

**W A R S Z A W S K A  
W Y Ź S Z A S Z K O Ł A I N F O R M A T Y K I**

---

---

**P R A C A M A G I S T E R S K A**

**Agnieszka Wrzos**

Numer albumu: 10350

**Badanie skuteczności modelu rozpoznawania dzieł malarstw  
z wykorzystaniem uczenia głębokiego**

**Promotor:**

**dr hab. inż. Zenon Gniazdowski, prof. WWSI**

**Konsultant:**

**mgr inż. Andrzej Ptasznik**

*Praca spełnia wymagania stawiane pracom dyplomowym na studiach drugiego stopnia.*



# Spis treści

<b>Wstęp</b> . . . . .	5
<b>Rozdział 1. Architektura sieci neuronowych</b> . . . . .	7
1.1. Wstęp do architektury sieci neuronowych . . . . .	7
1.2. Wybrane architektury . . . . .	12
1.3. Normalizacja danych . . . . .	15
1.4. Nadmierne niedouczenie i przeuczenie . . . . .	16
1.5. Regularyzacja danych . . . . .	17
1.6. Funkcje aktywacji i optymalizatory . . . . .	19
1.7. Augmentacja danych . . . . .	22
<b>Rozdział 2. Wstępna analiza i interpretacja zbioru danych</b> . . . . .	25
2.1. Przedstawienie danych . . . . .	25
2.2. Przykładowe obrazy . . . . .	30
2.3. Wnioski z wstępnej analizy danych . . . . .	33
<b>Rozdział 3. Modele rozpoznawania artystów dzieł malarstw</b> . . . . .	35
3.1. Model 0: Podstawowa sieć CNN . . . . .	39
3.2. Model 0.5: Sieć CNN z ulepszoną architekturą i augmentacją . . . . .	44
3.3. Model 1: Sieć CNN z ręcznym podziałem danych i bez augmentacji . . . . .	48
3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout . . . . .	52
3.4.1. Rozszerzenie Modelu 2 o metodę oversamplingu . . . . .	56
3.4.2. Rozszerzenie Modelu 2 o metodę walidacji krzyżowej . . . . .	60
3.4.3. Zastosowanie redukcji ilości klas w Modelu 2 . . . . .	63
3.5. Zastosowanie Transfer Learning z wykorzystaniem ResNet i Fastai . . . . .	65
<b>Podsumowanie i wnioski</b> . . . . .	69
3.6. Podsumowanie wyników wszystkich modeli . . . . .	69
3.7. Wnioski . . . . .	71
<b>Wykaz literatury</b> . . . . .	75

## Spis treści

<b>Spis rysunków</b>	77
<b>Spis tabel</b>	79

## **Wstęp**

Sztuka od zawsze była nieodłącznym elementem kultury, odzwierciedlając emocje, wartości i interpretacje subiektywnej rzeczywistości. Uczenie głębokie, jako zaawansowana dziedzina sztucznej inteligencji, otworzyło nowe horyzonty w analizie oraz tworzeniu sztuki. Stając się narzędziem, które nie tylko wspiera analizę dzieł sztuki, lecz także uczestniczy w ich tworzeniu, pozwoliło na rozwój innowacyjnych metod badawczych oraz kreatywnych zastosowań. Wzrost zapotrzebowania na technologię związaną z uczeniem maszynowym w różnych dziedzinach życia, w tym w sztuce, stał się zauważalny.

W niniejszej pracy przedstawiono różne podejścia do wykorzystania uczenia głębokiego w kontekście sztuki. Przeanalizowano, jak technologia może wpływać na postrzeganie, analizowanie i interpretowanie dzieł artystycznych. W teoretycznej części pracy omówiono podstawowe koncepcje teoretyczne związane z uczeniem głębokim. W części praktycznej, po wstępnej analizie zbioru danych, zaimplementowano najprostszy model uczenia głębokiego. Model ten następnie rozwinięto o coraz bardziej zaawansowane techniki przetwarzania danych. Celem tego etapu jest zrozumienie wpływu różnych technik na skuteczność modelu oraz jego zdolność do poprawnej klasyfikacji obrazów. W końcowej części podsumowano wszystkie uzyskane wyniki, a następnie omówiono wnioski.

Artyści wykorzystują różnorodność kolorów, technik i stylów, aby oddać subiektywną wizję świata w swoich dziełach. Celem pracy jest stworzenie modelu opartego na uczeniu głębokim, który będzie w stanie rozpoznać indywidualność artystów w ich podejściu do sztuki. Analiza wpływu technik przetwarzania danych jest istotna dla dobrego zrozumienia ich działania i wpływa na budowę modeli.



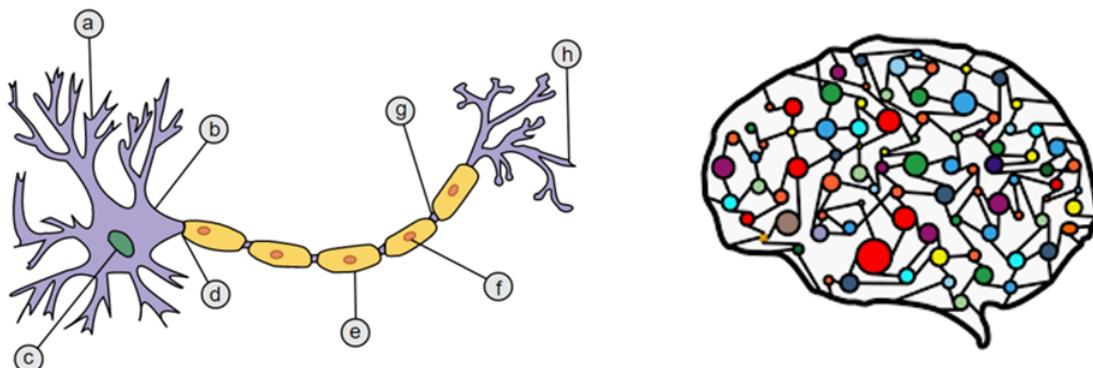
## Rozdział 1

### Architektura sieci neuronowych

#### 1.1. Wstęp do architektury sieci neuronowych

Współczesne techniki uczenia maszynowego są szeroko stosowane w różnych dziedzinach, takich jak systemy rekomendacyjne, filtrowanie spamów, wykrywanie oszustw, bezpieczeństwo czy medycyna. Dynamiczny rozwój uczenia maszynowego prowadzi do powstawania coraz bardziej zaawansowanych technologii, które nie tylko stają się co-dziennym elementem życia, ale także znacząco podnoszą jego jakość w kluczowych dziedzinach, takich jak medycyna czy bankowość. Tradycyjne metody uczenia maszynowego, które zaczęły napotykać pewne ograniczenia, wymusiły rozwój bardziej nowoczesnych technik. Głębokie sieci neuronowe, będące nowoczesnym narzędziem, są w stanie automatycznie odkrywać ukryte związki i wzorce w danych, które są trudne do uchwycenia dla ludzi. Umożliwia to rozwiązywanie złożonych problemów i efektywne przetwarzanie ogromnych ilości danych, które we współczesnym świecie mnożą się bardzo szybko. Uczenie głębokie charakteryzuje się również mniejszym wkładem ze strony programistów, co sprawia, że jest bardziej dostępnym i skutecznym narzędziem w porównaniu do tradycyjnych metod. Przyczynia się to do szybszego postępu w takich dziedzinach jak rozpoznawanie obrazów, analiza tekstu czy rozpoznawanie mowy, co wpływa na rozwój technologiczny i społeczny. W architekturze sieci neuronowych kluczowe jest zrozumienie fundamentalnych zasad i mechanizmów, bez których trudno byłoby projektować, trenować i optymalizować modele w sposób efektywny. Znajomość podstaw takich jak między innymi budowa sieci, funkcje aktywacji, warstwy konwolucyjne, techniki regularyzacji oraz metody optymalizacji pozwala na świadomie podejmowanie decyzji dotyczących konstrukcji sieci i jej dostosowywania do konkretnego zadania. Bez tego niecieżko jest popełnić błędy przy projektowaniu modelu, co może prowadzić do trudności w analizie oraz nieoptimalnych wyników. Przyswojenie podstawowych koncepcji jest zatem niezbędne, aby skutecznie zarządzać procesem tworzenia i doskonalenia sieci neuronowej, a także w pełni wykorzystać jej potencjał w praktycznych zastosowaniach.[1] [2]

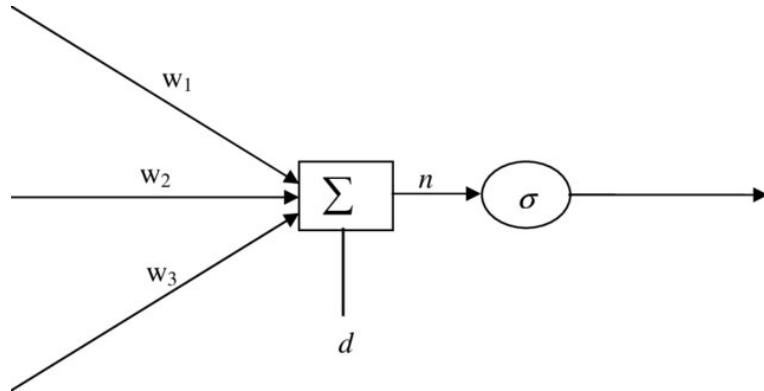
Podstawowym elementem każdej sieci neuronowej jest neuron. Neurony w sieciach sztucznych, inspirowane strukturą i funkcjonowaniem neuronów biologicznych, są odpowiedzialne za przetwarzanie i przesyłanie informacji. Na rysunku 1.1, został zilustrowany neuron biologiczny, który sam być może wydaje się prostą jednostką, jednak połączony w sieci złożonej z ogromnej ilości neuronów potrafi współpracować w celu wykonania bardzo złożonych obliczeń. Zrozumienie pochodzenia koncepcji neuronu oraz jego różnic w stosunku do biologicznych odpowiedników jest istotne dla pełnego zrozumienia działania sieci neuronowych. Biologiczne neurony to złożone komórki nerwowe, które komunikują się z innymi neuronami za pośrednictwem synaps. Każdy neuron odbiera sygnały od innych neuronów przez dendryty i przekazuje je dalej przez akson. W kontekście sieci neuronowych ta analogia jest uproszczona: neurony są zorganizowane w warstwy, a ich działanie polega na przetwarzaniu sygnałów wejściowych i generowaniu wyjściowych, bez konieczności pełnego odwzorowywania rzeczywistych struktur biologicznych. [3]



a – dendryty, b – ciało komórki, c – jądro komórkowe, d – akson, e – otoczka mielinowa, f – komórka Schwanna, g – przewężenie anwiera, h – zakończenia aksonu.

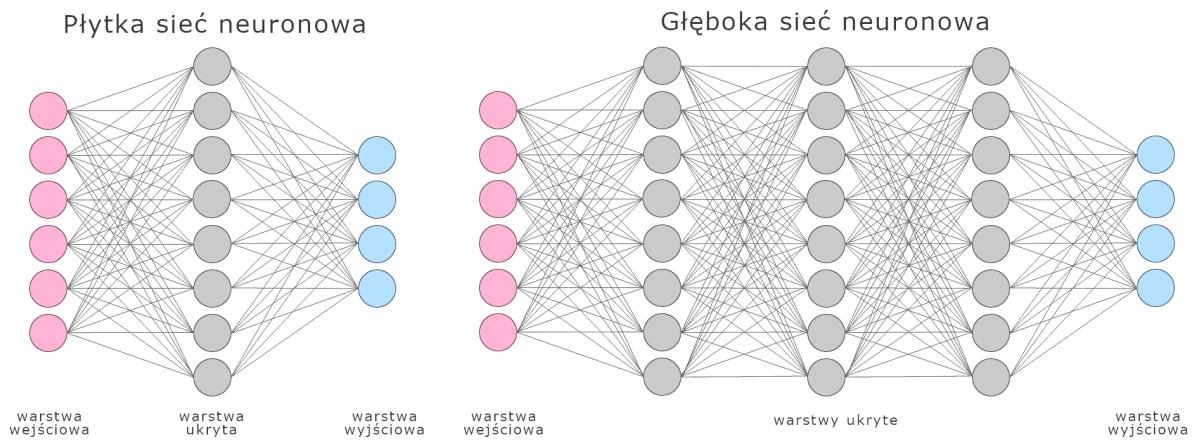
**Rysunek 1.1:** Neuron biologiczny [3]

**Sztuczne sieci neuronowe** składają się z dużej liczby elementów przetwarzających, ułożonych w formie warstw i połączonych ze sobą za pomocą łączys komunikacyjnych. Każde z tych łączys komunikacyjnych ma przypisaną wagę; wagi tych łączys są zmieniane, aby przybliżyć nieliniową zależność funkcyjną między wejściem a wyjściem. Każdy neuron ma również wewnętrzny proces, zwany funkcją aktywacji, który działa na skumulowane wejście do neuronu. [4] Podstawowa struktura sztucznego neuronu jest pokazana na rysunku 1.2.



Rysunek 1.2: Podstawowa struktura sztucznego neuronu [4]

W sieciach sztucznych neurony są organizowane w warstwy – wejściową, ukrytą i wyjściową. **Warstwa wejściowa** odbiera surowe dane, warstwy ukryte przetwarzają te dane, a **warstwa wyjściowa** dostarcza ostateczne wyniki modelu. Każda warstwa może zawierać wiele neuronów, które współpracują ze sobą, aby rozwiązać zadanie przetwarzania danych. Warstwy w sieciach neuronowych można klasyfikować jako płytkie lub głębokie w zależności od ich złożoności i liczby warstw. **Warstwy płytkie** zazwyczaj składają się z jednej lub kilku warstw neuronów, podczas gdy **warstwy głębokie** mogą obejmować setki lub tysiące warstw neuronów. Rysunek 1.3 ilustruje wspomniane rodzaje warstw.



Rysunek 1.3: Płytna i głęboka sieć neuronowa

Warstwy płytkie są przedstawione jako prostsze struktury z niewielką liczbą warstw, które są odpowiednie dla prostych zadań, takich jak klasyfikacja liniowa. W przeciwieństwie do nich, warstwy głębokie, które są bardziej złożone, składają się z wielu warstw i są zdolne do rozwiązywania bardziej skomplikowanych problemów, takich jak rozpoznanie obrazów czy przetwarzanie języka naturalnego. Tabela 1.1 w sposób dokładniejszy porównuje warstwy płytkie i głębokie pod względem kilku

kryteriów, takich jak architektura, zastosowania, efektywność obliczeniowa, optymalizacja, oraz zdolność do generalizacji.

**Tabela 1.1:** Porównanie warstw płytowych i głębokich

Kryterium	Warstwy płytowe	Warstwy głębokie
Architektura	Jednowarstwowe lub z niewielką liczbą warstw	Wielowarstwowe, często z setkami lub tysiącami neuronów
Zastosowania	Odpowiednie dla prostych zadań, takich jak klasyfikacja liniowa	Odpowiednie dla skomplikowanych zadań, takich jak rozpoznawanie obrazów, przetwarzanie języka naturalnego
Efektywność obliczeniowa	Mniejsze wymagania sprzętowe i energetyczne	Większe wymagania sprzętowe i energetyczne
Optymalizacja	Prostsza do przeprowadzenia optymalizacji	Bardziej złożona optymalizacja, ryzyko zanikającego gradientu
Wyrażanie złożoności	Ograniczone do prostszych, bardziej liniowych problemów	Potrafi rozwiązywać bardzo złożone, nieliniowe problemy
Przeciwdziałanie przeuczeniu	Mniejsze ryzyko przeuczenia	Wyższe ryzyko przeuczenia, wymaga technik regularyzacji
Zasoby danych	Efektywne przy mniejszych zbiorach danych	Wymagają dużych zbiorów danych do efektywnego trenowania
Zdolność do generalizacji	Mniejsza zdolność do uchwycenia skomplikowanych wzorców	Większa zdolność do uchwycenia skomplikowanych wzorców
Obliczeniowa wydajność	Szybsze trenowanie i predykcja	Wolniejsze trenowanie i predykcja
Złożoność modelu	Mniejsza złożoność, łatwiejsze do zrozumienia	Większa złożoność, trudniejsze do zrozumienia

Warstwy płytowe są mniej skomplikowane i mniej wymagające obliczeniowo, ale również mniej efektywne w uchwytwaniu złożonych wzorców. Z kolei warstwy

głębokie, mimo że wymagają więcej zasobów obliczeniowych i są bardziej złożone, oferują większą zdolność do rozwiązywania złożonych problemów i lepszą zdolność do generalizacji.

Kluczowym aspektem, wpływającym na zdolność sieci neuronowej do efektywnego uczenia się i generalizacji wzorców jest liczba neuronów w warstwach ukrytych. Podczas gdy liczba neuronów w warstwach wejściowej i wyjściowej jest ściśle określona przez strukturę danych, liczba neuronów w warstwach ukrytych jest zazwyczaj dobrana eksperymentalnie, zależnie od zadania. Przy dobiorze liczby neuronów należy mieć na uwadze, że zbyt mała liczba neuronów w warstwie może ograniczać zdolność sieci do skutecznego modelowania złożonych wzorców danych, co może prowadzić do utraty istotnych informacji. Zaleca się eksperymentalne zwiększanie liczby neuronów w warstwach do momentu, gdy sieć zaczyna przeuczać się na danych treningowych. Alternatywnie, można rozważyć budowanie modelu z większą liczbą warstw i neuronów niż jest to wymagane, a następnie zastosowanie technik regularyzacji, takich jak wczesne zatrzymywanie, aby zapobiec przeuczeniu. Podsumowując, dobór odpowiedniej liczby neuronów w warstwach ukrytych jest kluczowy dla efektywności i zdolności generalizacji sieci neuronowych, wymagając równowagi między złożonością modelu a jego zdolnością do reprezentacji złożonych wzorców danych. [3]

## 1.2. Wybrane architektury

W dziedzinie sztucznych sieci neuronowych istnieje wiele różnych architektur, z których każda jest zoptymalizowana pod kątem specyficznych rodzajów danych i zadań. Podstawowym modelem sieci neuronowej i zarazem fundamentem dla bardziej zaawansowanych struktur jest perceptron. Konwolucyjne sieci neuronowe są szczególnie skuteczne w analizie obrazów, natomiast rekurencyjne sieci neuronowe doskonale radzą sobie z danymi sekwencyjnymi, takimi jak tekst czy mowa. Impulsowe sieci neuronowe, zainspirowane biologicznymi mechanizmami przetwarzania informacji, znajdują zastosowanie w dziedzinach wymagających szybkiego i precyzyjnego przetwarzania sygnałów. Każda z tych architektur wnosi unikalne możliwości do analizy i przetwarzania danych, co czyni je istotnymi narzędziami w rozwijających się dziedzinach sztucznej inteligencji i uczenia maszynowego. [5]

**Perceptron** to jedna z najprostszych form sztucznych sieci neuronowych, stworzona przez Franka Rosenblatta. [6] [7] Można wyróżnić dwa główne rodzaje perceptronów:

- **Perceptrony jednowarstwowe** potrafiące uczyć się jedynie wzorców, które są liniowo separowalne, co oznacza, że mogą oddzielić dane w przestrzeni wejściowej za pomocą jednej prostej granicy,
- **Perceptrony wielowarstwowe (sieci neuronowe jednokierunkowe)** mające co najmniej dwie warstwy i większą moc przetwarzania. Wielowarstwowej struktury, pozwala na uchwycenie bardziej złożonych wzorców w danych.

Perceptron działa na zasadzie obliczania ważonej sumy swoich wejść i dodawania do niej stałej wartości, zwanej obciążeniem. Następnie stosuje prostą funkcję skokową, która decyduje o ostatecznym wyniku. Jeśli obliczona suma przekroczy pewien próg, perceptron klasyfikuje dane jako należące do jednej klasy, w przeciwnym razie do drugiej. W praktyce perceptron można zobrazować jako klasyfikator binarny, który na podstawie obliczonej sumy ważonej decyduje o przynależności do jednej z dwóch klas. W przeciwnieństwie do regresji logistycznej, która stosuje funkcję sigmoidalną do uzyskania wartości w zakresie od 0 do 1, perceptron używa prostego kryterium: jeśli obliczona suma przekracza zero, klasyfikuje dane jako należące do jednej klasy; w przeciwnym razie, przypisuje je do drugiej klasy.[3]

Podczas gdy perceptron jest skuteczny w prostych zadaniach klasyfikacyjnych, jego możliwości są ograniczone, gdy w grę wchodzi bardziej złożona analiza danych. Z tego powodu, w miarę rozwoju technologii, pojawiły się bardziej zaawansowane architektury sieci neuronowych, takie jak **konwolucyjne sieci neuronowe (CNN)**. Są specjalnym rodzajem sieci neuronowych, które zostały zaprojektowane z myślą o prze-

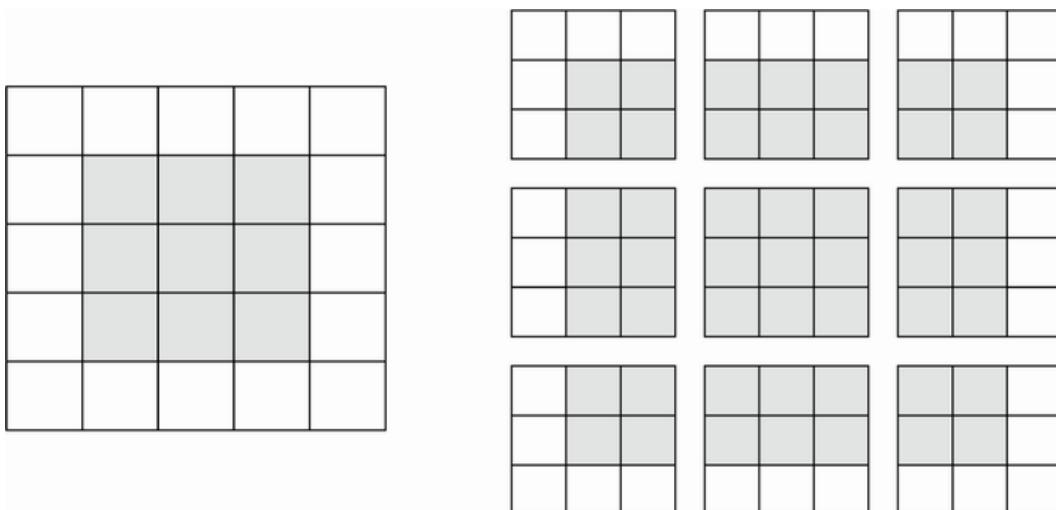
twarzaniu i analizowaniu obrazów. W odróżnieniu od tradycyjnych sieci neuronowych, które mogą być stosowane w różnych dziedzinach, koncentrują się na wydobywaniu cech z obrazów i ich klasyfikacji.

Podstawowym elementem konwolucyjnych sieci neuronowych jest **warstwa konwolucyjna**, znana również jako operacja splotu. W tej warstwie stosowane są małe macierze, zwane filtrami lub kernelami, które przesuwane są po obrazie wejściowym. Każdy filtr jest używany do wydobywania określonych cech obrazu, takich jak krawędzie, tekstury czy wzory. Proces ten polega na wykonywaniu operacji matematycznych, takich jak mnożenie i dodawanie, na wartościach pikseli w obrębie filtru. W wyniku tego procesu powstaje mapa cech, która jest następnie używana do dalszej analizy.

Konwolucje są zdefiniowane przez dwa kluczowe parametry:

- Rozmiar wycinków (patches) wyodrębnionych z danych wejściowych. Zazwyczaj są to wymiary  $3 \times 3$  lub  $5 \times 5$ .
- Głębokość wyjściowej mapy cech, czyli liczba filtrów obliczonych przez konwolucję. Często na liczbę filtrów wybiera się potęgę dwójki, co jest związane z zaletami w zakresie obliczeń i zarządzania pamięcią.

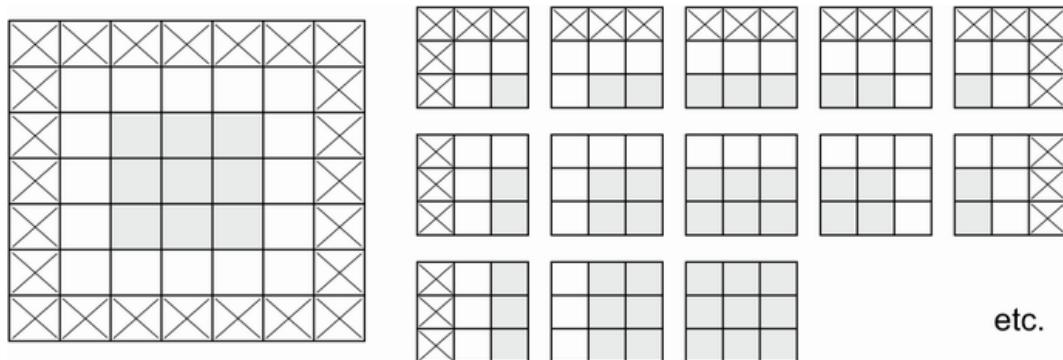
Według przykładu (z pozycji [8]) z mapą cech o wymiarach  $5 \times 5$  (łącznie 25 płytka) istnieje tylko 9 płytka, wokół których można umieścić okno  $3 \times 3$ , tworząc siatkę  $3 \times 3$  (rys. 1.4). W związku z tym mapa cech wyjściowych będzie miała rozmiar  $3 \times 3$ . Mapa ta kurczy się nieco: dokładnie o dwie płytki wzdłuż każdego wymiaru, jak w tym przypadku.



**Rysunek 1.4:** Prawidłowe położenia latek  $3 \times 3$  na mapie cech wejściowych  $5 \times 5$  [8]

Aby zaradzić problemowi utraty informacji na brzegach obrazu, można zastosować **padding (wypełnienie)**. Polega na dodaniu odpowiedniej liczby wierszy

i kolumn po każdej stronie mapy cech wejściowych, aby umożliwić umieszczenie centralnych okien konwolucyjnych wokół każdej płytki wejściowej (rys. 1.5). Dla okna  $3 \times 3$  dodaje się jedną kolumnę po prawej, jedną kolumnę po lewej, jeden wiersz na górze i jeden wiersz na dole. Dla okna  $5 \times 5$  dodaje się dwa wiersze.



**Rysunek 1.5:** Dopełnianie (padding) wejścia o wymiarach  $5 \times 5$ , aby móc wyodrębnić 25 latek  $3 \times 3$ . [8]

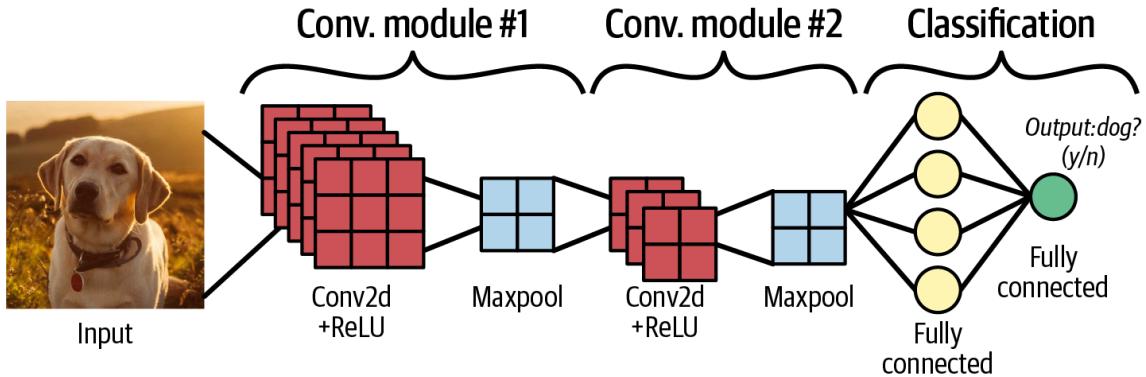
Istnieją różne metody wypełniania, takie jak zero padding (wypełnianie zerami), reflection padding (wypełnianie lustrzanym odbiciem) i border padding (wypełnianie ostatnią wartością).

Inną kluczową warstwą w konwolucyjnych sieciach neuronowych jest **warstwa łącząca (ang. pooling layer)**. Jej głównym zadaniem jest redukcja wymiarów map cech, co zmniejsza liczbę parametrów oraz obciążenie obliczeniowe modelu. Najpopularniejszą metodą w tej warstwie jest MaxPooling, która wybiera największą wartość z określonego obszaru mapy cech. Przykładowo, stosując filtr o wymiarach  $2 \times 2$ , każda czwórka pikseli jest redukowana do jednego, który ma najwyższą wartość.

Ostatnim rodzajem warstwy w konwolucyjnych sieciach neuronowych jest **warstwa w pełni połączona**. W tej warstwie każda jednostka jest połączona z wszystkimi jednostkami w poprzedniej warstwie. Dzięki tej warstwie można połączyć wydobyte cechy z warstw konwolucyjnych i poolingowych, aby uzyskać końcową klasyfikację lub regresję. Warstwa w pełni połączona zazwyczaj stosuje funkcje aktywacji, takie jak ReLU, aby poprawić wydajność modelu.

Rysunek 1.6 ilustruje jak CNN reprezentuje obraz pieska w formie macierzy, następnie dwa moduły konwolucyjne wydobywają przydatne cechy, które są przekazywane do dwóch ostatnich warstw w pełni połączonych, które przewidują, czy obraz przedstawia psa, czy nie. CNN są szczególnie skuteczne w zadaniach takich jak rozpoznawanie obrazów, ponieważ potrafią przetwarzać informacje zakodowane w obrazach, gdzie obrazy są reprezentowane jako macierze (mapy cech wejściowych). Dane wejściowe są następnie „konwolutowane” przez różne warstwy konwolucyjne w sieci. Kon-

wolutowanie jest procesem, który wydobywa informacje z macierzowej reprezentacji obrazów i tworzy nowe cechy, które uchwytyają bardziej subtelną informację o obrazie. Konwolucja pozwala również na spłaszczenie i kompresję informacji w obrazie, co jest efektywne pod względem obliczeniowym. [9]



**Rysunek 1.6:** Zastosowanie CNN do zaganienia klasyfikacji [9]

Konwolucyjne sieci neuronowe znalazły szerokie zastosowanie w różnych dziedzinach. Używane są w rozpoznawaniu obrazów, klasyfikacji obiektów, analizie medycznej, a także w systemach rozpoznawania pisma ręcznego, czyli dla przykładu w tworzeniu systemów autonomicznych pojazdów, analizie sytuacji drogowych oraz przewidywaniu ryzyka wystąpienia chorób. Ich zdolność do dokładnego wydobywania cech z obrazów i efektywnej klasyfikacji sprawia, że są one niezwykle wszechstronne i potężne w analizie danych wizualnych.[10] [11]

Innymi powszechnie stosowanymi architekturami są rekurencyjne sieci neuronowe (RNN) oraz splotowe sieci neuronowe (SNN). **Rekurencyjne sieci neuronowe** są sieciami bardzo dobrze przystosowanymi do użytku przy danych sekwencyjnych, czyli takich, które mają naturalny porządek czasowy lub sekwencyjny (tłumaczenie, analiza i generowanie tekstu, rozpoznawanie mowy itp.). **Splotowe sieci neuronowe** mają zastosowanie w dziedzinach wymagających przetwarzania danych w czasie rzeczywistym, takich jak rozpoznawanie wzorców w sygnałach sensorycznych, robotyka czy symulacje biologiczne. Dzięki ich zdolności do efektywnego przetwarzania informacji impulsowej mogą być używane w systemach, które wymagają niskiego opóźnienia i wysokiej precyzji. O RNN oraz SNN więcej można przeczytać między innymi w pozycjach [12] [13] [8] [14].

### 1.3. Normalizacja danych

**Normalizacja danych**, czyli ich przeskalowywanie z różnych zakresów do ustalonego zakresu, np.  $[0,1]$ , jest istotnym elementem wstępnego przetwarzania. [15] Dzięki

normalizacji eliminowane są skrajności w danych, co przekłada się na bardziej stabilne oraz szybsze procesy uczenia.

Dane nieznormalizowane mogą zawierać kolumny z wartościami w bardzo różnych zakresach, co może wpływać na efektywność algorytmów. Większość innych algorytmów zyskuje na normalizacji, gdyż zapewnia ona, że wszystkie cechy mają porównywalne zakresy, co może wpłynąć pozytywnie na efektywne uczenie się algorytmów i zapobiec dominacji cech o większych wartościach. W przypadku algorytmów opartych na odległościach, takich jak k-najbliżsi sąsiedzi, normalizacja jest szczególnie ważna, ponieważ zapewnia, że wszystkie cechy podczas obliczeń odległości są traktowane sprawnie. Dodatkowym plusem jest przyspieszenie procesu treningowego. [16]

Jedna z najpopularniejszych metod normalizacji to **Batch Normalization**, która zwiększa prędkość i dokładność treningu, poprzez zapobieganie problemom z aktywacjami, które mogą stać się zbyt małe (zanikać) lub zbyt duże (eksplodować). [17]

Wpływ normalizacji na uczenie sieci jest zatem fundamentalny. Przyspiesza ona konwergencję, stabilizuje proces uczenia oraz zwiększa ogólną wydajność sieci neuronowych. Dzięki normalizacji modele stają się bardziej odporne na problemy związane z niestabilnymi gradientami i nadmiernym dopasowaniem.

### 1.4. Nadmierne niedouczenie i przeuczenie

**Nadmierne niedouczenie (underfitting)** oraz **przeuczenie (overfitting)** są uznawane za jedne z głównych wyzwań w uczeniu głębokim, szczególnie w zadaniach związanych z detekcją obrazów o wysokiej złożoności danych. Zjawisko niedouczenia występuje, gdy sieć nie dysponuje wystarczającą mocą wyrażeniową do modelowania danych treningowych. W takim przypadku nie jest możliwe uchwycenie istotnych wzorców w danych treningowych, co prowadzi do problemów z poprawnym działaniem na nowych danych, a wyniki osiągane zarówno na danych treningowych, jak i testowych są słabe. W przypadku wystąpienia takich objawów zaleca się zastosowanie bardziej złożonej sieci z większą liczbą perceptronów. Z kolei przeuczenie występuje, gdy model jest zbyt skomplikowany w stosunku do dostępnej liczby danych treningowych lub gdy “zapamiętuje” nieistotne szczegóły i szum obecny w danych treningowych, które nie są reprezentatywne dla ogólnych wzorców. W efekcie model osiąga dobre wyniki na danych treningowych, ale napotyka trudności z generalizowaniem na nowych danych. Innymi słowy, sieć dostosowuje się do każdego zakamarka, skrętu i niuansu danych treningowych, co sprawia, że dobrze radzi sobie z danymi treningowymi, kosztem wydajności na danych testowych.

Aby wykryć, czy struktura modelu może mieć problem z overfittingiem, należy podzielić próbę danych na zbiór treningowy i testowy. Model jest trenowany na danych treningowych, a następnie oceniany na danych testowych w celu oszacowania jego wydajności. Jeśli model potrafi prawidłowo rozpoznawać wzorce, które uogólniają się z danych treningowych na cały zestaw danych, istnieje większe prawdopodobieństwo, że ta sama struktura modelu będzie dobrze generalizować również na populację. Strategie zapobiegania overfittingowi obejmują między innymi:

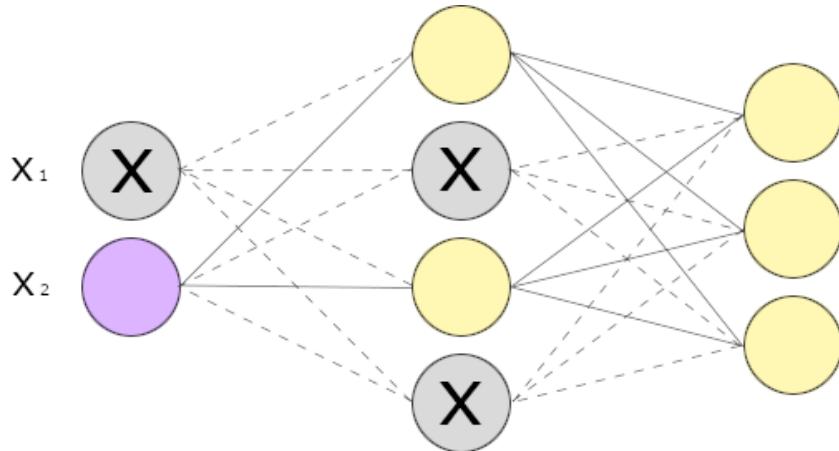
- **Walidację krzyżową**, która jest alternatywą dla zwykłego podziału na zestaw uczący i testowy, gdyż dzieli zbiór danych na k podzbiorów podobnych rozmiarów. Model trenowany jest k razy, używając po kolejne każdy z podzbiorów jako zbiór walidacyjny podczas gdy pozostałe k-1 podzbiorów służą jako zbiór treningowy. Wpływa to pozytywnie na stabilność i wiarygodność wyników, [18] [19]
- Metody **regularyzacji**, takie jak wprowadzenie kar za zbyt duże wartości wag modelu, opóźniają początek overfittingu, zmuszając model do nauki bardziej ogólnych reguł. Regularyzacja ogranicza zdolność modelu do nauki szczegółów specyficznych dla danych treningowych. Jedną z powszechnie używanych metod regularyzacji jest Dropout,
- **Wczesne zatrzymywanie (Early Stopping)** jest techniką, która powoduje zakończenie treningu, gdy tylko błąd walidacyjny zaczyna rosnąć, nawet jeśli błąd treningowy nadal maleje. Pozwala to uniknąć nadmiernego dopasowania do danych treningowych. [13]

Podsumowując, zjawisko niedopasowania występuje, gdy model nie jest w stanie odpowiednio odwzorować wzorców w danych treningowych, co prowadzi do problemów z generalizowaniem na nowe dane. Z kolei overfitting ma miejsce, gdy model dobrze radzi sobie z danymi treningowymi, ale nie z nowymi danymi, ponieważ nauczył się zbyt wielu szczegółów specyficznych dla danych treningowych, które mogą nie być obecne w szerszej populacji.[20] [21] [22] [23]

## 1.5. Regularyzacja danych

Techniki regularyzacji odgrywają kluczową rolę w poprawie generalizacji modeli sieci neuronowych i w zapobieganiu nadmiernemu dopasowaniu. Innymi słowy, jest stosowana w celu kontrolowania złożoności modelu i zmniejszenia ryzyka przeuczenia, w którym model uczy się zbyt dokładnie szczegółów danych treningowych, przez co słabnie jego zdolność do generalizacji na nowe, nieznane dane.

**Dropout** jest jedną z najbardziej efektywnych technik regularizacji dla głębowich sieci neuronowych. Polega na tym, że część węzłów zasilających warstwę zostaje losowo porzuconych, co oznacza, że są ignorowane podczas aktualizacji wag. Wyłączenie części neuronów zapobiega koadaptacji i jest skuteczną metodą zwiększającą moc predykcyjną modelu. [13] Tę metodę obrazuje rysunek 1.7.



**Rysunek 1.7:** Schemat wyłączania węzłów przez metodę Dropout

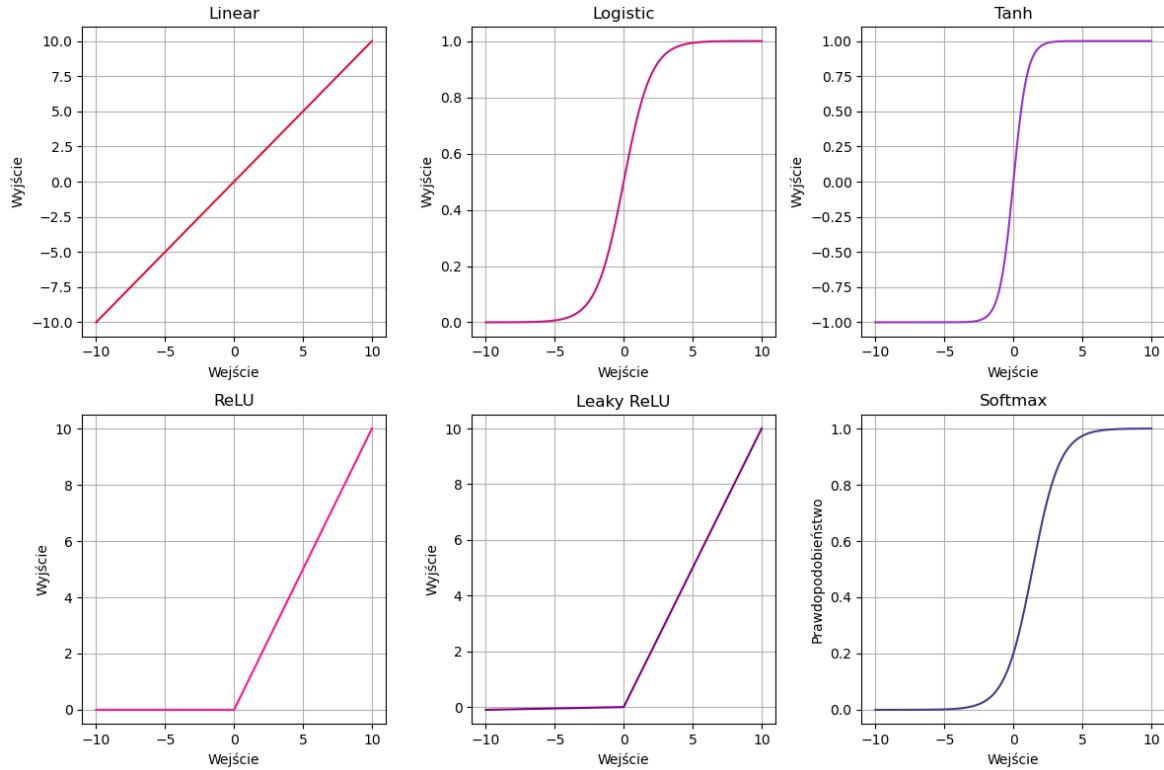
Hiperparametr  $p$ , zwany współczynnikiem dropout, zazwyczaj mieści się w zakresie od 10% do 50%. Technika ta zwiększa odporność sieci na drobne zmiany w danych wejściowych i prowadzi do bardziej odpornych modeli, które lepiej się uogólniają. Jego stosowanie może znacznie poprawić wyniki modelu, szczególnie w przypadku dużych modeli, jednak dzieje się to kosztem spowolnienia procesu konwergencji. Warto również wspomnieć, że podczas treningu generowana jest unikalna wersja sieci neuronowej na każdym kroku, co uśrednia wyniki z wielu różnych wersji sieci, implikując zwiększenie jej zdolności do generalizacji. Podczas analizowania wyników, trzeba być ostrożnym przy porównywaniu straty treningowej i walidacyjnej, gdyż dropout jest aktywny tylko podczas treningu, a nie w czasie testowania. Istnieje rozszerzenie techniki Dropout do **MC Dropout**, pozwalające na szacowanie niepewności modelu poprzez losowe wyłączanie neuronów także podczas fazy predykcji.

Jedną z innych powszechnie stosowanych technik regularizacji jest **regularyzacja L2**, która ogranicza wielkość wag połączeń w sieci neuronowej. Dzięki tej technice wagi są karane za ich zbyt dużą wartość, co zapobiega nadmierнемu dopasowaniu. W praktyce w Keras regularyzacja L2 jest stosowana poprzez dodanie współczynnika regularizacji do wag warstwy, co przyczynia się do redukcji wartości wag. Alternatywnie, **regularyzacja L1** może być używana do uzyskania modelu o rzadkich wagach, czyli takich gdzie wiele z nich jest równych zeru.

Podsumowując, techniki regularyzacji, są nieocenionymi narzędziami w kontekście głębokiego uczenia, gdyż pomagają w poprawie generalizacji modeli i w zapobieganiu nadmierнемu dopasowaniu. [3] [23]

## 1.6. Funkcje aktywacji i optymalizatory

Kluczową rolę w sieciach neuronowych odgrywają **funkcje aktywacji**, które wprowadzają nieliniowość, umożliwiając modelom uczenie się bardziej złożonych funkcji. Gdyby funkcje aktywacji były pominięte, sieć neuronowa działałaby jako klasyfikator liniowy, ucząc się funkcji będącej liniową kombinacją danych wejściowych. Funkcje aktywacji przekształcają ważone sumy wejść w nieliniowy sposób, co pozwala sieci neuronowej efektywnie rozdzielać i klasyfikować dane. Podstawowym celem funkcji aktywacji jest wprowadzenie nieliniowości do wyjścia neuronu, co umożliwia modelowi uchwycenie bardziej złożonych wzorców w danych. Funkcja aktywacji decyduje, czy dany neuron powinien być aktywowany, na podstawie obliczonej ważonej sumy wejść oraz dodanego przesunięcia (bias). Rysunek 1.8 ilustruje kilka powszechnie stosowanych funkcji aktywacji. Każda z tych funkcji ma swoje unikalne cechy, które wpływają na sposób, w jaki sieć neuronowa przetwarza dane. [3] [5]



Rysunek 1.8: Funkcje aktywacji

Po przedstawieniu wykresów funkcji aktywacji warto zwrócić uwagę na ich różnice oraz praktyczne zastosowanie w sieciach neuronowych. Tabela 1.2, która szczegółowo opisuje różne funkcje aktywacji, ich wzory matematyczne oraz zastosowanie pomaga w zrozumieniu, jak każda z funkcji wpływa na działanie sieci neuronowej oraz jakie ma zalety i wady w kontekście różnych zadań i architektur sieciowych. Funkcje takie jak ReLU są szeroko stosowane, gdyż przyspieszają trening dzięki swojej prostocie i efektywności obliczeniowej, a także minimalizują problem zanikania gradientu, który może występować przy użyciu funkcji takich jak sigmoid czy tanh. Funkcja liniowa jest używana w regresji, gdy chcemy przewidywać liczby. Funkcja logistyczna (sigmoid) jest przydatna w klasyfikacji binarnej, bo daje wyniki w zakresie od 0 do 1, co oznacza prawdopodobieństwo przynależności do jednej z dwóch klas. Tangens hiperboliczny (tanh) jest używany, gdy potrzebujemy symetrii wokół zera i gdy dane muszą być w zakresie od -1 do 1. Leaky ReLU pomaga uniknąć problemów z "martwymi neuronami", które mogą wystąpić w zwykłym ReLU, pozwalając na małe, ale niezerowe wartości. Softmax jest używany, gdy chcemy klasyfikować do jednej z wielu klas, bo przekształca wyniki na prawdopodobieństwa, które sumują się do 1. Funkcje aktywacji decydują, jak dobrze model radzi sobie z danymi i jak skutecznie się uczy.

**Tabela 1.2:** Funkcje aktywacji

Nazwa	Typ	Wzór funkcji	Opis
Linear	Wyjściowa	$f(x) = x$	<ul style="list-style-type: none"> <li>— Pozostawia wartości bez zmian</li> <li>— Rzadko używana</li> </ul>
Logistic	Wyjściowa	$f(x) = \frac{1}{1 + e^{-x}}$	<ul style="list-style-type: none"> <li>— Zmniejsza wartości do przedziału 0-1</li> <li>— często używana w klasyfikacji binarnej</li> </ul>
Tangent Hyperbolic	Ukryta	$f(x) = \text{tgh}(x)$	<ul style="list-style-type: none"> <li>— Pomaga w "wycentrowaniu" danych przez zbliżenie średniej do 0</li> </ul>
ReLU	Ukryta	$f(x) = \max(0, x)$	<ul style="list-style-type: none"> <li>— Ustawia wartości ujemne na 0</li> <li>— szybsza funkcja niż sigmoid i tanh</li> </ul>
Leaky ReLU	Ukryta	$f(x) = \begin{cases} x & \text{dla } x > 0 \\ 0.01x & \text{dla } x \leq 0 \end{cases}$	<ul style="list-style-type: none"> <li>— Mnoży wartości ujemne przez 0.01</li> <li>— Kontrowersyjna wersja ReLU, marginalizuje zamiast eliminować wartości ujemne</li> </ul>
Softmax	Wyjściowa	$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$	<ul style="list-style-type: none"> <li>— Zapewnia, że suma wartości wyjściowych wynosi 1.0</li> <li>— Przydatna do klasyfikacji wieloklasowej</li> </ul>

Wybór odpowiedniej funkcji aktywacji ma kluczowe znaczenie dla efektywności uczenia się i wydajności sieci neuronowej, ponieważ wpływa na zdolność modelu do nauki złożonych wzorców oraz stabilność procesu treningowego. Funkcje aktywacji wprowadzają nieliniowość do modelu, co pozwala na uchwycenie bardziej skomplikowanych relacji w danych. Bez takich funkcji, sieć neuronowa ograniczałaby się do modelowania tylko funkcji liniowych, co znacznie obniża jej zdolność do rozwiązywania problemów wymagających zrozumienia nieliniowych zależności. Dodatkowo wybór funkcji aktywacji wpływa na szybkość konwergencji oraz stabilność procesu uczenia się.

**Optymalizatory** są ważnymi elementami w procesie trenowania sieci neuronowych, ponieważ służą do dostosowywania parametrów modelu w celu minimalizacji funkcji straty. Wybór odpowiedniego algorytmu optymalizacji ma istotne znaczenie dla efektywności i skuteczności uczenia się, wpływając na tempo konwergencji oraz jakość końcowego rozwiązania. W tabeli 1.3 przedstawiono porównanie różnych algorytmów optymalizacji pod kątem szybkości i jakości konwergencji.

**Tabela 1.3:** Porównanie algorytmów optymalizacji

Klasa	Szybkość konwergencji	Jakość konwergencji
SGD	*	***
SGD(momentum=...)	**	***
SGD(momentum=..., nesterov=True)	**	***
Adagrad	***	* (kończy zbyt wcześnie)
RMSprop	***	** lub ***
Adam	***	** lub ***
AdaMax	***	** lub ***
Nadam	***	** lub ***

**Stochastic Gradient Descent (SGD)** jest podstawowym algorytmem optymalizacji, który charakteryzuje się stosunkowo wolnym tempem konwergencji, ale zapewnia stabilność w końcowej fazie uczenia się. Dodanie momentum do SGD przyspiesza konwergencję poprzez nadanie kierunku ruchowi, co przyczynia się do szybszego osiągania wyników. Włączenie gradientu Nesterova w połączeniu z momentum dodatkowo poprawia szybkość konwergencji oraz efektywność przez lepsze przewidywanie kierunku optymalizacji. **AdaGrad** jest algorymem, który adaptacyjnie dostosowuje tempo uczenia się do każdego współczynnika. Problematyczne może być zbyt szybkie zmniejszanie tempa uczenia się, co prowadzi do zbyt wcześniego zatrzymania procesu uczenia się. **RMSprop** jest poprawką do AdaGrad, redukując jego problematykę poprzez wprowadzenie mechanizmu wygładzania w historycznych gradientach. W re-

zultacie RMSprop nie cierpi na problem zbyt szybkiego zmniejszania tempa uczenia się i jest bardziej stabilny w długotrwałym procesie uczenia. **Adam** łączy zalety adaptacyjnego tempa uczenia się i momentum, co czyni go bardzo efektywnym w przyspieszaniu konwergencji oraz zapewnieniu stabilności. **AdaMax** i **Nadam** są podobne do Adama, oferując również szybkie tempo konwergencji i dobrą jakość wyników, ale z pewnymi różnicami w sposobie adaptacji. Algorytmy optymalizacji różnią się szybkością i jakością konwergencji, gdzie Adam i jego warianty oferują najlepszą równowagę między szybkością a stabilnością, podczas gdy inne metody, takie jak SGD czy AdaGrad, mają swoje specyficzne zalety i ograniczenia. [17] [23]

## 1.7. Augmentacja danych

**Augmentacja danych** jest techniką regularyzacji stosowaną w celu zwiększenia zbioru danych treningowych poprzez generowanie nowych przykładów z oryginalnych obrazów. Proces ten polega na zastosowaniu różnych transformacji geometrycznych, które nie zmieniają etykiet klas, lecz modyfikują wygląd obrazów w sposób, który może poprawić generalizację modelu. [11]

W ramach augmentacji danych dostępne są transformacje:

- **Rotacja:** Obracanie obrazu o określony kąt,
- **Przesunięcie:** Przesuwanie obrazu w poziomie lub pionie,
- **Zmiana skali:** Skalowanie obrazu, powiększanie lub pomniejszanie,
- **Odbicie lustrzane:** Odbicie obrazu w poziomie lub pionie,
- **Przycinanie:** Wycinanie określonego obszaru obrazu,
- **Zmienność jasności:** Zmianianie jasności obrazu,
- **Zmienność kontrastu:** Zmianianie kontrastu obrazu,
- **Zmienność nasycenia:** Zmianianie nasycenia kolorów obrazu,
- **Dodawanie szumów:** Dodawanie szumów do obrazu,
- **Zmiana kolorów:** Losowa zmiana kolorów obrazu,
- **Zmiana rozmiaru:** Zmiana rozmiaru obrazu do określonych wymiarów,
- **Filtracja:** Stosowanie filtrów takich jak wygładzanie, ostrość itp.

Augmentacja danych jest techniką, która może pomóc w redukcji nadmiernego dopasowania do zbioru treningowego. Na przykład, jeżeli w danych treningowych znajdują się wyłącznie koty skierowane w prawo, sieć CNN może nie nauczyć się, że koty skierowane w lewo również są kotami. Poprzez zastosowanie różnych technik augmentacji danych, takich jak obracanie, przycinanie, czy lustrzane odbicie, można wprowadzić dodatkowe reprezentacje tego samego obiektu do zbioru danych, co może pomóc sieci

CNN w nauce generalizowania detekcji obiektów. Standaryzacja obrazów sprawia, że praca z danymi staje się łatwiejsza, zapewniając, że obrazy mają wysokość i szerokość zbliżone do siebie. W tym etapie wstępnego przetwarzania obrazy są zmieniane rozmiarowo, aby mieściły się w określonym zakresie szerokości i wysokości. [9]



## Rozdział 2

### Wstępna analiza i interpretacja zbioru danych

Część praktyczna pracy bazuje na zbiorach obrazów 50-ciu znanych artystów. Celem jest przetestowanie różnych modeli i próba stworzenia modelu rozpoznawania dzieł malarskich z wykorzystaniem technik uczenia głębokiego.

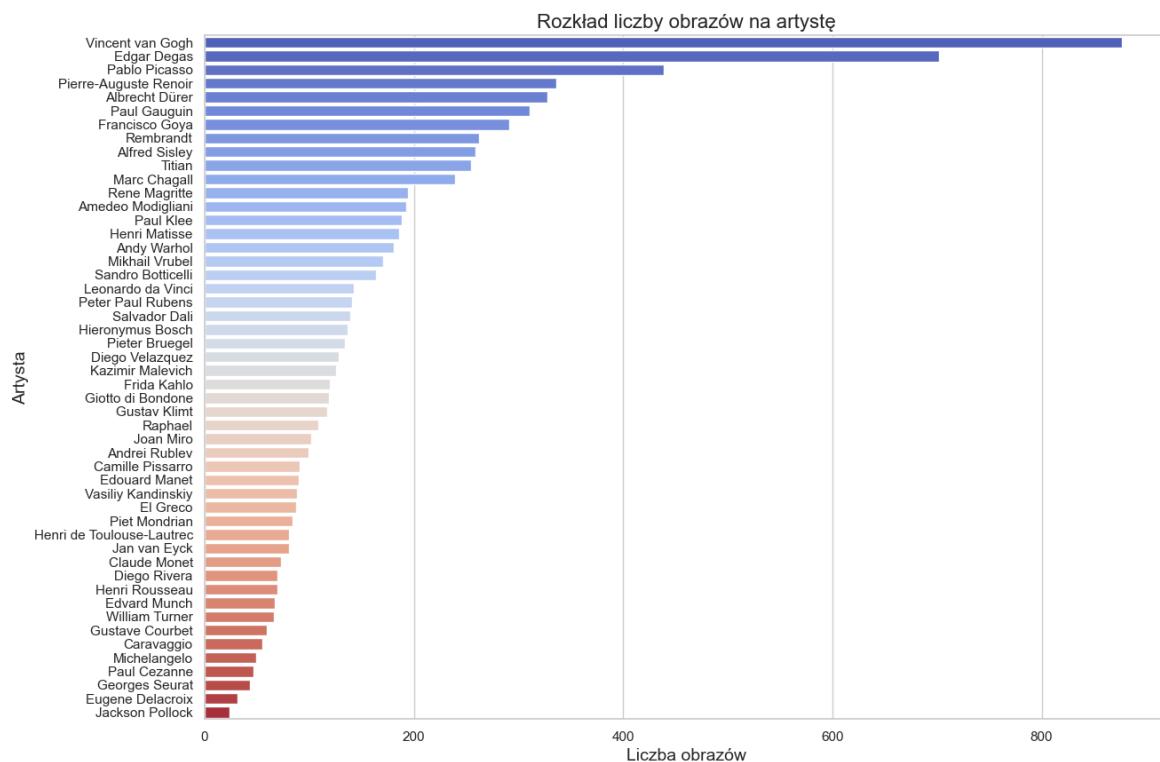
#### 2.1. Przedstawienie danych

Rysunek 2.1 opisuje podzbiór (5 z 50-ciu wierszy) danych opisujących artystów, których obrazy w późniejszym etapie pracy są wykorzystane do stworzenia modelu uczenia głębokiego. Tabela jest pomocą we wstępnej analizie danych oraz w prawidłowemu zrozumieniu ich zawartości. Ilustrując analizowany zbiór danych, można poznać nazwiska artystów, których obrazy będą analizowane w późniejszej części. Dostępne są również informacje opisujące malarzy takie jak lata działalności, gatunek, narodowość, bio, link do wikipedii. Istotna dla nas jest ostatnia kolumna reprezentująca liczbę obrazów, gdyż informacja ta wykorzystana będzie później do wstępnego zrozumienia danych poprzez wykresy podsumowujące dane.

		id	name	years	genre	nationality	bio	wikipedia	paintings
0	0	Amedeo Modigliani	1884 - 1920		Expressionism	Italian	Amedeo Clemente Modigliani (Italian pronunciation: [a'mɛdɛo klemente mɔdilja'ni]; French: [amə'dɛo klemɑ̃tɛ mɔdiljan]) was an Italian painter and sculptor who spent most of his career in France. He had a major influence on the art world and is considered one of the precursors of Cubism. His portraits of artists such as Pablo Picasso and Jean Cocteau, and of his numerous lovers, are among his best-known works.	<a href="http://en.wikipedia.org/wiki/Amedeo_Modigliani">http://en.wikipedia.org/wiki/Amedeo_Modigliani</a>	193
1	1	Vassily Kandinsky	1866 - 1944		Expressionism, Abstractionism	Russian	Vassily Wassilyevich Kandinsky (Russian: Васи́лий Васи́льевич Ка́ндиски; German: Wassily Wassiljewitsch Kandinskiy) was a Russian painter and art theorist. He is widely regarded as one of the most important figures in modern art. He was a central figure in the development of abstract painting.	<a href="http://en.wikipedia.org/wiki/Wassily_Kandinsky">http://en.wikipedia.org/wiki/Wassily_Kandinsky</a>	88
2	2	Diego Rivera	1886 - 1957		Social Realism, Muralism	Mexican	Diego María de la Concepción Juan Nepomuceno Esteban de la Rivera y Barrientos Acosta y Rodríguez was a Mexican painter, known for his large murals depicting social and political subjects. He was a leading figure in the Mexican mural movement.	<a href="http://en.wikipedia.org/wiki/Diego_Rivera">http://en.wikipedia.org/wiki/Diego_Rivera</a>	70
3	3	Claude Monet	1840 - 1926		Impressionism	French	Oscar-Claude Monet (French: [ɔksaʁ klawd mɔnɛ]; 14 July 1840 – 5 December 1926) was a French Impressionist painter. He is best known for his series of landscapes, particularly those of the River Seine at Giverny.	<a href="http://en.wikipedia.org/wiki/Claude_Monet">http://en.wikipedia.org/wiki/Claude_Monet</a>	73
4	4	Rene Magritte	1898 - 1967		Surrealism, Impressionism	Belgian	René François Ghislain Magritte (French: [ʁəne fʁɑ̃swa ɡisɛ̃ɛ̃ maɡʁit]; 21 November 1898 – 15 August 1967) was a Belgian surrealist painter. He became well known for his oil paintings, particularly his "surrealist" scenes.	<a href="http://en.wikipedia.org/wiki/René_Magritte">http://en.wikipedia.org/wiki/René_Magritte</a>	194

Rysunek 2.1: Podzbiór danych opisujących artystów

Analizując dostępne dane, od razu można dostrzec niesamowitą różnorodność stylów i technik malarskich, ale również nierówny zbiór danych przypadającą na różnych artystów. Liczbę obrazów przypadających na każdego artystę widać na wykresie na rysunku 2.2.



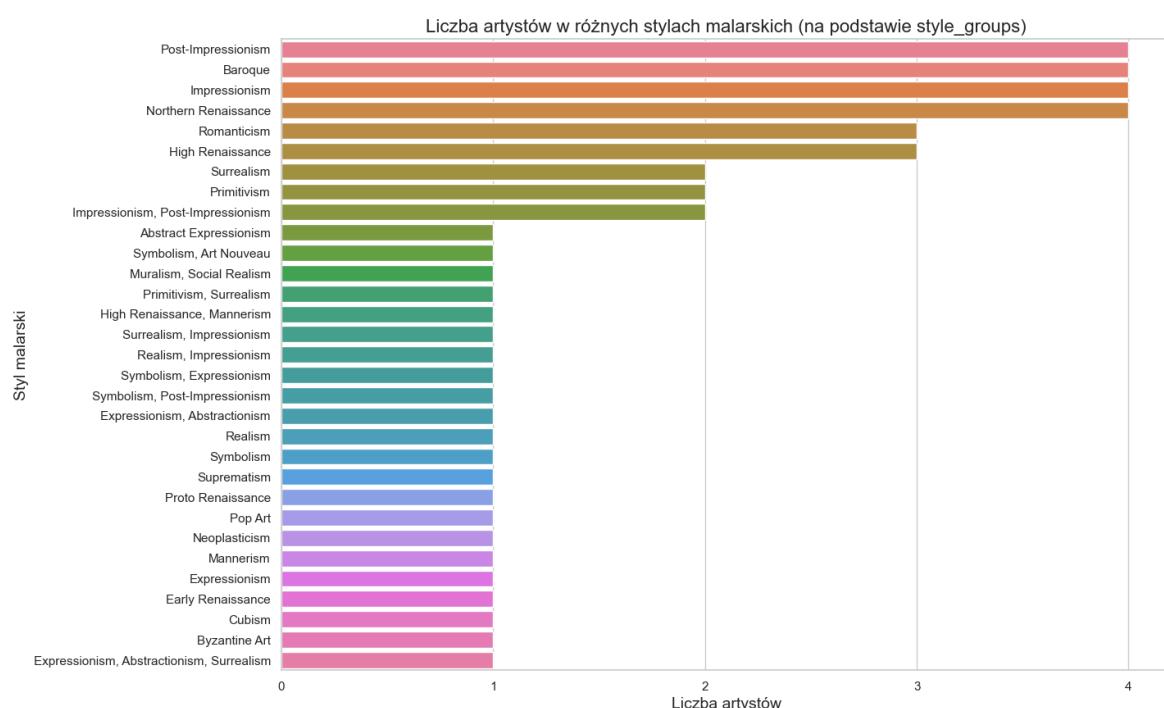
**Rysunek 2.2:** Rozkład liczby obrazów na artystę

Różnica jest ogromna, zaczynając od Vincenta Van Gogha, gdzie liczba danych wynosi 877 obrazów, kończąc na Jacksonie Pollock, gdzie jest to jedynie 24. Tylko 11 z 50-ciu artystów posiada liczbę obrazów większą niż 200. W tabeli 2.1 zostało przedstawione, jak liczba artystów rozkłada się w różnych przedziałach liczby obrazów.

**Tabela 2.1:** Podział artystów według liczby obrazów

Przedział liczby obrazów	Liczba artystów
1 - 49	5
50 - 99	15
100 - 149	12
150 - 199	7
200 - 249	1
250+	10

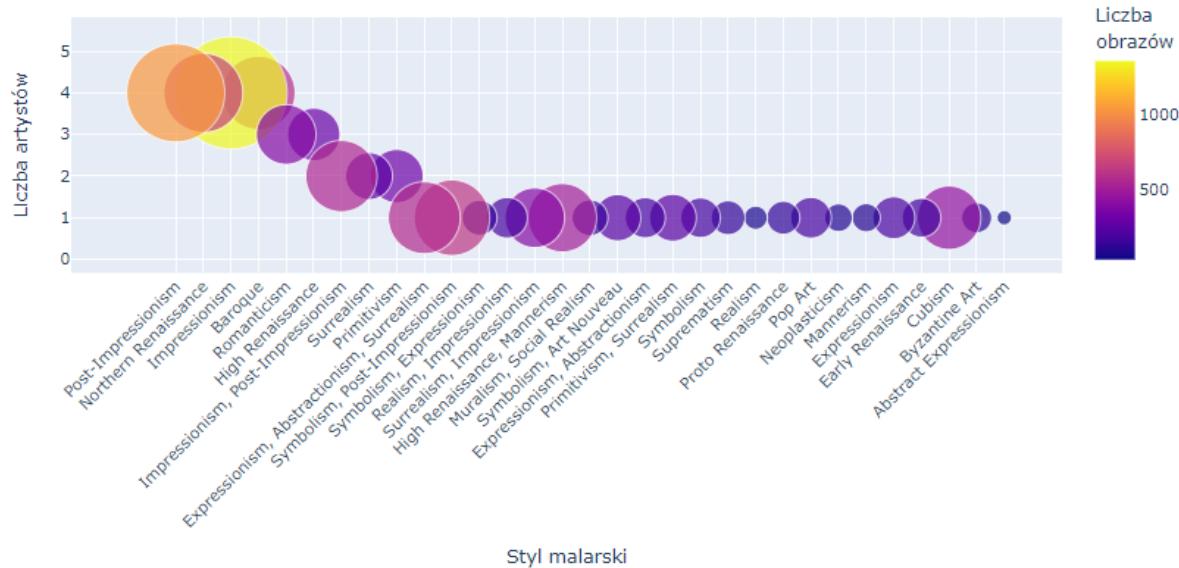
Taka dysproporcja w liczbie obrazów przypadających na poszczególnych artystów ma istotne konsekwencje dla procesu budowy modeli uczenia głębokiego, które omówione zostaną na końcu sekcji we wnioskach. Po dostrzeżeniu różnorodności w liczbie danych przypadających na artystę nasuwa się pytanie jak wygląda rozkład w różnych stylach malarstw. Na rysunku 2.3 dostrzec można, że niektóre gatunki wyróżniają się znacznie. Podejrzewać można, iż wpłynie to na to jak dobrze model nauczy się rozpoznawać jakie style malarstw, posiadając różnicę liczbę danych.



**Rysunek 2.3:** Liczba artystów w różnych stylach malarstw

Idąc dalej w zrozumieniu danych, warto sprawdzić oprócz nierównego rozkładu między ilością gatunków malarstw, jak wygląda liczność w danych gatunkach. Rysunek 2.4 najdokładniej odzwierciedla nierówność w ilości danych przypadających na każdego artystę i gatunek. Im więcej artystów na gatunek — tym więcej obrazów. Jedynie kubizm wyróżnia się jako gatunek reprezentowany w naszym przypadku przez jednego artystę, który posiada liczbę artystów porównywalną do grup z trzema artystami. Jest to Pablo Picasso wyróżniający się dużą liczbą dzieł malarstw (439), jako trzeci po Vincencie Van Gogh'u i Edgarze Degas. Realizm wygląda obok manieryzmu i abstrakcjonizmu dość ubogo. Z kolei ekspresjonizm abstrakcyjny reprezentowany przez wspomnianego wcześniej Jackson'a Pollock'a wyróżnia się znów dużo mniejszą liczbą danych.

Liczba artystów, liczba obrazów i styl malarski



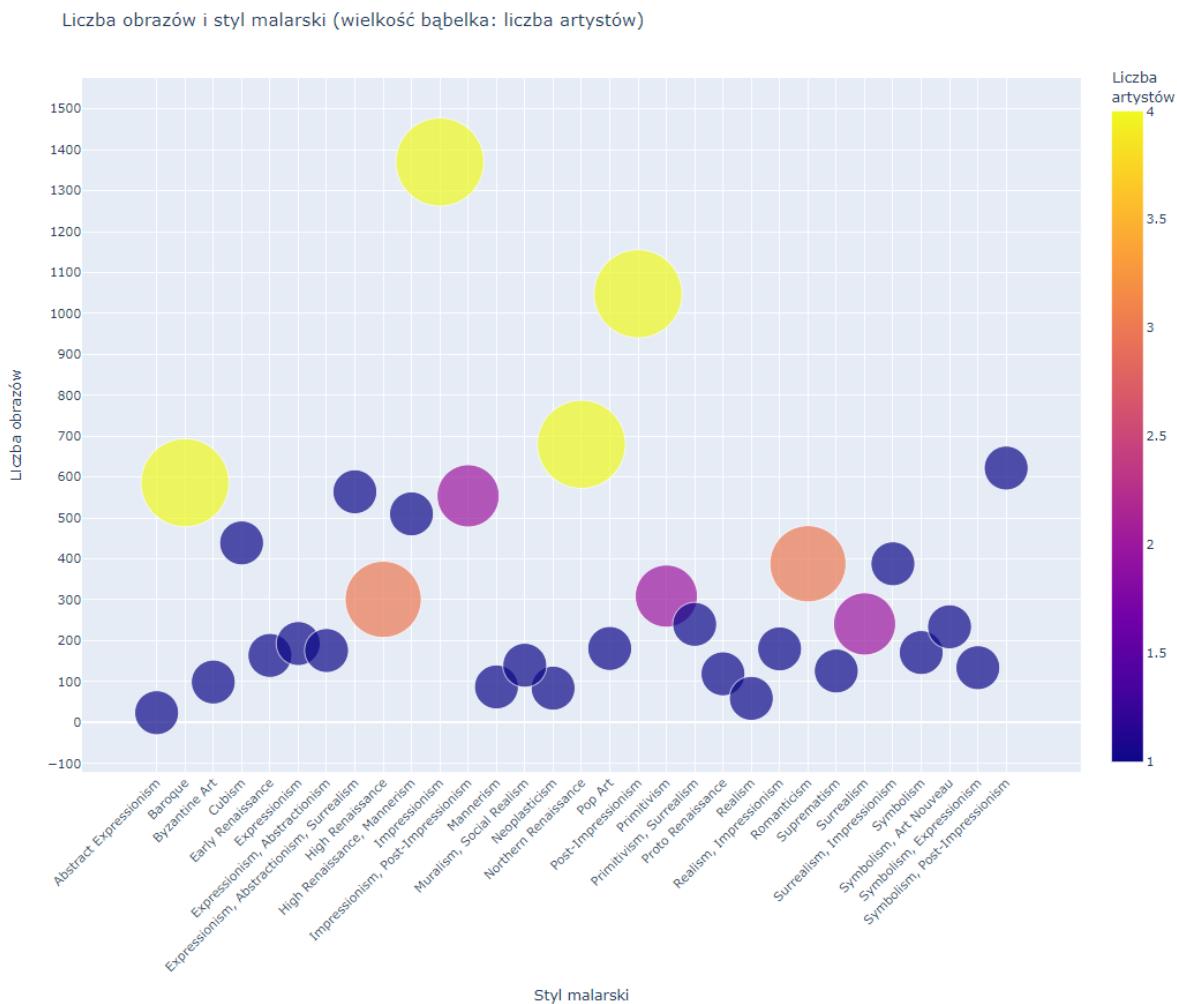
Rysunek 2.4: Liczba artystów i obrazów dla każdej grupy stylu malarskiego

Zestawienie na rys. 2.5 przedstawia sumaryczną liczbę obrazów przypisanych do różnych stylów malarskich, uporządkowaną według liczby obrazów w kolejności malejącej. Najbardziej reprezentowane style to impresjonizm (1370 obrazów) i postimpresjonizm (1048), które wyraźnie dominują w zestawieniu. Mniejsze grupy, takie jak realizm (59) czy abstrakcyjny ekspresjonizm (24), są mniej liczne. Podkreśla to różnorodność zbioru danych.

	Total_Paintings		
Impressionism	1370	Pop Art	181
Post-Impressionism	1048	Symbolism	171
Northern Renaissance	680	Early Renaissance	164
Baroque	586	Suprematism	126
Cubism	439	Primitivism, Surrealism	120
Romanticism	388	Proto Renaissance	119
Symbolism, Post-Impressionism	311	Symbolism, Art Nouveau	117
Primitivism	309	Byzantine Art	99
High Renaissance	301	Realism, Impressionism	90
Impressionism, Post-Impressionism	277	Expressionism, Abstractionism	88
High Renaissance, Mannerism	255	Mannerism	87
Surrealism	241	Neoplasticism	84
Surrealism, Impressionism	194	Muralism, Social Realism	70
Expressionism	193	Symbolism, Expressionism	67
Expressionism, Abstractionism, Surrealism	188	Realism	59

Rysunek 2.5: Liczba obrazów dla każdej grupy stylu malarskiego

Warto przyjrzeć się raz jeszcze różnorodności zbioru, używając drugi już raz wykresu bąbelkowego (Rys. 2.6). Na pierwszy rzut oka wyróżniają się dwie grupy impresjonizm oraz postimpresjonizm, które posiadają najwyższą liczbę obrazów i artystów. Renesans europy północnej i barok są reprezentowane przez czterech artystów każdy, jednak mają liczbę dzieł sztuki porównywalną do pięciu innych grup z niższą liczbą artystów.

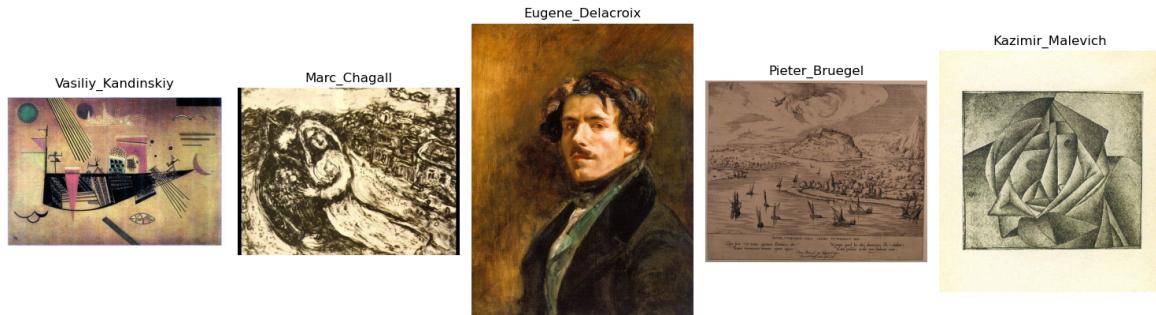


**Rysunek 2.6:** Liczba obrazów i artystów dla każdej grupy stylu malarskiego

Zestaw danych charakteryzuje się dużą różnorodnością zarówno pod względem liczby artystów przypadających na dany gatunek, jak i liczby obrazów przypadających na poszczególnych artystów oraz style. Te czynniki wskazują na wysoką złożoność i potencjalną trudność analizy tego zbioru danych.

## 2.2. Przykładowe obrazy

Patrząc na rysunek 2.7, który przedstawia losowo wybrane ze zbioru obrazy (Vasiliy'ego Kandinsky'ego, Marc Chagalla, Eugène'a Delacroix, Pietera Bruegela oraz Kazimira Malevicha), można dostrzec, że każdy z nich reprezentuje inny styl artystyczny, co sprawia, że różnice między nimi są widoczne.



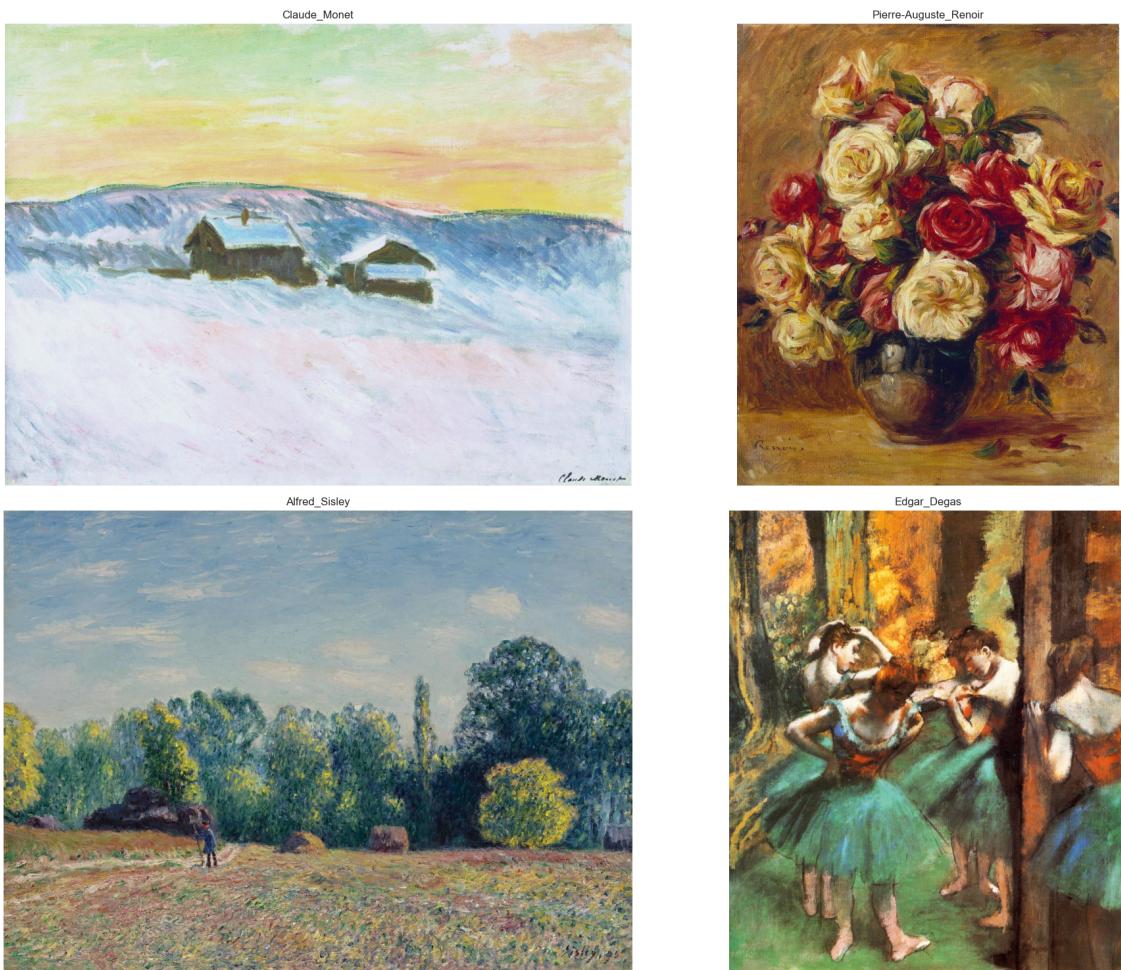
**Rysunek 2.7:** Pięć przykładowych, losowo wybranych obrazów

Analizując te obrazy, można zauważać różnice w technice i tematyce.

- Wasilij Kandinsky, zaliczany do nurtu *ekspresjonizmu i abstrakcjonizmu*, rezygnuje z realizmu na rzecz wyrażania emocji poprzez formę i kolor. Jego prace różnią się od siebie: niektóre cechują się łagodnym, stonowanym stylem, podczas gdy inne są dynamiczne i pełne ruchu, ale wspólnym elementem jest intensywna kolorystyka i silny kontrast. W jego twórczości można dostrzec zarówno obrazy o subtelnym i spokojnym charakterze, jak i te o agresywnej i wysoce abstrakcyjnej formie. Ta siła przekazu różnych emocji wskazuje na bardzo świadomy ruch pędzla i dobór kolorów,
- Marc Chagall, reprezentujący prymitywizm, tworzył obrazy łączące realizm z fantazją. Jego obrazy charakteryzują się prostą formą i różnorodną kolorystyką — od stonowanych czarno-białych tonów po intensywne, żywe barwy. Sztuka Chagalla przypomina naiwne i pełne wyobraźni rysunki dziecięce, w których prostota formy i kolorystyki tworzy wyjątkowy, niemal bajkowy świat,
- Eugène Delacroix, jako przedstawiciel romantyzmu, tworzył obrazy pełne dramatyzmu i silnych emocji. Dynamiczne kompozycje, intensywne kolory i wyrazisty światłocień, tworzą razem pewną głębię i oddają wewnętrzne przeżycia postaci,
- Pieter Bruegel, przedstawiciel baroku, jest znany z realizmu i szczegółowych przedstawień życia codziennego, zwłaszcza z okresu renesansu. W swoich pracach przedstawiał złożoność społeczeństwa i jego obyczajów, w sposób pełny drobnych szczegółów,

- Kazimir Malevich, twórca suprematyzmu, używała podstawowych form geometrycznych i ograniczonej palety kolorystycznej, aby skupić się na formie przekazu w minimalistyczny sposób.

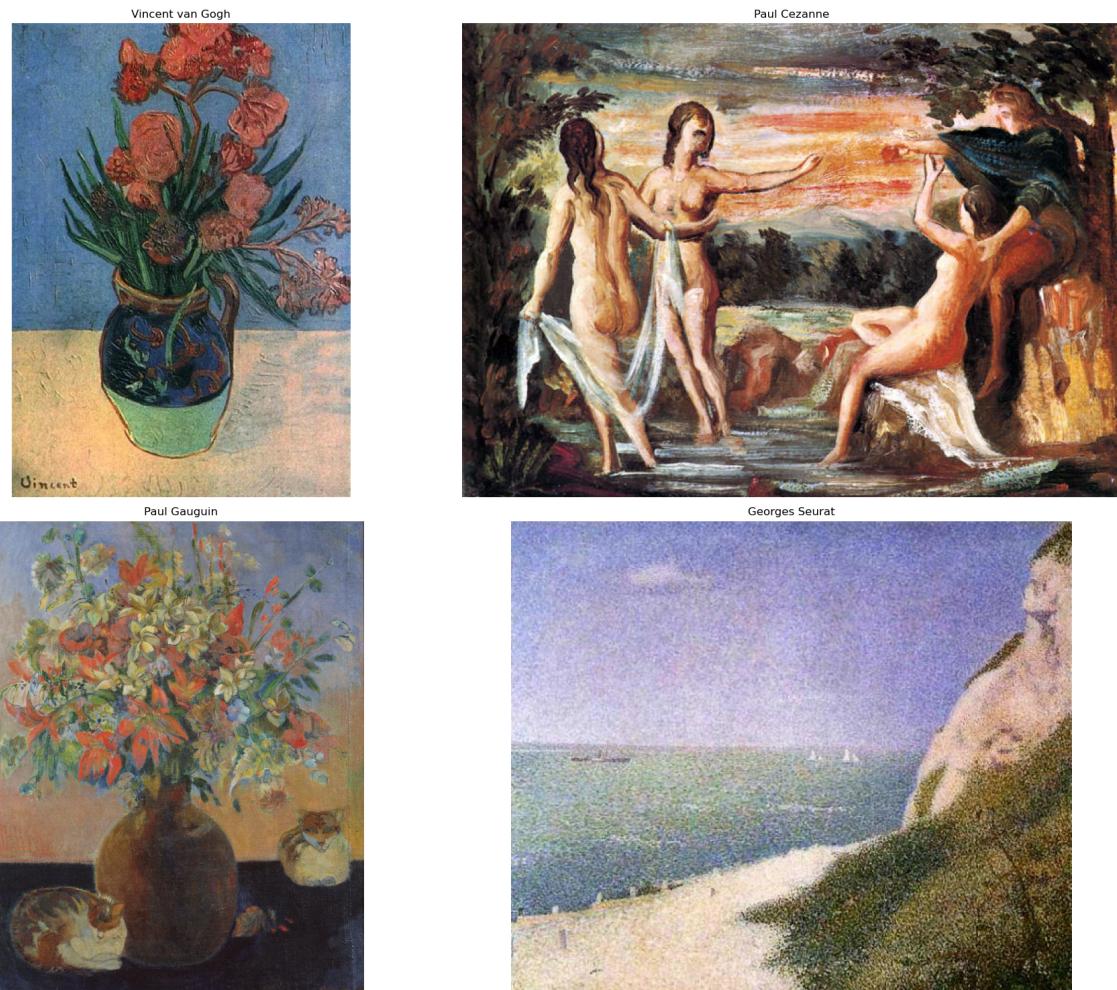
Mimo że Kandinsky i Malevich różnią się nurtem artystycznym, obaj skupiają się na formie i kolorze, dążąc do minimalizmu i abstrakcji. Chagall, Delacroix i Bruegel, mimo różnych technik i tematów, używali kolorów i kompozycji, aby wyrazić emocje i narracje, co podkreśla wspólne cechy ich indywidualnych stylów i wpływów artystycznych. Dla przykładu analizy wewnętrzgatunkowej, biorąc pod lupę obrazy np. impresjonistyczne różnych artystów, na rysunku 2.8 widać podobieństwa płynące z kolorystyki czy delikatnych, nieco rozmazanych liniach. Ważnym elementem jest tutaj przekazanie danego momentu, piękności chwili, czy emocji. Obrazy niemal nie posiadają konturów.



**Rysunek 2.8:** Grupa przykładowych, losowo wybranych obrazów impresjonistycznych

Po okresie impresjonizmu, w dziełach postimpresjonistycznych (widocznych na rys. 2.9) widoczna jest kontynuacja intensywnego użycia koloru i koncentracji na emo-

cjach, ale z wprowadzeniem nowych stylów i technik, takich jak pointylizm Seurata czy strukturalizm Cézanne'a. Dodatkowo poza uchwyceniem piękna chwili, postimprejonisiści dodają więcej własnych emocji, co czyni obrazy przedstawiającymi nie tylko wartość i piękno zewnętrznej rzeczywistość, ale także głębsze, subiektywne odczucia i wewnętrzne przeżycia.



**Rysunek 2.9:** Grupa przykładowych, losowo wybranych obrazów postimpresjonistycznych

Przegląd przykładowych obrazów ukazuje wyraźne różnice w technikach, kompozycjach i użyciu koloru między artystami. Różnorodność ta stanowi wyzwanie dla modelu, który musi nauczyć się rozpoznawać specyficzne style i podejścia artystyczne. Przykłady impresjonizmu i postimpresjonizmu pokazują ewolucję w wyrażaniu emocji i koloru, wymagającą od modelu zdolności do wychwytywania subtelnych różnic między artystami. Te obrazy wskazują na potrzebę odpowiednich technik przetwarzania danych i strategii modelowania, aby skutecznie rozpoznać dzieła malarskie.

### 2.3. Wnioski z wstępnej analizy danych

Taka dysproporcja w liczbie obrazów przypadających na poszczególnych artystów ma istotne konsekwencje dla procesu budowy modeli uczenia maszynowego. Dzięki wstępnej analizie zawartości zbioru danych nasuwają się wnioski, które mogą być przydatne w późniejszej części pracy:

- **Stronniczość w Modelu:** Należy zwrócić uwagę na ogromną różnorodność w ilości obrazów, które przypisane są do poszczególnych artystów. Patrząc dla przykładu na skrajny przypadek Van Gogh'a i Pollock'a, gdzie liczba obrazów wynosi odpowiednio 877 i 24, dostrzec można, że klasy są nierówno reprezentowane pod względem liczby danych. Konsekwencją owej nierówności jest ryzyko faworyzowania przez model klas z większą ilością danych. Oznacza to, że dla pewnych grup model może nauczyć się rozpoznawać obrazy lepiej, a dla innych mieć trudności z identyfikacją,
- **Overfitting i Generalizacja:** Model może przeładować się szczegółami i wzorcami specyficznymi dla artystów z większą liczbą obrazów, co może prowadzić do problemów z generalizacją. W praktyce oznacza to, że model będzie dobrze działał na danych, które przypominają te, na których był trenowany, jednocześnie posiadając słabsze wyniki na nowych, nieznanych danych,
- **Nierównomierność w Ocenie:** Ważne jest pamiętać o fakcie, że liczba danych dla każdej klasy jest bardzo różna. Aby nie wyciągnąć mylnych wniosków o skuteczności modelu, należy patrzeć nie tylko na dokładność ogólną, ale również na wyniki dla poszczególnych klas oddzielnie. Ponadto, aby model działał bardziej sprawiedliwie dla różnych grup, próbować można metod balansowania (over-,undersampling), stosowania różnych metryk. Przydatne może być także użycie technik walidacji krzyżowej, które pozwalają na lepsze zrozumienie, jak model radzi sobie na różnych fragmentach danych,
- **Strategie Modelowania:** Warto eksperymentować z różnymi architekturami i algorytmami, które lepiej radzą sobie z rozkładem danych. To dobra okazja do porównania i ocenienia różnic między różnymi podejściami, ponieważ wybranie odpowiedniej metody nie jest oczywiste,
- **Analiza Błędów:** Ze względu na wcześniej wspomniane potencjalne wyzwania, a także dużą liczbę klas ważna będzie regularna analiza błędów modelu, szczególnie w kontekście mniej reprezentowanych artystów. Pomoże to na bieżąco zauważać problemy i w miarę możliwości dobrać odpowiednie techniki. Skomplikowana struktura danych może sprawić, że w pewnym momencie zagadnienie nas przytłoczy i

spowoduje trudności lub niedopatrzenia, dlatego ważne jest, aby podchodzić ze zrozumieniem i kontrolą,

- **Eksperymenty z Preprocessingiem:** Przydatne może być skorzystanie z technik preprocessingowych, jak np. zmiana rozmiaru czy augmentacja. Mogą one znacznie wpływać na wyniki modelu.

Dzięki uwzględnieniu powyższych aspektów w procesie budowy i oceny modeli uczenia maszynowego można lepiej zarządzać wpływem dysproporcji w danych na końcowe wyniki i zwiększyć dokładność oraz skuteczność modelu.

## Rozdział 3

# Modele rozpoznawania artystów dzieł malarstw

Ten rozdział dotyczy różnych narzędzi stosowanych w uczeniu głębokim. Na przykładzie problemu rozpoznawania dzieł malarstw jako pierwszy jest przedstawiony najprostszy model, który stanowi pewien punkt odniesienia dla dalszych modeli.

```
# Operacje matematyczne i macierzowe
import numpy as np
from numpy.random import seed

# Manipulacja i analiza danych
import pandas as pd

# Generowanie liczb losowych
import random

# Praca z danymi w formacie JSON
import json

# Operacje na systemie plików i ścieżkach
import os
import shutil

# Pasek postępu
from tqdm import tqdm, tqdm_notebook

# Wizualizacja danych
import matplotlib.pyplot as plt
import seaborn as sns

# Podział danych na zbiory treningowe i testowe
from sklearn.model_selection import train_test_split

# Ocena wyników modelu
from sklearn.metrics import classification_report, confusion_matrix

# TensorFlow i Keras - Tworzenie i trenowanie modeli głębokiego uczenia
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers, initializers, applications, optimizers, callbacks
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img, array_to_img

# Ustawienie losowości dla TensorFlow
tf.random.set_seed(1)
```

Rysunek 3.1: Biblioteki użyte przy tworzeniu modelu

Tabela 3.1 przedstawia podstawowe biblioteki użyte w projekcie oraz ich główne funkcje.

**Tabela 3.1:** Funkcje bibliotek wykorzystanych do budowy modeli

Biblioteka	Funkcja
<b>numpy</b>	Operacje matematyczne i macierzowe
<b>pandas</b>	Manipulacja i analiza danych w formie tabelarycznej
<b>matplotlib.pyplot</b>	Tworzenie wykresów i wizualizacja danych
<b>seaborn</b>	Zaawansowana wizualizacja danych, oparta na <b>matplotlib</b>
<b>tqdm</b>	Wyświetlanie pasków postępu w pętlach
<b>json</b>	Praca z danymi w formacie JSON
<b>os</b>	Operacje na systemie plików, zarządzanie ścieżkami
<b>shutil</b>	Operacje na plikach i katalogach, takie jak kopowanie i przenoszenie
<b>random</b>	Generowanie liczb losowych
<b>sklearn.model_selection</b>	Podział danych na zbiory treningowe i testowe
<b>sklearn.metrics</b>	Ocena wyników modelu, w tym macierz pomyłek i raport klasyfikacji
<b>tensorflow.keras</b>	Tworzenie, trenowanie i ewaluacja modeli głębokiego uczenia

Biblioteka Keras została opracowana przez François'a Chollet'a w projekcie badawczym, a następnie w 2015 roku udostępniona jako open source. Keras wyróżnia się jako API do głębokiego uczenia, cenione za prostotę w trenowaniu i wdrażaniu sieci neuronowych. Ze względu na wysokopoziomowość oraz intuicyjny interfejs, chroni użytkowników przed skomplikowanymi operacjami, umożliwiając szybkie prototypowanie i eksperymentowanie z różnymi architekturami. Zintegrowany z TensorFlow (Keras nie działa bez TensorFlow, TensorFlow automatycznie instaluje Keras), wykorzystuje moc obliczeniową TensorFlow do efektywnego wykonania intensywnych zadań, takich jak obliczanie gradientów i optymalizacja modeli. [3]

Biblioteka TensorFlow udostępniona w 2015 roku przez Google wykorzystywana jest w uczeniu głębokim do tworzenia i trenowania modeli sztucznej inteligencji. TensorFlow wspiera różne platformy sprzętowe i systemy operacyjne, co czyni go wszechstronnym narzędziem w dziedzinie uczenia maszynowego. Cechy, którymi TensorFlow wyróżnia się to skalowalność, elastyczność i szeroki ekosystem narzędzi oraz bibliotek

wspomagających proces tworzenia modeli. Warto jednak mieć na uwadze, że konstruowanie głębokich architektur jest stosunkowo powolne, co może być niewygodne i frustrujące przy testowaniu różnych podejść oraz optymalizacji parametrów. [13] [3]

Modele w dalszej części pracy bazują na modułach z biblioteki **tensorflow.keras**, które są kluczowe w procesie tworzenia i trenowania modeli głębokiego uczenia:

- **tensorflow.keras** to wysokopoziomowe API TensorFlow do tworzenia i trenowania modeli sieci neuronowych. W skład tego modułu wchodzą różne podmoduły, takie jak:
  - **regularizers** oferuje funkcje regularyzacyjne (L1, L2, L1\_L2), które pomagają zapobiegać nadmiernemu dopasowaniu modelu (overfitting) do danych treningowych,
  - **initializers** dostarcza metody do inicjalizacji wag w modelu sieci neuronowej przed rozpoczęciem treningu. Różne sposoby inicjalizacji, takie jak **GlorotUniform**, **HeNormal** lub **RandomNormal**, wpływają na szybkość uczenia i zbieżność modelu,
  - **applications** dostarcza gotowe do użycia, pretrenowane modele głębokiego uczenia, takie jak **ResNet**, **VGG**, **Inception**, czy **MobileNet**. Te modele mogą być używane do transfer learningu, co pozwala na skrócenie czasu trenowania oraz uzyskanie lepszych wyników na mniejszych zbiorach danych, dzięki wykorzystaniu już wytrenowanych wag na dużych zbiorach takich jak ImageNet,
- **tensorflow.keras.callbacks** umożliwia zarządzanie i kontrolę procesu trenowania modeli. Callbacks, czyli funkcje wywoływane w trakcie i po zakończeniu epoki treningowej, pozwalają na dynamiczne dostosowanie procesu uczenia,
- **EarlyStopping** monitoruje wybraną metrykę podczas trenowania modelu, a gdy zauważa, że poprawa metryki zatrzymała się, zatrzymuje trening, aby uniknąć przeuczenia modelu. Parametry, takie jak **monitor**, **patience** i **restore\_best\_weights**, pozwalają na dokładne dostosowanie działania tej funkcji,
- **ReduceLROnPlateau** to callback, który obniża tempo uczenia (learning rate) w momencie, gdy obserwowana metryka przestaje się poprawiać. Jest to przydatne do dalszego udoskonalania modelu po osiągnięciu stagnacji w jego wydajności. Parametry, takie jak **monitor**, **factor** i **patience**, pozwalają na precyzyjną kontrolę działania tej funkcji,
- **tensorflow.keras.layers** dostarcza zestaw warstw, które są podstawowymi elementami budulcowymi sieci neuronowych. Warstwy takie jak **Conv2D**, **MaxPooling2D**, **Flatten**, **Dense**, **Dropout**, i **BatchNormalization** są używane do konstrukcji architektury modelu,

- **tensorflow.keras.optimizers** to moduł zawierający różne algorytmy optymalizacji, które regulują proces uczenia modeli. Najczęściej stosowanym optymalizatorem jest **Adam**, który łączy zalety metod **AdaGrad** i **RMSProp**,
- **tensorflow.keras.preprocessing.image** umożliwia ładowanie, przekształcanie i augmentację obrazów w celu przygotowania danych wejściowych dla modeli. Funkcje takie jak **img\_to\_array**, **load\_img**, i **ImageDataGenerator** są kluczowe do pracy z obrazami w Keras,
- **from tensorflow.keras.models** zawiera klasy do definiowania architektury modelu sieci neuronowej takie jak:
  - **Sequential** to prosty sposób na tworzenie modeli w Keras, gdzie warstwy są ułożone w sekwencyjny stos. Każda warstwa przyjmuje wyjście z poprzedniej jako swoje wejście. Jest używany, gdy model ma prostą, liniową strukturę, gdzie każda warstwa ma tylko jedno wejście i jedno wyjście,
  - **Model** jest bardziej elastyczną klasą, pozwalającą na tworzenie modeli z bardziej skomplikowanymi, nieliniowymi architekturami. Umożliwia definiowanie modeli z wieloma wejściami, wyjściami lub łączącymi różne warstwy w bardziej zaawansowany sposób. Jest używany, gdy model nie jest sekwencyjny lub gdy potrzebne są niestandardowe architektury sieci neuronowej,
- Ustawienie losowości dla TensorFlow służy zachowaniu powtarzalności wyników w trakcie trenowania modeli, w projekcie zastosowano ustawienie ziarna losowości za pomocą funkcji **tf.random.set\_seed(1)**.

### 3.1. Model 0: Podstawowa sieć CNN

W tej sekcji zostanie opisano model kontrolny, który jest podstawową siecią CNN bez wcześniejszego przygotowania danych. Celem jest sprawdzenie, jakie wyniki można uzyskać przy minimalnym przetwarzaniu danych. Podział danych skutkował 6779 obrazami w zbiorze treningowym oraz 1667 obrazami w zbiorze walidacyjnym, podzielonymi na 50 klas.

**Tabela 3.2:** Szczegóły modelu kontrolnego

Parametr	Wartość
Opis	Model kontrolny bez przygotowania danych
Architektura sieci	CNN z 2 warstwami konwolucyjnymi, 2 warstwami MaxPooling, Flatten, 2 warstwami Dense (128 neuronów), Softmax
Rozmiar obrazów wejściowych	150x150 pikseli
Liczba klas	50
Batch size	32
Liczba epok	25
Optymalizator	Adam
Funkcja straty	Categorical Crossentropy
Metryki	Dokładność (Accuracy)

Na rysunku 3.2 widać kod w języku Python, służący do wczytania odpowiednio danych treningowych i walidacyjnych.

```
# Wczytywanie danych treningowych
train_generator = data_gen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
Found 6779 images belonging to 50 classes.

# Wczytywanie danych walidacyjnych
validation_generator = data_gen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
Found 1667 images belonging to 50 classes.
```

**Rysunek 3.2:** Kod w Pythonie służący wczytaniu danych treningowych i walidacyjnych

Model sieci konwolucyjnej (CNN), którego kod widoczny jest na rysunku 3.3, składa się z dwóch warstw konwolucyjnych, dwóch warstw MaxPooling, warstwy Flatten oraz dwóch warstw Dense, z których ostatnia używa aktywacji Softmax, aby zwrócić prawdopodobieństwo każdej z 50 klas. Model został skompilowany z optymalizato-

rem Adam oraz funkcją straty Categorical Crossentropy. Następnie trenowano model przez 25 epok.

```
In [11]: # Budowanie modelu CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')
])

In [12]: # Kompilacja modelu
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

**Rysunek 3.3:** Kod w Pythonie służący do budowy i komplikacji modelu

Model kontrolny został zbudowany bez żadnego wcześniejszego przygotowania danych, bez użycia augmentacji czy transfer learning. Wyniki pokazują wyraźne problemy z generalizacją. Pojawia się wysoka dokładność na zbiorze treningowym, ale niska na zbiorze walidacyjnym. W tabeli przedstawiono dokładność oraz stratę dla każdej epoki treningu i walidacji.

Zazwyczaj symbol In[\*] informuje nas, że komórka jest w trakcie wykonywania. Dzięki bibliotece tqdm możemy jednak na bieżąco śledzić postęp wykonania kodu. Na rysunku 3.4 widoczny jest proces trenowania modelu w trakcie drugiej epoki. W momencie uchwycenia zrzutu ekranu model przetworzył 82 z 212 dostępnych porcji (batchy) danych. Współczynnik ETA (Estimated Time of Arrival) podpowiada nam przewidywany czas pozostały do zakończenia bieżącego zadania. W przypadku widocznym na rysunku 3.4, ETA: 1:33, oznacza, że szacowany czas, który pozostał do zakończenia obecnie trwającej epoki, wynosi 1 minutę i 33 sekundy.

```
In [*]: # Trenowanie modelu
history = model.fit(
    train_generator,
    epochs=25,
    validation_data=validation_generator
)

Epoch 1/25
212/212 [=====] - 251s 1s/step - loss: 27.1747 - accuracy: 0.1075 - val_loss: 3.6915 - val_accuracy: 0.1248
Epoch 2/25
82/212 [=====>.....] - ETA: 1:33 - loss: 2.7177 - accuracy: 0.3688
```

**Rysunek 3.4:** Kod w Pythonie służący do trenowania modelu i pokazanie przebiegu progresu drugiej epoki

Wyniki dla końcowej fazy treningu są widoczne na rysunku 3.5. Wartość val\_accuracy rośnie powoli, zaczynając od 0.1116 i kończąc na 0.1458. Od drugiej do ostatniej epoki wartość ta wskazuje na stagnację dokładności i wahając się między 0.1248, a 0.1632. Jednocześnie wartość accuracy bardzo szybko zwiększa się, w czwartym kroku osiąga już 0.9211, a w ostatnim nawet 0.9940. Na zbiorze walidacyjnym dokładność jest niska przy jednocześnie bardzo wysokiej wartości na zbiorze treningowej. Zauważając przy tym dodatkowo jednokrotnie szybki wzrost strat (od 3.6699 w pierwszej epoce, po 20.4516 w ostatniej), można wyciągnąć wnioski o silnym przeuczeniu modelu. Prościej ujmując, model bardzo dobrze dopasowuje się do zbioru treningowego, jednak ma duże problemy z dokładnością na zbiorze walidacyjnym.

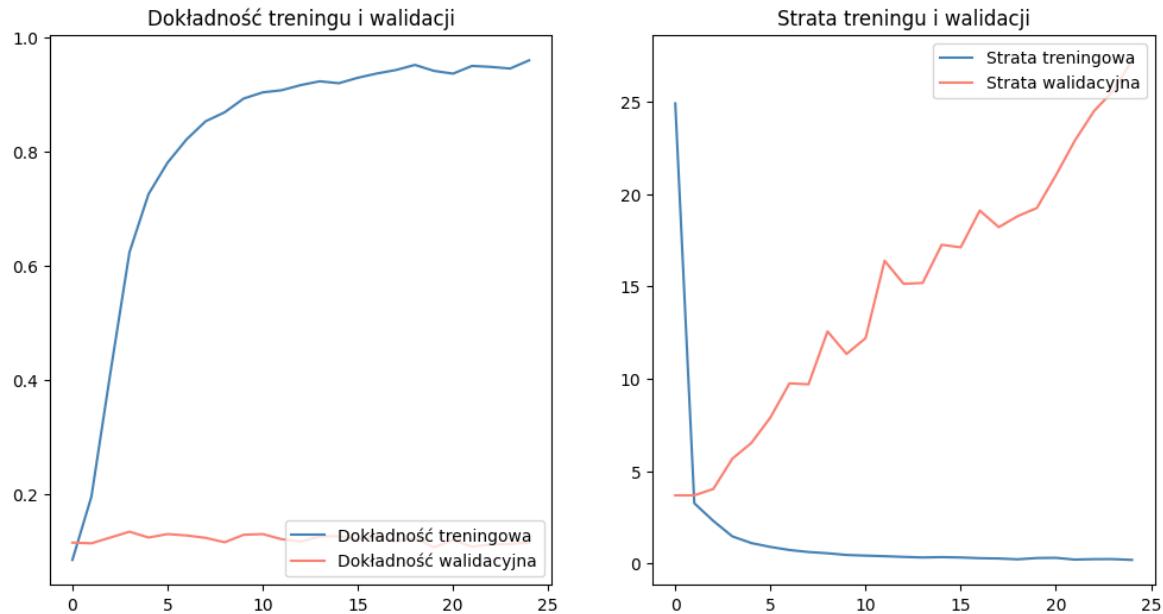
```

Epoch 20/25
212/212 124s 576ms/step - accuracy: 0.9460 - loss: 0.2603
- val_accuracy: 0.1056 - val_loss: 19.2384
Epoch 21/25
212/212 116s 539ms/step - accuracy: 0.9438 - loss: 0.2734
- val_accuracy: 0.1188 - val_loss: 21.0108
Epoch 22/25
212/212 116s 541ms/step - accuracy: 0.9498 - loss: 0.2063
- val_accuracy: 0.1068 - val_loss: 22.8932
Epoch 23/25
212/212 119s 554ms/step - accuracy: 0.9543 - loss: 0.1996
- val_accuracy: 0.1110 - val_loss: 24.4824
Epoch 24/25
212/212 118s 550ms/step - accuracy: 0.9477 - loss: 0.2253
- val_accuracy: 0.1116 - val_loss: 25.5711
Epoch 25/25
212/212 116s 541ms/step - accuracy: 0.9686 - loss: 0.1654
- val_accuracy: 0.1140 - val_loss: 27.2335

```

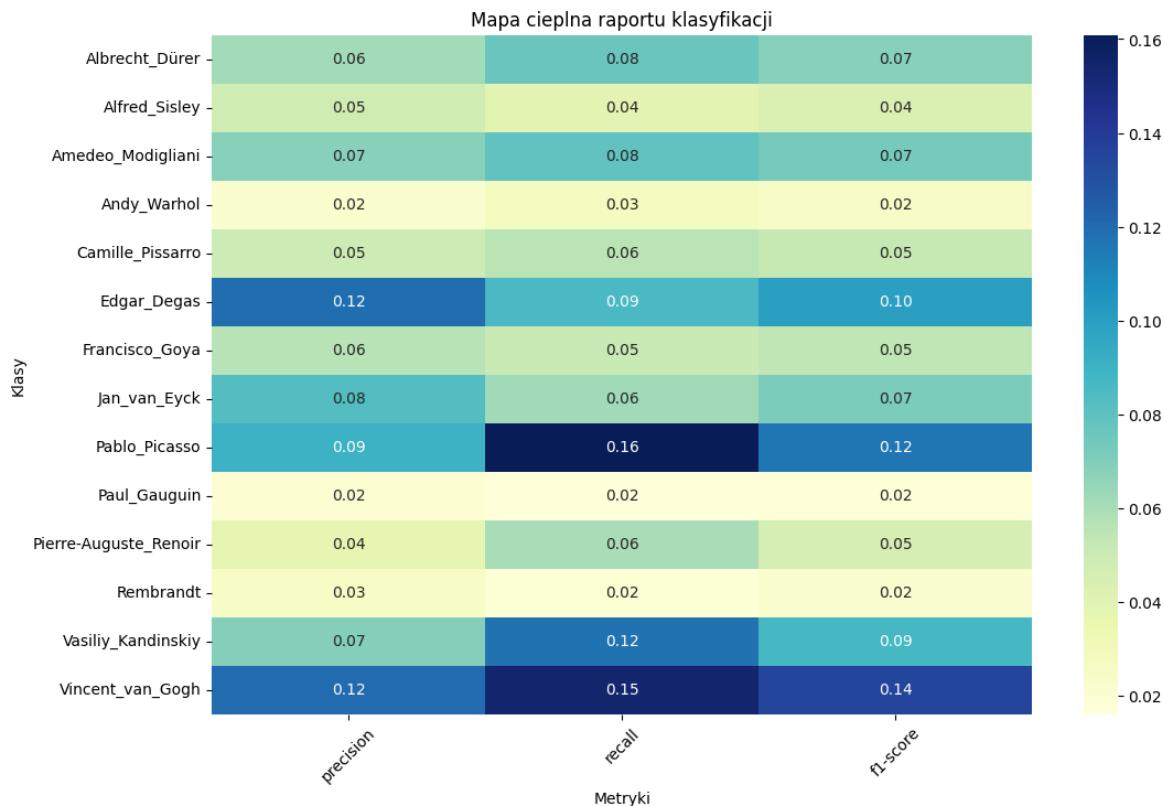
**Rysunek 3.5:** Wyniki treningu dla epok 20-25

Warto zerknąć teraz na wyniki treningu i walidacji dla wszystkich epok. Aby lepiej je zobrazować, przedstawiono je na wykresach, na rysunku 3.6. Pierwszy wykres pokazuje dokładność treningową i walidacyjną, a drugi straty treningowe i walidacyjne. Bardzo łatwo dostrzec tu wcześniej już omówione wnioski. Wartość dokładności na zbiorze treningowym rośnie nieporównywalnie szybko do wartości na zbiorze testowym, która utrzymuje się nisko. Wykres strat treningowych i walidacyjnych pokazuje, że model zaczyna się przeuczać. Straty na zbiorze treningowym szybko spadają, ale na walidacyjnym po początkowym spadku zaczynają rosnąć. To rosnące różnice między stratami wskazują, że model ma problem z uogólnieniem na nowe dane.



Rysunek 3.6: Wyniki treningu i walidacji

Rysunek 3.7 przedstawia wyniki oceny modelu w trzech kluczowych miarach: precyzyji (precision), czułości (recall) oraz ich połączenia, czyli miary F1 (F1-score).



Rysunek 3.7: Mapa cieplna dla precyzyji >0

Na wykresie cieplnym pokazane są wyniki dla każdej z tych miar, nie uwzględniając jednak klas z wartością precyzji równą zeru. Każda z metryk wskazuje na słabą skuteczność w przypisywaniu odpowiednich klas. Wysoka liczba zerowych wartości wskazuje na potrzebę dalszej optymalizacji modelu.

### 3.2. Model 0.5: Sieć CNN z ulepszoną architekturą i augmentacją

Model 0.5 jest rozwinięciem Modelu 0 i ma na celu zbadanie wpływu technik augmentacji danych na skuteczność rozpoznawania dzieł malarstwic. Kluczową różnicą między tymi dwoma modelami jest wprowadzenie augmentacji danych w Modelu 0.5. Poprzez zwiększenie różnorodności danych treningowych, augmentacja może poprawić zdolność modelu do generalizacji i dokładność rozpoznawania obrazów. Ponadto, w Modelu 0.5 zmieniono architekturę sieci neuronowej poprzez dodanie dwóch dodatkowych warstw konwolucyjnych oraz dwóch dodatkowych warstw MaxPooling, co umożliwia modelowi uchwycenie bardziej złożonych wzorców w danych. Wzrost liczby neuronów w warstwie Dense do 512 zwiększa moc obliczeniową modelu, co może wpływać na lepsze wyniki klasyfikacji. Wszystkie szczegóły architektury modelu są opisane w tabeli 3.3. Liczba obrazów w zbiorze treningowym wynosi 6779, podczas gdy zbiór walidacyjny zawiera 1667 obrazów.

**Tabela 3.3:** Szczegóły modelu 0.5

Parametr	Wartość
Opis	Model z podstawową augmentacją danych (flip, zoom, shear)
Architektura sieci	CNN z 4 warstwami konwolucyjnymi, 4 warstwami MaxPooling, Flatten, Dense (512 neuronów), Softmax
Rozmiar obrazów wejściowych	150x150 pikseli
Liczba klas	50
Batch size	32
Liczba epok	25
Optymalizator	Adam
Funkcja straty	Categorical Crossentropy
Metryki	Dokładność (Accuracy)
Augmentacja danych	Przeskalowanie (1./255), flip poziomy, zoom (0.2), shear (0.2)

Kod użyty do budowy Modelu 0.5 (widoczny na rysunku 3.8) tworzy model konwolucyjnej sieci neuronowej (CNN) przy użyciu frameworka Keras. Proces ten zaczyna się od zdefiniowania warstw konwolucyjnych, które odpowiadają za ekstrakcję cech

z obrazów. Pierwsza warstwa konwolucyjna z 32 filtrami o rozmiarze 3x3 wprowadza funkcję aktywacji ReLU, a następnie warstwę MaxPooling, która redukuje wymiarowość map cech. Kolejne warstwy konwolucyjne zwiększą liczbę filtrów oraz stosują podobne operacje spoolingowe, co pozwala modelowi na uchwycenie bardziej złożonych wzorców. Po warstwach konwolucyjnych dane są spłaszczone za pomocą warstwy Flatten, a następnie przekazywane do warstwy Dense z 512 neuronami i funkcją aktywacji ReLU, która dodaje nieliniowość do modelu. Ostatecznie, warstwa Dense z liczbą neuronów równą liczbie klas, z funkcją aktywacji Softmax, umożliwia modelowi klasyfikację obrazów do jednej z 50 klas. Model jest komplikowany z użyciem optymalizatora Adam oraz funkcji straty Categorical Crossentropy, co pozwala na skuteczną naukę i optymalizację parametrów podczas treningu. Proces trenowania modelu trwa 25 epok, podczas których model jest uczony na danych treningowych i walidowany na osobnym zbiorze, co umożliwia monitorowanie jego wydajności i precyzyjności.

```
In [10]: # Budowanie modelu CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(num_classes, activation='softmax')
])

In [11]: # Kompilacja modelu
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

In [12]: # Trenowanie modelu
history = model.fit(
    train_generator,
    epochs=25,
    validation_data=validation_generator
)
```

Rysunek 3.8: Kod w Pythonie pozwalający na zbudowanie i wytrenowanie modelu

Odnosząc się do rysunku 3.9 i porównując go z poprzednim modelem, można dostrzec, że udało się nieco poprawić wyniki. Wzrosła dokładność na zbiorze walidacyjnym, zmalała funkcja straty. Oznacza to, że techniki augmentacji mają istotny wpływ na redukcję przeuczenia i efektywność modelu.

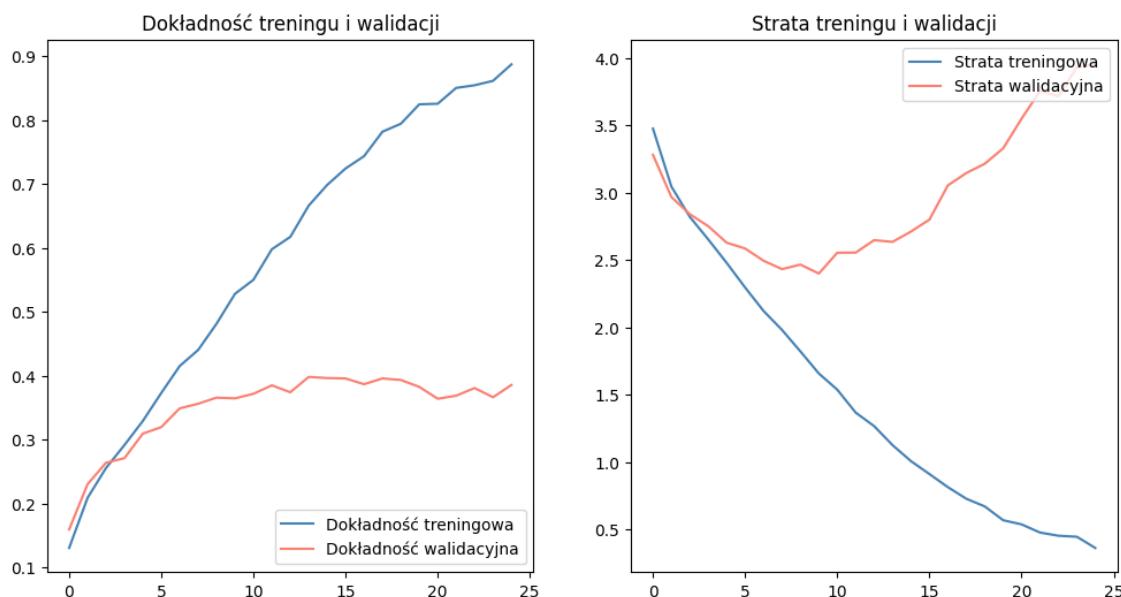
```

Epoch 20/25
212/212 [=====] - 243s 1s/step - loss: 0.3968 - accuracy: 0.8724
- val_loss: 3.9091 - val_accuracy: 0.3761
Epoch 21/25
212/212 [=====] - 244s 1s/step - loss: 0.3793 - accuracy: 0.8774
- val_loss: 4.0191 - val_accuracy: 0.3827
Epoch 22/25
212/212 [=====] - 245s 1s/step - loss: 0.3534 - accuracy: 0.8867
- val_loss: 3.9611 - val_accuracy: 0.3785
Epoch 23/25
212/212 [=====] - 243s 1s/step - loss: 0.3390 - accuracy: 0.8900
- val_loss: 4.2985 - val_accuracy: 0.3695
Epoch 24/25
212/212 [=====] - 242s 1s/step - loss: 0.3433 - accuracy: 0.8942
- val_loss: 4.2597 - val_accuracy: 0.3755
Epoch 25/25
212/212 [=====] - 244s 1s/step - loss: 0.2665 - accuracy: 0.9131
- val_loss: 4.4780 - val_accuracy: 0.3665

```

**Rysunek 3.9:** Wyniki treningu dla epok 20-25

Jak widać na rysunku 3.10, wyniki treningu i walidacji poprawiły się.

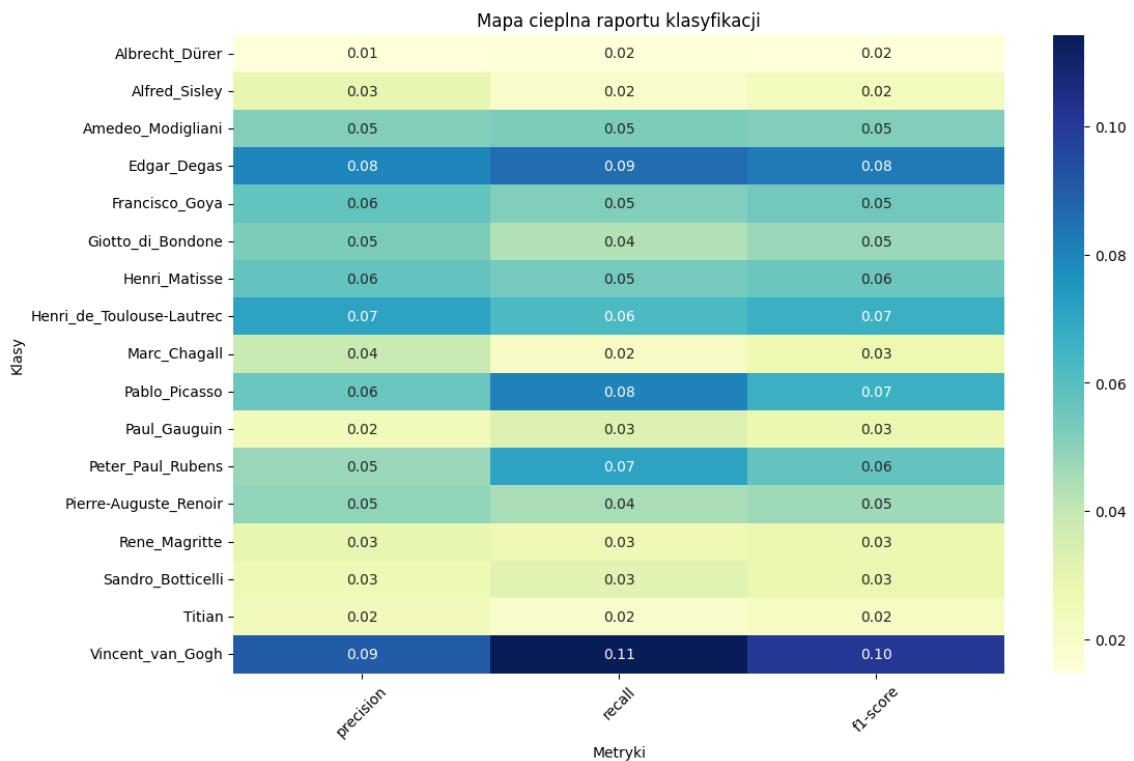


**Rysunek 3.10:** Wyniki treningu i walidacji

W przypadku walidacyjnej dokładności, w Modelu 0.5 występuje wartość początkowa równa 0.1596, która rosła stopniowo do 0.3983 w trakcie trenowania. W Modelu 0 dokładność walidacyjna była dużo niższa. Model 0.5 zaczynał z niską dokładnością, ale z każdą kolejną epoką była ona systematycznie poprawiana, aż osiągnęła wartość 0.8863 na końcu 25-tej epoki. W Modelu 0 również pojawia się wzrost dokładności, ale model ten, pomimo dobrych wyników na zbiorze treningowym, słabo generalizuje na zbiorze walidacyjnym. Straty (loss) w Modelu 0.5 również są mniejsze i bardziej stabilne w porównaniu do Modelu 0. Model 0 ma problemy z generalizacją i jest prze-

### 3.2. Model 0.5: Sieć CNN z ulepszoną architekturą i augmentacją

uczony. Pomimo, że Model 0.5 wypadł lepiej niż Model 0, to wciąż ma problemy z generalizacją na danych walidacyjnych. Na rysunku 3.11 przedstawiona jest mapa cieplna raportu klasyfikacji, zawierająca trzy wartości trzech metryk. Widzimy więcej niezerowych wartości, a także wyniki sugerujące poprawę dla wielu artystów.



**Rysunek 3.11:** Mapa cieplna dla prezycji>0

Augmentacja danych nie tylko poprawia dokładność klasyfikacji, ale także pomaga w lepszym wydobyciu istotnych cech z obrazów, co jest kluczowe w zadaniu rozpoznawania dzieł malarstw, gdzie detale mogą mieć znaczący wpływ na wynik klasyfikacji.

### 3.3. Model 1: Sieć CNN z ręcznym podziałem danych i bez augmentacji

W Modelu 1 wprowadzone jest kilka istotnych zmian, które mają pokazać, jak model działa przy prostszych założeniach. Najważniejszą zmianą jest ręczne przygotowanie danych. Zamiast korzystać z automatycznego podziału na zbiory treningowe i walidacyjne (jak to miało miejsce w poprzednich modelach), dane są dzielone ręcznie przy użyciu funkcji `train_test_split`, a następnie przeniesione do odpowiednich folderów. Sposób owego postępowania widoczny jest na rysunku 3.12 Dzięki temu uzyskano pełną kontrolę nad tym, które dane trafiają do treningu, a które do walidacji, co daje większą możliwość eksperymentowania i bardziej przejrzyste porównywanie wyników.

```
# Tworzenie folderów dla zbiorów treningowych i walidacyjnych
os.makedirs(train_dir, exist_ok=True)
os.makedirs(val_dir, exist_ok=True)

# Iteracja przez foldery artystów
for artist in os.listdir(data_dir):
    if os.path.isdir(os.path.join(data_dir, artist)):
        artist_dir = os.path.join(data_dir, artist)
        images = os.listdir(artist_dir)

    # Podział danych w proporcji 80:20
    train_images, val_images = train_test_split(images, test_size=0.2, random_state=42)

    # Tworzenie folderów dla artystów w zbiorze treningowym i walidacyjnym
    os.makedirs(os.path.join(train_dir, artist), exist_ok=True)
    os.makedirs(os.path.join(val_dir, artist), exist_ok=True)

    # Przenoszenie obrazów do odpowiednich folderów
    for image in train_images:
        shutil.copy(os.path.join(artist_dir, image), os.path.join(train_dir, artist, image))
    for image in val_images:
        shutil.copy(os.path.join(artist_dir, image), os.path.join(val_dir, artist, image))
```

Rysunek 3.12: Kod reprezentujący manualny podział danych na zbiory treningowy i walidacyjny

Budowa modelu reprezentowana przez tabelę 3.4 obejmuje użycie TensorFlow i Keras do stworzenia modelu CNN. Architektura modelu składa się z trzech warstw konwolucyjnych, każda z warstwą MaxPooling, a następnie warstwy Flatten oraz dwóch warstw Dense. Model jest komplikowany z użyciem optymalizatora Adam oraz funkcji straty Categorical Crossentropy, a następnie trenowany przez 25 epok z użyciem danych dostarczonych przez ImageDataGenerator. W przeciwieństwie do wcześniejszych modeli Model 1 nie stosuje augmentacji danych, co prawdopodobnie ma negatywne przełożenie na jego zdolność do generalizacji.

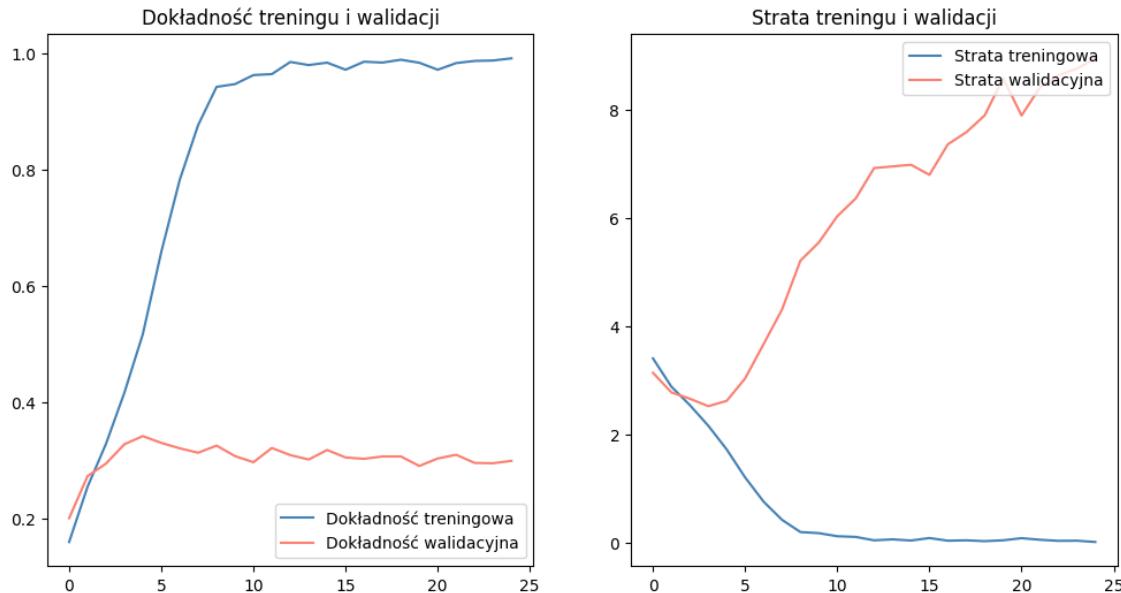
**Tabela 3.4:** Szczegóły Modelu 1

Parametr	Wartość
Opis	Model z normalizacją danych bez augmentacji
Architektura sieci	CNN z 3 warstwami konwolucyjnymi, 3 warstwami MaxPooling, Flatten, Dense (128 neuronów), Softmax
Rozmiar obrazów wejściowych	150x150 pikseli
Liczba klas	50
Batch size	32
Liczba epok	25
Optymalizator	Adam
Funkcja straty	Categorical Crossentropy
Metryki	Dokładność (Accuracy)
Przetwarzanie danych	Standaryzacja rozmiarów obrazów do 150x150, normalizacja wartości pikseli (rescale=1./255)
Liczba obrazów treningowych	6735
Liczba obrazów walidacyjnych	1711

W odróżnieniu od Modelu 0.5, z augmentacją danych (np. obracanie obrazów, zmiany kontrastu), Model 1 ogranicza się jedynie do skalowania wartości pikseli. Celem jest sprawdzenie, jak model radzi sobie z danymi bez dodatkowych modyfikacji. Dzięki temu można zobaczyć, czy wcześniejsze wyniki są implikacją bardziej złożonego przetwarzania, czy może sama struktura modelu ma większe znaczenie. Model 1 zawiera trzy warstwy konwolucyjne i trzy warstwy MaxPooling, ale nie użyto mechanizmów takich jak Dropout czy Batch Normalization, które w poprzednich modelach wpływają na zapobieganie przeuczeniu. Chodzi o to, by sprawdzić, jak dobrze model radzi sobie bez dodatkowych zabiegów regulujących.

Wyniki treningu i walidacji dla Modelu 1 są widoczne na rysunku 3.13. W trakcie treningu widać, że od pierwszych epok model stopniowo poprawia swoją dokładność, a funkcja straty spada zarówno dla zbioru treningowego, jak i walidacyjnego. Już w 3cej epoce dokładność modelu na treningu wynosi 35%, a na walidacji 32%. Z kolei od 5tej epoki model zaczyna mocniej przetrenowywać się na danych treningowych – osiągając 63% dokładności, podczas gdy na walidacji wyniki są gorsze (34%). Widać,

że w tym czasie straty walidacyjne dalej rosną – od 7.8 w 20tej epoce do 8.9 w 25tej epoce, co potwierdza, że model traci zdolność generalizacji i dochodzi do przeuczenia.



**Rysunek 3.13:** Wyniki treningu i walidacji

Na rysunku 3.14 pokazane są wyniki z epok 20-25.

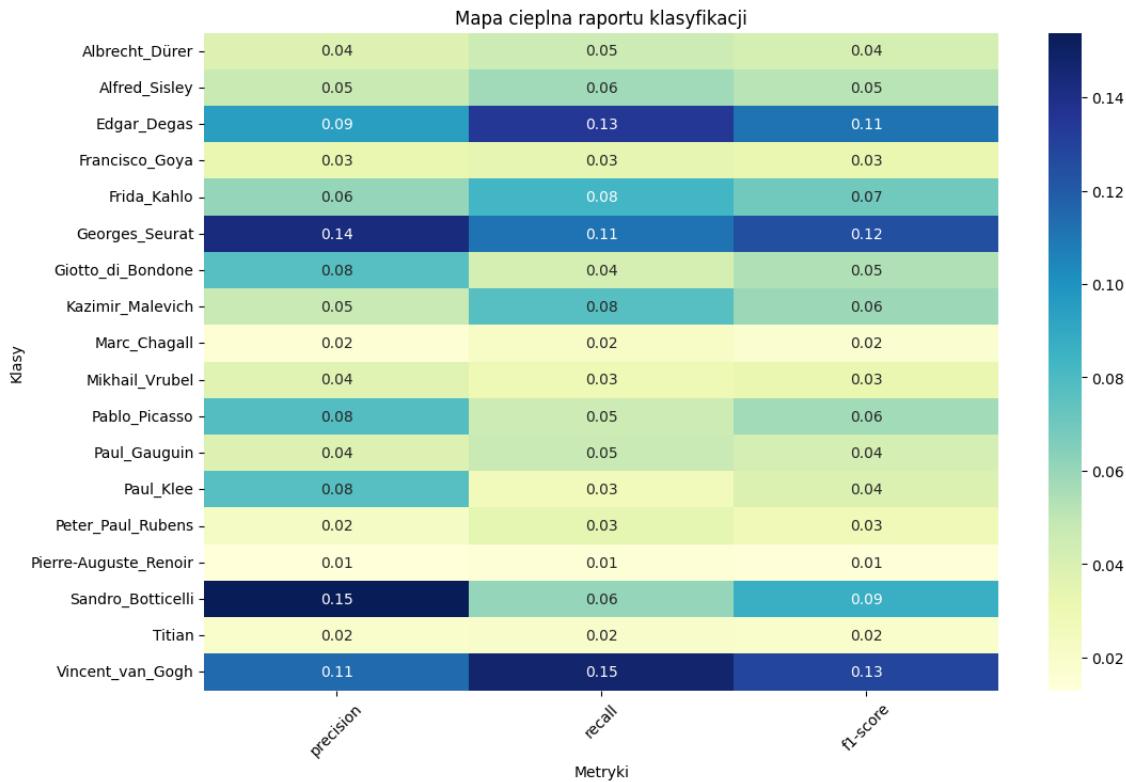
```
Epoch 20/25
211/211 148s 688ms/step - accuracy: 0.9887 - loss: 0.0368
- val_accuracy: 0.2899 - val_loss: 8.5763
Epoch 21/25
211/211 162s 757ms/step - accuracy: 0.9688 - loss: 0.1065
- val_accuracy: 0.3027 - val_loss: 7.8898
Epoch 22/25
211/211 153s 717ms/step - accuracy: 0.9858 - loss: 0.0579
- val_accuracy: 0.3092 - val_loss: 8.4107
Epoch 23/25
211/211 141s 662ms/step - accuracy: 0.9886 - loss: 0.0362
- val_accuracy: 0.2951 - val_loss: 8.6317
Epoch 24/25
211/211 159s 747ms/step - accuracy: 0.9856 - loss: 0.0562
- val_accuracy: 0.2946 - val_loss: 8.7535
Epoch 25/25
211/211 166s 776ms/step - accuracy: 0.9953 - loss: 0.0130
- val_accuracy: 0.2987 - val_loss: 8.9443
```

**Rysunek 3.14:** Wyniki treningu dla epok 20-25

Wyniki pokazują, że Model 1 ma znaczące problemy z klasyfikacją większości artystów, z niskimi wartościami precyzji i czułości dla większości kategorii. W szczególności, wiele artystów ma precyzyję równą zeru, co oznacza, że model nie rozpoznaje ich wcale. Na rysunku 3.15 widoczna jest mapa cieplna, która pomija te kategorie,

### 3.3. Model 1: Sieć CNN z ręcznym podziałem danych i bez augmentacji

gdzie precyza wynosi 0, i koncentruje się na artystach, dla których model wykazuje jakiekolwiek możliwości klasyfikacyjne. Mimo to nie widać dobrych wyników nawet dla klas z najwyższymi metrykami, co wskazuje na potrzebę dalszego udoskonalania modelu.



**Rysunek 3.15:** Mapa cieplna dla precyzyj $>0$

Podsumowując, przeprowadzone eksperymenty z Modelem 1 dostarczają istotnych informacji dotyczących wpływu uproszczonego przetwarzania danych i prostszej architektury na wydajność modelu. Użycie ręcznego podziału danych oraz ograniczenie przetwarzania do skalowania wartości pikseli, zamiast stosowania bardziej złożonej augmentacji, pozwalała ocenę wpływu tych czynników na rezultaty modelu. Stwierdzono, że pomimo uproszczenia, Model 1 osiągnął dokładność na poziomie około 33%, co sugeruje, że efektywność modelu może być mocno uzależniona od technik przetwarzania wstępnego danych oraz architektury. Wnioski te wskazują na potrzebę dalszego eksperymentowania z bardziej zaawansowanymi metodami augmentacji i bardziej złożonymi architekturami sieci, aby uzyskać lepsze wyniki i stabilność modelu.

### 3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout

Celem Modelu 2 jest nie tylko sprawdzenie, czy zaawansowana architektura oraz intensywna augmentacja mogą poprawić dokładność modelu, ale także zrozumienie, jak te zmiany wpływają na ogólną efektywność i zdolność modelu do generalizacji. Szczegóły dotyczące architektury są opisane w tabeli 3.5. Liczba obrazów treningowych wynosi 13393, zaś walidacyjnych 1711.

**Tabela 3.5:** Szczegóły Modelu 2

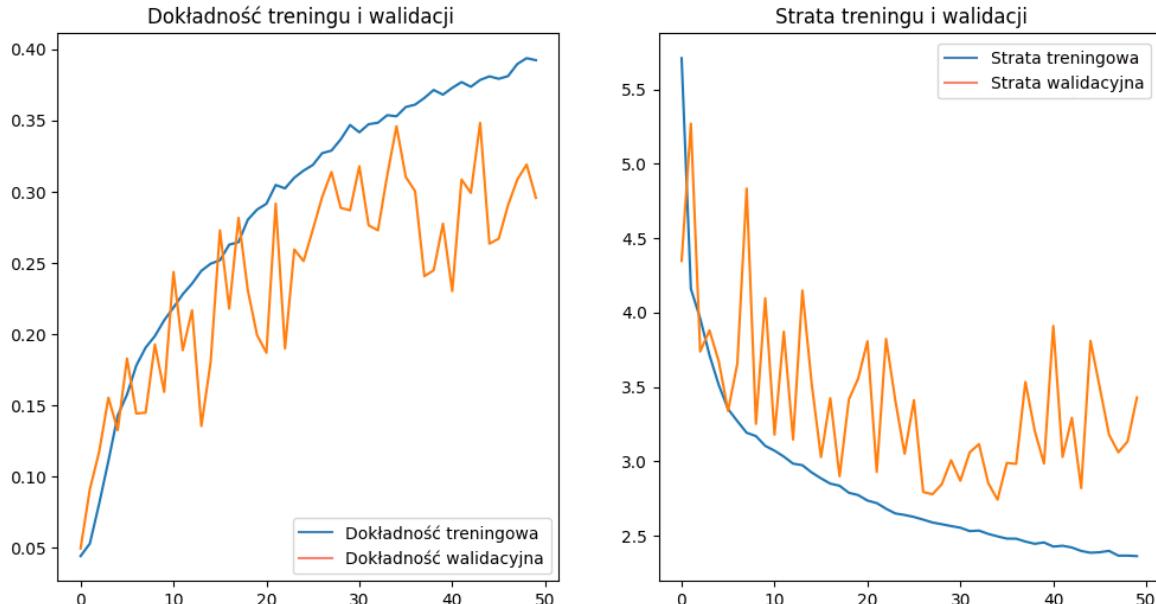
Parametr	Wartość
Opis	Model z augmentowanymi danymi, Batch Normalization i Dropout
Architektura sieci	CNN z 4 warstwami konwolucyjnymi (w tym z Batch Normalization), 4 warstwami MaxPooling, Flatten, Dense (512 neuronów), Dropout, Softmax
Rozmiar obrazów wejściowych	150x150 pikseli
Liczba klas	50
Batch size	32
Liczba epok	50
Optymalizator	RMSprop
Funkcja straty	Categorical Crossentropy
Przetwarzanie danych	Augmentacja danych (rotacja, przesunięcie, zmiana rozmiaru, przycięcie, zoom, flip) oraz normalizacja wartości pikseli (rescale=1./255)
regularyzacja	L2 regularyzation (0.001) dla warstwy Dense, Dropout (0.5)
Batch Normalization	Tak, po każdej warstwie konwolucyjnej

Model 2 wprowadza kilka kluczowych różnic w porównaniu do Modelu 0 i Modelu 0.5. W przeciwieństwie do Modelu 0, który korzysta z prostszej architektury sieci konwolucyjnej i nie stosuje augmentacji danych, Model 2 charakteryzuje się znacznie bardziej zaawansowaną strukturą z dodatkowymi warstwami konwolucyjnymi oraz technikami regularyzacji, jak Batch Normalization i Dropout. Dzięki temu możliwe jest lepsze uchwycenie złożonych cech obrazów i poprawa ogólnej wydajności modelu. W porównaniu do Modelu 0.5, który używa augmentacji danych i prostszego podejścia do

### 3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout

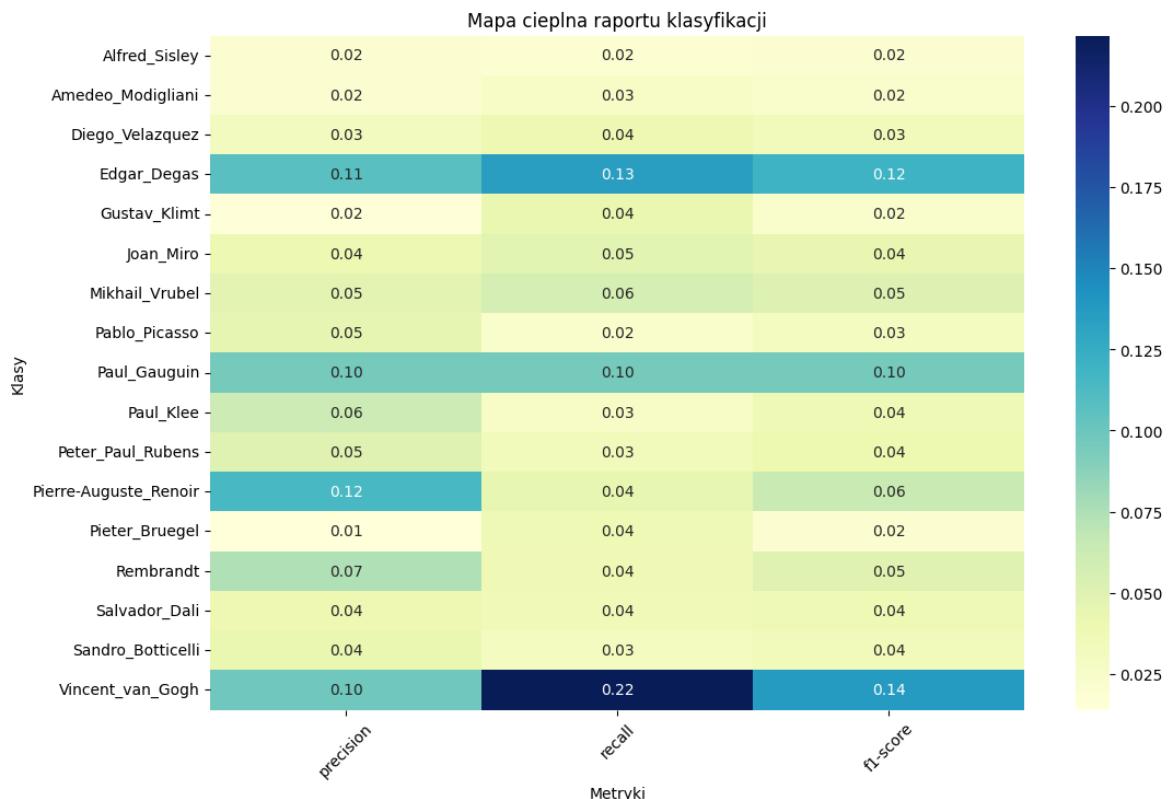
przetwarzania, Model 2 wprowadza bardziej intensywną augmentację oraz dłuższy czas treningu. Model 2 nie tylko używa augmentacji, ale również wzbogaca ją o dodatkowe techniki przetwarzania, co ma na celu lepsze dostosowanie modelu do różnorodnych danych treningowych. Dłuższy czas trenowania w Modelu 2 pozwala na dokładniejsze dostosowanie się do danych, co różni się od krótszego okresu treningu w Modelu 0.5.

Wyniki epok przedstawione na rysunku 3.16 pokazują, że dokładność modelu wzrasta zarówno na danych treningowych, jak i walidacyjnych w miarę upływu kolejnych epok. Strata na zbiorze treningowym maleje, co wskazuje na poprawę dopasowania modelu do danych. Strata na zbiorze walidacyjnym również się zmniejsza, sugerując dobrą generalizację. Jednakże, jeśli strata na zbiorze walidacyjnym zaczęła wzrastać, może to świadczyć o przeuczeniu modelu. Ostatecznie, analiza wykresów wskazuje, że kluczowe jest monitorowanie zarówno dokładności, jak i strat na zbiorze walidacyjnym, aby uniknąć problemów z generalizacją.



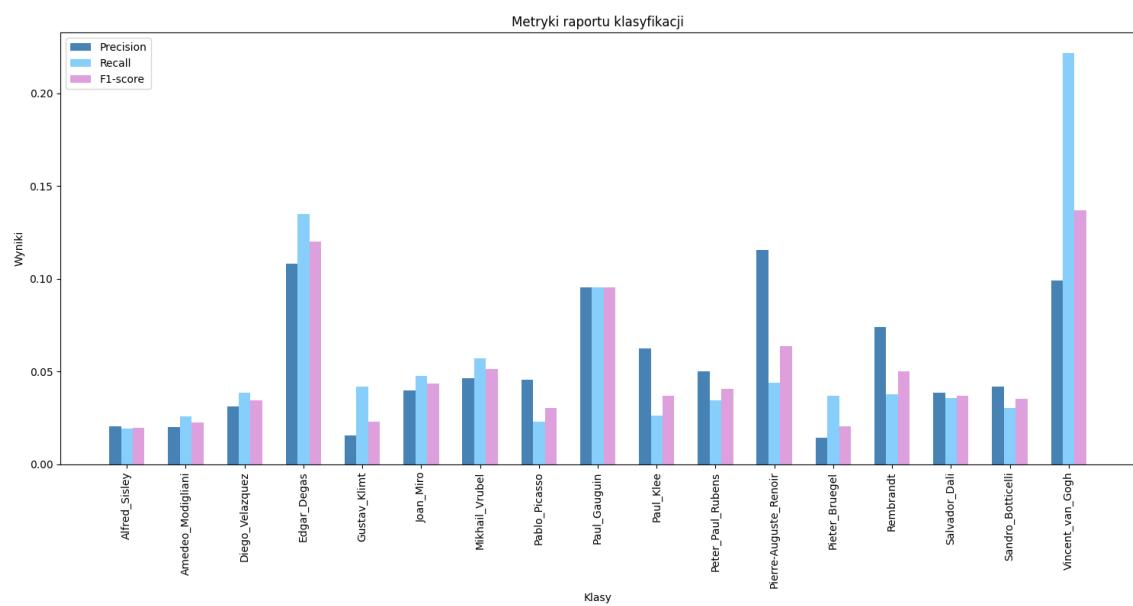
Rysunek 3.16: Wyniki treningu i walidacji

Mapa cieplna (rys. 3.17) przedstawia precyzję modelu dla różnych klas, przy jednoczesnym wykluczeniu (aż 33-ech) wartości równych zeru. Klasy takie jak Vincent Van Gogh oraz Edgar Degas wyróżniają się wyższymi wynikami, większość jednak stanowią wartości zerowe lub zeru bliskie. Ogólnie rzecz biorąc, mapa cieplna ilustruje nie tylko nierównomierną wydajność modelu w klasyfikacji różnych klas, ale również słabą skuteczność.



**Rysunek 3.17:** Mapa cieplna dla precyzji>0

Na rysunku 3.18, przedstawione zostały te same wyniki w nieco innej formie, w dobry sposób podkreślające niskie wartości metryk dla każdej z klas.

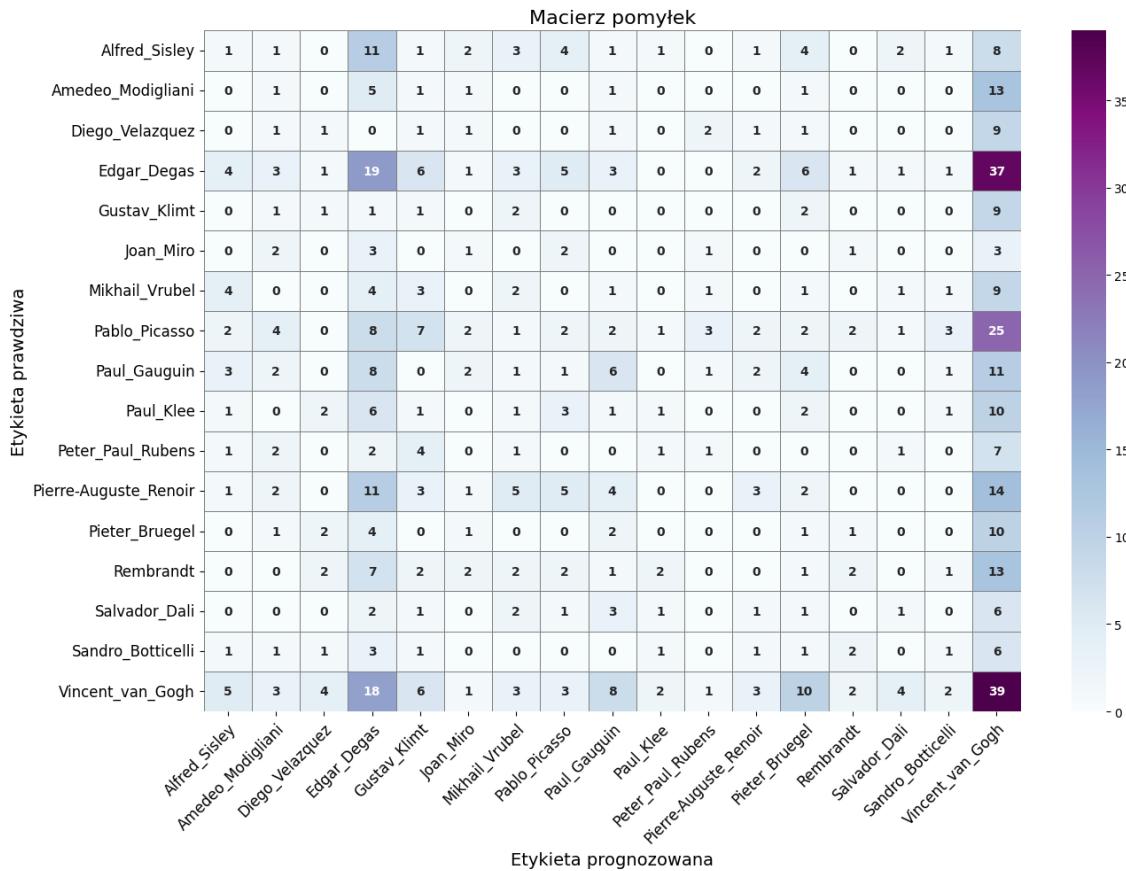


**Rysunek 3.18:** Wyniki metryk na wykresie słupkowym (precyzja>0)

Na macierzy pomyłek, widocznej na rysunku 3.19, można dostrzec ciemniejsze

### 3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout

pola na klasach, które wyróżniają się większą liczebnością. Może to wskazywać na faworyzowanie artystów, dla których jest więcej danych.



Rysunek 3.19: Mapa cieplna dla precyzji>0

Podsumowując modele omówione do tej pory:

- Model 0: Najprostszy, silne przebiasowanie, bardzo niska wydajność na zbiorze walidacyjnym,
- Model 05: Większa złożoność sieci prowadzi do lepszej generalizacji, choć nadal wyniki są dalekie od zadowalających,
- Model 1: Największe przebiasowanie, najwyższa dokładność na treningu, ale słaba na walidacji,
- Model 2: Najbardziej zaawansowana architektura z mechanizmami regularyzacji, ale bez znaczącej poprawy wyników walidacyjnych.

W porównaniu z Modelami 0, 0.5, 1, pod względem struktury, Model 2 oferuje najlepszą architekturę, ale żaden z modeli nie osiąga satysfakcjonujących wyników w zadaniu klasyfikacji obrazów. Kolejnym krokiem jest przetestowanie wpływu metod oversamplingu, walidacji krzyżowej oraz redukcji klas na przykładzie Modelu 2.

### 3.4.1. Rozszerzenie Modelu 2 o metodę oversamplingu

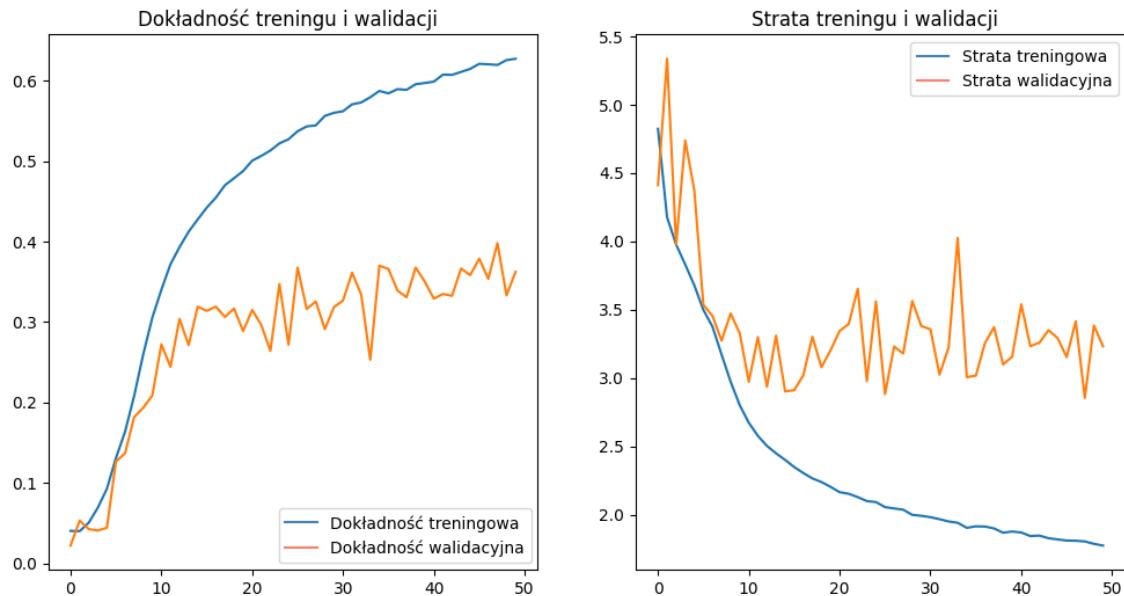
Ze względu na różnorodność klas warto pójść o krok dalej i zbadać wpływ zastosowania metody oversampling przy użyciu biblioteki imbalanced-learn (imblearn). Model 2a różni się od Modelu 2 tym, że w Modelu 2a stosowany jest oversampling w celu zbalansowania klas w zbiorze danych, przy zachowaniu identycznej architektury sieci. Celem tej zmiany jest sprawdzenie, czy sztuczne tworzenie dodatkowych przykładów dla niedoreprezentowanych klas przyczynia się do zwiększenia skuteczności modelu w klasyfikacji mniejszych liczących klas. Undersampling nie został tutaj zastosowany, ponieważ może prowadzić do utraty istotnych danych z nadreprezentowanych klas, co mogłoby obniżyć ogólną jakość modelu i jego zdolność do nauki wzorców w tych klasach. Metoda oversamplingu pozwala na wzbogacenie zbioru danych bez eliminowania informacji, co może poprawić ogólne wyniki klasyfikacji, szczególnie dla klas, które są rzadziej reprezentowane.

```
Epoch 45/50
1096/1096 1016s 926ms/step - accuracy: 0.6122 - loss: 1.8163
- val_accuracy: 0.3583 - val_loss: 3.2921
Epoch 46/50
1096/1096 1017s 927ms/step - accuracy: 0.6266 - loss: 1.7908
- val_accuracy: 0.3787 - val_loss: 3.1529
Epoch 47/50
1096/1096 1014s 924ms/step - accuracy: 0.6241 - loss: 1.7936
- val_accuracy: 0.3536 - val_loss: 3.4149
Epoch 48/50
1096/1096 1015s 925ms/step - accuracy: 0.6168 - loss: 1.8144
- val_accuracy: 0.3980 - val_loss: 2.8540
Epoch 49/50
1096/1096 1016s 926ms/step - accuracy: 0.6289 - loss: 1.7638
- val_accuracy: 0.3331 - val_loss: 3.3840
Epoch 50/50
1096/1096 1017s 927ms/step - accuracy: 0.6240 - loss: 1.7801
- val_accuracy: 0.3624 - val_loss: 3.2321
```

Rysunek 3.20: Wyniki treningu dla epok 40-50

Na rysunku 3.21 widać, że Model 2a osiąga lepsze wyniki w zakresie dokładności i straty w porównaniu do Modelu 2. Pod koniec treningu, Model 2a osiąga średnią dokładność na poziomie 62.40% i średnią stratę na poziomie 1.7801, podczas gdy Model 2 kończy z dokładnością 39.86% i stratą 2.3544. Różnica ta wskazuje na znaczną poprawę w zdolności modelu do poprawnego klasyfikowania próbek. Model 2a również lepiej radzi sobie z walidacją, osiągając dokładność 39.80% i stratę 3.2321 w porównaniu do 31.91% i 3.4291 otrzymanego dla Modelu 2. Wartości val\_accuracy i val\_loss Modelu 2a są wyższe niż jego wartości dokładności i strat w treningu, ale wciąż są lepsze w porównaniu do Modelu 2, co sugeruje, że oversampling pomógł poprawić ogólne wyniki modelu.

### 3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout



**Rysunek 3.21:** Wyniki treningu i walidacji

Dla dobrego porównania wyniki zostały przedstawione w tabeli 3.6.

**Tabela 3.6:** Porównanie wyników trenowania Modelu 2 i Modelu 2a

Metryka	Model 2		Model 2a	
	Początkowa	Końcowa	Początkowa	Końcowa
<b>Accuracy</b>	4.12%	39.86%	4.21%	62.40%
<b>Loss</b>	8.5360	2.3544	5.6440	1.7801
<b>Val_Accuracy</b>	4.97%	31.91%	2.22%	39.80%
<b>Val_Loss</b>	4.3467	3.4291	4.4119	3.2321

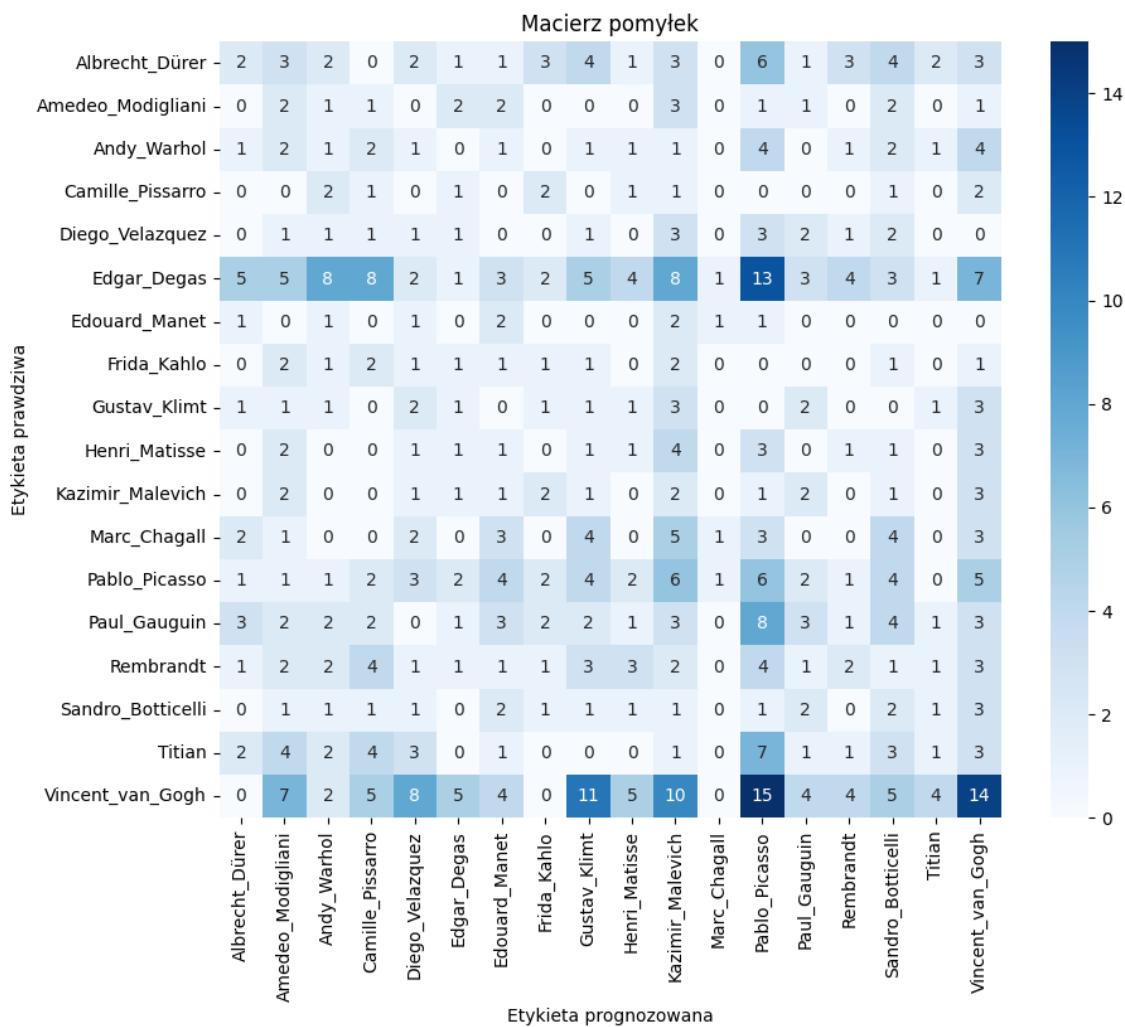
Metryki widoczne na tabeli 3.7 pokazują, że oversampling nie przynosi zamierzonych korzyści dla modelu ogólnie. Oversampling w Modelu 2a przyniósł znaczną poprawę w wynikach treningowych i walidacyjnych w porównaniu do Modelu 2. Choć walidacja nie osiągnęła idealnych wyników, widoczna jest wyraźna poprawa.

**Tabela 3.7:** Porównanie wartości metryk Modelu 2 i Modelu 2a

	Accuracy	Precision	Recall	F1-Score
<b>Model 2</b>	0.05	0.04	0.05	0.04
<b>Model 2a</b>	0.03	0.03	0.03	0.03

Z analizy macierzy Modelu 2 (rys. 3.19), można by wyciągnąć wnioski jakoby model faworyzował Edgara Degas'a i Vincent'a Van Gogh'a, gdyż przypisuje im obrazy innych artystów. Znacznie przewyższają inne klasy ilością danych, gdyż posiadają

odpowiednio 702 i 877 obrazów (różnice przedstawione były podczas wstępnej analizy danych na rys. 2.2). Mając na uwadze tą nierównomierność rozkładu klasy można na rysunku 3.22 zauważyc jak model przypisuje większą ilość etykiet prognozowanych innym artystom. Nie robi tego co prawda tak, jakby się wymarzyło, mimo wszystko jest to ciekawa obserwacja.



Rysunek 3.22: Macierz pomylek dla precyzji  $>0$

Mimo że Model 2a osiąga wyższą dokładność w porównaniu do Modelu 2, nie można automatycznie stwierdzić, że jest on lepszy pod względem ogólnej wydajności. Zastosowanie oversamplingu może prowadzić do różnych problemów, które mogą wpływać na jakość modelu. Oversampling może wprowadzać nadmierne uproszczenie danych. Może to skutkować tym, że model jest zbyt mocno dostosowany do danych treningowych, co w efekcie prowadzi do przeuczenia. Aby poprawić ogólną wydajność modelu, mogłoby być konieczne przeprowadzenie dalszych eksperymentów z dostosowywaniem parametrów oversamplingu lub wypróbowaniem innych technik augmentacji.

### 3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout

---

tacji danych. Ważnym wnioskiem jest tutaj potrzeba, aby wyniki modelu oceniać na podstawie różnych metryk. W Modelu 2a na początku wydawało się, że dodanie oversamplingu poprawiło skuteczność, jednak dopiero po głębszym zrozumieniu dostrzec można było większy obraz i komplikacje, jakie się pojawiły.

### 3.4.2. Rozszerzenie Modelu 2 o metodę walidacji krzyżowej

Model 2b jest taki sam pod względem architektury jak Model 2, ale poszerzony o metodę walidacji krzyżowej. Dlatego poza bibliotekami wspomnianymi wcześniej (3.1) dodano technikę KFold z biblioteki sklearn.model\_selection. Metoda ta polega na podziale zbioru danych na k równych części, nazywanych przekrojami. W tym przypadku liczba przekrojów wynosi pięć. W każdej iteracji jeden z pięciu przekrojów będzie wykorzystywany jako zbiór testowy, a pozostałe cztery jako zbiór treningowy. Dzięki lepszemu wykorzystaniu danych można zredukować ryzyko przeuczenia modelu. Następnie wyniki są uśredniane, dzięki czemu ocena wydajności modelu jest bardziej wiarygodna. Dzięki walidacji krzyżowej wyniki uzyskiwane są w sposób bardziej wszechstronny i dokładny, ponieważ model jest testowany na różnych podziałach danych, co zwiększa pewność co do jego ogólnej wydajności. Wyniki są uśredniane, co pozwala uzyskać bardziej wiarygodną ocenę efektywności modelu. Wyniki dla Modelu 2b są widoczne w tabeli 3.8 i wskazują na pewne poprawy w porównaniu do wyników Modelu 2.

**Tabela 3.8:** Wyniki walidacji krzyżowej dla pięciu przekrojów

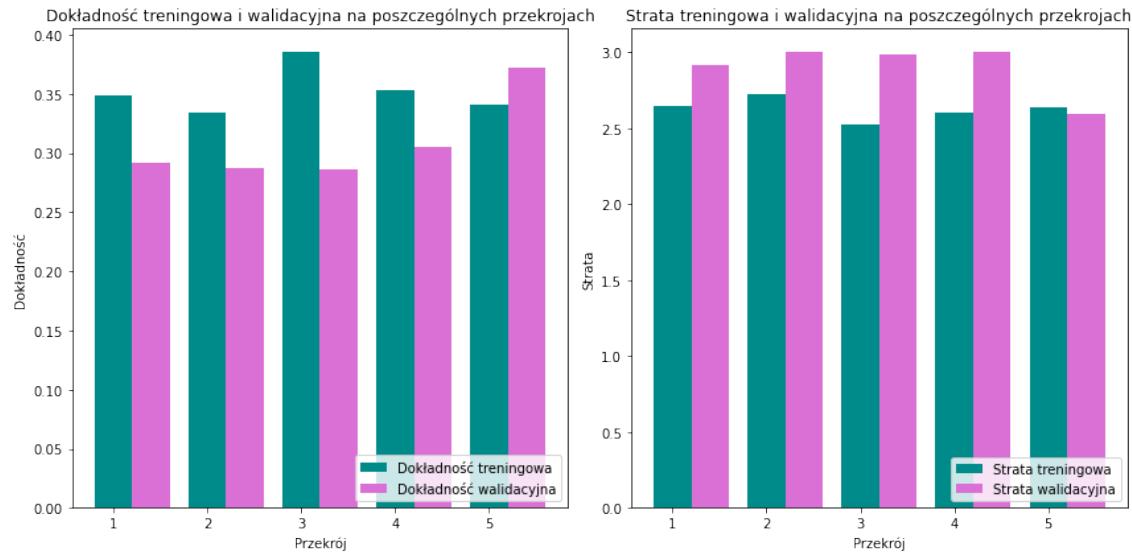
Fold	Accuracy (%)	Loss	Validation Accuracy (%)	Validation Loss
1	34.89	2.6483	29.17	2.9192
2	33.43	2.7236	28.77	2.9997
3	38.60	2.5205	28.66	2.9820
4	35.36	2.6021	30.55	3.0047
5	34.09	2.6365	37.18	2.5906
<b>Średnia</b>	<b>35.27</b>	<b>2.6262</b>	<b>30.87</b>	<b>2.8992</b>

Dokładność uzyskana w poszczególnych przekrojach waha się od 33.43% do 38.60%, co wskazuje na pewne zróżnicowanie w skuteczności modelu w różnych iteracjach walidacji. Wartości straty w poszczególnych przekrojach, które wynoszą od 2.5205 do 2.7236, sugerują, że model dobrze radzi sobie z dopasowaniem do danych treningowych. Warto zwrócić uwagę również na różnice w dokładności walidacyjnej, która waha się od 28.66% do 37.18%, co może sugerować wpływ różnych zestawów walidacyjnych na ocenę modelu. Średnia dokładność osiąga wartość 35.27%, a średnia strata wynosi 2.6262, co sugeruje, że model jest w stanie lepiej generalizować na danych testowych w porównaniu do Modelu 2.

Dokładność i strata są bardziej zrównoważone, a ogólna zdolność modelu do generalizacji nieco się poprawiła. Chociaż wyniki są nadal dalekie od idealnych, zastosowanie walidacji krzyżowej pomaga w uzyskaniu bardziej stabilnych i wiarygodnych ocen wydajności. Walidacja krzyżowa pozwala na lepsze wykorzystanie dostępnych da-

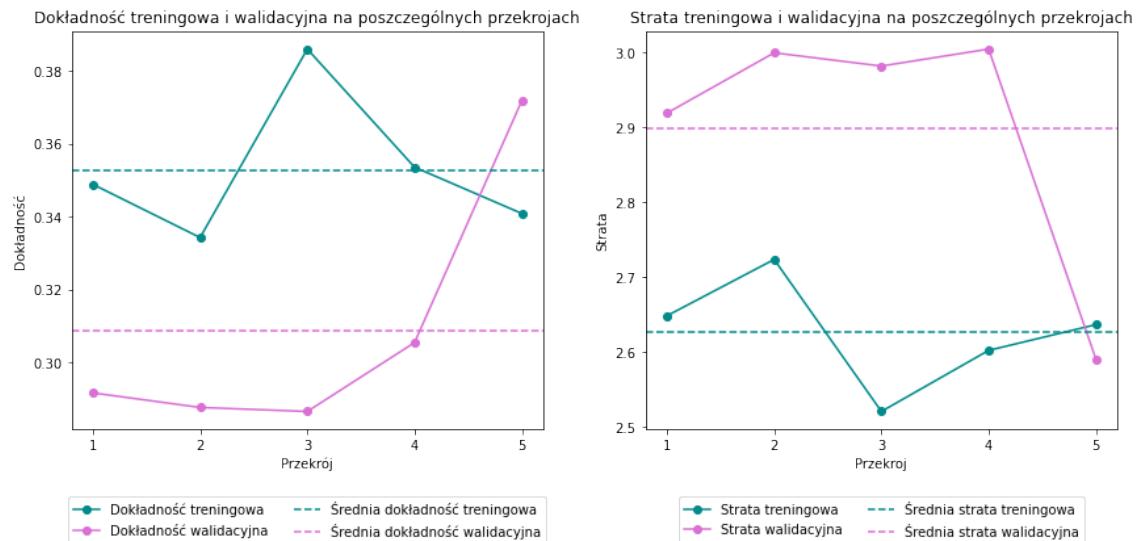
### 3.4. Model 2: Zaawansowana sieć CNN z augmentacją danych, Batch Normalization i Dropout

nych i dostarcza bardziej precyzyjnych informacji na temat wydajności modelu, co jest kluczowe dla minimalizacji ryzyka przeuczenia. Na wykresie w sposób nieco bardziej przystępny, dostrzec można, że wyniki oscylują blisko siebie (rys. 3.23).



Rysunek 3.23: Wyniki treningowe i walidacyjne przedstawione na wykresie słupkowym

Choć wyniki wydają się być bardzo zbliżone, warto je porównać na większej skali, aby lepiej zobaczyć szczegóły. Na rysunku 3.24 poza większą skalą wprowadzono też średnie wartości.



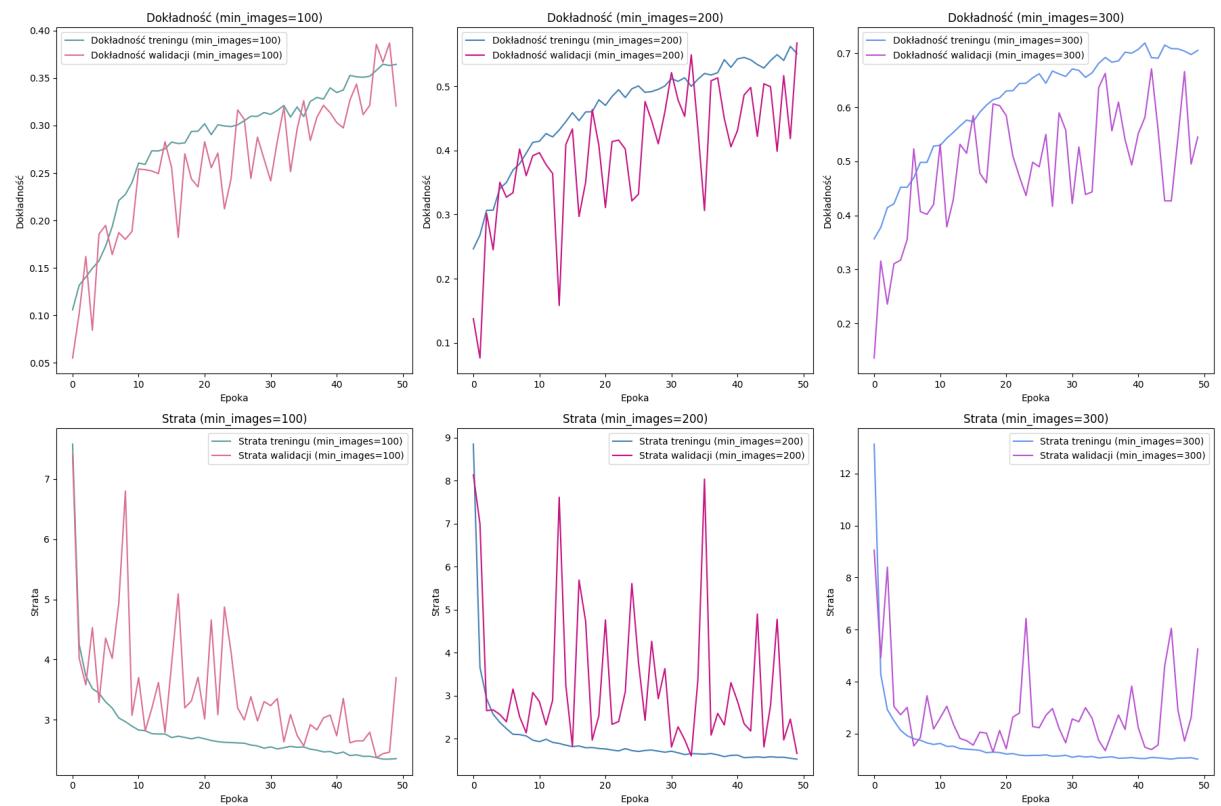
Rysunek 3.24: Porównanie wyników treningowych i walidacyjnych ze średnimi wynikami

Dzięki zastosowaniu walidacji krzyżowej uzyskano bardziej stabilne i wiarygodne wyniki. Losowość podziału danych nie wpływa tak silnie na model i można czuć się bardziej pewnie, że wyniki nie są zniekształcone przez przypadkowy wybór mniej

lub korzystniejszych danych. Model 2b, będący rozszerzeniem Modelu 2, skupia się na wykorzystaniu techniki walidacji krzyżowej (K-Fold) w celu uzyskania bardziej wiarygodnej oceny wydajności modelu. Prezentuje lepszą ogólną wydajność w porównaniu do wcześniejszych modeli, ale nie osiąga zadowalających wyników w klasyfikacji obrazów. Podsumowując, skorzystanie z walidacji krzyżowej może zapewnić bardziej spójne, stabilne i sprawiedliwe dla wszystkich klas wyniki i dlatego warto uwzględnić ją w optymalizacjach modeli.

### 3.4.3. Zastosowanie redukcji ilości klas w Modelu 2

Ze względu na niewielką liczbę obrazów przypisanych do niektórych artystów, trzy modele są zoptymalizowane poprzez wykluczenie artystów z liczbą obrazów mniejszą niż odpowiednio 100, 200 i 300. Celem tej analizy jest zbadanie, jak zmieniają się wyniki Modelu 2 w zależności od liczby dostępnych klas. Poprzez wykluczenie artystów z liczbą obrazów mniejszą niż 100, 200 i 300 uzyskano modele z odpowiednio z liczebnością klas: 30, 11 i 6. Rysunek 3.25 przedstawia wykresy ilustrujące zmiany dokładności i straty modeli w trakcie 50 epok treningu. Wykresy są podzielone na trzy części, każda reprezentująca różne zestawy danych: z minimalną liczbą obrazów 100, 200 i 300. Na wykresach przedstawiono krzywe dokładności i straty dla danych treningowych oraz walidacyjnych.



**Rysunek 3.25:** Wyniki treningowe i walidacyjne dla trzech wersji Modelu 2 po redukcji liczb klas

Modele po redukcji klas wykazują lepsze wyniki zarówno w dokładności, jak i stracie. Im więcej klas zostało usuniętych tym lepsze wyniki i stabilność osiągał model. Takie porównanie pokazuje, że podejście do zarządzania danymi, w tym redukcja klas i liczba obrazów, ma kluczowe znaczenie dla wydajności modelu.

Wykresy ilustrują różnice w efektywności modeli w zależności od liczby obrazów w zestawach treningowych i walidacyjnych:

- Model z minimalną liczbą obrazów 100 (30 klas): Wykres dokładności pokazuje początkowy wzrost, który później stabilizuje się, osiągając wartość około 0.3643 w 50. epoce. Krzywa straty wykazuje duże fluktuacje, z wyraźnym spadkiem na początku treningu, a następnie tendencją do wzrostu i spadku w późniejszych epokach,
- Model z minimalną liczbą obrazów 200 (11 klas): Krzywa dokładności na wykresie przedstawia stały wzrost, osiągając około 0.5820 w 50. epoce, podczas gdy krzywa straty wykazuje stopniowy spadek, osiągając wartość około 1.4992,
- Model z minimalną liczbą obrazów 300 (6 klas): Dokładność modeli pokazuje znaczny wzrost i stabilizację w okolicach 0.5676, a krzywa straty wykazuje stały spadek do wartości około 1.6604.

Tabela 3.9 zawiera wyniki dla wersji Modelu 2 z redukcjami liczby klas.

**Tabela 3.9:** Porównanie wyników dla Modelu 2 i jego wersji ze zredukowaną liczbą klas

Model (minimalna liczba obrazów)	accuracy	loss	val_accuracy	val_loss
Model1 (300)	0.6869	1.0892	0.6628	1.3492
Model1 (200)	0.6096	1.2947	0.6628	1.3492
Model1 (100)	0.6030	2.1267	0.6030	2.1267
Model2 (-)	0.3986	2.3544	0.3191	3.1342

Po zbadaniu wpływu redukcji klas na skuteczność modelu, można wyciągnąć wnioski, że wysoka ilość klas, zwłaszcza tak zróżnicowanych, może znacznie wpływać na model. Zaimplementowanie modelu dla 50ciu artystów jest sporym wyzwaniem, zwłaszcza gdy dodatkowo danych jest mało, a rozdysytrybuowane one są nierównomiernie pomiędzy klasy.

### 3.5. Zastosowanie Transfer Learning z wykorzystaniem ResNet i Fastai

Warto także zwrócić uwagę na dostępność gotowych metod i narzędzi, które mogą znacząco ułatwić pracę nad projektem. Choć zrozumienie podstawowych architektur sieci neuronowych jest kluczowe, rozsądne korzystanie z już zaimplementowanych rozwiązań może być bardzo pomocne. W tej sekcji zostanie zbadane, jak wykorzystanie biblioteki Fastai może przyspieszyć i uprościć proces tworzenia modeli do rozpoznawania artystów na podstawie dzieł malarstw. Fastai oferuje przyjazne interfejsy, które umożliwiają szybkie przygotowanie danych i trenowanie modeli, co jest szczególnie cenne w praktyce.

W Fastai, **transfer learning** z modelem ResNet polega na wykorzystaniu pretrenowanych modeli, które zostały wcześniej wytrenowane na dużych zbiorach danych, takich jak ImageNet. Te pretrenowane modele nauczyły się rozpoznawać różne cechy i wzorce w obrazach, co stanowi solidną bazę do dalszego uczenia. Aby dostosować model ResNet do konkretnego zadania, należy wymienić ostatnią warstwę modelu na warstwę dopasowaną do liczby klas w nowym zadaniu. Następnie, możliwe jest fine-tuning, czyli dalsze trenowanie modelu, co pozwala na dostosowanie wag całego modelu lub tylko niektórych jego warstw do nowych danych. Proces ten pozwala modelowi nauczyć się nowych cech i wzorców, które są bardziej związane z konkretnym zadaniem niż te, na których model był pierwotnie trenowany. Po dokonaniu tych zmian model jest trenowany na nowych danych, co pozwala na osiągnięcie lepszych wyników. Na rysunku 3.26 przedstawione są biblioteki użyte w modelu.

```
import cv2
from pathlib import Path
from fastai.vision import *
from fastai.vision.all import *
from wordcloud import WordCloud, STOPWORDS
from collections import Counter
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob
```

Rysunek 3.26: Biblioteki użyte w modelu

Oprócz bibliotek, które zostały omówione we wcześniejszej części pracy, dodano cv2, fastai, wordcloud oraz glob. Biblioteka cv2 jest używana do przetwarzania obrazów, co pozwala na ich efektywne wczytywanie i manipulowanie nimi. Fastai dostarcza narzędzi do łatwego budowania i trenowania modeli sieci neuronowych, co znaczająco upraszcza proces tworzenia modeli. Wordcloud umożliwia tworzenie chmur tagów,

co może być przydatne do wizualizacji częstotliwości występowania słów, jeśli projekt wymaga analizy tekstów. Natomiast glob pomaga w zarządzaniu plikami i katalogami, co jest nieocenione przy pracy z dużymi zbiorami danych. Zostanie zbadane, czy wykorzystanie tych narzędzi przyspieszy zbudowanie dobrego modelu. Pierwszym po wczytaniu bibliotek krokiem widocznym na rysunku 3.27 jest zdefiniowanie DataBlock (klasa z biblioteki Fastai), które jest niezbędne do przygotowania danych do modelu, ponieważ pozwala zdefiniować jak mają być wczytywane i przetwarzane dane, a także jak mają być dzielone na zbiory treningowy i walidacyjny.

```
# Ścieżka do katalogu z obrazami
img_dir = 'data/images/images'
path = Path(img_dir)

# Definicja DataBlock
dblock = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files, # Pobiera wszystkie pliki obrazów z folderu
    splitter=RandomSplitter(valid_pct=0.2, seed=42), # Split na dane treningowe i walidacyjne
    get_y=parent_label, # Pobiera etykiety z nazw folderów
    item_tfms=Resize(299), # Rozmiar obrazu po załadowaniu
    batch_tfms=aug_transforms(size=299) # Augmentacja danych
)

# Tworzenie DataLoaders
data = dblock.dataloaders(path, bs=64, num_workers=0)
```

Rysunek 3.27: Zdefiniowanie DataBlock

Funkcje użyte w DataBlock:

- **blocks**: Określa typy danych wejściowych i etykiet: obrazy (**ImageBlock**) i kategorie (**CategoryBlock**),
- **get\_items**: Funkcja **get\_image\_files** pobiera wszystkie pliki obrazów z określonego katalogu,
- **splitter**: Funkcja **RandomSplitter(valid\_pct=0.2, seed=42)** dzieli dane na zbiory treningowe (80%) i walidacyjne (20%),
- **get\_y**: Funkcja **parent\_label** pobiera etykiety klas na podstawie nazw folderów, w których znajdują się obrazy,
- **item\_tfms**: Przekształcenie **Resize(299)** zmienia rozmiar obrazów do 299x299 pikseli po ich załadowaniu,
- **batch\_tfms**: Augmentacja danych z użyciem **aug\_transforms(size=299)**, mająca na celu zwiększenie różnorodności danych treningowych.

Po zdefiniowaniu obiektu DataBlock i przygotowaniu danych, kolejnym krokiem jest stworzenie i wytrenowanie modelu. Jak pokazano na rysunku 3.28, proces ten rozpoczyna się od utworzenia obiektu Learner za pomocą funkcji **cnn\_learner**. Funkcja ta przyjmuje kilka kluczowych argumentów: **data**, który jest obiektem DataLoaders zawierającym przygotowane zbiory danych; **resnet50**, czyli pretrenowany model sie-

ci ResNet50, który posłuży jako baza dla naszego modelu; oraz metrics=accuracy, co oznacza, że jako miarę skuteczności modelu będziemy monitorować dokładność. Dodatkowo model\_dir wskazuje katalog, w którym model będzie zapisywany.

```
# Tworzenie modelu i jego uczenie
learn = cnn_learner(data, resnet50, metrics=accuracy, model_dir=model_dir)
learn.fit_one_cycle(20)
```

**Rysunek 3.28:** Kod pozwalający na tworzenie modelu i jego uczenie

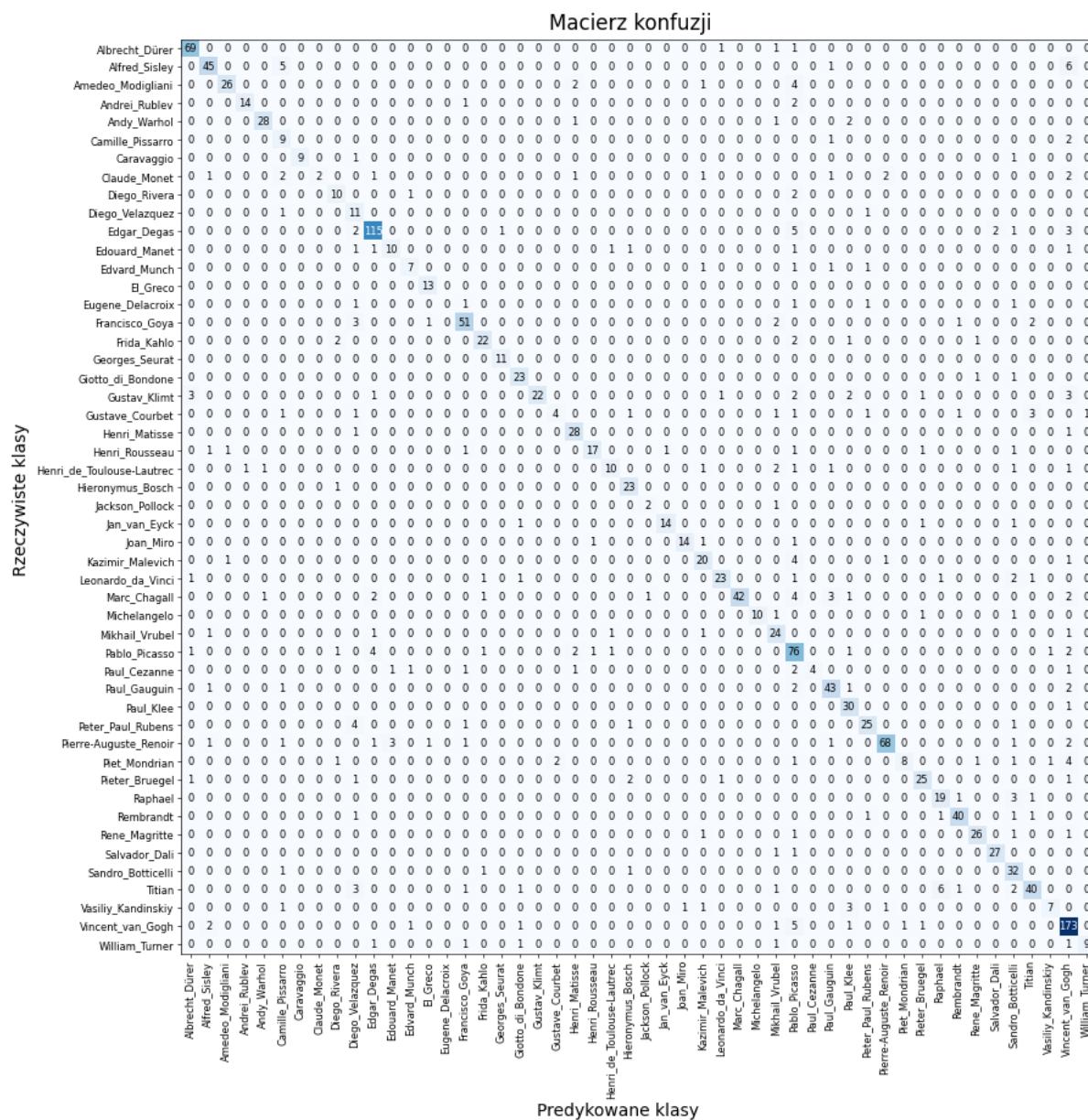
Po skonfigurowaniu modelu, uruchamiany jest proces uczenia za pomocą metody fit\_one\_cycle(20). Ta metoda przeprowadza trening modelu przez 20 epok, stosując strategię cyklicznego uczenia się. Strategia ta jest znana z efektywności w optymalizacji, polegającą na dynamicznej zmianie szybkości uczenia się podczas treningu, co może poprawić wyniki modelu oraz przyspieszyć jego konwergencję. Cały proces trenowania modelu jest monitorowany, a wyniki są oceniane na podstawie dokładności, co pozwala na bieżąco śledzić postępy i efektywność modelu. Wyniki modelu dla poszczególnych epok są przedstawione na rysunku 3.29.

epoch	train_loss	valid_loss	accuracy	time	epoch	train_loss	valid_loss	accuracy	time
0	1.033388	1.055819	0.725281	1:09:16	10	0.322353	0.763166	0.796329	1:08:23
1	1.002850	1.011871	0.725873	1:08:30	11	0.271402	0.696869	0.807578	1:08:59
2	0.953898	0.949781	0.733570	1:07:24	12	0.246747	0.700440	0.803434	1:09:50
3	0.837294	0.922275	0.738307	1:07:18	13	0.225577	0.681022	0.813499	1:10:56
4	0.775030	0.880827	0.751924	1:07:45	14	0.191035	0.696167	0.817052	1:10:55
5	0.650132	0.799276	0.774423	1:07:54	15	0.182599	0.690006	0.814091	1:09:10
6	0.547391	0.805859	0.775015	1:08:06	16	0.160454	0.670956	0.818828	1:09:44
7	0.501788	0.765031	0.786856	1:07:46	17	0.144002	0.678308	0.818828	1:08:56
8	0.407507	0.770148	0.782712	1:08:14	18	0.157262	0.685385	0.815275	1:08:57
9	0.381860	0.758412	0.788632	1:08:09	19	0.142765	0.681762	0.817052	1:12:26

**Rysunek 3.29:** Wyniki treningu dla poszczególnych epok

Z danych wynika, że w miarę postępu treningu wartość train\_loss maleje, co wskazuje na poprawę dopasowania modelu do danych treningowych. Na początku treningu strata wynosi 1.033388, a na końcu spada do 0.142765. Systematyczne zmniejszanie się tej wartości w kolejnych epokach sugeruje, że model staje się coraz lepszy w rozpoznawaniu wzorców w danych treningowych. Strata na zbiorze walidacyjnym (valid\_loss) również zmniejsza się na początku treningu, osiągając minimum w epoce 14, a następnie wzrasta. Spadek wartości valid\_loss z 1.055819 do 0.681022 sugeruje poprawę modelu, ale późniejszy wzrost do 0.681762 w ostatniej epoce może wskazywać

na przeuczenie modelu (overfitting). To oznacza, że model może zacząć tracić zdolność do generalizacji do nowych danych po pewnym etapie treningu. Dokładność na zbiorze walidacyjnym wzrasta przez większość treningu, od 0.725281 na początku do 0.817052 na końcu, co wskazuje, że model poprawia swoje umiejętności klasyfikacji z każdą epoką. Na rysunku 3.30 została przedstawiona macierz pomyłek.



**Rysunek 3.30:** Macierz pomyłek

Analizując wyniki uzyskane dzięki narzędziu Fastai i architektury ResNet, można śmiało powiedzieć, że jest ona odpowiednim narzędziem dla osób, które potrzebują efektywnego rozwiązania do rozpoznawania obrazów. Jej zaawansowane funkcje, łatwość w konfiguracji oraz pretrenowane modele umożliwiają szybkie uzyskanie wysokiej jakości wyników.

## Podsumowanie i wnioski

W pracy zaprezentowana została analiza i porównanie różnych modeli klasyfikacji obrazów, które różniły technikami i parametrami trenowania. Przeprowadzono eksperymenty, dzięki którym można ocenić, jak różne cechy modeli, takie jak augmentacja danych, regularyzacja, oraz liczba epok, wpływają na efektywność i dokładność klasyfikacji. Pomimo pewnych trudności spowodowanych bardzo dużą liczbą klas oraz różnorodnością pomiędzy nimi, a także nieosiągnięciem optymalnej skuteczności, analiza wyników dostarcza cennych informacji na temat sposobu implementacji różnych technik oraz ich wpływu na jakość modelu. Zrozumienie tych zależności jest kluczowe dla dalszego rozwoju i doskonalenia technik uczenia głębokiego, a zdobyte doświadczenie może stanowić solidną bazę dla przyszłych badań i aplikacji w dziedzinie uczenia maszynowego.

### 3.6. Podsumowanie wyników wszystkich modeli

Tabela 3.10 opisuje różnice i podobieństwa w cechach charakteryzujących każdy z modeli.

**Tabela 3.10:** Porównanie modeli na podstawie cech

Cecha	Model 0	Model 0.5	Model 1	Model 2	Model ResNet
Augmentacja	Tak	Tak	Nie	Tak	Tak
Regularyzacja	Nie	Tak	Nie	Tak	Tak
Normalizacja	Tak	Tak	Tak	Tak	Tak
Rozmiar obrazów	150x150	150x150	150x150	150x150	150x150
Batch size	32	32	32	32	32
Liczba epok	25	25	25	50	50
Optymalizator	Adam	Adam	Adam	RMSprop	Adam
Batch Normalization	Nie	Nie	Nie	Tak	Tak
Dropout	Nie	Nie	Nie	Tak	Tak

Wszystkie modele w badaniu zastosowały rozmiar obrazów równy 150x150 pikseli oraz batch size wynoszący 32. Liczba klas wszędzie wynosi 50 poza wariantami Modelu 2 z redukcją klas. Normalizacja obrazów jest obecna we wszystkich modelach, co może przyczynić się do ujednolicenia danych wejściowych i poprawy efektywności trenowania. Różnice w liczbie epok, które są kluczowe dla procesu trenowania, są znaczące. Modele 0, 0.5 i 1 są trenowane przez 25 epok, podczas gdy modele 2 i ResNet przez 50 epok. Większa liczba epok zazwyczaj pozwala na lepsze dopasowanie modelu do danych, jednak może również prowadzić do ryzyka przeuczenia, jeśli nie zostaną zastosowane odpowiednie techniki regularyzacji.

Nie ulega wątpliwościom, że najlepszym modelem jest Model ResNet. Dzięki metodzie transfer learningu można osiągnąć lepsze wyniki przy mniejszej ilości danych, ponieważ model posiada już podstawową wiedzę, a jedynie doucza się na otrzymanych danych. Analizując przypadki Modelu 2c, można dostrzec, że przy redukcji klas znacznie wzrasta jakość modelu, co potwierdza, że zbiór danych posiada bardzo dużo różnorodnych klas i co za tym idzie, jest bardzo trudny do wytrenowania.

Rozszerzenie Modelu 2 zarówno o oversampling jak i walidację krzyżową okazało się korzystne dla zapewnienia sprawiedliwości między klasami. Warto jednak pamiętać o tym, że metody te znacznie wydłużają czas trenowania modelu. Jednak odpowiednio zastosowane potrafią zwiększyć stabilność i ogólną jakość modelu.

W tabeli 3.11 przedstawiono szczegółowe wyniki dla każdego modelu, w tym czas trenowania, dokładność na zbiorze treningowym (Acc), dokładność na zbiorze walidacyjnym (Val Acc), stratę (Loss) oraz stratę na zbiorze walidacyjnym (Val Loss).

**Tabela 3.11:** Porównanie wyników wszystkich modeli

Model	Time	Acc	Val Acc	Loss	Val Loss
Model 0	52min i 50s	0.9686	0.1140	0.1654	27.2335
Model 0.5	1h 16min 53s	0.8863	0.3857	0.3643	3.9544
Model 1	58min 44s	0.9953	0.2987	0.0130	8.9443
Model 2	6h 5min 55s	0.3986	0.2957	2.3544	3.4291
Model 2a	14h 7min 43s	0.6240	0.3624	1.7801	3.2321
Model 2b	16h 39min 56s	0.3409	0.3718	2.6365	2.5906
Model 2c (6 klas)	1h 19min 43s	0.6869	0.5449	1.0892	5.2616
Model 2c (11 klas)	1h 55min 37s	0.5361	0.5676	1.5470	1.6604
Model 2c (30 klas)	3h 5min 17s	0.3616	0.3205	2.3719	3.6974
Model ResNet	22h 35min 7s	0.817052	-	0.142765	0.681762

Czas trenowania może wpływać na całkowity koszt obliczeniowy i efektywność

procesu, co zdecydowanie należy brać pod uwagę. Dokładność na zbiorze treningowym oraz walidacyjnym dostarcza informacji na temat zdolności modelu do generalizacji i jego skuteczności w rozwiązywaniu zadania klasyfikacji. Strata na zbiorze treningowym i walidacyjnym odzwierciedla poziom błędu popełnianego przez model podczas nauki, co ma bezpośredni wpływ na jego wydajność. Analiza wyników przedstawionych w tabeli ujawnia różnorodne efekty działania poszczególnych modeli oraz ich efektywność w kontekście czasu trenowania, dokładności oraz strat.

Należy zachować balans w ocenie tych metryk. Model 1 uzyskuje najwyższą dokładność na danych treningowych (0.9953), co sugeruje, że jest bardzo dobrze dopasowany do tych danych. Jednak mimo tej imponującej dokładności, jego strata walidacyjna (8.9443) jest bardzo wysoka, co może wskazywać na przeuczenie i problemy z generalizacją na dane walidacyjne. Model 0, mimo że osiąga najniższą wartość straty treningowej (0.1654), jego dokładność walidacyjna (0.1140) jest niepokojąco niska, co może sugerować problemy z oceną skuteczności modelu lub niedostosowanie do danych.

Podsumowując, wyniki wskazują, że każdy model ma swoje unikalne mocne i słabe strony. Modele, które osiągają wysoką dokładność na danych treningowych, niekoniecznie przekładają się na lepsze wyniki walidacyjne, co podkreśla konieczność precyzyjnego dostosowania parametrów i architektury w celu optymalizacji wyników. Kluczowe jest zrozumienie, jak czas treningu, struktura modelu oraz liczba klas wpływają na efektywność modeli, aby skutecznie dostosować je do specyficznych wymagań zadania i poprawić ogólne wyniki.

### 3.7. Wnioski

Na podstawie przeprowadzonych eksperymentów i analizy wyników można wyizzare kilka istotnych wniosków dotyczących efektywności zastosowanych modeli oraz wpływu ich cech na końcowe rezultaty. Wprowadzenie augmentacji i regularyzacji ma wyraźny wpływ na poprawę wyników modeli. Augmentacja, poprzez generowanie nowych, zmodyfikowanych wersji obrazów, zwiększa różnorodność danych treningowych i pozwala na lepsze uogólnienie modelu, co może prowadzić do poprawy dokładności klasyfikacji. Regularyzacja, w tym techniki takie jak Dropout, które pomagają w redukcji przeuczenia, poprawia zdolność modeli do generalizacji i zwiększa ich efektywność na zbiorze walidacyjnym. Wprowadzenie Batch Normalization oraz Dropout w modelach może wpłynąć na poprawę stabilności i efektywności procesu trenowania. Batch Normalization, poprzez normalizację aktywacji warstw, przyczynia się do szybszej konwergencji i stabilizacji procesu trenowania, podczas gdy Dropout, poprzez

## Podsumowanie i wnioski

---

losowe wyłączanie jednostek, pomaga w uniknięciu przeuczenia i poprawie zdolności modelu do generalizacji. Różnice w liczbie epok, również mogą mieć istotny wpływ na wyniki końcowe. Jednak warto uwzględnić fakt, że większa liczba epok, choć pozwala na lepsze dopasowanie modelu do danych, może również prowadzić do przeuczenia, jeśli nie jest wspierana przez odpowiednie techniki regularyzacji. Omówione w pracy techniki to kluczowe elementy, które należy uwzględnić przy budowie efektywnych modeli. Dokładna analiza wpływu tych elementów na wyniki modeli jest niezbędna do osiągnięcia optymalnych rezultatów i skutecznego rozwiązania problemów związanych z klasyfikacją obrazów.

Zapoznanie się z budową architektury podczas budowania jej od zera jest niezastąpioną techniką, która stanowi solidne fundamenty wiedzy na temat uczenia głębokiego. Owszem, teoria jest istotna i dostarcza niezbędnych podstaw, jednak nic nie zastąpi praktycznego doświadczenia zdobytego poprzez eksperymentowanie z danymi. Proces ten pozwala na głębsze zrozumienie mechanizmów działania sieci neuronowych, ich architektury oraz optymalizacji. Testowanie różnych konfiguracji i technik uczy, jak subtelne zmiany mogą wpływać na efektywność modelu.

Mimo że w tej pracy optymalną skuteczność modelu udało osiągnąć się dopiero przy skorzystaniu z Modelu ResNet, przeprowadzone eksperymenty i porównania są niezwykle wartościowe. Pozwalają one na zgłębienie różnych aspektów budowy i trenowania modeli, takich jak wpływ augmentacji danych, technik regularyzacji oraz liczby epok na wyniki. Tego typu praktyczne doświadczenie jest bezcenne, gdyż umożliwia lepsze zrozumienie, jak różne komponenty wpływają na końcowe rezultaty.

Głównym wnioskiem pracy jest, że nauka na podstawie eksperymentów oraz budowanie modeli od podstaw, znaczco przyczynia się do pogłębienia wiedzy i umiejętności w dziedzinie uczenia maszynowego. Takie podejście daje możliwość testowania teorii w praktyce i lepszego rozumienia, jak działa dana technika lub architektura. Warto nie pomijać owych eksperymentów, gdyż pójście od razu w zaimplementowane metody bez żadnej intuicji może prowadzić do błędów w analizie wyników oraz do niereagowania na alarmujące, podejrzane wyniki. Praktyczne doświadczenie zdobyte podczas budowy modeli od podstaw pozwala na lepsze rozpoznawanie i interpretowanie takich nieprawidłowości, co jest kluczowe dla skutecznego rozwiązywania problemów i optymalizacji modeli. Niemniej jednak, w praktycznych zastosowaniach, warto korzystać z gotowych, zaimplementowanych metod i bibliotek, które często oszczędzają mnóstwo czasu i zasobów. Gotowe rozwiązania, takie jak predefiniowane architektury sieci, optymalizatory czy techniki regularyzacji, są często dobrze przetestowane i dostosowane do różnych scenariuszy. Korzystanie z takich narzędzi pozwala na skoncentrowanie się na kluczowych aspektach projektu, takich jak dobór danych, tuning

hiperparametrów i interpretacja wyników, zamiast na tworzeniu każdego komponentu od podstaw. Optymalne połączenie praktycznego eksperymentowania z wykorzystaniem sprawdzonych narzędzi może prowadzić do efektywniejszego rozwiązywania problemów i szybszego osiągania zadowalających rezultatów.



## Wykaz literatury

- [1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015. [Online]. Available: <https://doi.org/10.1038/nature14539>
- [2] D. Knickerbocker, *Network Science with Python*. Springer, 2021.
- [3] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*. Manning Publications, 2022.
- [4] R. Chandramohan, “Adaptive critic flight control for a general aviation aircraft: Simulations for the beech bonanza fly-by-wire testbed,” *Journal of Guidance, Control, and Dynamics*, 2020.
- [5] T. Nield, *Essential Math for Data Science*. Manning Publications, 2021.
- [6] F. Rosenblatt, *Principles of Neurodynamics*. Washington D.C.: Spartan Press, 1962.
- [7] ——, “The perceptron: A probabilistic model for information storage and organisation in the brain,” *Psychological Review*, vol. 65, pp. 386–408, 1958. [Online]. Available: <https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf>
- [8] F. Chollet, *Deep Learning with Python, Second Edition*. Manning Publications, 2021.
- [9] S. S. Chang, *Machine Learning Interviews: Kickstart Your Machine Learning and Data Career*. O'Reilly Media, 2023.
- [10] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv*, 2015. [Online]. Available: <https://arxiv.org/abs/1511.08458>
- [11] V. H. Phung and E. J. Rhee, “A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets,” *MDPI*, 2019. [Online]. Available: <https://www.mdpi.com/2076-3417/9/21/4500>
- [12] A. Nielsen, *Praktyczna analiza i predykcja z wykorzystaniem statystyki i uczenia maszynowego*. Helion, 2020.
- [13] B. Ramsundar and R. B. Zadeh, *Głębokie uczenie z TensorFlow. Od regresji liniowej po uczenie przez wzmacnianie*. Helion, 2020.
- [14] R. Tadeusiewicz and M. Szaleniec, *Leksykon sieci neuronowych*. PWN, 2022.
- [15] P. J. Deitel and H. Deitel, *Python dla Programistów. Big Data i AI. Studia przypadków*. Helion, 2020.
- [16] J. Prosise, *Applied Machine Learning and AI for Engineers*. O'Reilly Media, 2021.

## Wykaz literatury

---

- [17] L. R. Gonzalez and A. Stubberfield, *Cracking the Data Science Interview*. CareerMonk Publications, 2019.
- [18] A. C. Müller and S. Guido, *Machine Learning, Python i data science*. Helion, 2016.
- [19] Z.-H. Zhou, *Machine Learning*. Springer, 2016.
- [20] A. Glassner, *Deep Learning*. Morgan Kaufmann, 2019.
- [21] A. W. Trask, *Grokking Deep Learning*. Manning Publications, 2019.
- [22] S. Weidman, *Deep Learning from Scratch*. Self-published, 2019.
- [23] K. Chaudhury, A. H. Ashok, S. Narumanchi, and D. Shankar, *Math and Architectures of Deep Learning*. Manning Publications, 2024.

## Spis rysunków

1.1	Neuron biologiczny [3] . . . . .	8
1.2	Podstawowa struktura sztucznego neuronu [4] . . . . .	9
1.3	Płytką i głęboką sieć neuronową . . . . .	9
1.4	Prawidłowe położenia latek $3 \times 3$ na mapie cech wejściowych $5 \times 5$ [8] . . . . .	13
1.5	Dopełnianie (padding) wejścia o wymiarach $5 \times 5$ , aby móc wyodrębnić 25 latek $3 \times 3$ . [8] . . . . .	14
1.6	Zastosowanie CNN do zaganienia klasyfikacji [9] . . . . .	15
1.7	Schemat wyłączania węzłów przez metodę Dropout . . . . .	18
1.8	Funkcje aktywacji . . . . .	19
2.1	Podziór danych opisujących artystów . . . . .	25
2.2	Rozkład liczby obrazów na artystę . . . . .	26
2.3	Liczba artystów w różnych stylach malarstw . . . . .	27
2.4	Liczba artystów i obrazów dla każdej grupy stylu malarstwego . . . . .	28
2.5	Liczba obrazów dla każdej grupy stylu malarstwego . . . . .	28
2.6	Liczba obrazów i artystów dla każdej grupy stylu malarstwego . . . . .	29
2.7	Pięć przykładowych, losowo wybranych obrazów . . . . .	30
2.8	Grupa przykładowych, losowo wybranych obrazów impresjonistycznych . . . . .	31
2.9	Grupa przykładowych, losowo wybranych obrazów postimpresjonistycznych . . . . .	32
3.1	Biblioteki użyte przy tworzeniu modelu . . . . .	35
3.2	Kod w Pythonie służący wczytaniu danych treningowych i walidacyjnych . . . . .	39
3.3	Kod w Pythonie służący do budowy i komplikacji modelu . . . . .	40
3.4	Kod w Pythonie służący do trenowania modelu i pokazanie przebiegu progresu drugiej epoki . . . . .	40
3.5	Wyniki treningu dla epok 20-25 . . . . .	41
3.6	Wyniki treningu i walidacji . . . . .	42
3.7	Mapa cieplna dla przyciągania $>0$ . . . . .	42
3.8	Kod w Pythonie pozwalający na zbudowanie i wytrenowanie modelu . . . . .	45
3.9	Wyniki treningu dla epok 20-25 . . . . .	46
3.10	Wyniki treningu i walidacji . . . . .	46

## Spis rysunków

---

3.11	Mapa cieplna dla $\text{precyzji} > 0$	47
3.12	Kod reprezentujący manualny podział danych na zbiory treningowy i walidacyjny	48
3.13	Wyniki treningu i walidacji	50
3.14	Wyniki treningu dla epok 20-25	50
3.15	Mapa cieplna dla $\text{precyzji} > 0$	51
3.16	Wyniki treningu i walidacji	53
3.17	Mapa cieplna dla $\text{precyzji} > 0$	54
3.18	Wyniki metryk na wykresie słupkowym ( $\text{precyzja} > 0$ )	54
3.19	Mapa cieplna dla $\text{precyzji} > 0$	55
3.20	Wyniki treningu dla epok 40-50	56
3.21	Wyniki treningu i walidacji	57
3.22	Macierz pomyłek dla $\text{precyzji} > 0$	58
3.23	Wyniki treningowe i walidacyjne przedstawione na wykresie słupkowym	61
3.24	Porównanie wyników treningowych i walidacyjnych ze średnimi wynikami	61
3.25	Wyniki treningowe i walidacyjne dla trzech wersji Modelu 2 po redukcji liczb klas	63
3.26	Biblioteki użyte w modelu	65
3.27	Zdefiniowanie DataBlock	66
3.28	Kod pozwalający na tworzenie modelu i jego uczenie	67
3.29	Wyniki treningu dla poszczególnych epok	67
3.30	Macierz pomyłek	68

## **Spis tabel**

1.1	Porównanie warstw płytowych i głębokich . . . . .	10
1.2	Funkcje aktywacji . . . . .	20
1.3	Porównanie algorytmów optymalizacji . . . . .	21
2.1	Podział artystów według liczby obrazów . . . . .	26
3.1	Funkcje bibliotek wykorzystanych do budowy modeli . . . . .	36
3.2	Szczegóły modelu kontrolnego . . . . .	39
3.3	Szczegóły modelu 0.5 . . . . .	44
3.4	Szczegóły Modelu 1 . . . . .	49
3.5	Szczegóły Modelu 2 . . . . .	52
3.6	Porównanie wyników trenowania Modelu 2 i Modelu 2a . . . . .	57
3.7	Porównanie wartości metryk Modelu 2 i Modelu 2a . . . . .	57
3.8	Wyniki walidacji krzyżowej dla pięciu przekrojów . . . . .	60
3.9	Porównanie wyników dla Modelu 2 i jego wersji ze zredukowaną liczbą klas . . . . .	64
3.10	Porównanie modeli na podstawie cech . . . . .	69
3.11	Porównanie wyników wszystkich modeli . . . . .	70