AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

INSTYTUT TELEKOMUNIKACJI

Projekt dyplomowy

Implementacja algorytmu opartego o uczenie maszynowe do rozpoznawania figur szachowych na zdjęciach szachownicy

Implementation of a machine learning-based algorithm for recognizing chess pieces on chessboard images

Autor:	*Bartosz Szymański*
Kierunek studiów:	*Teleinformatyka*
Opiekun pracy:	*dr inż. Jarosław Bułat*

Kraków, 2024

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

Wstęp 3
1. Omówienie wybranych architektur sieci neuronowych 4
1.1. Sieci splotowe 4
1.2. Algorytm YOLO 6
2. Zbiór danych 9
2.1. Opis zbioru danych 9
2.2. Przygotowanie zbioru danych dla YOLO 11
2.3. Przygotowanie zbioru danych dla CNN dla całej szachownicy 12
2.4. Przygotowanie zbioru danych dla CNN dla pojedynczego pola 15
3. Implementacja modeli 16
3.1. Środowisko uczenia 16
3.2. Budowa oraz trenowanie modeli 17
3.3. Normalizacja i przekształcanie danych wejściowych 23
4. Analiza otrzymanych wyników 27
4.1. Wyniki oraz porównanie modeli 27
Podsumowanie 36
Bibliografia 39

Wstęp

Szachy są grą której popularność bardzo wzrosła [1], a technologie z nią związane rozwinęły się na przestrzeni ostatnich lat. Obecnie, z wykorzystaniem silników szachowych i cyfryzacji, analiza partii stała się bardziej zaawansowana oraz oferuje możliwość śledzenia rozgrywki w czasie rzeczywistym. W tym celu stworzone zostały elektroniczne szachownice DGT (ang. *Digital Game Technology*), które pozwalają przenieść fizyczną rozgrywkę na formę cyfrową. Jednakże, takie rozwiązanie jest kosztowne i nie zawsze dostępne dla każdego z graczy, zwłaszcza dla tych początkujących.

Celem tej pracy jest zaprojektowanie algorytmu opartego o uczenie maszynowe, który będzie wykorzystany do analizy zdjęć planszy szachowej i identyfikacji położenia figur. Taka metoda stanowi podstawę do niedrogiej i dostępnej alternatywy dla elektronicznych szachownic DGT. Dodatkowym aspektem tego rozwiązania jest zniwelowanie cyfrowej przewagi, którą oferuje rozgrywka internetowa, dzięki możliwości analizy partii w realnym świecie. Pozwoli to na bezpośrednie konfrontacje z przeciwnikiem, co jest istotnym elementem tradycyjnej rozgrywki w szachy, równocześnie zachowując korzyści, jakie oferuje platforma cyfrowa.

W pierwszym rozdziale omówiono teoretyczne założenia algorytmów uczenia maszynowego w kontekście analizy obrazów. Drugi rozdział poświęcono przygotowaniu danych oraz metod ich przetwarzania. W trzecim rozdziale szczegółowo przedstawiono implementację wykorzystanych modeli, takich jak CNN (ang. *Convolutional neural network*) oraz YOLO (ang. *You Only Look Once*). Czwarty rozdział skupia się na dokładnej analizie otrzymanych wyników i porównania modeli, co pozwoli na wybór optymalnego rozwiązania problemu.

1. Omówienie wybranych architektur sieci neuronowych

W poniższym rozdziale skupiono się na wstępnym wprowadzeniu do wybranych sieci neuronowych, w tym sieci splotowych (CNN) oraz architektury YOLO. Przedstawione zostaną ich podstawowe koncepcje, rozwój oraz zastosowania w różnych dziedzinach, takich jak rozpoznawanie obrazów.

1.1. Sieci splotowe

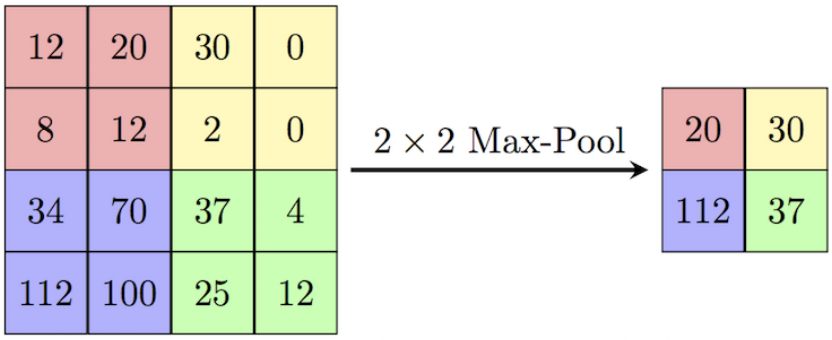
Sieci splotowe nazywane również konwolucyjnymi, to jedne z najbardziej popularnych sieci w dziedzinie przetwarzania obrazów. Posiadają one zdolności do efektywnego wykrywania wzorców i cech bezpośrednio z surowych danych, takich jak piksele obrazów. Obecnie te algorytmy wykorzystywane są w rozpoznawaniu twarzy, autonomicznych pojazdach, samoobsługowych marketach oraz w medycynie. Różnorodność tych zastosowań świadczy o ich wszechstronności i umiejętności rozwiązywania problemów, które kiedyś wydawały się nieosiągalne [2].

Sieci CNN składają się z trzech głównych warstw:

a) Warstw splotowych - używają one zestawu małych, uczących się filtrów (ang. *kernels*), które przesuwają się po obrazie wejściowym. Każdy filtr jest odpowiedzialny za wykrywanie cech w różnych częściach obrazu, tworząc mapę aktywacji, która wskazuje, gdzie dana cecha występuje na obrazie. Dzięki temu CNN mogą wykrywać cechy niezależnie od ich położenia na obrazie [3][4].

b) Warstw łączących - redukują one wymiarowość danych poprzez agregowanie informacji z małych regionów obrazu. Operacje takie jak max-pooling (rysunek 1), która wybiera maksymalną wartość z małego fragmentu obrazu pomagają zmniejszyć liczbę parametrów i obliczeń [3].

c) Warstw w pełni połączonych - w tych warstwach każdy neuron jest połączony ze wszystkimi aktywowanymi neuronami z poprzedniej warstwy, co przypomina strukturę tradycyjnych sieci neuronowych. Te warstwy zwykle znajdują się na końcu architektury CNN i są odpowiedzialne za podejmowanie ostatecznych decyzji, takich jak klasyfikowanie obrazu do określonej kategorii na podstawie wykrytych cech [3].



Rys. 1. Wizualizacja operacji max-pooling [5].

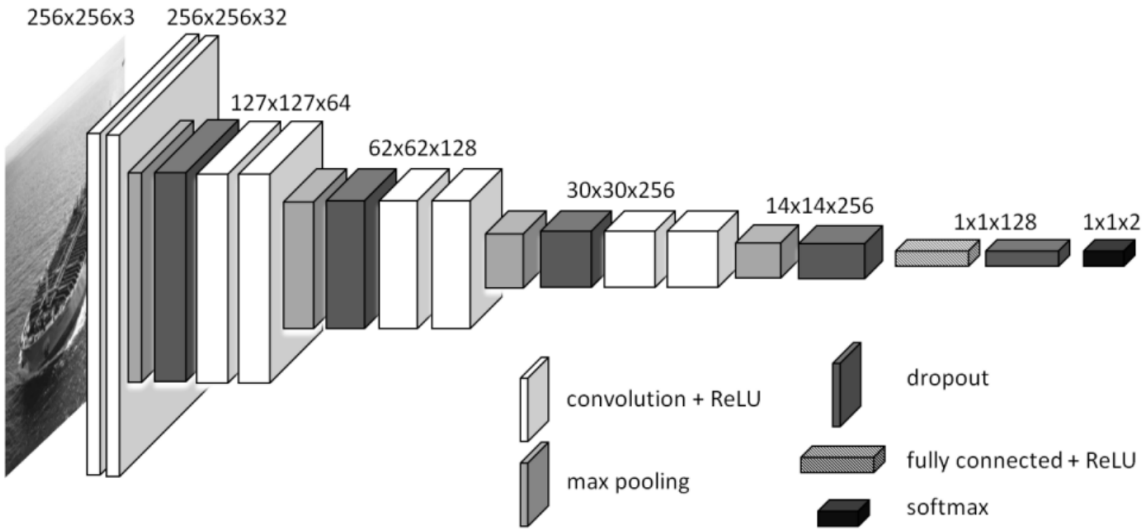
Warstwy sieci mogą być łączone na różne sposoby, dlatego nie ma konkretnego schematu w jakiej kolejności mają być ze sobą połączone. Przykładowa struktura modelu sieci splotowej przedstawiona jest na rysunku 2. Widać na nim, że warstwy zazwyczaj grupowane są w sekwencję trzech głównych typów: warstwy splotowe, po nich warstwy łączące, a na końcu algorytmu warstwy w pełni połączone. Dodatkowo po warstwach łączących i w pełni połączonych można zastosować tzw. warstwę dropout, która zapobiega nadmiernemu dopasowaniu (przeuczeniu). Warstwy splotowe oraz w pełni połączone są powiązane z funkcją aktywacji, ponieważ wprowadza ona nieliniowość do modelu. Skuteczność sieci jest ściśle powiązana z rozwiązywanym problemem, w przypadku bardziej skomplikowanych stosuje się głębsze i bardziej zaawansowane sieci, dla których wraz z ich głębokością rośnie ich trudność uczenia. Owy problem powoduje zanikający gradient (ang. *vanishing gradient*), który pojawia się w przypadku zbyt małych zmian wag sieci oraz eksplodujący gradient (ang. *exploding gradient*), który oznacza bardzo duże zmiany wag [6]. Złożoność i efektywność architektury CNN są zależne od wyboru hiperparametrów którymi są (między innymi):

– liczba i właściwości warstw ukrytych, opisujące ilość i parametry warstw,

– rodzaj operacji na warstwie łączącej, wpływający na sposób połączenia danych między warstwami,

– techniki regularyzacji, pomagające w uniknięciu przeuczenia,

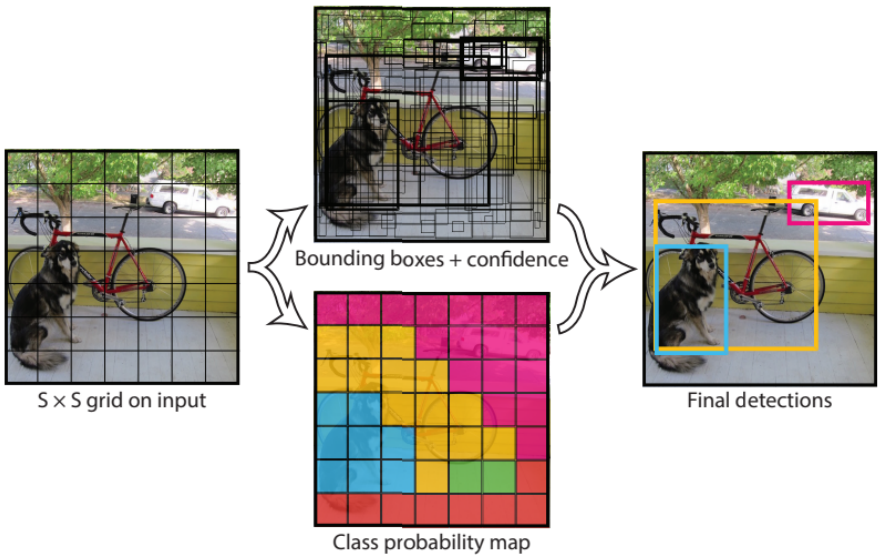
– funkcja kosztu, determinująca metodę oceny dokładności modelu [7][8].



Rys. 2. Przykładowa architektura modelu CNN [9].

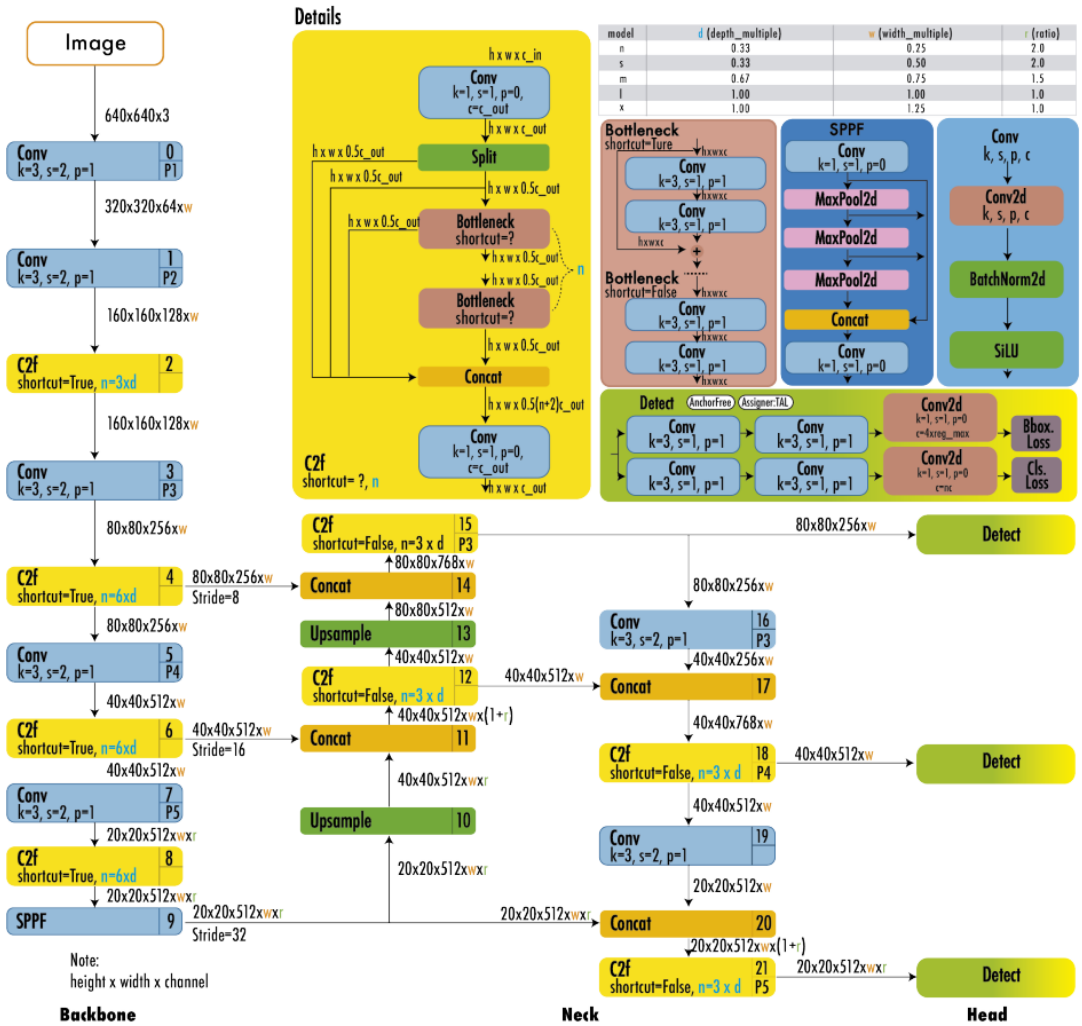
1.2. Algorytm YOLO

YOLO to system wykrywania obiektów w czasie rzeczywistym, który został stworzony przez Josepha Redmona i współpracowników w 2016 roku [10]. Różnił się on od poprzednich dostępnych rozwiązań, ponieważ jak sama nazwa wskazuje analizuje on obraz tylko jednokrotnie. Algorytm działa poprzez podzielenie obrazu wejściowego na siatkę, gdzie każda komórka reprezentuje pewne ograniczające ramy (ang. *bounding boxes*) i prawdopodobieństwa klas [11]. Na rysunku 3 przedstawiono metodę działania algorytmu YOLO.

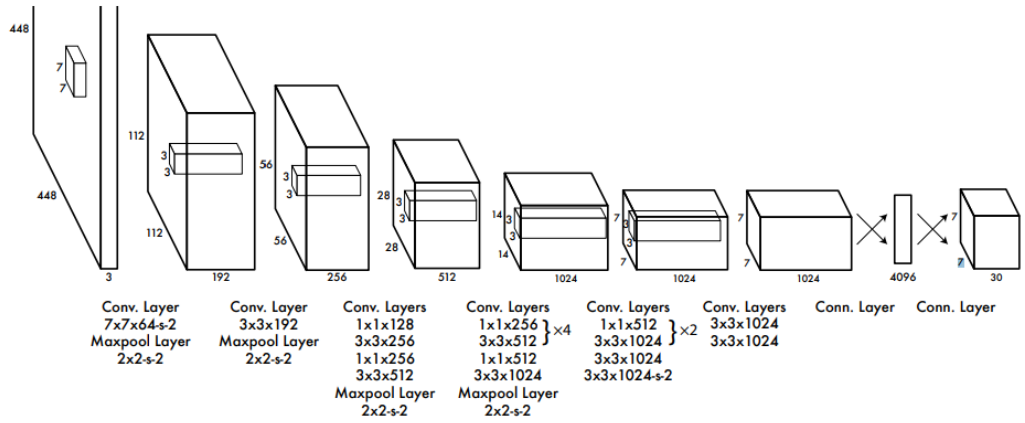


Rys. 3. Metoda działania algorytmu YOLO [10].

YOLOv8 to najnowsza wersja z rodziny algorytmów YOLO, które sukcesywnie się rozwijały pokonując coraz większe ograniczenia oraz nieustannie podnosząc stopień wydajności. Każda nowa wersja YOLO wnosiła poprawki oraz udoskonalenia względem poprzedniej wersji, a co za tym idzie poprawiała ona dokładność i szybkość detekcji obiektów. Najlepszym tego przykładem jest porównanie architektur YOLOv8 (rysunek 4) oraz YOLOv1 (rysunek 5). Widać jak z początkowego, prostszego układu 24 warstw splotowych i dwóch w pełni połączonych warstw YOLOv1, ewoluowano do znacznie bardziej zaawansowanego i złożonego YOLOv8. W tej nowszej wersji zaimplementowano moduły C2f, będące ulepszoną wersją techniki CSP (ang. *Cross Stage Partial Bottleneck*), która wykorzystuje dwie warstwy splotowe. Moduły te łączą cechy z różnych poziomów sieci, zwiększając efektywność i skuteczność modelu [12].



Rys. 4. Architektura modelu YOLOv8 [12].



Rys. 5. Architektura modelu YOLOv1 [10].

Nie jest to jedyny algorytm służący do detekcji obrazów, gdyż równie popularnymi są:

a) R-CNN (ang. *Region-based Convolutional Neural Networks*), który ma swoje ograniczenia, ponieważ korzysta z trzech różnych modeli:

   – CNN do wydobywania cech,

   – liniowego SVM (ang. *Support Vector Machine*) do klasyfikacji obiektów,

   – modelu regresji do tworzenia ramek ograniczających.

   Złożoność ta prowadzi do wydłużenia czasu obróbki, co sprawia, że model ten jest najdłużej przetwarzającym spośród wszystkich wymienionych [13].

b) Fast R-CNN - ulepszony R-CNN, w którym w celu ograniczenia czasu oczekiwania, zmieniono ostatnią warstwę łączącą CNN na warstwę łączącą obszary zainteresowania (ROI), a końcową w pełni połączoną warstwę rozdzielono na dwie części: pierwszą, tzw. gałąź softmax, służącą do klasyfikacji obiektów oraz drugą, określaną jako gałąź regresji ramek ograniczających, odpowiedzialną za lokalizację. W tym ujęciu, obraz wejściowy jest przetwarzany przez CNN do wygenerowania mapy cech, z której następnie wybierane są regiony do analizy [13].

c) SSD (ang. *Single Shot MultiBox Detector*) - wykonuje detekcję w jednym przejściu, eliminując potrzebę osobnego etapu proponowania regionów. Model ten dzieli obraz wejściowy na małe, z góry określone obszary, nazywane “siatkami” i zastosowaniu dla nich tzw. ramek kotwiczących (ang. *anchor boxes*). Te ramki są trenowane, aby reagować na elementy o podobnych rozmiarach i proporcjach. Na końcowym etapie, algorytm przeprowadza klasyfikację oraz estymację prostokątów dla zaproponowanych obiektów. SSD wykazuje szybszą detekcję od R-CNN oraz Fast R-CNN [13][14].

2. Zbiór danych

Ten rozdział koncentruje się na analizie zbioru danych, podkreślając jego znaczenie oraz sposób wykorzystania w trzech różnych modelach:

1. **YOLO:** Ten algorytm wymaga danych zawierających parametry ramek ograniczających. Są one kluczowe dla efektywnego lokalizowania i identyfikowania obiektów w ramach przetwarzanych obrazów.

2. **CNN dla całej szachownicy:** W tym modelu konieczne jest posiadanie danych, które nie tylko zawierają informacje o klasie obiektów na każdym z 64 pól, ale także zachowują dokładną kolejność tych pól na szachownicy. Ta kolejność jest istotna, aby model mógł poprawnie interpretować układ planszy.

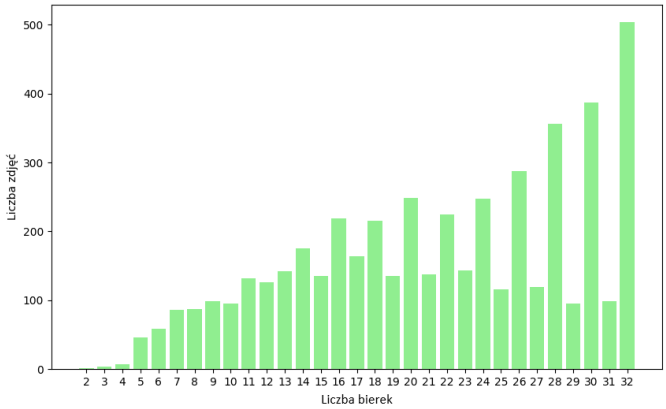
3. **CNN dla pojedynczego pola planszy:** W tym podejściu, zdjęcie jest dzielone na pojedyncze pola, a każde z nich jest analizowane osobno. Wymagane jest dokładne oznaczenie klasy obiektu znajdującego się na każdym z tych pól.

2.1. Opis zbioru danych

Obrazy oraz etykiety pozyskano z bazy [15]. Zbiór ten zawiera 4888 syntetycznych obrazów stanu partii szachowej, które zostały wygenerowane na podstawie 2851 partii rozegranych przez Magnusa Carlsena [16]. Obrazy zostały wyrenderowane w programie Blender[17] pod różnymi kątami i w różnych warunkach oświetleniowych.

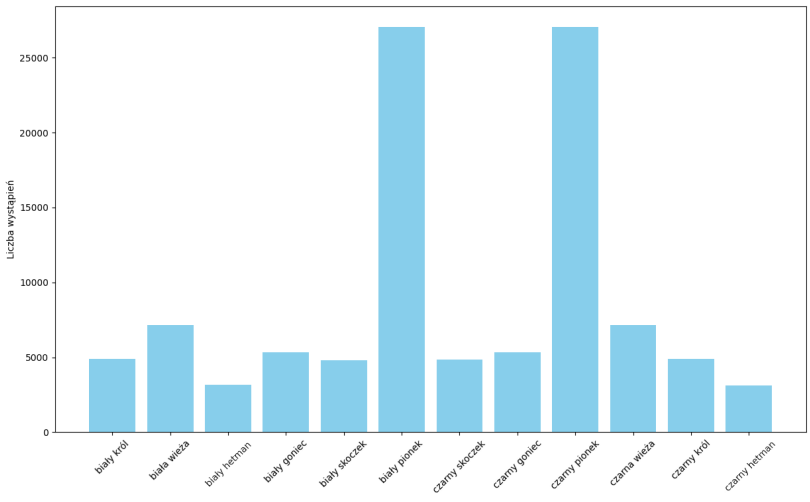
Do początkowego zrozumienia i interpretacji zbioru danych bardzo ważny był wykres, który dostarcza wizualnej reprezentacji liczebności bierek na zdjęciu (rysunek 6). Słowo “bierek” odnosi się do dwóch kategorii elementów używanych w grze: “figur” i “pionów”. Piony to mniejsze bierki, zajmujące pierwszy rząd na początku planszy. Figury natomiast to większe bierki, takie jak król, hetman, wieże, gońce, skoczki. Na wykresie przedstawionym na rysunku 6 widać czy występują jakieś szczególne trendy, na przykład czy częściej pojawiają się obrazy z większą liczbą bierek, co może sugerować etapy początkowe oraz środkowe partii szachowych.

Dodatkowo, widać na nim przewagę parzystych liczb bierek w początkowych etapach, ponieważ często są one wymieniane parami. Na przykład, gdy gracz traci piona często jest w stanie od razu zbić piona przeciwnika, co powoduje utrzymanie się parzystej liczby bierek.



Rys. 6. Rozkład liczby zdjęć względem ilości występowania bierek na planszy.

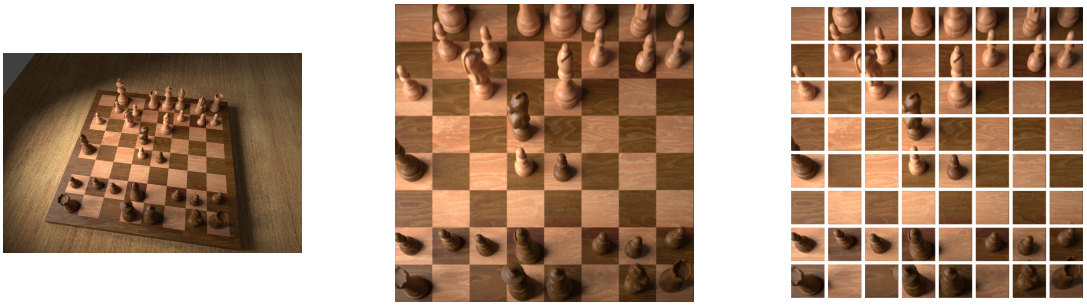
Drugi wykres znajdujący się na rysunku 7 reprezentuje liczebność każdej z bierek w całym zbiorze danych. Pomaga on w interpretacji, czy wszystkie bierki występują wystarczającą ilość razy, czyli czy zbiór treningowy jest zbalansowany. Nadprezentowanie lub niewystarczająca liczba próbek każdej z klas mogłoby wpływać na jakość modeli uzyskanych w trakcie trenowania.



Rys. 7. Wykres liczby wystąpień każdej bierki na szachownicy.

Obrazy w zbiorze wymagały przycięcia (normalizacji) tak, aby ograniczały się one do rogów szachownicy. Takie podejście pozwoliło uzyskać prawidłową predykcję pozycji bierki.

Oryginalne etykiety zawierały wystarczającą ilość danych, takich jak ramki ograniczające dla obiektów oraz informacje o figurach szachowych umieszczonych na konkretnych polach. Dzięki temu możliwe było dostosowanie etykiet danych do specyficznych wymagań oraz formatów trenowanych modeli. Proces polegał na przekształceniu danych etykietujących w taki sposób, aby były one zgodne z oczekiwaniami algorytmu, co obejmowało zarówno modyfikację struktury danych, jak i ewentualne przekształcenie kategorii etykiet. Cel tego działania to zapewnienie spójności procesu uczenia, co ma bezpośredni wpływ na jakość i wiarygodność wyników uzyskanych z modelu. Na rysunku 8 przedstawiono przykładowe zdjęcie po przetworzeniu dla danego modelu.



Rys. 8. Od lewej: Oryginalny obraz, wejściowy obraz dla modeli YOLO oraz CNN dla całej szachownicy, wejściowe obrazy dla modelu CNN dla pojedynczego pola.

2.2. Przygotowanie zbioru danych dla YOLO

Przygotowanie zbioru danych wymagało zmiany formatu etykiet na odpowiedni dla modeli YOLO. Fragment etykiety jest pokazany na listing 2.1.

Listing 2.1. Etykieta dla modelu YOLO.

```
11 0.0625 0.93515625 0.125 0.1328125
7 0.41171875 0.92890625 0.1296875 0.1453125
```

W każdym z wierszy liczby te oznaczają kolejno: klasę, pozycję poziomą (oś X), pozycję pionową (oś Y), szerokość ramki ograniczającej, wysokość ramki ograniczającej. Wszystkie te wartości są proporcjonalne, co oznacza, że są one stosowane w kontekście, w którym wymiary są skalowane w zależności od rozmiarów obrazu.

Przy użyciu platformy i biblioteki Roboflow[18], która upraszcza i przyspiesza proces obróbki zbioru danych, podzielono zbiór w stosunku 74% dla zbioru treningowego, 13% dla zbioru walidacyjnego i 13% dla zbioru testowego. Dodatkowo zastosowano 3-krotną augumentacje dla każdego z przykładu zbioru treningowego. Wykorzystano następujące wzbogacenia obrazu:

– odbicie horyzontalne lub wertykalne,

– obrót obrazu o $90°$ w możliwości: zgodnie z ruchem wskazówek zegara, przeciwnie do ruchu wskazówek zegara lub obrót o $180°$,

– obrotu w zakresie od - $-2°$ do $2°$,

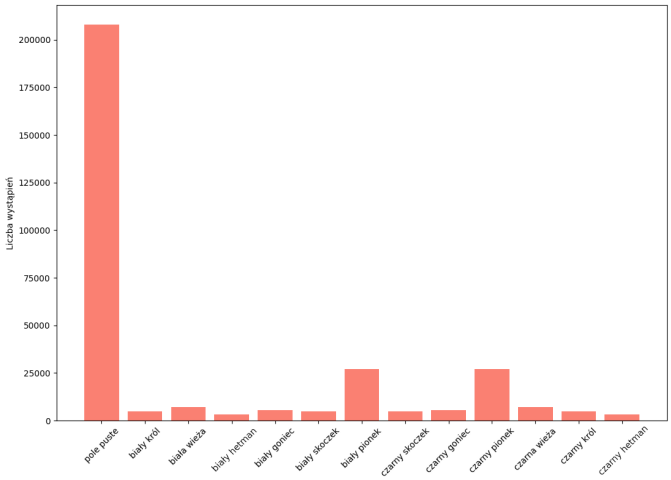
– nasycenie kolorów w zakresie od -18% do 18%,

– jasność obrazu w zakresie od -35% do 35%,

– ekspozycję w zakresie od -5% do 5%.

2.3. Przygotowanie zbioru danych dla CNN dla całej szachownicy

Ten zbiór będzie wykorzystywany do treningu modelu podobnie jak w przypadku algorytmu YOLO, jednak istotna różnica polega na sposobie etykietyzacji danych wejściowych, gdyż model będzie dawał wynik dla każdego z pól szachownicy. Oznacza to, że na wyjściu oprócz bierek możliwe jest również puste pole. Na rysunku 9 przedstawiono wykres liczby wystąpień po zmianie etykiet oraz widać jego główny problem - znaczącą dominacje pustych pól. Jak wiadomo, na początku każdej partii szachowej, zanim wykonane zostaną jakiekolwiek ruchy, puste pola stanowią dokładnie połowę wszystkich pól szachownicy (32 z 64 możliwych). Jest to unikalna cecha danych w analizie całego zdjęcia szachownicy, gdzie puste pola są tak samo ważnym elementem analizy, jak bierki.



Rys. 9. Wykres liczby wystąpień każdej bierki na szachownicy z uwzględnieniem pustych miejsc.

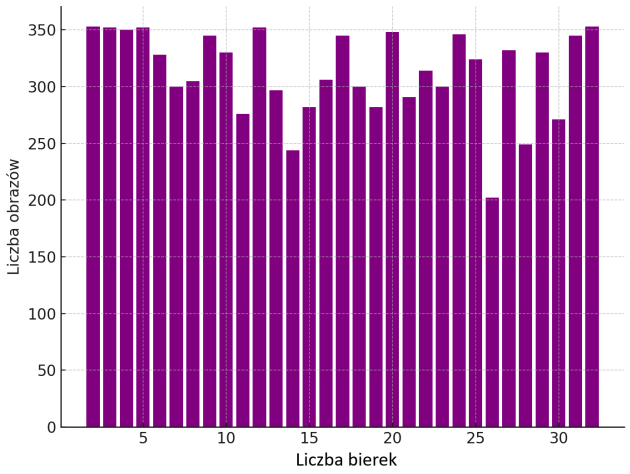
Bez jednolitego systemu reprezentacji, analiza wzorców rozmieszczenia pustych pól i bierek mogłaby być znacznie utrudniona. Dzięki standaryzacji, gdzie pole H1 zawsze znajduje się w lewym górnym rogu, a A8 w prawym dolnym, algorytm będzie jednoznacznie interpretować i analizować rozmieszczenie bierek i pustych pól na planszy. Przykład takiego ustandaryzowanego systemu reprezentacji użytego w etykietowaniu danych znajduje się w listingu 2.2. Ten sposób etykietowania jest bardzo ważny dla modelu, ponieważ zapewnia że nie dojdzie do sytuacji, w której etykieta nie odzwierciedla rzeczywistych pozycji bierek na planszy szachowej.

Listing 2.2. Etykieta dla modelu CNN.

```
    "h1": "puste_pole",
    "g1": "K",
    "f1": "puste_pole",
    "e1": "R",
    "d1": "Q",
    "c1": "B",
    "b1": "N",
    "a1": "R",
    "h2": "puste_pole",
    "g2": "P",
    "f2": "P",
```

Aby zminimalizować ryzyko przeuczenia modelu oraz poprawić jego zdolność do generalizacji, zastosowano technikę augmentacji danych, polegającą na nadpróbkowaniu tych obrazów, na których ilość bierek była reprezentowana w niewystarczającej liczbie. Dzięki temu zbiór danych stał się bardziej zrównoważony i jest on około 3% dokładniejszy oraz o mniej więcej 10% wykazuje lepszy wynik w uśrednionej dokładności klas, niż koncepcyjny zbiór, w którym zwiększano występowanie tylko zdjęć z początkowych etapów partii.

Wykres słupkowy przedstawiony na rysunku 10 pokazuje rezultaty tego procesu. Każdy słupek odpowiada liczbie obrazów w danej kategorii po augmentacji, co pozwala zaobserwować, że różnice między klasami zostały zniwelowane, dążąc do równomiernego rozkładu danych.



Rys. 10. Rozkład ilości względem liczby bierek na planszy zbioru treningowego (po zastosowaniu augumentacji).

Kluczowym aspektem zbioru danych jest także równomierne rozłożenie etapów gry, tak aby każdy podzbiór zawierał zdjęcia, na których bierki są częściowo zasłonięte przez inne, a także takie, na których widoczne są kształty bierek bez dodatkowych szumów.

W celu osiągnięcia tego zastosowano skrypt znajdujący się w listingu 2.3, który za pomocą funkcji `split_dataset`, przetwarza katalogi zawierające obrazy reprezentujące poszczególne kategorie. Skrypt ten wykonuje podział danych na zbiory treningowe, walidacyjne i testowe według określonych proporcji, jednocześnie stosując augmentację danych z różną intensywnością dla każdej kategorii. Zmienna `aug_values` zawiera liczby określające, ile razy każda kategoria obrazów ma być nadpróbkowana.

Listing 2.3. Zrównoważenie etapów rozgrywki.

```
aug_values = [352, 175, 69, 10, 7, 4, 4, 4, 4, 2, 3, 2, 1, 2, 1, 2, 1, 2,
    1, 2, 1, 2, 1, 3, 0, 3, 0, 4, 0, 4, 0]
for i, aug_value in zip(range(0, 31), aug_values):
    split_dataset(f"./blender_dataset/{2+i}", train_dir, val_dir, test_dir,
        train_pct=0.8, val_pct=0.10, aug_range=aug_value, random_seed=42)
```

Przebieg działania skryptu można przedstawić krok po kroku:

1. Wybierana jest kategoria obrazów (klasa) według iteracji w pętli.
2. Dla każdej klasy, określana jest liczba augmentacji z tablicy `aug_values`.
3. Funkcja `split_dataset` przydziela obrazy do zbiorów treningowego, walidacyjnego i testowego z zachowaniem odpowiednio proporcji 80%, 10%, 10%.

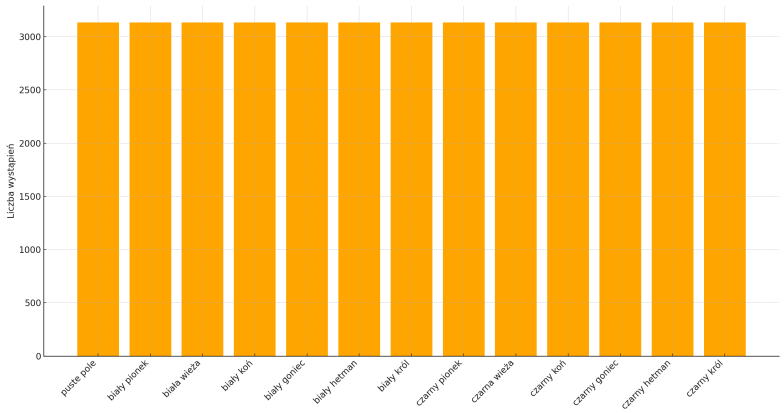
4. W procesie dzielenia, obrazy są augmentowane (nadpróbkowane) według wartości z `aug_values` dla danej klasy.

5. Operacje te są powtarzane dla każdej klasy (odpowiadającej katalogowi z obrazami), co zapewnia równomierne rozłożenie danych w zbiorze.

Dzięki takiemu podejściu, model uczący się na tych danych jest w stanie lepiej rozpoznawać i interpretować różne konfiguracje planszy, które mogą pojawić się w trakcie rzeczywistej partii szachów.

2.4. Przygotowanie zbioru danych dla CNN dla pojedynczego pola

Dla tego modelu wymagane było rozdzielenie każdego z pól na każdym zdjęciu szachownicy, co zwiększyło liczbę próbek do 312 832. Ponadto takie rozwiązanie pozwalało na większą swobodę odnośnie podpróbkowania zbioru danych. Zastosowano tę technikę obróbki danych, ponieważ po podziale szachownicy klasa `puste pole` wynosiła około 180 tysięcy przypadków, a klasa `czarny hetman` 3133. To znaczące przeważenie jednej klasy nad innymi mogłoby prowadzić do problemu przeuczenia modelu. Podpróbkowanie pozwoliło na uniknięcie tego zagrożenia, ponieważ umożliwiało skupienie nauki modelu na najbardziej istotnych przypadkach, a nie na sztucznie utworzonych lub nadreprezentowanych danych. Dlatego liczba danych została zmniejszona do najmniejszej wartości występowania pojedynczej klasy, czyli do 3133 dla klasy `czarny hetman`. Liczba ta została uznana za wystarczającą do efektywnego przeprowadzenia nauki modelu. W rezultacie, występowanie pozostałych klas zostało zrównoważone dla tej wartości poprzez wybór losowych próbek z danej klasy. Po tej operacji, zbiór danych został podzielony w proporcjach: 74% na trening, 13% na walidację i 13% na testowanie.



Rys. 11. Liczby wystąpień każdej bierki na szachownicy po podpróbkowaniu.

3. Implementacja modeli

W rozdziale dotyczącym implementacji skoncentrowano się na szczegółowym opisie implementacji i trenowania modeli. Zaprojektowano dwa modele sieci splotowej: CNN dla całej szachownicy i CNN dla pojedynczego pola szachownicy oraz przy pomocy metody dotrenowania zaimplementowano algorytm YOLO. Przedstawiono budowę każdej z nich, sposób działania oraz wykorzystane hiperparametry trenowania.

3.1. Środowisko uczenia

Podczas implementacji korzystano zarówno ze środowisk lokalnych, jak i chmury obliczeniowej Kaggle i Google Colab. Kluczowym elementem każdego z tych środowisk była karta graficzna NVIDIA T4 posiadająca 16 GB pamięci GDDR6. Ten komponent używany był na każdej z wymienionych platform zapewniając tą samą wydajność, przez co nie było konieczności edycji hiperparametrów z uwagi na przykładowe zmniejszenie pamięci VRAM. Obok środowisk chmurowych, korzystano również z lokalnego komputera o parametrach:

- procesor: Intel i7-4790,
- RAM: 16GB 1600MHz,
- karta graficzna: NVIDIA GTX 970 4GB DDR5.

Chociaż ma ono znacznie mniejszą moc obliczeniową, niż środowiska chmurowe, lokalne środowisko pozwoliło na szybkie prototypowanie i testowanie początkowych koncepcji modeli.

W całym procesie implementacji zdecydowano się użyć języka Python oraz środowiska PyTorch[19]. Python jest szeroko stosowany w dziedzinie nauki o danych ze względu na swoją czytelność, bogaty zestaw bibliotek oraz wsparcie społeczności. Ponadto jest on wspierany przez Jupyter Notebook, w którym była przetwarzana cała implementacja. PyTorch z kolei oferuje elastyczność i intuicyjny interfejs, co ułatwia eksperymentowanie z modelami.

3.2. Budowa oraz trenowanie modeli

Model YOLO

Model ten obsługuje na wejściu zdjęcia całej szachownicy oraz jako jedyny był uczony metodą "douczenia" (ang. *transfer learning*). Oznacza to, że algorytm wcześniej nauczony na jednym zestawie danych, jest ponownie trenowany na innym, specyficznym zestawie danych. Pozwala to modelowi lepiej dostosować się do nowych zadań, wykorzystując wcześniej nabyte umiejętności. Jednakże, taki proces wiążę się z ryzykiem utraty wiedzy ogólnej (ang. *catastrophic forgetting*), gdzie model traci zdolności do wykonywania zadań, dla których został pierwotnie wytrenowany [20]. W przypadku modelu, który jest przystosowany tylko do rozpoznawania bierek szachowych, nie stanowi to problemu. W tym celu wykorzystano bibliotekę *ultralytics*[21], która odpowiada za dystrybucje modeli YOLO. Architektura modelu YOLOv8s znajduję się na rysunku 12, na którym widać 225 warstw i ponad 111 milionów parametrów, wykazując złożoność obliczeniową na poziomie 28.7 GFLOPs, co świadczy o jego potencjale do efektywnej detekcji obiektów przy znaczących wymaganiach obliczeniowych. Podczas uczenia wykorzystywano 11.6 GB pamięci karty graficznej na 16 GB dostępnych.

	from	n	params	module	arguments
0	-1	1	928	ultralytics.nn.modules.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.Conv	[32, 64, 3, 2]
2	-1	1	29056	ultralytics.nn.modules.C2f	[64, 64, 1, True]
3	-1	1	73984	ultralytics.nn.modules.Conv	[64, 128, 3, 2]
4	-1	2	197632	ultralytics.nn.modules.C2f	[128, 128, 2, True]
5	-1	1	295424	ultralytics.nn.modules.Conv	[128, 256, 3, 2]
6	-1	2	788480	ultralytics.nn.modules.C2f	[256, 256, 2, True]
7	-1	1	1180672	ultralytics.nn.modules.Conv	[256, 512, 3, 2]
8	-1	1	1838080	ultralytics.nn.modules.C2f	[512, 512, 1, True]
9	-1	1	656896	ultralytics.nn.modules.SPPF	[512, 512, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.Concat	[1]
12	-1	1	591360	ultralytics.nn.modules.C2f	[768, 256, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.Concat	[1]
15	-1	1	148224	ultralytics.nn.modules.C2f	[384, 128, 1]
16	-1	1	147712	ultralytics.nn.modules.Conv	[128, 128, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.Concat	[1]
18	-1	1	493056	ultralytics.nn.modules.C2f	[384, 256, 1]
19	-1	1	590336	ultralytics.nn.modules.Conv	[256, 256, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.Concat	[1]
21	-1	1	1969152	ultralytics.nn.modules.C2f	[768, 512, 1]
22	[15, 18, 21]	1	2120692	ultralytics.nn.modules.Detect	[12, [128, 256, 512]]

Model summary: 225 layers, 11140244 parameters, 11140228 gradients, 28.7 GFLOPs

Rys. 12. Architektura modelu YOLOv8s.

Proces trenowania tego algorytmu uproszczony jest do jednej linijki, którą zaprezentowano w listingu 3.1.

Listing 3.1. Kod do nauki modelu YOLO.

```
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.
    yaml epochs=28 imgsz=640 plots=True
```

Parametry w nim użyte oznaczają:

– `task` - określa zadanie do wykonania, w tym przypadku jest to `detect`, co wskazuje, że celem jest detekcja obiektów,

– `mode` - określa tryb pracy. Wartość `train` wskazuje, że model trenowano na podstawie dostarczonych danych,

– `model` - wskazuje na model, który ma być zastosowany. W tym przypadku użyto `yolov8s.pt`,

– `data` - ścieżka do pliku YAML, który zawiera konfigurację danych treningowych i walidacyjnych,

– `epochs` - określa liczbę epok treningowych,

– `imgsz` - rozmiar obrazu (w pikselach) używany podczas treningu. W tym przypadku obrazy skalowano do rozmiaru 640x640 pikseli,

– `plots` - ta opcja wskazuje, że po zakończeniu treningu wygenerowano wykresy pokazujące metryki treningu, takie jak krzywe straty i dokładności.

Pozostałe użyte hiperparametry podczas douczania algorytmu to:

– wielkość partii o wartości 16,

– kryterium SGD (ang. *Stochastic Gradient Descent*),

– współczynnik uczenia ustawiono na 0.01,

– regularyzacja L2 o wartości 0.001 dla 64 parametrów wag.

Nie została zamrożona żadna z warstw wzorując się na dokumentacji *ultralytics*, w której w artykule pt.“Transfer Learning with Frozen Layers” pokazano jak model YOLO bez zamrożonych warstw osiąga najlepsze wyniki, ale nauka tego modelu trwa najdłużej.

Model CNN dla całej szachownicy

Zaprojektowana architektura algorytmu (rysunek 13) do identyfikacji bierek na całej planszy szachowej składa się z 26 warstw. Obraz wejściowy ma rozmiar 320x320 pikseli, dzięki czemu otrzymano liczbę 8 076 608 parametrów. Wykorzystanie 11 warstw regularyzacyjnych ma na celu zapobieganie przeuczeniu się modelu podczas jego implementacji. Struktura rozpoczyna się od warstw z relatywnie mniejszą liczbą neuronów, co pozwala na wstępne przetwarzanie i

rozpoznawanie prostych cech, takich jak krawędzie czy wierzchołki. Następnie, w miarę przechodzenia do głębszych warstw, liczba neuronów wzrasta, umożliwiając coraz bardziej złożone reprezentacje danych, taki projekt struktury pozwala na płynne przejście od prostych do skomplikowanych cech. Każda z warstw splotowych jest uzupełniona o:

a) Warstwę normalizacji wsadowej, która pozwala modelowi działać stabilniej. Jej wartość dobrano na podstawie ilości neuronów wyjściowych z warstwy splotowej.

b) Funkcję aktywacji ReLU wprowadzającą nieliniowości do modelu.

c) Warstwę łączącą z użyciem MaxPool, która pozwala na koncentrację na najistotniejszych cechach, jednocześnie redukując ryzyko przeuczenia oraz ilość danych przetwarzanych przez sieć.

d) Warstwę dropout wynoszącą w początkowych warstwach 0.2, a w późniejszych 0.25. Technika ta jest kolejnym elementem regularyzacji zwiększając zdolność modelu do generalizacji. Polega ona na losowym dezaktywowaniu neuronów tylko podczas treningu algorytmu.

Po przejściu przez wszystkie warstwy splotowe następuje spłaszczenie (textttFlatten) do jednowymiarowego wektora, aby zastosować warstwę w pełni połączoną. Na wyjściu z modelu otrzymano wektor o długości 832 neuronów, który został za pomocą funkcji `shape(3)` podzielony na 64 segmenty po 13 elementów każdy. Każdy element odpowiada prawdopodobieństwu wystąpienia jednej z 13 klas, a segmenty oznaczają określone pole szachowe. Liczba 3 w funkcji `shape(...)` wynika z trójwymiarowości tensora, który został świadomie zaprojektowany na podstawie trzech kluczowych parametrów i ich kolejności:

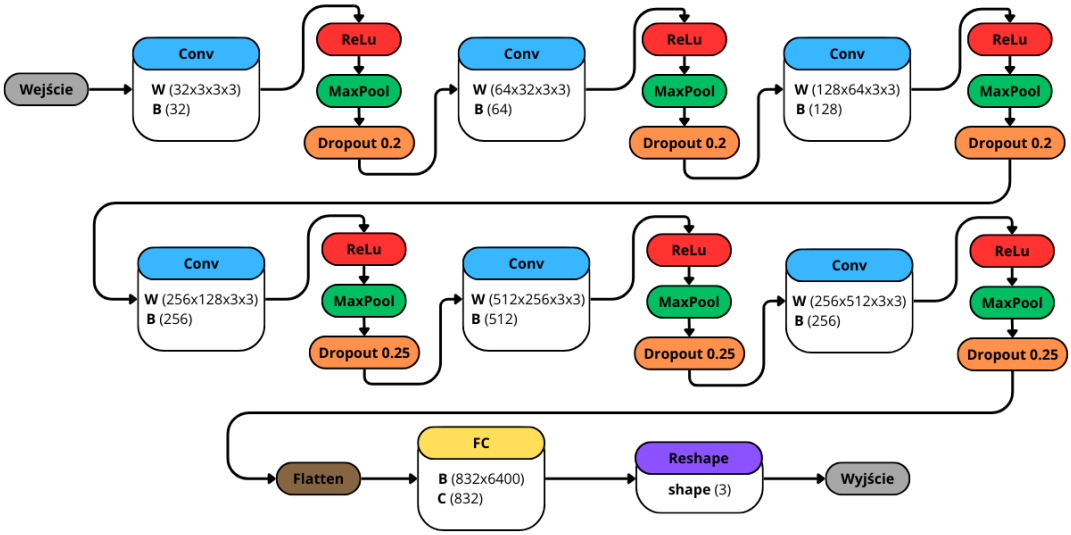
– wielkości partii danych, reprezentowanej jako pierwszy wymiar,

– iczby pól na szachownicy, stanowiącej drugi wymiar,

– ilości różnych klas, które mogą być przypisane do każdego pola szachownicy, będącej trzecim wymiarem.

Tak więc, kształt tensora jest zdefiniowany jako [-1, 64, 13], gdzie '-1' reprezentuje nieokreśloną wielkość partii, '64' odnosi się do pól na szachownicy, a '13' do możliwych klas. Pozwala to na szczegółowe rozróżnienie nie tylko rodzajów figur szachowych, ale również ich położenia na planszy. Na rysunku 13 każda z warstw splotowych posiada oznaczenia, które odnoszą się odpowiednio 'W' do wag (ang. *weigths*) oraz 'B' do obciążeń (ang. *biases*). Wagi są parametrami filtra splotowego, a obciążenia pozwalają na dostosowanie progu aktywacji w warstwie. Na przykład pierwsza warstwa ma 32 filtry (W), każdy o rozmiarze 3x3x3 oraz 32 wartości obciążeń (B). W przypadku warstw w pełni połączonych literka 'B' również oznacza obciążenia,

które są dodawane do ważonej sumy wejść. Natomiast 'C' odnosi się do liczby neuronów w tej warstwie, która na prezentowanej architekturze wynosi 832.



Rys. 13. Architektura modelu CNN dla całej szachownicy.

Przykładowy fragment zastosowanej pętli uczącej znajduje się w listingu 3.2. Ważne, aby podczas nauki obserwować wyniki znajdujące się w zmiennej `train_class_accuracy`, która pokazuje jak algorytm radzi sobie z każdą z klas. Widać na niej jakich hiperparametrów użyto:

a) Liczby epok ustaloną na 28 co oznacza, że cały zbiór danych przetwarzany jest przez model 28 razy w procesie uczenia.

b) Wielkość partii (ang. *batch size*) wynoszącą 16 wskazuje ona, że w każdej iteracji sieć przetwarza grupę 16 zdjęć, co pozwala na efektywniejsze i stabilniejsze uczenie.

c) Współczynnika uczenia na poziomie 0.0001 parametr ten reguluje szybkość aktualizacji wag sieci w trakcie treningu, zapobiegając skrajnym zmianom.

d) Wartości regularyzacji L2 ustalonej na 0.001 dla wag (bez obciążeń) na wszystkich warstwach modelu, pomaga zapobiegać przeuczeniu modelu poprzez nakładanie kar na zbyt duże wagi.

e) Optymalizatora Adam, który aktualizuje wagi sieci na podstawie aktualnego gradientu oraz poprzednich gradientów.

f) Kryterium CrossEntropyLoss, służąc jako funkcja straty, która odpowiada za mierzenie różnicy między przewidywaniami modelu a rzeczywistymi etykietami. Jest szczególnie przydatna, gdy modele generują logity, które są następnie przekształcane w rozkłady

prawdopodobieństwa. W PyTorch te kryterium automatycznie stosuje funkcję softmax do logitów i oblicza negatywną logarytmiczną prawdopodobieństwa dla rzeczywistej klasy. Strata jest obliczana następująco:

$$H(y, \hat{y}) = -\frac{1}{N}\sum_{n=1}^{N} \log\left(\frac{\exp(\hat{y}_{n,y_n})}{\sum_{j=1}^{C}\exp(\hat{y}_{n,j})}\right)$$

gdzie:

- N to liczba próbek w zestawie danych.
- y_n to indeks rzeczywistej klasy dla n-tej próbki.
- $\hat{y}_{n,j}$ to logit dla klasy j dla n-tej próbki.
- C to liczba klas.

Listing 3.2. Fragment pętli uczącej Pytorch.

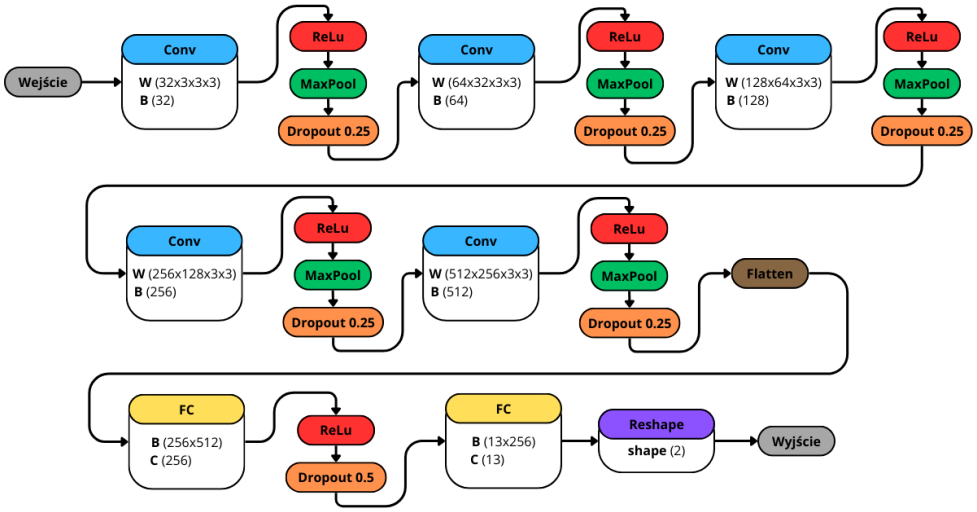
```
num_epochs = 28
lr = 0.0001
lambda_l2 = 1e-3
weight_list = [p for n, p in model.named_parameters() if 'weight' in n]
bias_list = [p for n, p in model.named_parameters() if 'bias' in n]
optimizer = optim.Adam([
    {'params': weight_list, 'weight_decay': lambda_l2},
    {'params': bias_list, 'weight_decay': 0.0}], lr=lr)
train_losses, train_accuracies = [], []
criterion = nn.CrossEntropyLoss()
for epoch in tqdm(range(num_epochs)):
    model.train()
    for images, labels in tqdm(train_data_loader, leave=False):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()

        outputs = model(images)
        loss = criterion(outputs.view(-1, 13), labels.view(-1))
        loss.backward()
        optimizer.step()

        predicted_classes = torch.argmax(outputs, dim=2)
        train_running_loss += loss.item()
        train_correct += (predicted_classes == labels).sum().item()
        train_total += labels.size(0) * 64
        for i in range(batch_size):
            for j in range(64):
                label = labels[i, j]
                train_class_correct[label] += c[i, j].item()
                train_class_total[label] += 1
    train_class_accuracy = [correct / total * 100 for correct, total in zip
        (train_class_correct, train_class_total)]
    train_accuracies.append((train_correct / train_total)*100)
    train_losses.append(train_running_loss / len(train_data_loader))
```

Model CNN dla pojedynczego pola

Koncepcja tego modelu wyłoniła się przy problemie z podpróbkowaniem danych podczas implementacji modelu dla całej szachownicy i znalezieniu podejścia opracowanego przez Karię i współautorów [22]. Zaprojektowana architektura znajdująca się na rysunku 14 w dużej mierze pokrywa się ona z modelem CNN dla całej szachownicy. Jednakże, ten algorytm będzie rozpoznawał bierki oraz puste pole na pojedynczym polu. Ich położenia nie będzie obsługiwał. Z uwagi na założenie, że zdjęcie planszy szachowej zostanie podzielone na 64 pola następuje wywołanie predykcji implementowanego algorymtu dla każdego z pól szachowych. Zaprojektowana struktura składa się z 24 warstw, a rozmiar zdjęć na wejściu to 50x50 pikseli, co przekłada się na 1 705 229 parametrów. Występują w niej, w odróżnieniu od modelu CNN dla całej szachownicy dwie warstwy w pełni połączone. Pierwsza z nich uzupełniona jest o funkcję aktywacji ReLU oraz występuje po niej warstwa Dropout. Jej wyższa wartość równa 0.5, wynika z potrzeby dezaktywowania większej ilości neuronów pomiędzy pierwszą, a drugą warstwą w pełni połączoną.



Rys. 14. Architektura modelu CNN dla pojedynczego pola.

Podczas implementacji najlepszy wynik udało się uzyskać przy pomocy następujących hiperparametrów:

- liczba epok: 30,
- wielkość serii na wejściu: 16,
- rozmiar wejściowy: 50x50,
- współczynnik uczenia: 0.00005,

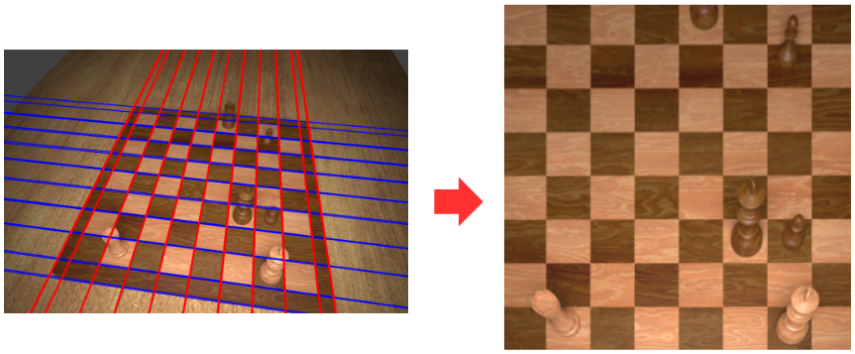
– optymalizator: Adam,

– kryterium: CrossEntropyLoss.

3.3. Normalizacja i przekształcanie danych wejściowych

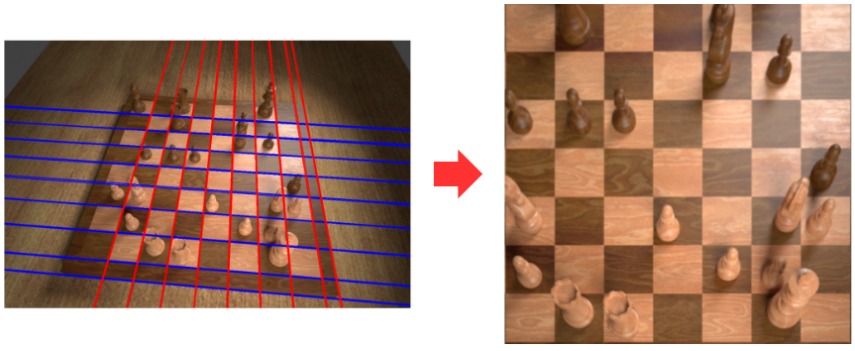
System wykorzystujący techniki przetwarzania obrazu

Ten system opiera się na 6 etapach przetwarzania obrazu, aby na wejściu modelu znajdowało się odpowiednie zdjęcie do interpretacji. Podczas projektowania wzorowano się na rezultatach pracy autorstwa Athanasios'a Masouris'a [23]. Najważniejszą częścią jest prawidłowe znalezienie przecięć bocznych planszy szachowej (rysunek 15), ponieważ to od nich zależy późniejsza predykcja.



Rys. 15. Prawidłowo wykryta szachownica.

Przykładowe błędne rozpoznanie szachownicy pokazane jest na rysunku 16, na którym zamiast standardowej szachownicy 8x8, wykryto szachownicę 7x7.



Rys. 16. Błędnie wykryta szachownica.

W celu osiągnięcia skutecznego wykrycia planszy wykorzystano sprawdzone metody detekcji krawędzi, co zostało zilustrowane na rysunku 17 oraz opisano ich działanie poniżej.

1. **Krok 1:** Zmiana koloru na czarno-biały, z powodu tego iż krawędzie są zdefiniowane przez zmiany intensywności, a nie przez zmiany koloru. Przetwarzanie obrazu w odcieniach szarości pozwala skoncentrować się na strukturze obrazu bez rozpraszania uwagi przez kolor.

2. **Krok 2:** Rozmycie gaussowskie, które redukuje szumy i nieistotne detale, co przekłada się na lepszą interpretację i analizę obrazu.

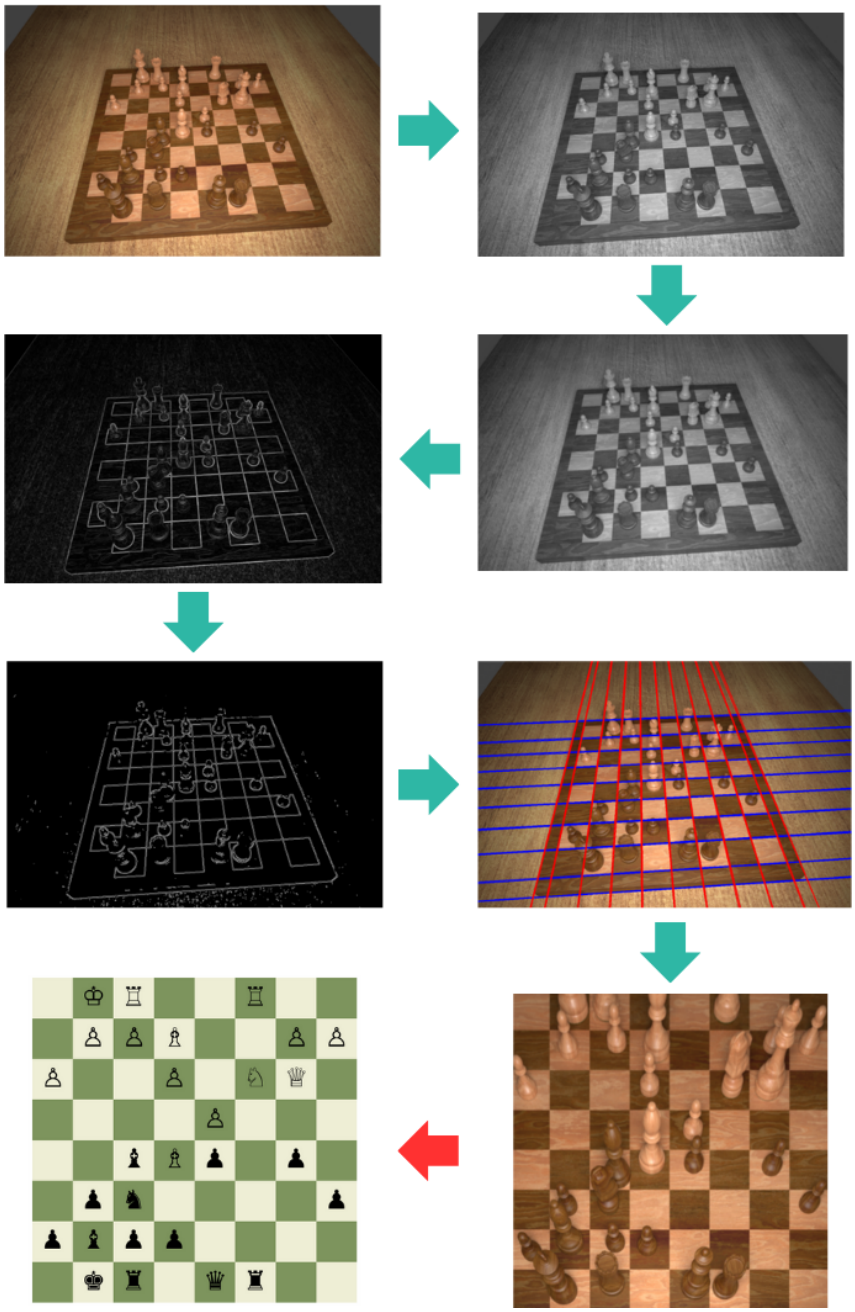
3. **Krok 3:** Operator Sobela jest stosowany do wykrywania krawędzi obrazu poprzez obliczanie gradientu intensywności pikseli. Zastosowano dwa filtry Sobela, jeden do wykrywania krawędzi w poziomie, a drugi w pionie. Stosowane są one do uzyskania wyodrębnienia krawędzi.

4. **Krok 4:** Detektor Cannego to technika wykrywania krawędzi, skuteczna w identyfikacji cienkich linii oraz eliminacji szumów.

5. **Krok 5:** Wykrywanie i grupowanie linii - za pomocą transformacji Hougha dla linii, wykrywane są linie, które są następnie klasyfikowane jako poziome lub pionowe.

6. **Krok 6:** Wyszukiwanie przecięć linii - identyfikacja potencjalnych narożników, gdzie linie się przecinają.

7. **Krok 7:** Wyznaczenie narożników oraz transformacja perspektywiczna z ich wykorzystaniem.

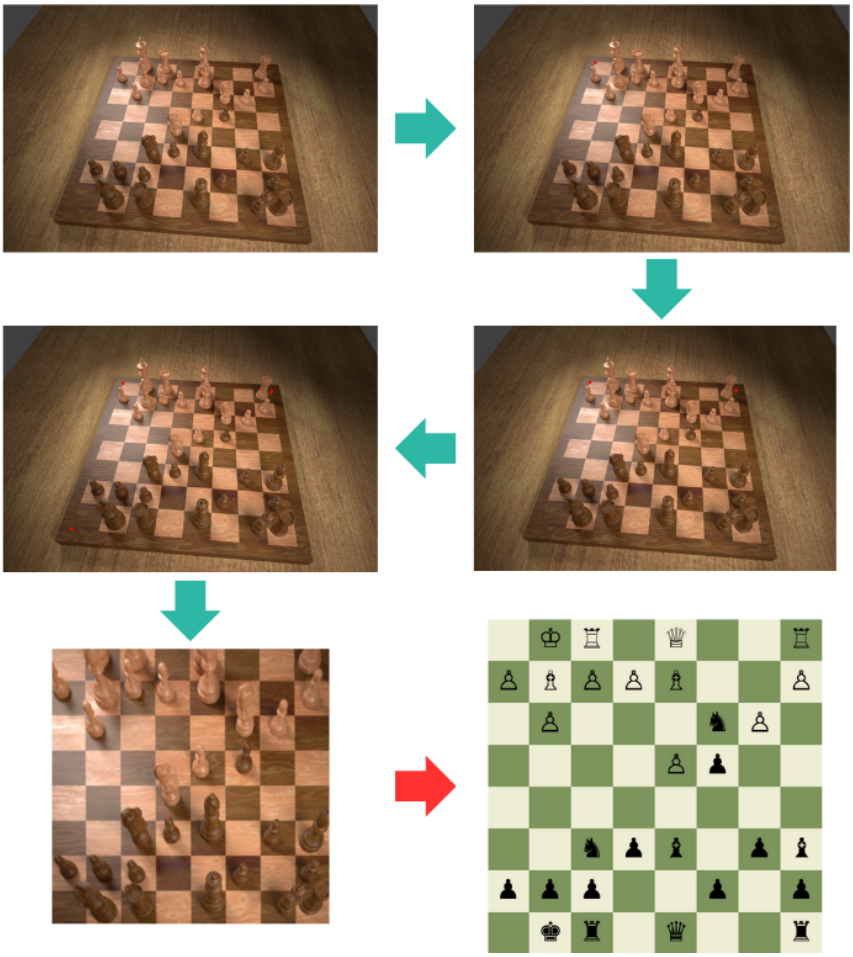


Rys. 17. Schemat działania systemu z technikami przetwarzaniem obrazu.

System interaktywny

System prezentowany w tej części pracy bazuje na interaktywnym zaangażowaniu użytkownika w wykryciu wierzchołków szachownicy. Takie rozwiązanie może posłużyć jako aplikacja do analizy gry w szachy, wspomagająca osobę korzystającą w podejmowaniu decyzji o następnym ruchu szachowym. Inspiracją dla tego rozwiązania były aplikacje do skanowania dokumentów za pomocą telefonu komórkowego, gdzie użytkownik jest zwykle proszony o zaznaczenie rogów skanowanego dokumentu [24].

Schemat znajdujący się na rysunku 18 przedstawia proces interakcji użytkownika z systemem. W pierwszym etapie użytkownik wykonuje zdjęcie szachownicy. Kolejnym krokiem jest zaznaczenie rogów szachownicy - tym sposobem zlokalizowano położenie planszy szachowej. Następnie system dokonuje transpozycji szachownicy zgodnie z zaznaczonymi rogami. W wyniku tego procesu, na ekranie telefonu użytkownika pojawia się cyfrowa reprezentacja rozgrywki. Przedstawiony system pozwala na wykorzystanie technologii rozpoznawania obrazu i interaktywnego interfejsu dla użytkownika końcowego.



Rys. 18. Schemat działania systemu interaktywnego.

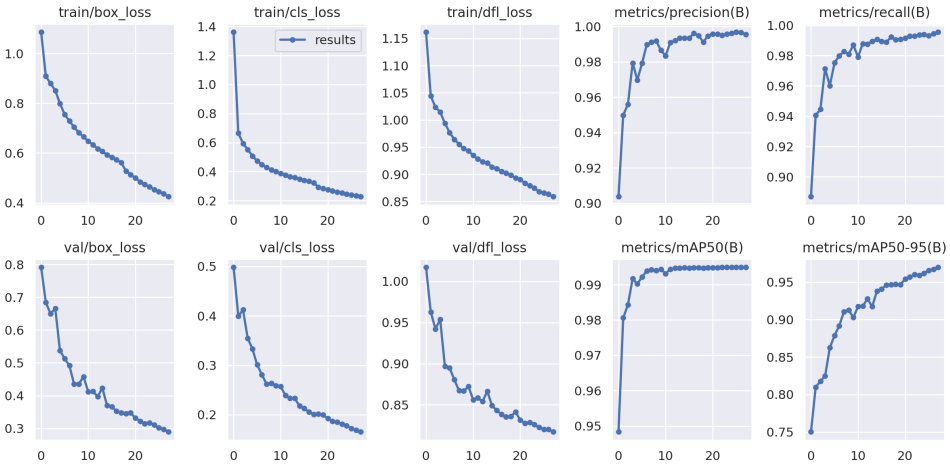
4. Analiza otrzymanych wyników

W tym rozdziale dokonana zostanie analiza i porównanie wyników uzyskanych z modeli zaimplementowanych w poprzedniej części pracy. Przedstawione zostaną zarówno prawidłowe, jak i błędne predykcje każdego z modeli. Dodatkowo przeprowadzona zostanie analiza krzywych uczenia i macierzy pomyłek, co pozwoli na ocenę postępów i poprawności modeli. Na koniec, przedstawione zostanie porównanie wszystkich zaimplementowanych algorytmów, umożliwiające ocenę ich wzajemnej skuteczności.

4.1. Wyniki oraz porównanie modeli

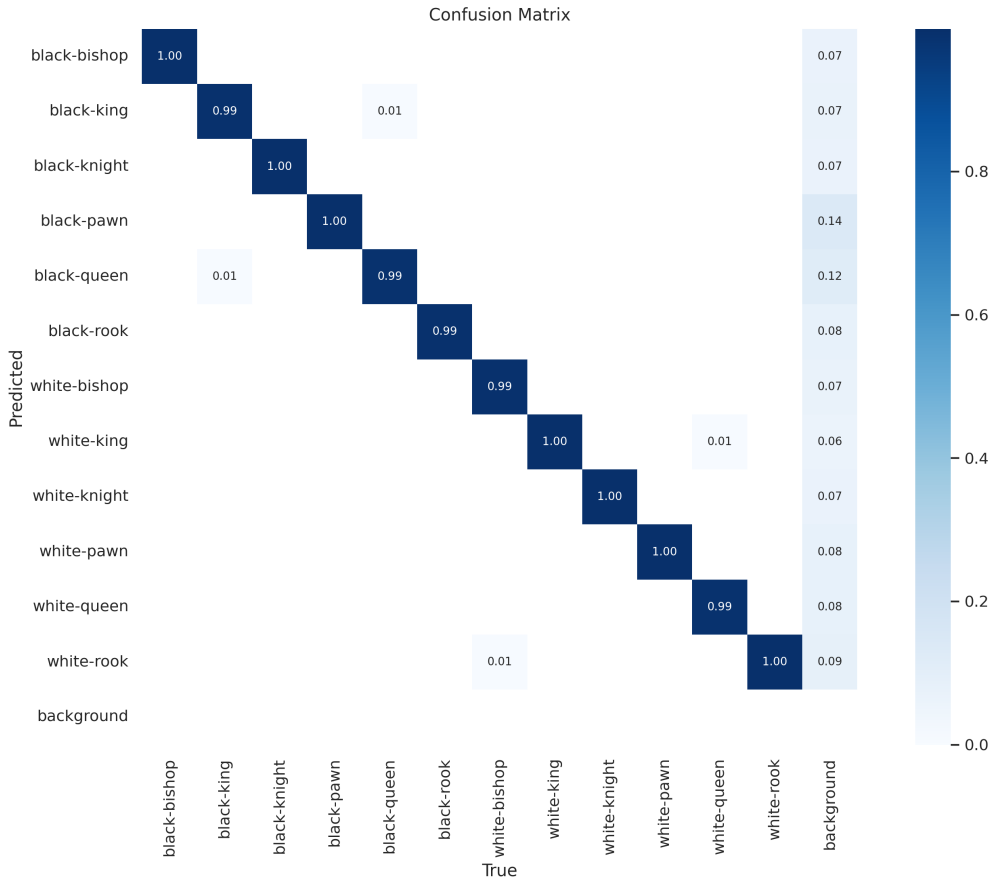
Model YOLO

Wykresy krzywych uczenia z rysunku 19 wskazują na efektywne uczenie się modelu w kolejnych epokach. Spadek wartości funkcji strat oraz wzrost kluczowych metryk jakości, takich jak dokładność (ang. *precision*), czułość (ang. *recall*) i średniej uśrednionej precyzji (ang. *mAP*) świadczy o rosnącej skuteczności i niezawodności modelu w detekcji oraz klasyfikacji obiektów. Brak wyraźnych oznak przeuczenia, ponieważ wyniki na zestawie walidacyjnym są porównywalne lub lepsze niż na zestawie treningowym, sugeruje dobrą generalizację modelu.



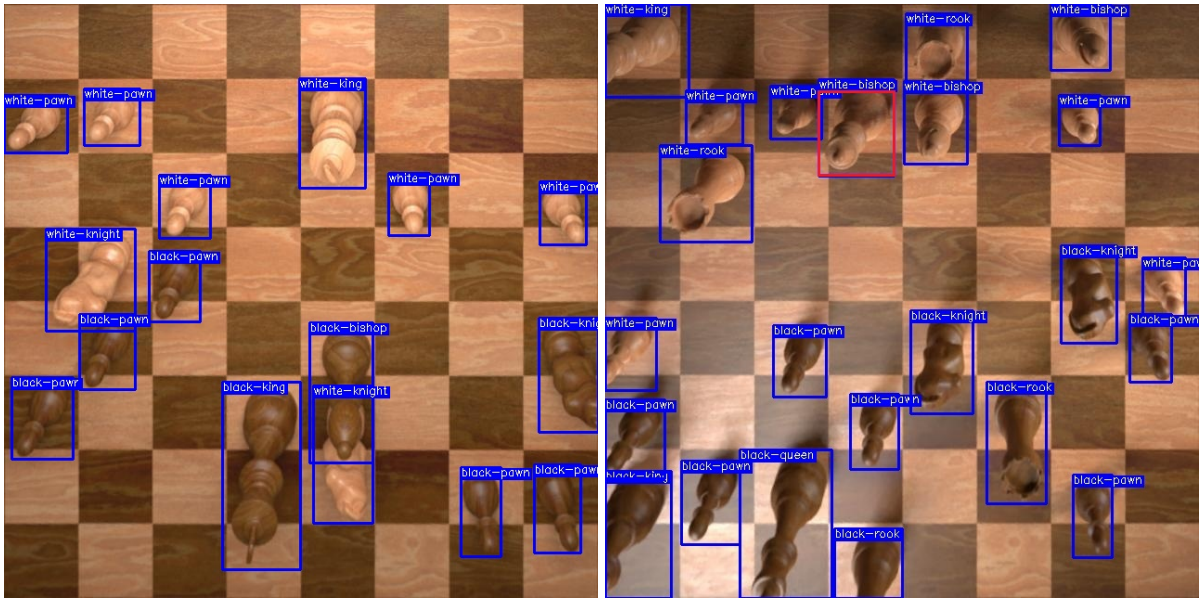
Rys. 19. Krzywe uczenia modelu YOLO.

Analizując całą macierz pomyłek, którą widać na rysunku 20, można zauważyć, że praktycznie wszystkie przewidywania są poprawne, co widać po wysokich wartościach na przekątnej. To wskazuje na bardzo wysoką dokładność modelu. Jednak pojawiają się pojedyncze błędne klasyfikacje, takie jak mylenie klas `czarny król` z `czarnym hetmanem`, czy `białej wieży` z `białym gońcem`, co sugeruje, że model może mieć trudności z rozróżnianiem pomiędzy bardzo podobnymi bierkami lub w specyficznych warunkach oświetleniowych. Dodatkowo na macierzy, widać klasę `background`, która oznacza, że model bardzo dobrze radzi sobie z identyfikacją obszarów bez obiektów. To jest istotne, ponieważ w detekcji obiektów, kluczowe jest nie tylko wykrywanie obiektów, ale także unikanie sytuacji, gdy model błędnie wykrywa obiekt tam, gdzie go nie ma. Macierz ta pokazuje, że model działa bardzo skutecznie.



Rys. 20. Znormalizowana macierz pomyłek modelu YOLO.

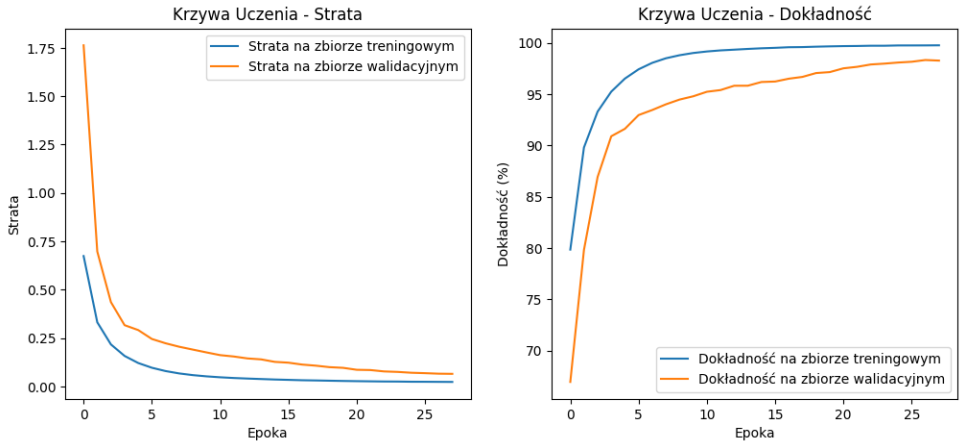
Po 28 epokach model osiągnął imponujące wyniki: 99.5% dokładności, 99.6% czułości oraz 99.5% średniej uśrednionej precyzji. W trakcie procesu uczenia model wymagał 11.6 GB zasobów pamięciowych. Przykładowe predykcje zaprezentowano na rysunku 21, na którym po prawej stronie widać, jak model pomylił jedną z figur.



Rys. 21. Po lewej stronie bezbłędna predykcja modelu YOLO, po prawej z jednym błędem.

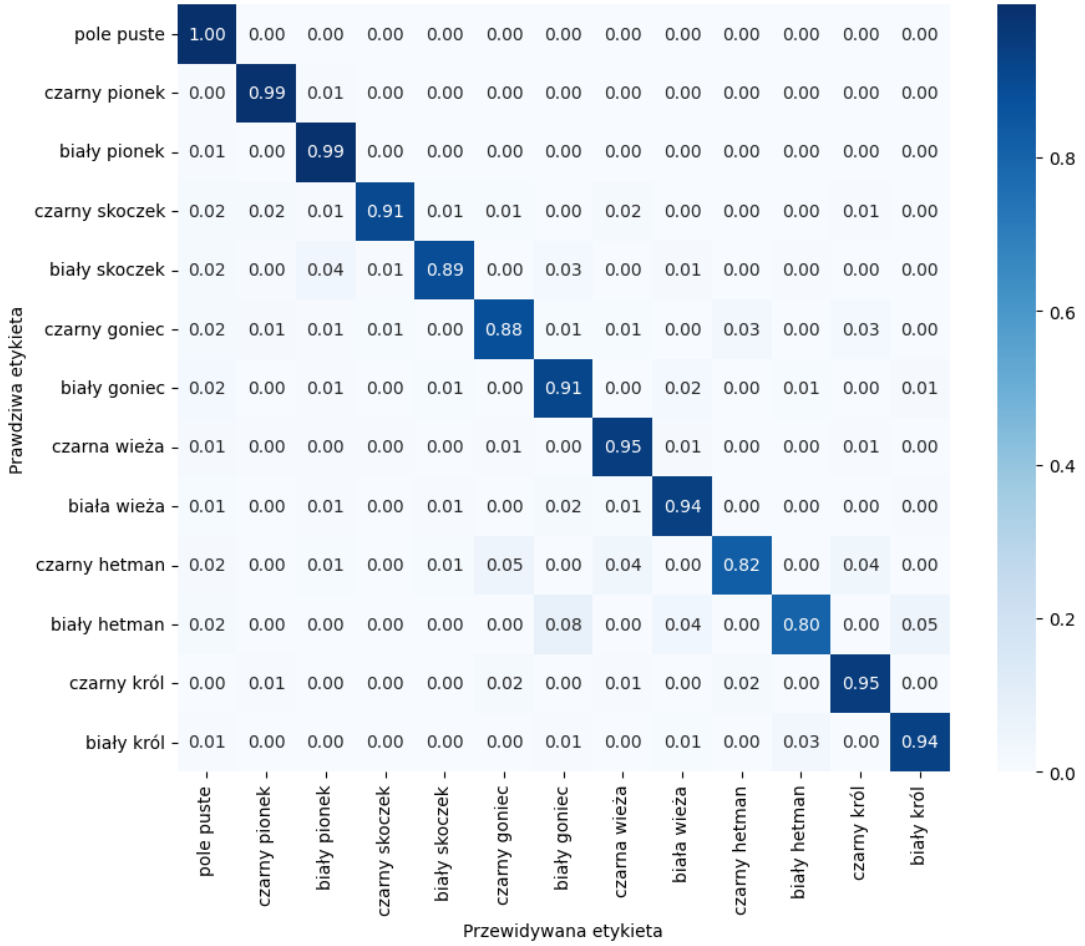
Wyniki modelu CNN dla całej szachownicy

Model osiągnął 98.27% dokładności na zbiorze testowym oraz wykorzystywał zaledwie 2.2 GB pamięci karty graficznej. Jednakże, w przypadku tego algorytmu wyznacznikiem dokładności jest uśredniona precyzja klas na zestawie testowym wynoszącą 91.99%, która lepiej odzwierciedla dokładność algorytmu z uwagi na liczbę próbek takich klas jak `puste pole`, `czarny pionek`, `biały pionek`. Proces uczenia, przedstawiony na wykresach krzywych uczących (rysunek 22), wskazuje na brak przeuczenia, co obrazuje dobrą generalizację modelu. Stopniowe zwiększanie się dokładności i stopniowy spadek wartości straty w kolejnych epokach treningu, świadczy o efektywnym procesie nauki.



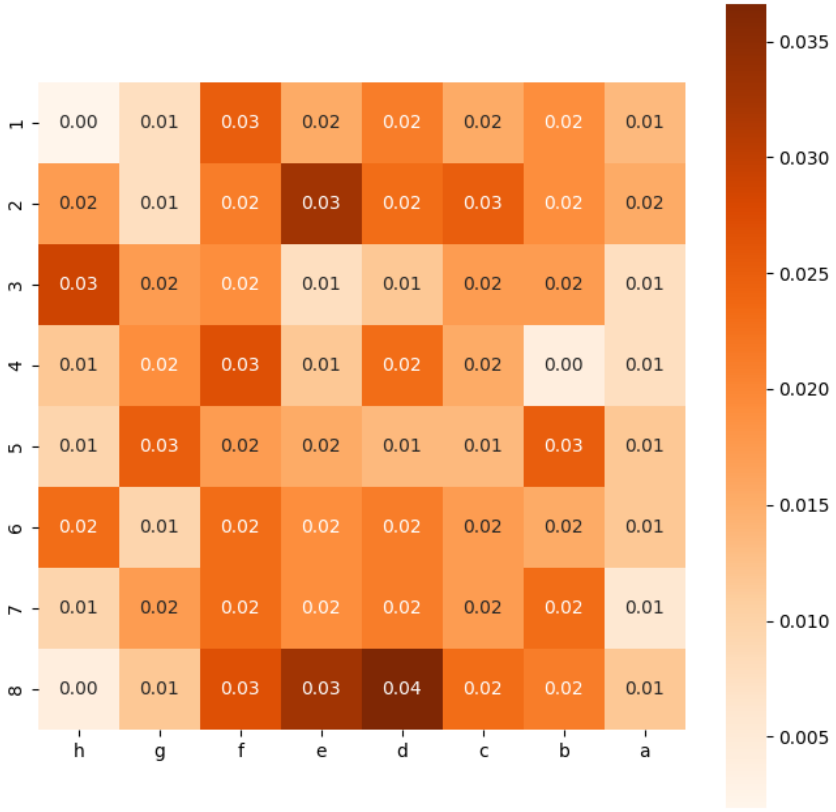
Rys. 22. Krzywe uczenia modelu CNN dla całej szachownicy.

Analizując macierz pomyłek (rysunek 23), widać iż model wykazuje bardzo dobrą precyzję w rozpoznawaniu większości bierek szachowych oraz pola pustego. Najwyższe wartości należą do pionów i figur takich jak `czarna wieża` i `biała wieża` oraz do `pola pustego`. W przypadku niektórych figur szachowych, takich jak `czarny hetman` i `biały hetman`, występują niższe wartości na przekątnej macierzy, z uwagi na trudności wynikające z podobieństwa pomiędzy figurami gońca, a hetmana. Szczególnie w sytuacji, gdy ich górne części są obcięte i model musi rozróżnić bierki na podstawie dolnej części figury.



Rys. 23. Znormalizowana macierz pomyłek modelu CNN dla całej szachownicy.

Na rysunku 24, zaprezentowano szachownicę, na której na każdym polu przedstawiony jest jego stopień niedokładności. Największe wartości znajdują się na środku w górnej oraz dolnej części planszy, oznacza to, że model popełnia najwięcej błędów w miejscach, w których jest najbardziej aktywnie z punktu widzenia rozgrywki. Jednakże, błędy widoczne na dolnej części szachownicy najprawdopodobniej wynikają z transformacji perspektywicznej obrazu.



Rys. 24. Pola szachownicy z największa niedokładnością.

Na rysunku 25 przedstawiono przypadek bezbłędnej predykcji modelu z dokładnością 100%, gdzie wszystkie bierki na planszy zostały sklasyfikowane poprawnie, co demonstruje potencjał modelu do precyzyjnej identyfikacji bierek w różnych konfiguracjach.

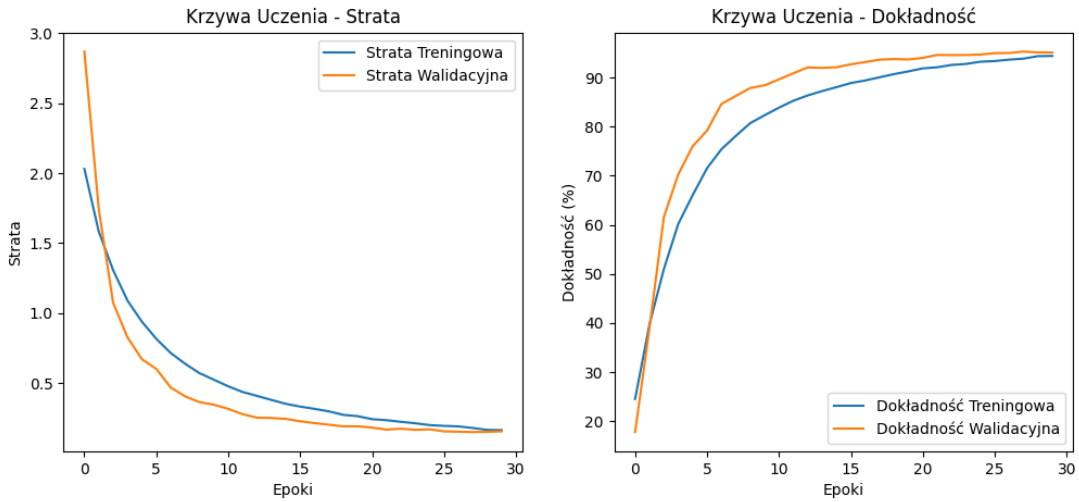
Rys. 25. Bezbłędna predykcja szachownicy modelu CNN.

Równocześnie rysunek 26 prezentuje sytuacje, w którym model osiągnął dokładność na poziomie 90.62%. Jest to najniższy wynik predykcji na zbiorze testowym, który pomimo bycia relatywnie wysokim, wskazuje na obszary, w których model ma swoje ograniczenia.

Rys. 26. Najgorsza predykcja szachownicy modelu CNN.

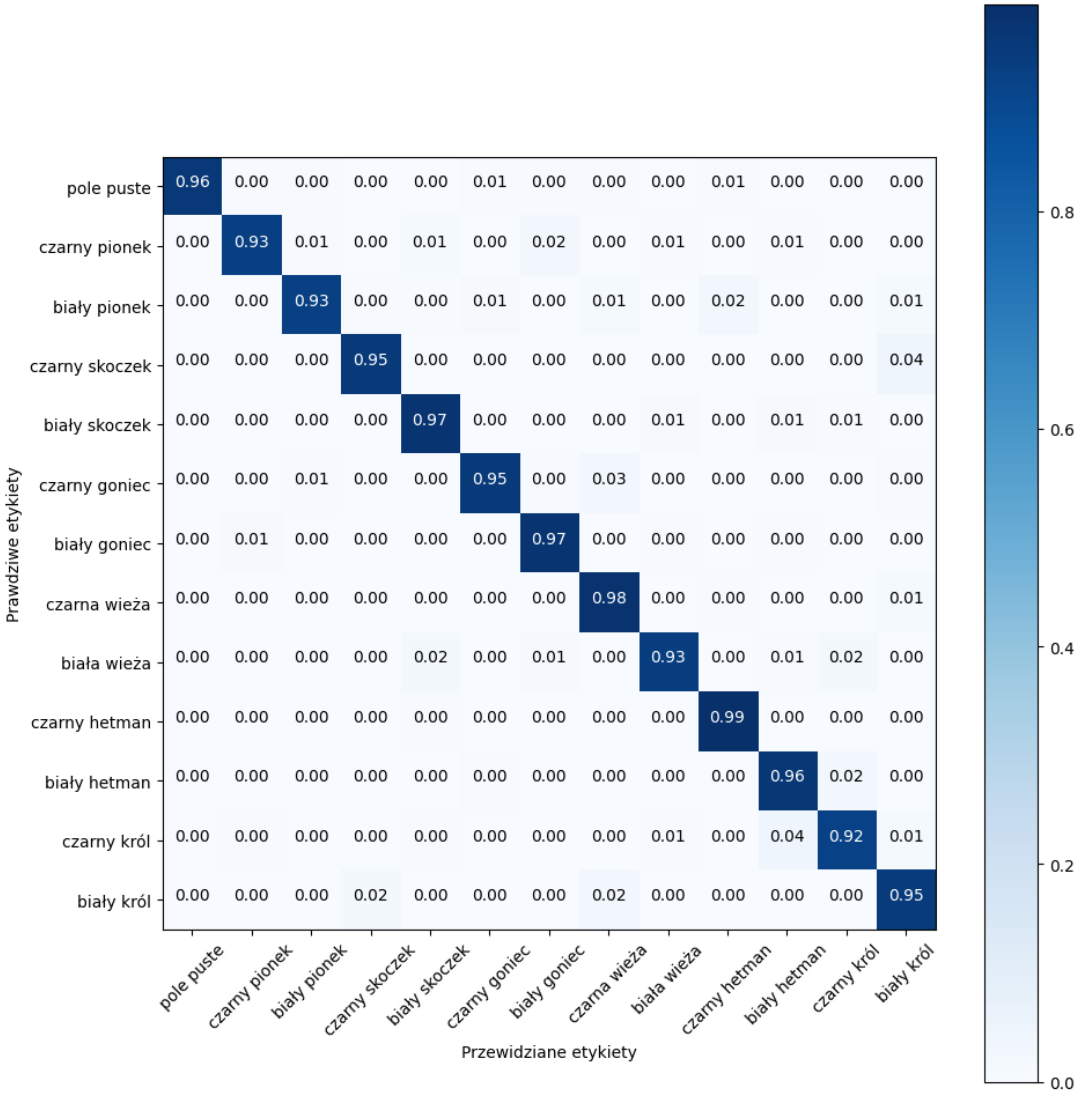
Wyniki modelu CNN dla pojedyńczego pola

W wyniku implementacji zaprezentowano krzywe uczenia dla zbudowanego modelu sieci neuronowej (rysunek 27). Lewy wykres pokazuje, jak zmienia się wartość funkcji straty w zależności od liczby epok. Można zauważyć oczekiwany trend spadkowy straty zarówno dla zbioru treningowego, jak i walidacyjnego, co świadczy o poprawie modelu w procesie uczenia. Strata treningowa maleje od około 2.8 w pierwszej epoce do wartości 0.5 w okolicy 10-tej epoki, podczas gdy strata walidacyjna również zmniejsza się. Należy zaznaczyć, że niższe wartości straty dla zbioru walidacyjnego są częściowo wynikiem zastosowania regularyzacji, a także faktu, że walidacja przeprowadzana jest po treningu w danej epoce. Na wykresie po prawej stronie widać, że dokładność zarówno treningowa, jak i walidacyjna wzrasta z każdą epoką. Chociaż dokładność walidacyjna wzrasta wolniej, to oba przebiegi świadczą o stopniowej poprawie wydajności modelu. Wartość dokładności na zbiorze walidacyjnym odzwierciedla zdolność modelu do generalizacji na danych, których nie widział podczas treningu.



Rys. 27. Krzywe uczenia modelu CNN dla pojedynczego pola.

Diagonalne wartości macierzy pomyłek (rysunek 28) są wyraźnie wyższe od pozostałych wartości w odpowiednich wierszach i kolumnach, wskazują na wysoką skuteczność modelu w poprawnym identyfikowaniu klasyfikowanych kategorii. Dla każdej bierki szachowej oraz pustego pola, model wykazuje zdolność do dokładnej predykcji z niewielką ilością błędów. Wartości poza główną przekątną reprezentują błędne klasyfikacje. Na przykład klasę `czarny król`, model kilkukrotnie błędnie sklasyfikował jako `biały hetman`. Podobne pomyłki występują dla innych bierek, ale ich liczba jest stosunkowo mała w porównaniu do liczby poprawnych klasyfikacji.



Rys. 28. Znormalizowana macierz pomyłek CNN dla pojedynczego pola.

Na rysunku 29 przedstawiono zestaw sześciu miniatur, które demonstrują wyniki procesu klasyfikacji. Dla każdej miniatury podane są dwie informacje: etykieta oczekiwana przez model oraz etykieta przewidziana na podstawie analizy obrazu. Wśród tych przykładów znajdują

się zarówno trafne klasyfikacje, jak i błędne przypisania. Na obrazach, gdzie model dokonał błędnej klasyfikacji, można zauważyć, że nieprawidłowe przewidywania mogły wynikać z podobieństw kształtów i konturów między różnymi bierkami. Na przykład, figura, która powinna być zidentyfikowana jako `czarny hetman`, została mylnie zaklasyfikowana jako `czarny król`, co wskazuje na to, że model mógł nie wychwycić kluczowych cech rozróżniających te dwie figury.

Rys. 29. Skuteczne oraz nieprawidłowe predykcje modelu CNN dla pojedynczego pola.

Model wykazał wysoką dokładność, osiągając 95.30% na zbiorze testowym, przy czym zużycie pamięci karty graficznej wynosiło jedynie 0.3 GB.

Porównanie zaimplementowanych modeli

Tabela 4.1. Porównanie wydajności badanych modeli

	YOLO	**CNN dla całej szachownicy**	**CNN dla pojedynczego pola**
Dokładność	99.5%	92.26%	95.30%
Zużycie pamięci karty graficznej	11.6 GB	2.2 GB	0.3 GB

W ramach przeprowadzonej analizy, dokonano porównania trzech modeli pod kątem ich dokładności, zużycia pamięci karty graficznej oraz czasu uczenia. Dane przedstawione w tabeli 4.1 pozwalają na ocenę wydajności każdego z modeli w kontekście zastosowania do detekcji i klasyfikacji bierek szachowych.

Model YOLO wyróżnia się najwyższą precyzją na poziomie 99.5%, co świadczy o jego bardzo dużej zdolności do identyfikacji obiektów. Wysoka dokładność idzie w parze z istotnie

większym zapotrzebowaniem na pamięć karty graficznej (11.6 GB), co może być krytyczne w przypadku zastosowań z ograniczonymi zasobami sprzętowymi. Biorąc pod uwagę lokalne środowisko, którego parametry zostały zaprezentowane w rozdziale 3, nie pozwoliłoby ono na wytrenowanie tego modelu z powodu jego ograniczeń mocy obliczeniowej.

Z kolei model CNN dla całej szachownicy zachowuje równowagę pomiędzy dokładnością, a zużyciem zasobów osiągając 98.03% dokładności przy zużyciu pamięci na poziomie 2.1 GB.

Model CNN dla pojedynczego pola charakteryzuje się najniższym zapotrzebowaniem na pamięć (0.3 GB), co czyni go atrakcyjnym wyborem, pod względem posiadania ograniczonych zasobów. Jego dokładność, chociaż najniższa spośród porównywanych modeli, wynosi nadal imponujące 95.30%.

Wybór modelu zależy od wymagań odbiorcy, w tym dostępnych zasobów sprzętowych, długości czasu przetwarzania oraz stopnia dokładności. Model YOLO będzie najlepszą opcją w przypadku zastosowań, gdzie priorytetem jest maksymalna dokładność, natomiast modele CNN oferują bardziej zrównoważone podejście, zwłaszcza w kontekście ograniczeń sprzętowych.

Podsumowanie

W ramach projektu zaimplementowano dwa algorytmy rozpoznające figury szachowe na zdjęciach szachownicy oraz jeden model rozpoznający figury na polu szachowym, co pozwoliło na porównanie tych dwóch podejść do rozwiązania problemu. Najwyższą dokładnością wykazał się model YOLO ze skutecznością 99.5%, którą osiągnięto przy pomocy douczania modelu. Wartość ta wynika z tego, że ten algorytm jest efektem wieloletniej pracy. Przez te lata, zespół pracujący nad nim nie tylko zgromadził ogromną wiedzę i doświadczenie, ale także wykorzystał jednostki o bardzo dużych mocach obliczeniowych do przeprowadzania rozległych eksperymentów i testów. Oba zaprojektowane modele CNN również zaprezentowały bardzo dobre wyniki, jednak w kontekście wykrywania figur szachowych, gdzie wymagana jest wyjątkowa precyzja, model YOLO okazał się lepszy.

Największą trudnością podczas realizacji projektu okazała się implementacja modelu CNN dla całej szachownicy. Problemy wynikały głównie z rozkładu klas w zbiorze danych, co utrudniało efektywne przeprowadzenie treningu algorytmu. W celu uniknięcia przeuczenia modelu niezbędne było dokładne dostosowanie hiperparametrów, co wymagało licznych prób i testów. Ta praca nad hiperparametrami była konieczna dla osiągnięcia optymalnej wydajności modelu, jednocześnie zapewniając, że model nie będzie nadmiernie dostosowany do danych treningowych.

Dalszy rozwój projektu mógłby obejmować rozwinięcie wybranego modelu o zwiększenie liczby obsługiwanych odmian figur szachowych, dzięki czemu odbiorca mógłby wykorzystywać algorytm na swoich własnych bierkach. Kolejnym obszarem mogłoby być stworzenie aplikacji mobilnej zintegrowanej z silnikami szachowymi, która pozwoliłaby na śledzenie partii szachowych w czasie rzeczywistym i na naukę strategii szachowych. W tym celu konieczne byłoby wcześniejsze udoskonalenie przedstawionego systemu wykorzystującego techniki przetwarzania obrazu (rozdział 3.3). Realizacja takiej aplikacji wymagałaby dalszego rozwoju i badań, aby zagwarantować niezawodność i precyzyjność, co jest kluczowe dla jej sukcesu i użyteczności.

Bibliografia

[1] CHESScom. „*Chess.com Reaches 100 Million Members!*" [Online; Data uzyskania dostępu 26 Grudnia, 2023]. 2022.

[2] Zewen Li i in. „*A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*". W: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), s. 6999–7019. DOI: *10.1109/TNNLS.2021.3084827*.

[3] Keiron O'Shea i Ryan Nash. „*An Introduction to Convolutional Neural Networks*". 2015. arXiv: *1511.08458* [cs.NE].

[4] C. Jose L. Flores, A. E. Gladys Cutipa i R. Lauro Enciso. „*Application of convolutional neural networks for static hand gestures recognition under different invariant features*". W: *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*. 2017, s. 1–4. DOI: *10.1109/INTERCON.2017.8079727*.

[5] FirelordPhoenix. „*Pictorial example of max-pooling*". [Online; Data uzyskania dostępu 17 Grudnia, 2023]. 2018.

[6] Charu C. Aggarwal. „*Neural Networks and Deep Learning: A Textbook*". 1st. Springer Publishing Company, Incorporated, 2018. ISBN: 3319944622.

[7] Teja Kattenborn i in. „*Review on Convolutional Neural Networks (CNN) in vegetation remote sensing*". W: *ISPRS Journal of Photogrammetry and Remote Sensing* 173 (2021), s. 24–49. ISSN: 0924-2716. DOI: *https://doi.org/10.1016/j.isprsjprs.2020.12.010*.

[8] Eva Tuba i in. „*Convolutional Neural Networks Hyperparameters Tuning*". W: *Artificial Intelligence: Theory and Applications*. Red. Endre Pap. Cham: Springer International Publishing, 2021, s. 65–84. ISBN: 978-3-030-72711-6. DOI: *10.1007/978-3-030-72711-6_4*.

[9] Mario Milicevic i in. „*Application of Transfer Learning for Fine-Grained Vessel Classification Using a Limited Dataset*". W: wrz. 2018.

[10] Joseph Redmon i in. „*You Only Look Once: Unified, Real-Time Object Detection*". W: *CoRR* abs/1506.02640 (2015). arXiv: *1506.02640*.

[11] Zhiwei Li, Zhihong Liu i Xiangke Wang. „*On-Board Real-Time Pedestrian Detection for Micro Unmanned Aerial Vehicles Based on YOLO-v8*". W: *2023 2nd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*. 2023, s. 250–255. DOI: *10.1109/MLCCIM60412.2023.00042*.

[12] Juan Terven i Diana Cordova-Esparza. „*A Comprehensive Review of YOLO: From YOLOv1 and Beyond*". 2023. arXiv: *2304.00501* [cs.CV].

[13] Meghana Pulipalupula i in. „*Object Detection using You Only Look Once (YOLO) Algorithm in Convolution Neural Network (CNN)*". W: *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*. 2023, s. 1–4. DOI: *10.1109/I2CT57861.2023.10126213*.

[14] Hideaki Yanagisawa, Takuro Yamashita i Hiroshi Watanabe. „*A study on object detection method from manga images using CNN*". W: *2018 International Workshop on Advanced Image Technology (IWAIT)*. 2018, s. 1–4. DOI: *10.1109/IWAIT.2018.8369633*.

[15] Georg Wölflein i Ognjen Arandjelović. „*Dataset of Rendered Chess Game State Images*". Lut. 2023. DOI: *10.17605/OSF.IO/XF3KA*.

[16] Georg Wölflein i Ognjen Arandjelović. „*Determining Chess Game State from an Image*". W: *Journal of Imaging* 7.6 (czer. 2021), s. 94. ISSN: 2313-433X. DOI: *10.3390/jimaging7060094*.

[17] Blender Online Community. „*Blender - a 3D modelling and rendering package*". Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018.

[18] Roboflow. „*Roboflow Python Package*".

[19] Adam Paszke i in. „*PyTorch: An Imperative Style, High-Performance Deep Learning Library*". W: *Advances in Neural Information Processing Systems 32*. Red. H. Wallach i in. Curran Associates, Inc., 2019, s. 8024–8035.

[20] Steven Gutstein i Ethan Stump. „*Reduction of catastrophic forgetting with transfer learning and ternary output codes*". W: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, s. 1–8. DOI: *10.1109/IJCNN.2015.7280416*.

[21] Glenn Jocher, Ayush Chaurasia i Jing Qiu. „*YOLO by Ultralytics*". Wer. 8.0.0. Sty. 2023.

[22] Preet Karia i in. „*Digitization of Chess Board and Prediction of Next Move*". W: *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*. 2022, s. 1–5. DOI: *10.1109/I2CT54291.2022.9825379*.

[23] Athanasios Masouris i Jan van Gemert. „*End-to-End Chess Recognition*". 2023. arXiv: *2310.04086* [cs.CV].

[24] Prathmesh Tirodkar i in. „*Buddy Scanner – A Scanning Application*". W: *2021 2nd International Conference for Emerging Technology (INCET)*. 2021, s. 1–9. DOI: *10.1109/INCET51464.2021.9456316*.