranemu polu percepcji dla warstwy. Poprzez przemnożenie kolejnych rejonów percepcji z obrazu wejściowego z macierzą wag (i otrzymaniu wartości funkcji aktywacji z uzyskanej sumy kumulatywnej), dla każdego filtra otrzymana zostaje tak zwana mapa cech, wydobywająca z obrazu bardziej abstrakcyjne cechy. Warstwy konwolucyjne najczęściej łączone są naprzemiennie z warstwami ściągającymi (pooling layers), jak np. warstwy max-pooling, które pomagają zredukować ilość danych wejściowych w kolejnych warstwach, zmniejszając uzyskane mapy cech, poprzez przepuszczenie dalej tylko najsilniejszych sygnałów. Tym głębsza sieć, tym ułożone w ten sposób warstwy konwolucji i ściągania wydobywają z obrazu bardziej złożone cechy, począwszy od prostych informacji o krawędziach,

Wydział Matematyki Stosowanej, Politechnika Śląska
Kaszubska 28, 44-100 Gliwice
M. Bober, mb300327@student.polsl.pl
A. Jaszcz, aj303181@student.polsl.pl

przez informacje o kształcie, aż po własności obrazu bezpośrednio uczestniczących w obliczeniach następującej warstwy gęstej, takiej jak np. obecność oczu na obrazie. Wyjście każdej warstwy konwolucyjnej można opisać równaniem 1, gdzie $z_{i,j,k}$ jest wyjściem neuronu w i-tym wierszu, j-tej kolumnie i k-tej mapie cech warstwy konwolucyjnej l, s_h i s_w są odpowiednio poziomymi i pionowymi krokami, f_h i f_w są wysokością i wagą pola percepcji, $f_{n'}$ oznacza liczbę kanałów w poprzedniej $(l-1)$ warstwie, $x_{i',j',k'}$ jest wyjściem neuronu w poprzedniej warstwie, w odpowiedniej pozycji, b_k jest obciążeniem mapy cech k.

$$z_{i,j,k} = b_k \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k=0}^{f_n-1} x_{i',j',k'} \cdot w_{u,v,k',k}, \quad gdzie \quad i' = i \times s_h + u, j' = j \times s_w + v \tag{1}$$

B. Architektura YOLO

Model You Only Look Once (YOLO) to jednoetapowa sieć wykrywania obiektów, w której kluczowym elementem jest podział obrazu wejściowego na siatkę i dokonywanie predykcji bezpośrednio z całego obrazu podczas pojedynczego przejścia. Każda komórka siatki przewiduje stałą liczbę ograniczających pól i prawdopodobieństwa wystąpienia w nich możliwych klas. Po podzieleniu obrazu na macierz s x s pól, w każdej komórce dokonywana jest predykcja b ramek ograniczających (*ang.* bounding boxes) - wartości x, y, w, h, miarę obiektowości (miara prawdopodobieństwa, że obiekt istnieje w proponowanym obszarze zainteresowania, Rów. 2) oraz prawdopodobieństwa wystąpienia w obrębie ramki danej obiektu danej klasy.

$$c = P(object) \times IoU_{pred}^{true} \tag{2}$$

Z uwagi na złożoność zadań poszczególnych etapów sieci, podstawowa funkcja kosztu przy trenowaniu sieci typu YOLO składa się z trzech osobnych funkcji kosztu: lokalizacji, pewności oraz klasyfikacji.

$$LocalizationLoss = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2 \right] \tag{3}$$

$$ConfidenceLoss = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[1_{ij}^{obj} (c_i - \hat{c}_i)^2 + \lambda_{noobj} 1_{ij}^{noobj} (c_i - \hat{c}_i)^2 \right] \tag{4}$$

$$ClassificationLoss = \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{5}$$

$$TotalLoss = \lambda_{coord} \times LocalizationLoss + \lambda_{obj} \times ConfidenceLoss + \lambda_{class} \times ClassificationLoss \tag{6}$$

- S: Rozmiar siatki (np. siatka $S \times S$).
- B: Liczba prognozowanych ramek ograniczających na każdą komórkę siatki.
- x_i, y_i: Współrzędne środka prognozowanej ramki ograniczającej.
- $\hat{x}_i, \hat{y}_i$: Współrzędne środka ramki ograniczającej dla prawdziwej wartości.
- w_i, h_i: Szerokość i wysokość prognozowanej ramki ograniczającej.
- $\hat{w}_i, \hat{h}_i$: Szerokość i wysokość ramki ograniczającej dla prawdziwej wartości.
- c_i: Prognozowany wynik pewności dla ramki ograniczającej.
- $\hat{c}_i$: Wynik pewności dla prawdziwej wartości.
- $p_i(c)$: Prognozowane prawdopodobieństwo klasy c dla ramki ograniczającej.
- $\hat{p}_i(c)$: Prawdopodobieństwo klasy c dla prawdziwej wartości.
- 1_{ij}^{obj}: Funkcja wskaźnikowa, która wynosi 1, jeśli j-ta ramka ograniczająca w komórce i zawiera obiekt, w przeciwnym razie 0.
- 1_{ij}^{noobj}: Funkcja wskaźnikowa, która wynosi 1, jeśli j-ta ramka ograniczająca w komórce i nie zawiera obiektu, w przeciwnym razie 0.
- λ_{coord}: Współczynnik wagowy dla straty lokalizacji.
- λ_{obj}: Współczynnik wagowy dla straty pewności.
- λ_{noobj}: Współczynnik wagowy dla straty pewności dla ramek bez obiektów.
- λ_{class}: Współczynnik wagowy dla straty klasyfikacji.

III. EKSPERYMENTY

A. Dane

W naszym projekcie zajmowaliśmy się detekcją dronów na wykonanych zdjęciach. W wykonanych eksperymentach, posłużyliśmy otwarto-źródłową bazą zdjęć dronów, prezentującą jednostki latające w różnych sytuacjach. Wszystkie próbki wraz z etykietami dostępne są na platformie Kaggle. Każdej próbce odpowiada odpowiada plik tekstowy, gdzie ramki ograniczające zostały zapisane w formacie YOLO (4 znormalizowane wartości zmiennoprzecinkowe, określające odpowiednio środek ramki oraz jej długość i szerokość). W danych wyróżniamy tylko 1 klasę główną (nie licząc tła), która reprezentuje drony. Zbiór zawiera łącznie 1486 próbek. Przykładowe próbki zostały zaprezentowane w Rys. 1. Zdjęcia są w formacie RGB. Każda próbka została uprzednio znormalizowana do wartości [0, 1] przed wejściem na sieć. Dane zostały podzielone w proporcjach 7:2:1 na zbiór treningowy, walidacyjny i testowy. Struktura folderów wyglądała następująco ze względu na wymagania YOLO

```
train/
   images/
   labels/

val/
   images/
   labels/

test/
    images/
    labels/
```

Dodatkowo w przypadku trenowanie modelów o architekturze YOLO przygotwany został następujący plik konfiguracyjny data.yaml.

```
train: D:\detection_drones\dataset\yolo_dataset\train
val: D:\detection_drones\dataset\yolo_dataset\val
test: D:\detection_drones\dataset\yolo_dataset\test

nc: 1
names: ['drone']
```

Rysunek 1: Przykładowe próbki w użytej bazie danych

Rysunek 2: Próbki z nałożonymi etykietami

B. Augmentacja danych

W celu zwiększenia reprezentatywności danych względem całej populacji (dotyczącej wykrywania dronów w różnych otoczeniach), zastosowaliśmy dwie proste techniki augmentacji danych: losowy przerzut obrazu w poziomie oraz losową rotację $\pm 30°$. Dzięki temu, modele poznawały więcej pozycyjnych układów, w jakich mogą znajdować się drony, zakładając, że wykrywane drony są w pionie lub delikatnym przechyle. W przypadku uczenia modeli YOLO wystarczyło dodać następujące dwie linijki do pliku konfiguracyjnego data.yaml

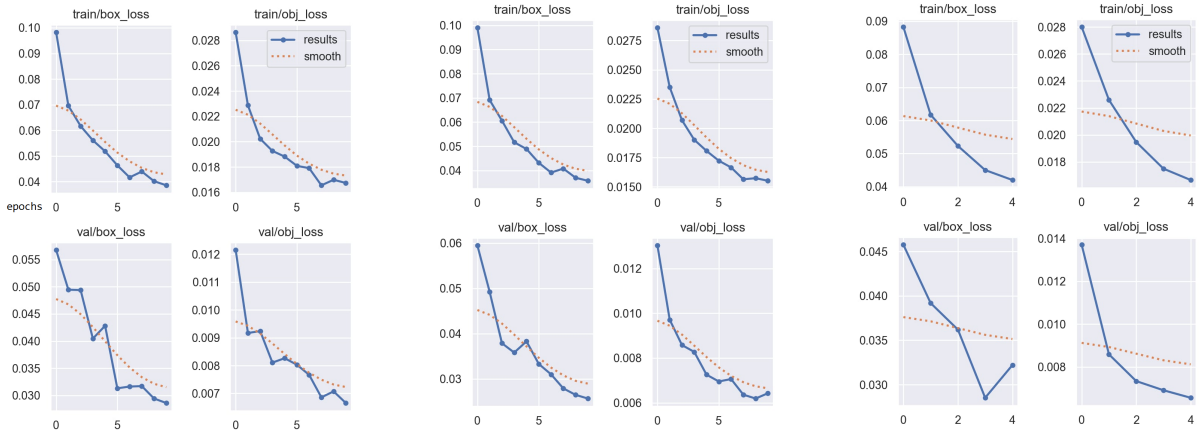
```
degrees: 30.0
fliplr: 0.5
```

C. Ustawienia eksperymentalne

Wszystkie obliczenia były prowadzone na komputerze z GPU nvidia rtx 2060, która była wykorzystywana w obliczeniach. Trening odbywał się mini grupami, liczącymi 16 próbek. Każdy z testowanych modeli był uczony przez 10 epok. Wybranym algorytmem optymalizacji był Adam, z maksymalnym wskaźnikiem uczenia ustawionym na 1e-3.

D. Dotrenowanie modeli

Aby przyśpieszyć proces uczenia, w naszych eksperymentach korzystaliśmy z transfer learningu (TL). W przypadku widzenia komputerowego, opiera się ona na przeniesieniu wag enkodera z innego, pre-trenowanego modelu i wykorzystanie ich jako tzw. kręgosłupa (*ang* backbone), aby ułatwić proces uczenia w skomplikowanym zadaniu, takim jak detekcja obiektów. Dzięki temu, model nie musi dodatkowo uczyć się (jeśli wagi są zamrożone) lub delikatnie korygować proces ekstrakcji cech w warstwach konwolucyjnych. W naszych eksperymentach wykorzystaliśmy przetrenowane wagi modeli VGG16, ResNet50 (trenowane na zbiorze ImageNet) oraz YOLOv5, YOLOv8 (trenowane na zbiorze COCO). Podczas uczenia modeli, warstwy konwolucyjne zostały zamrożone, a trenowanie polegało na dotrenowaniu warstw gęstych sieci, odpowiadających za klasyfikację/regresję. Wykresy wartości funkcji kosztu dla modeli YOLOv5 podczas uczenia zostały pokazane na Rys. 3. Wykresy dokładności podczas uczenia w przypadku prostszych sieci CNN zostały przedstawione na Rys. 4.



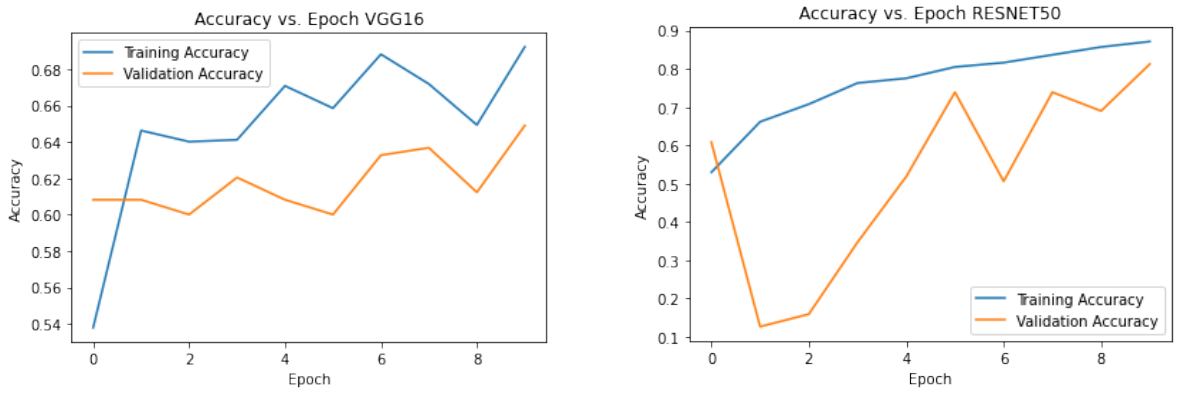
(a) YOLOv5n (b) YOLOv5s (c) YOLOv5m

Rysunek 3: Porównanie wykresów funkcji kosztu podczas uczenia dla różnych wersji modelu YOLOv5

E. Implementacja uczenia

Do eksperymentów wykorzystaliśmy język Python. Trenowanie modeli VGG16 oraz ResNet50 odbyło się z użyciem biblioteki TensorFlow. Do zamodelowania, trenowania i walidacji sieci wykorzystaliśmy moduły dostępne w interfejsie Keras. Sieci typu YOLO trenowane były przy użyciu biblioteki PyTorch oraz części gotowych implementacji sieci. Cały proces trenowania w obydwu przypadkach skonfigurowany był tak, aby proces uczenia mógł wykorzystywać GPU. Aby skutecznie nadzorować wykorzystywane przez proces uczenia zasoby, napisany został prosty dekorator, wykorzystujące bibliotekę GPUtil oraz psutil.

```
@monitor_performance(f'../results/{model_used}_memory_usage')
def train_yolo_model():
    model = YOLO(f'{model_used}.pt')
    model.train(data='D:\detection_drones\dataset\data.yaml', epochs=20, imgsz=640)
    metrics = model.val()
    map50 = metrics.box.map50
    return map50
```



(a) VGG16 (b) ResNet50

Rysunek 4: Porównanie dokładności podczas uczenia dla modeli CNN

(a) (b)

(c)

Rysunek 5: Przykładowe predykcje modelu YOLOv8n na zbiorze testowym

F. Wyniki

Metodyka mierzenia wydajności składała się z pomiarów 3 typów wartości: zużycia zasobów pamięciowych i czasowych (Tab. I), całkowitej liczby parametrów modelu ooraz uzyskanej dokładności mAP50 (Tab. II). Na wykorzystany czas składała się całkowita długość treningu oraz predykcje zbioru testowego. Najkrótszy czas uczenia osiągnął model ResNet50. W przypadku wykorzystania zasobów pamięciowych, najmniejsze zużycie było przy trenowaniu YOLOv8n. Najmniejsza liczba parametrów testowanych modelów to niecałe 2mln (dla YOLOv5n). Najwyższą dokładność osiągnął model YOLOv5s, z wartością mAP50 równą 91.60%. Przewidywania modelu YOLOv8n zostały przedstawione na Fig. 5. Kod źródłowy wszystkich eksperymentów oraz wyniki dostępne są na platformie GitHub.

Model	peak memory usage (MB)	mean memory usage (MB)	peak gpu usage (MB)	mean gpu usage (MB)	runtime (s)
VGG16	4799	4540	5738	5642	270
ResNet50	5012	4860	5747	5686	**240**
YOLOv5n	2367	1664	3308	2290	403
YOLOv5s	2438	1810	5587	3364	383
YOLOv5m	4821	3578	5775	4305	1353
YOLOv8n	**2256**	**1635**	**3011**	**2118**	451
YOLOv8s	2473	1721	5302	3073	391

Tabela I: Porównanie zużycia zasobów dla różnych modeli

Model	Ilość parametrów (mln.)	Acc. (mAP 50)
VGG16	18.9	68.38
ResNet50	40.3	77.94
YOLOv5n	**1.9**	90.69
YOLOv5s	7.2	**91.60**
YOLOv5m	21.2	90.42
YOLOv8n	3.2	91.48
YOLOv8s	11.2	90.55

Tabela II: Porównanie ilości parametrów w testowanych modelach oraz dokładności mAP50

IV. WNIOSKI

W naszym projekcie zaprezentowaliśmy zastosowanie sztucznych sieci neuronowych do detekcji dronów na zdjęciach. W wykonanych eksperymentach, testowaliśmy zarówno klasyczne architektury CNN (VGG16 oraz ResNet50), jak i różne warianty architektur sieci YOLOv5 oraz YOLOv8 (różniących się rozmiarem). Wszystkie modele wykorzystywały wagi pre-trenowanych modeli i były dotrenowywane do zadania przez 10 epok.

Z uzyskanych wyników widać, że wszystkie architektury typu YOLO okazały się lepsze, od klasycznych CNN, osiągając ponad 90-procentową dokładność w każdym przypadku. Co więcej, okazały się znacznie efektywniejsze pod względem wielkości modelu jak i zużyciem zasobów podczas trenowania. Choć model YOLOv8n ma niższą dokładność, niż YOLOv5s, jest to różnica zaledwie 0.12 punktów procentowych. Jest to niewielka strata, szczególnie w porównania z ewidentną przewagą YOLOv8n w zakresie ilości parametrów (ponad połowę mniejsza) oraz użycia karty graficznej. Z uzyskanych rezultatów wynika, że model YOLOv8n jest najbardziej optymalnym modelem do zadania detekcji dronów. Oczywiście, otrzymane wyniki mają poglądowy, ponieważ hiper-parametry nie były dostrajane do każdego modelu z osobna, a dotrenowanie trwało tylko 10 epok. Stanowi to ograniczenie prezentowanych wyników.

LITERATURA

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[2] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[5] D. Połap, A. Jaszcz, N. Wawrzyniak, and G. Zaniewicz, "Bilinear pooling with poisoning detection module for automatic side scan sonar data analysis," *IEEE Access*, vol. 11, pp. 72 477–72 484, 2023.

[6] A. Jaszcz and D. Połap, "Semantic segmentation for moon rock recognition using u-net with pyramid-pooling-based se attention blocks," in *Proceedings of the 16th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART*, INSTICC. SciTePress, 2024, pp. 965–971.