



Kierunek studiów: Informatyka stosowana

STUDIA STACJONARNE

Metody Inżynierii Wiedzy - Projekt

ZASTOSOWANIE KONWOLUCYJNYCH SIECI NEURONOWYCH
I UCZENIA MASZYNOWEGO DO ROZPOZNAWANIA ZNAKÓW
DROGOWYCH NA PODSTAWIE ZDJĘĆ

Wykonwacy:

Krystian Dutka

Gabriela Pałk

Adrian Maślanka

Artur Kasprzyk

Kraków, rok akad. 2021/2022

1. Wstęp	3
2. Cel i zakres	3
3. Dane	4
4. Wprowadzenie teoretyczne	7
4.1. Rozpoznawanie obrazu - na czym polega i jak działa	7
4.2. Analiza metod rozpoznawania znaków	9
4.2.1. Metody z wykorzystaniem sztucznych sieci neuronowych	9
4.2.2. Metody wykorzystujące Ukryte Modele Markowa	9
4.2.3. Metody wykorzystujące momenty geometryczne	10
4.2.4. Metody wykorzystujące transformatę Fouriera	10
4.2.5. Metody wykorzystujące transformacje czasowo-częstotliwościowe	10
4.2.6. Metody wykorzystujące przekształcenie obrazu w przestrzeń parametryczną	10
4.3 Sieci neuronowe	11
4.3.1 Sieć neuronowa	11
4.3.2 Typy sieci neuronowych	11
4.4 Konwolucyjne sieci neuronowe	13
4.5 Wykorzystanie konwolucyjnych sieci neuronowych w rozpoznawaniu obrazów	15
4.5.1 Współdzielone Wagi i Współczynniki Odchyлеń	16
4.5.2 Ilość Filtrów w Warstwach Konwolucyjnych Głębokość	18
4.5.3 Rozmiar Obrazu Wyjściowego	18
5. Część projektowa	19
5.1 Biblioteki wykorzystane w projekcie	19
5.2 Model	20
5.3 Wizualizacja filtrów konwolucyjnych	24
5.4 Wartości przewidziane i prawdziwe w warstwie Softmax	25
5.5 Aplikacja	25
6. Bibliografia	27

1. Wstęp

W dobie cyfryzacji wartość bezpieczeństwa wzrosła do punktu, gdzie odłamy informatyki zaczęły tworzyć następne terminy zajmujące się tylko działaniem zwiększającym ochronę osobistą. Skupiając się na teraźniejszości, świat oferuje wiele rzeczywistych rozwiązań wspomagających poczucie zaufania w wielu dziedzinach życia. Jednym z nich jest transport i pomysł stworzenia poruszającego się samodzielnie samochodu. To zagadnienie jest problematyczne ze względu na wiele czynników, np. wielu kierowców zachowuje się nieprzewidywalnie. Człowiek posiada zmysły, dzięki którym może analizować otoczenie, lecz czy jest możliwe ich odwzorowanie? Jednym z pomysłów zwiększającym poczucie pewności na drodze jest Traffic Sign Recognition, czyli system rozpoznawania znaków drogowych, który informuje kierowcę w czasie rzeczywistym o napotkanym oznakowaniu.

Podobne badania w tej dziedzinie są prowadzone od dawna, a zagadnienie detekcji i rozpoznawania znaków drogowych staje się obecnie coraz bardziej popularne. Przykłady takich systemów można znaleźć już na współczesnych drogach.

2. Cel i zakres

Celem naszej pracy jest próba stworzenia modelu do rozpoznawania znaków drogowych. Ważnym założeniem jest próba osiągnięcia tego celu jak najprostszymi metodami. Nasze działania będą opierać się o jak najmniej złożone algorytmy.

W projekcie zastosowaliśmy konwolucyjne sieci neuronowe, które świetnie radzą sobie z przetwarzaniem obrazu. Ponadto, wykorzystaliśmy kilka bibliotek z dziedziny uczenia maszynowego, np. tensorflow, która cechuje się wydajnością i skalowalnością. Środowisko, w którym pracowaliśmy, to Jupyter z interpreterem języka Python.

Opracowanie teoretyczne zrealizowaliśmy w kilku podpunktach:

- Rozpoznawanie obrazu - na czym polega i jak działa
- Analiza metod rozpoznawania obrazu
- Sieci neuronowe
- Konwolucyjne sieci neuronowe
- Wykorzystanie konwolucyjnych sieci neuronowych w rozpoznawaniu obrazów

3. Dane

Projekt wymagał przygotowania lub znalezienia zbioru danych. Skorzystaliśmy ze zbioru Traffic Signs Classification ze strony [Traffic Signs Classification | Kaggle](#).

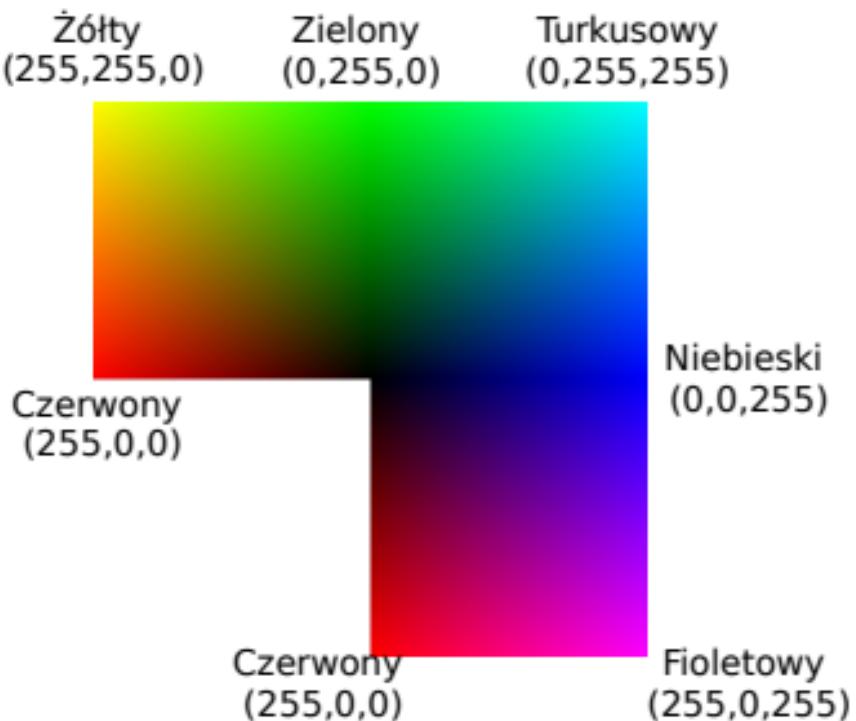
Znaki drogowe charakteryzują się dwoma cechami: posiadają typowe dla siebie kolory oraz mają geometryczne kształty. Biorąc pod uwagę tylko 4 najważniejsze kategorie znaków, które zostały przedstawione na rysunku 3.1., nakazu, zakazu, ostrzegawcze i informacyjne - możemy zobaczyć, że każda z tych kategorii ma swój dominujący kształt. Dla przykładu - znaki ostrzegawcze posiadają głównie kolor żółty i trójkątny kształt. Szukając znaków czerwonych, czyli zakazu, można skupić się na okręgach. Natomiast przeszukując obszary niebieskie trzeba szukać zarówno prostokątów, jak i okręgów.



Rys. 3.1. Przykład znaków ostrzegawczych, zakazu, informacyjnych i nakazu.

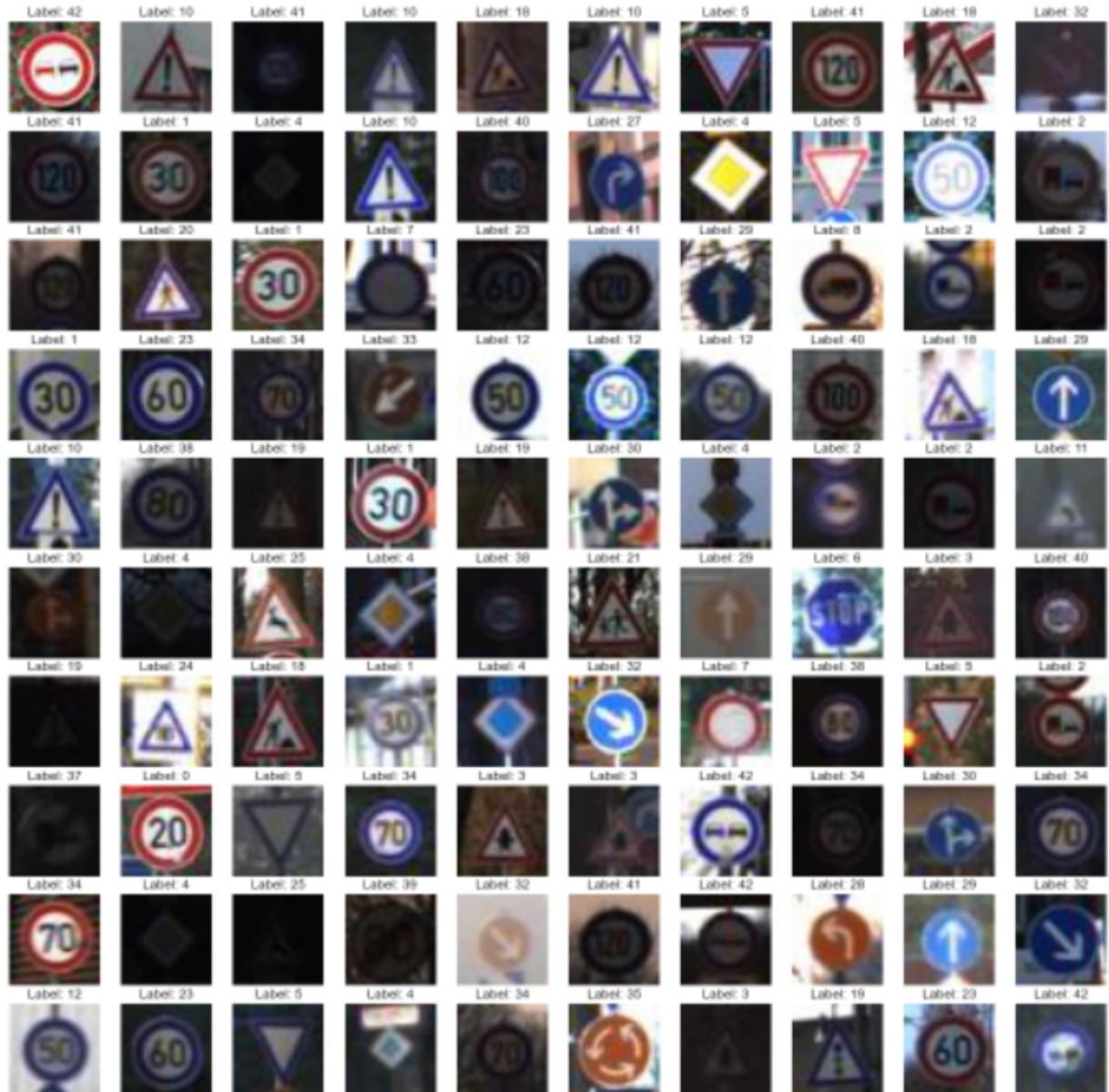
Ważnym elementem analizy obrazu jest rozdzielcość, która decyduje o ilości szczegółów. Chcielibyśmy znać ich jak najwięcej, ale powoduje to wzrost czasu przetwarzania obrazu. W wykorzystanych przez nas danych występują zdjęcia o rozmiarze 32x32 pikseli, co pozwala nam na szybką analizę. Oprócz tego, ważnym elementem analizy obrazów kolorowych jest dobór odpowiedniej przestrzeni barw. Nasz zbiór danych zawiera zdjęcia modelu RGB, który był wzorowany na właściwościach odbiorczych ludzkiego oka.

Model, przedstawiony na rysunku 3.2., powstał ze złożenia 3 kolorów - czerwonego, zielonego i niebieskiego.



Rys. 3.2. Przestrzeń barw RGB

Nasze dane zawierają 73 139 zdjęć, które zostały podzielone na 43 klasy w zależności od tego co dane zdjęcie ilustruje, co przedstawiono na rysunku 3.3. W zbiorze do trenowania znalazło się 80% próbek, a pozostałe 20% zostało przeznaczone do testowania.



Rys. 3.3. Przykład próbek wraz z przydzieloną etykietą z naszego zbioru danych.

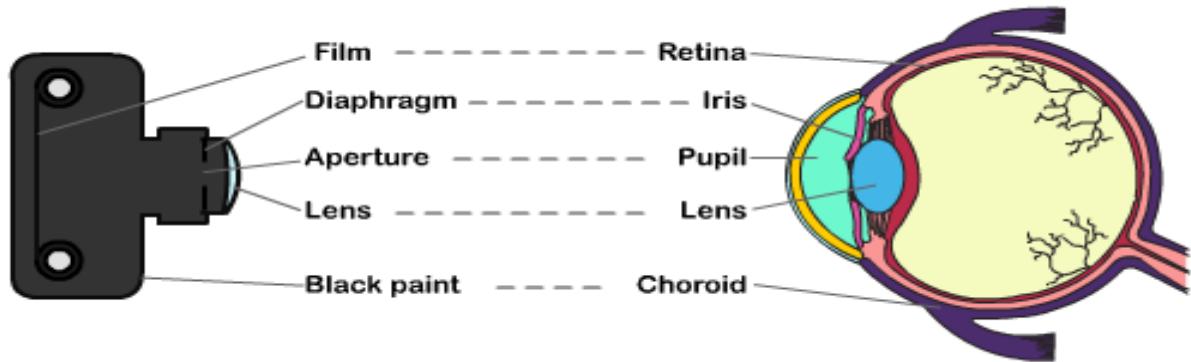
4. Wprowadzenie teoretyczne

4.1. Rozpoznawanie obrazu - na czym polega i jak działa

Dla lepszego zrozumienia badanego problemu musimy wyjaśnić kilka pojęć z nim związanych.

Pierwszym z nich jest rozpoznawanie obrazów. Jak pisze Wikipedia „Rozpoznawanie obrazu (także: widzenie komputerowe, od ang. computer vision) – przetwarzanie obrazu przez maszynę za pomocą urządzeń zewnętrznych (np. skaner) w opis cyfrowy tegoż obrazu w celu dalszego przetwarzania.”

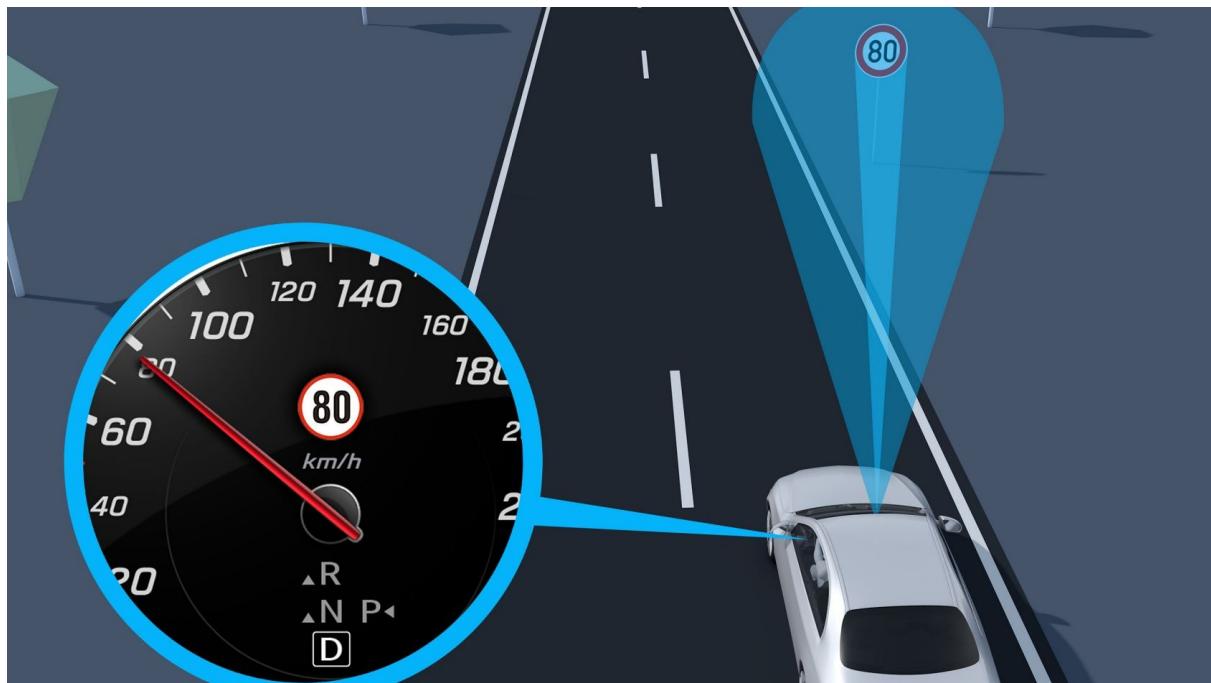
Dla przeciętnego człowieka nie stanowi to żadnego problemu, ale dla komputera już tak. Ale po co tak naprawdę nam komputer, który będzie w stanie rozpoznawać poszczególne elementy na zdjęciach lub wideo? Od wielu lat naukowcy pracują nad stworzeniem idealnego systemu, który będzie w stanie analizować świat wokół nas i reagować na niego w czasie rzeczywistym. Kamery cyfrowe mają wiele wspólnego z ludzkim okiem, działają na podobnej zasadzie jak przedstawia to rysunek 4.1. Promienie światła przechodzą odpowiednio przez obiektyw, a w przypadku oka przez soczewkę. Następnie ulegają załamaniu i padają na powierzchnię siatkówki, a w przypadku kamery na fotoreceptory, gdzie promienie są przetwarzane na sygnał cyfrowy. Ten sygnał z kolei jest przetwarzany przez komputer, a u człowieka przez mózg.



Rys. 4.1. Porównanie budowy oka oraz aparatu cyfrowego

W przypadku komputerów analizą i przetwarzaniem takich sygnałów najczęściej zajmują się odpowiednie algorytmy, które wcześniej muszą być stworzone przez ludzkie ręce oraz wyuczone na odpowiednio przygotowanych danych. Systemy informatyczne już są w stanie rozpoznawać znaki drogowe i wyświetlać informację dla kierowcy np. z ograniczeniem prędkości, znakiem stop. Mogą też zauważać jakiś obiekt na jezdni i odpowiednio szybko zareagować, aby nie doszło do tragedii. Naukowcy dążą do stworzenia samochodu autonomicznego, który będzie w stanie sam jeździć po drogach, ale do tego potrzeba wielu kamer,

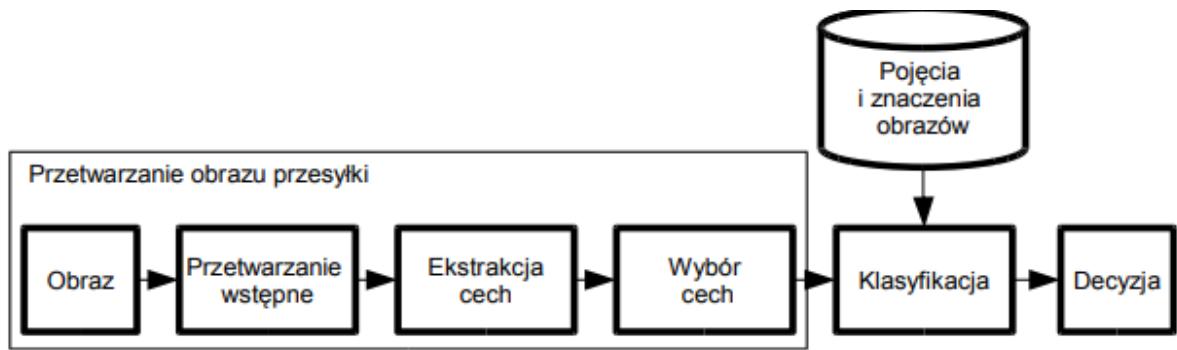
które przetwarzają świat na obraz cyfrowy. Obraz ten następnie jest analizowany przez stworzone, wyuczone algorytmy, a następnie wysyłając odpowiednią informację do systemów sterujących pojazdem. Przykładowy komunikat oraz zastosowanie ukazuje rysunek 4.2.



Rys. 4.2. Przykład działania rozpoznawania znaków drogowych

Ogólnie mówiąc zadaniem rozpoznawania obrazów jest określenie przynależności różnego typu wzorców lub obiektów widocznych na obrazie do pewnych, wcześniej zdefiniowanych klas. Do realizacji tego typu zadań wykorzystuję się sztuczne sieci neuronowe – czyli system przetwarzający informację oparty na budowie biologicznych układów nerwowych i procesach zachodzących w nim.

Dzięki rozwojowi komputerów i ich mocy obliczeniowej na przestrzeni lat, a także stopniowym rozwojom algorytmicznych metod służących do analizy i przetwarzania cyfrowych informacji, rozpoznawanie obrazów stawało się coraz popularniejsze i wykorzystywane w szerszej skali. Na początku w takich dziedzinach jak medycyna, przemysł czy właśnie bezpieczeństwo, a w obecnych czasach służy nam w naszym codziennym życiu i ułatwia nam je.



Rys. 4.3. Schemat procesu rozpoznawania obrazów

Jak pokazuje schemat rysunku 4.3, obraz w postaci cyfrowej dostarczony przez odpowiednie narzędzie poddawany jest wstępemu przetwarzaniu, a następnie wykonuje się proces ekstrakcji cech charakterystycznych reprezentujące dane są grupowane i kodowane dla potrzeb systemu i stają się wektorem cech. W procesie rozpoznawania można wyróżnić etap pozyskiwania i selekcji cech oraz etap klasyfikacji, w której przypisujemy cyfrowej reprezentacji obrazu odpowiednią klasę.

4.2. Analiza metod rozpoznawania znaków

4.2.1. Metody z wykorzystaniem sztucznych sieci neuronowych

To najczęściej stosowana technika do rozpoznawania znaków. Metoda wykorzystuje wielowarstwową sieć neuronową wspieraną przez algorytm uczenia z nauczycielem. Metoda świetnie radzi sobie z rozpoznawaniem pisma odręcznego w 93% oraz dla znaków drukowanych w 100%. Metod opartych na sieci MLP cieszą się popularnością z powodu łatwego procesu uczenia oraz szybkiego procesu decyzyjnego w trakcie klasyfikacji. Głównymi wadami są złożone procesy optymalizacji oraz konieczność przechowywania ogromnej ilości danych uczących.

4.2.2. Metody wykorzystujące Ukryte Modele Markowa

Rozwiążanie bazujące na modelach znaków, które niosą za sobą pewne wspólne elementy, które na etapie przechowywania i porównania pewne informacje są powielane. Rozwiążaniem tego problemu okazało się utworzenie modelu, którego podstawowe jednostki składały się z różnych fragmentów znaku (ang. subcharacter). Podział na segmenty generuje trudności w procesie tworzenia modeli, ponieważ na etapie segmentacji mogą być wydzielone pojedyncze znaki. Metoda ta dobrze sprawdza się przy przetwarzaniu pisma, ale wadą jest duża złożoność obliczeniowa.

4.2.3. Metody wykorzystujące momenty geometryczne

Metoda ta wykorzystuje moment geometryczny, dzięki temu świetnie radzi sobie z rozpoznawaniem, np. pisma niezależnie od orientacji, jego położenia czy rozmiaru. Do opisów obiektów używamy parametrów, które są kombinacją momentów centralnych oraz są niezależne od translacji i obrotu.

4.2.4. Metody wykorzystujące transformatę Fouriera

Metoda wykorzystuje transformatę Fouriera FT (ang. Fourier Transform), którą najczęściej stosuje się w przetwarzaniu sygnałów oraz w systemach rozpoznawania obrazów. Umożliwia ona przejście z funkcji opisanej w dziedzinie czasu do opisanej w dziedzinie częstotliwości oraz pozwala na operację odwrotną. Wadą tego rozwiązania jest to, że nie radzi sobie z rozpoznaniem kiedy kontury nie są ciągłe, a obrazy zazwyczaj posiadają rozmyte i zanikające krawędzie oraz różne szумy.

4.2.5. Metody wykorzystujące transformacje czasowo-częstotliwościowe

Metoda wykorzystuje transformację czasowo-częstotliwościowe, a przede wszystkim falkową. Dzięki niej możemy stworzyć reprezentację sygnału w czasie i w częstotliwości. Opisana metoda jest wykorzystywana w dziedzinie przetwarzania obrazu. Najczęściej wykorzystuję się transformację falkową i dyskretną falkową.

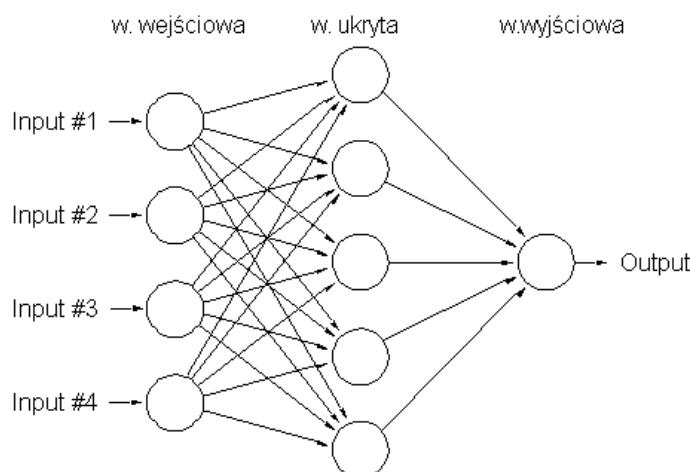
4.2.6. Metody wykorzystujące przekształcenie obrazu w przestrzeń parametryczną

Metoda wykorzystuje przekształcenie obrazu w przestrzeni parametrycznej, czyli przekształcanie obrazu znaku w przestrzeń parametrów (z wykorzystaniem przekształcenia Hougha lub Radona). Dzięki takiemu zabiegowi otrzymujemy informacje o lokalnych właściwościach obrazu, np. linie proste. Wadą tej metody jest wrażliwość reprezentacji parametrycznej na rotację, zmianę skali i przesunięcie obiektu. Mimo to w pewnych przypadkach metoda ta jest dość skuteczna, podczas eksperymentów skuteczność dla obrazów cyfrowych wykazano na poziomie 96%.

4.3 Sieci neuronowe

4.3.1 Sieć neuronowa

System przeznaczony do przetwarzania informacji. Budowa i zasada działania są wzorowane na funkcjonowaniu rzeczywistego systemu nerwowego. Cechą sieci neuronowej jest jej zdolność uczenia się na podstawie przykładów oraz generalizacja, czyli umiejętność do uogólniania zdobytej wiedzy (generalizacja \neq dokładność).



Rys. 4.4. Ogólna postać sieci neuronowej (w przypadku sieci wielowarstwowej)

Wspólną cechą wszystkich sieci neuronowych jest to, że na ich strukturę składają się neurony połączone ze sobą synapsami. Z synapsami związane są wagi, czyli wartości liczbowe, których interpretacja zależy od modelu.

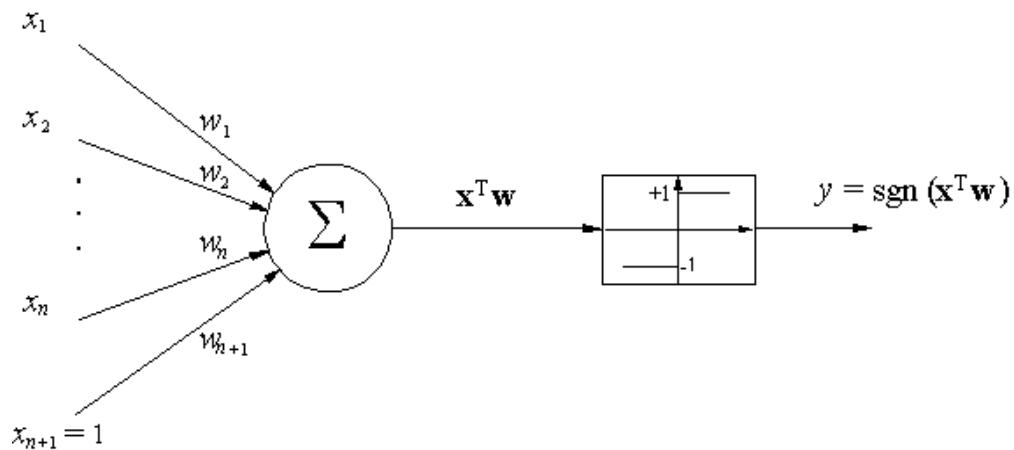
4.3.2 Typy sieci neuronowych

Rodzaj sieci neuronowej zależy od sposobu połączenia neuronów tej sieci oraz od kierunku przepływu sygnałów w sieci. Każdy typ sieci ma własne metody doboru wag, czyli uczenia.

Rodzajów sieci neuronowych:

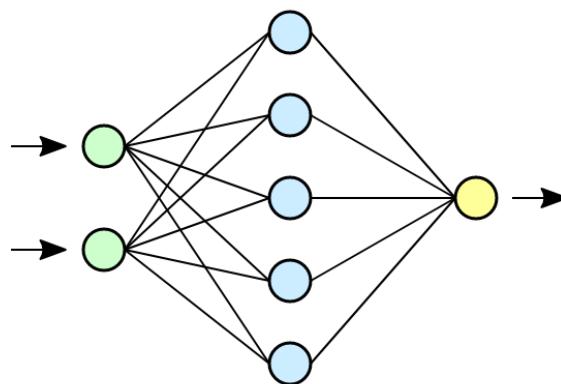
- Sieci jednokierunkowe
 - jednowarstwowe
 - wielowarstwowe (przykładowy wygląd na rysunku 4.4 powyżej)

Typowym przykładem jest perceptron jednowarstwowy, przedstawiony na rysunku 4.5 poniżej, który składa się z neuronów ułożonych w warstwach o jednym kierunku przepływu sygnałów i połączeniach międzywarstwowych jedynie między kolejnymi warstwami.



Rys. 4.5. Sztuczny neuron perceptron

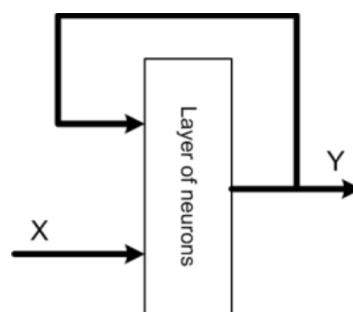
Sieć tego typu posiada warstwę wejściową, wyjściową i warstwy ukryte. Z funkcjonalnego punktu widzenia układ taki można traktować jako układ aproksymacji funkcji nieliniowej wielu zmiennych $y = f(u)$.



Rys. 4.6. Uproszczony schemat jednokierunkowej sieci neuronowej.

- Sieci rekurencyjne

W sieciach tego typu występuje przynajmniej jedno sprzężenie zwrotne. Sygnały wyjściowe warstwy podawane są na jej wejścia, co powoduje pewną dynamikę w pracy sieci.

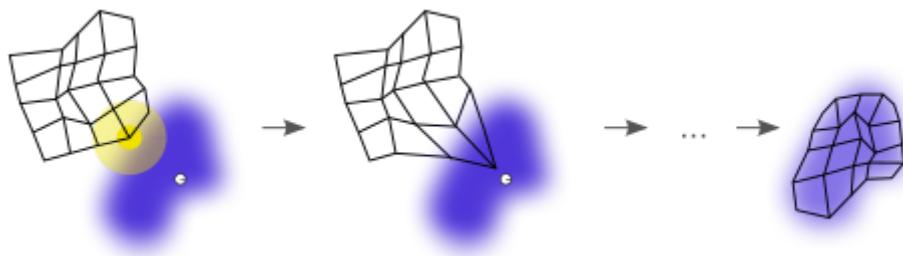


Rys. 4.7. Schemat sieci rekurencyjnej.

Sygnały wejściowe w takiej sieci zależą zarówno od aktualnego stanu wejścia jak i od sygnałów wyjściowych w poprzednim cyklu. Strukturę ogólną takiej sieci przedstawia poniższy rysunek.

- Sieci komórkowe

Sieci sprzężone wzajemnie między elementami jedynie najbliższego sąsiedztwa. Połączenia te są w ogólności nieliniowe i opisane poprzez układ równań różniczkowych.



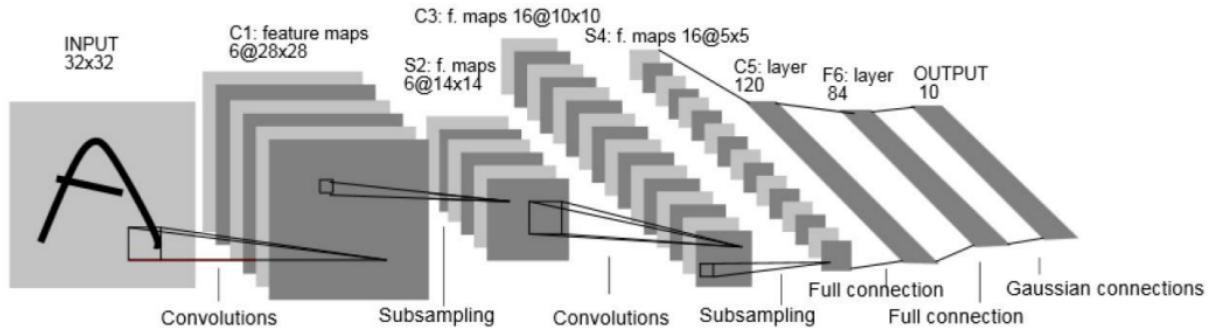
Rys. 4.8. Trening sieci Kohonena (najbardziej znana i najczęściej stosowana sieć samoucząca się, realizująca zasadę samoorganizacji)

Podstawową trudność w stosowaniu tego typu sieci stanowi opracowanie skutecznej, efektywnej i uniwersalnej metody projektowania.

4.4 Konwolucyjne sieci neuronowe

Specjalnym rodzajem sieci neuronowej, która wyjątkowo dobrze radzi sobie z przetwarzaniem obrazu, jest konwolucyjna sieć neuronowa. Konwolucyjne sieci neuronowe wykorzystują system podobny do perceptronu wielowarstwowego, który został opracowany w celu zminimalizowania wymagań przetwarzania. Warstwy CNN zawierają warstwę wyjściową, warstwę wejściową, a także warstwę ukrytą, obejmującą wiele warstw zbiorczych - warstwy konwolucyjne, warstwy normalizacji i warstwy w pełni połączone.

Dzięki zwiększonej wydajności i minimalnym ograniczeniom, sieci neuronowe fałdowe są znacznie bardziej efektywne i łatwiejsze w szkoleniu do naturalnego przetwarzania obrazu i języka. Głębokie sieci konwolucyjne (CNN) potrafią stopniowo filtrować różne części danych uczących i wyostrzać ważne cechy w procesie dyskryminacji wykorzystanym do rozpoznawania lub klasyfikacji wzorców.

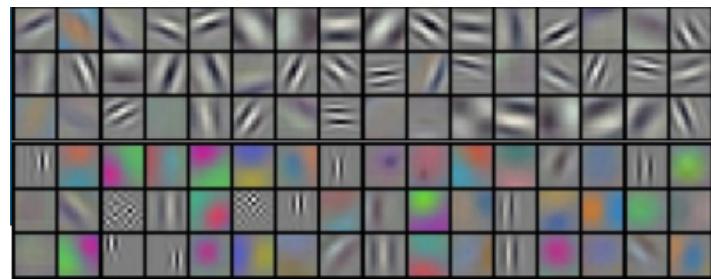


Rys. 4.9. Głębokie sieci konwolucyjne CNN

W każdej konwolucji możemy wyróżnić:

- Ilość parametrów w każdej warstwie:
 - Ilość kanałów * Ilość filtrów * szerokość filtru * wysokość filtru
- Ilość jednostek ukrytych w każdej warstwie:
 - Ilość filtrów * szerokość wzorca * wysokość wzorca

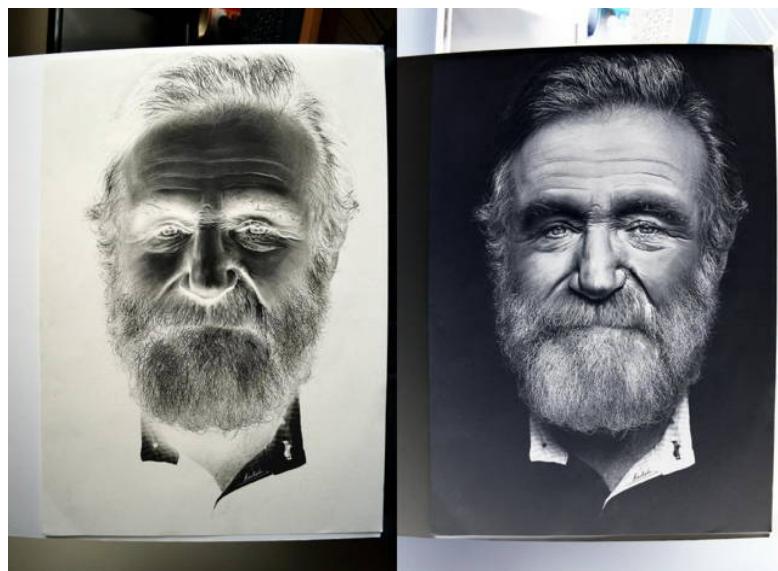
Konwolucje pozwalają na ekstrakcję prostych cech w początkowych warstwach sieci, np. rozpoznają krawędzie o różnej orientacji lub różnokolorowe plamy, a następnie plastry w kolejnych warstwach. Każda warstwa konwolucyjna zawiera cały zbiór filtrów (np. 8 filtrów), a każdy z nich generuje osobną mapę aktywacji 2D. Układamy te mapy aktywacyjne na stercie, wzdłuż wymiaru głębokości sieci i produkujemy obraz wyjściowy.



Rys. 4.10. Próbkowanie i Konwolucje

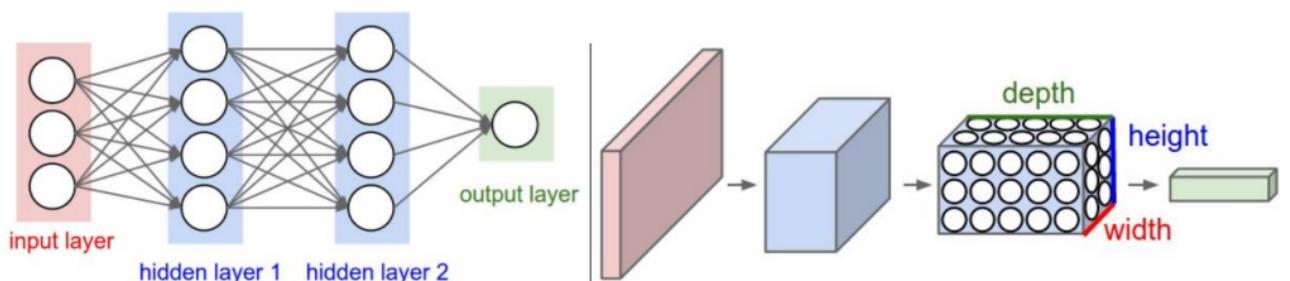
4.5 Wykorzystanie konwolucyjnych sieci neuronowych w rozpoznawaniu obrazów

Formalnie zdefiniowana matematyczna operacja splotu, czy konwolucji, jest dość zawiła i wykorzystuje całki. Obrazowo operacja konwolucji jest przekształceniem macierzowym fragmentów zdjęcia, które ma na celu wydobycie informacji o konkretnych cechach obrazu. Negatyw jest to po prostu przemnożenie zdjęcia przez macierz -1, aby odwrócić kolory.



Rys. 4.11. Portret w negatywie

Sieci konwolucyjne porządkują jednostki obliczeniowe w strukturze 3D: szerokość, wysokość i głębokość. Neurony w każdej warstwie są połączone do małego regionu warstwy poprzedniej zamiast do wszystkich, jak to jest w typowych sieciach neuronowych.

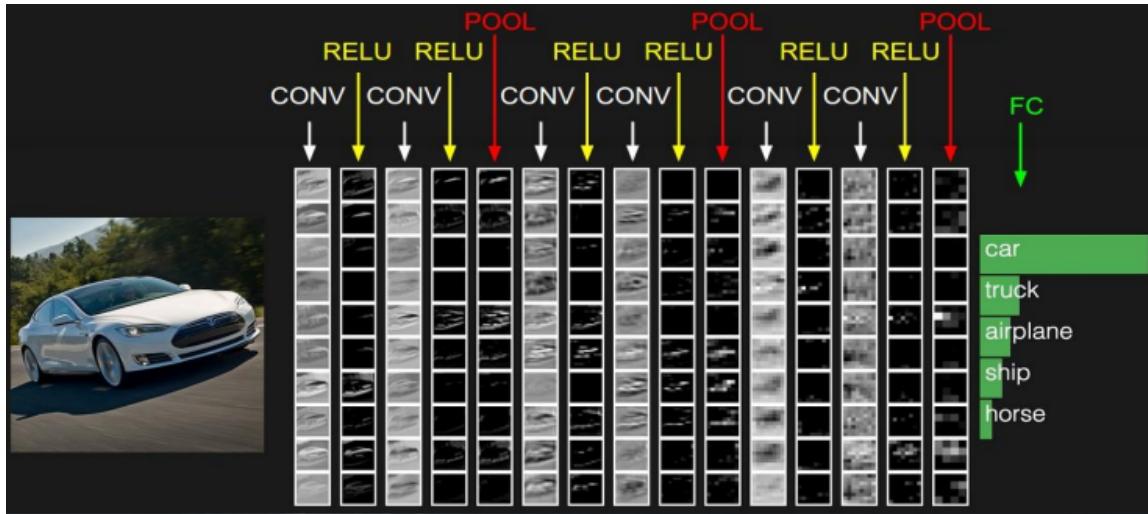


Rys. 4.12. Porównanie typowych i głębokich konwolucyjnych architektur sieci

Sieć konwolucyjna jest zwykle sekwencją warstw, które transformują jeden obraz danych do drugiego poprzez funkcję różniczkowalną.

CNNs zwykle składają się z trzech typowych warstw:

- Warstwa konwolucyjna zawierająca zbiór adaptowalnych filtrów, np. [5x5x3]
- Warstwę łączenia,
- W pełni połączoną sieć implementującą sieci MLP,SVM, czy SoftMax.

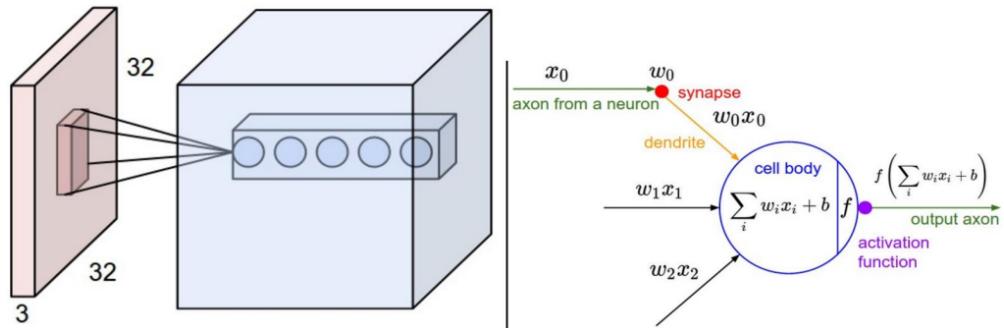


Rys. 4.13. Konwolucyjne sieci neuronowe na przykładzie samochodu

4.5.1 Współdzielone Wagi i Współczynniki Odchyleń

Każdy plaster (macierz, slice) w warstwie wykorzystuje te same wag i współczynniki odchyleń (biases) dla wszystkich neuronów. W praktyce każdy neuron wyznacza gradient dla wszystkich wag w trakcie propagacji wstecznej, ale te gradienty zostają dodane w każdym plastrze (macierzy), a aktualizacji wag dokonujemy tylko w pojedynczym zwycięskim plastrze. W taki sposób, wszystkie neurony w pojedynczym plastrze wykorzystują te same wektory wagowe. Warstwa konwolucyjna wykorzystuje te wektory do obliczenia konwolucji wag z obrazu danych wejściowych. Ponieważ ten sam zbiór wag jest wykorzystywany do obliczenia wartości wyjściowych w plastrze, dlatego można takie przekształcenie nazwać **filtrem adaptacyjnym** powodującym przekształcenie danych wejściowych na skalar wyjściowy.

Na poniższym rysunku 4.14 mamy różne neurony (5 z nich oblicza iloczyn skalarny ich wag z wydzielonym kawałkiem wejścia) dla całej głębokości (kolejnych plastrów), z których wszystkie są podłączone do tego samego regionu w obrazie wejściowym, gdzie połączenia są ograniczone do lokalnego wycinka obrazu (przestrzeni wejściowej):



Rys. 4.14. Warstwy konwolucyjne w sieci CNN

Warstwa konwolucyjna działa jak filtr adaptacyjny, który pozwala na wyznaczenie wartości w takich macierzach filtracji:

$$W = [W_{11}, W_{12}, W_{13} \\ W_{21}, W_{22}, W_{23} \\ W_{31}, W_{32}, W_{33}]$$

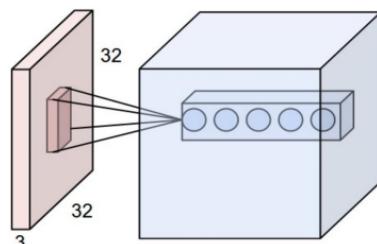
Wykorzystując również inne dobrze znane filtry możemy przekształcać obraz wejściowy tak jak przedstawiono poniżej na rysunku 4.15. (metody przetwarzania obrazów).

	Operation	Filter	Convolved Image
Identity		$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection		$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
		$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
		$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen		$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)		$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)		$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Rys. 4.15. Filtracja adaptacyjna

4.5.2 Ilość Filtrów w Warstwach Konwolucyjnych Głębokość

Każdy filtr uczy się rozpoznawać coś innego w obrazie wejściowym, np. pierwsza warstwa konwolucyjna bierze jako obraz wejściowy jedną składową koloru R, G lub B surowego obrazu, a różne neurony względem wymiaru głębokości tej sieci, które tworzą kolumnę nazywaną włóknem/filamentem, mogą aktywować się na skutek prezentacji różnie zorientowanych krawędzi lub kolorowych plam.



Rys. 4.16. Ilość filtrów w warstwach konwolucyjnych.

4.5.3 Rozmiar Obrazu Wyjściowego

Możemy obliczyć rozmiar przestrzenny obrazu wyjściowego wg $(W-F+2\cdot P)/S + 1$

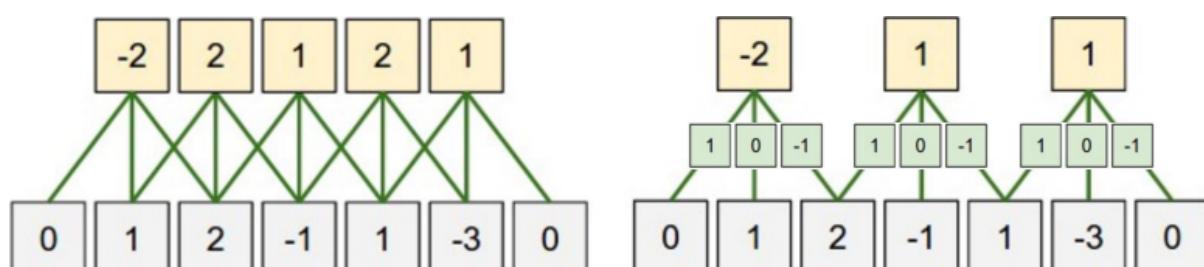
LEGENDA:

- W funkcja wielkości obrazu wejściowego
- F wielkości pola receptivejnego warstwy konwolucyjnej neuronów
- S krok
- P ilości zer tworzących otoczkę (padding) na krawędziach obrazu

przykład:

Dla wejście 7x7 i filtr 3x3 z krokiem 1 i otoczką 0

Wynik wyjście 5x5: $(7-3+2\cdot 0)/1+1=5$



Rys. 4.17. Rozmiar obrazu wyjściowego dla jednego wymiaru (szerokości lub wysokości) i takich samych zestawów wag współdzielonych przez wszystkie neurony.

5. Część projektowa

Sieć neuronowa składa się z warstwy wejściowej, ukrytych i wyjściowej. Stworzenie architektury dla danej sieci jest niezmiernie ciężkie, ponieważ istnieje niemalże nieskończona liczba kombinacji elementów tych warstw. Aby przejść do dokładnej analizy modelu i warstw, musimy najpierw omówić użyte przez nas biblioteki.

5.1 Biblioteki wykorzystane w projekcie

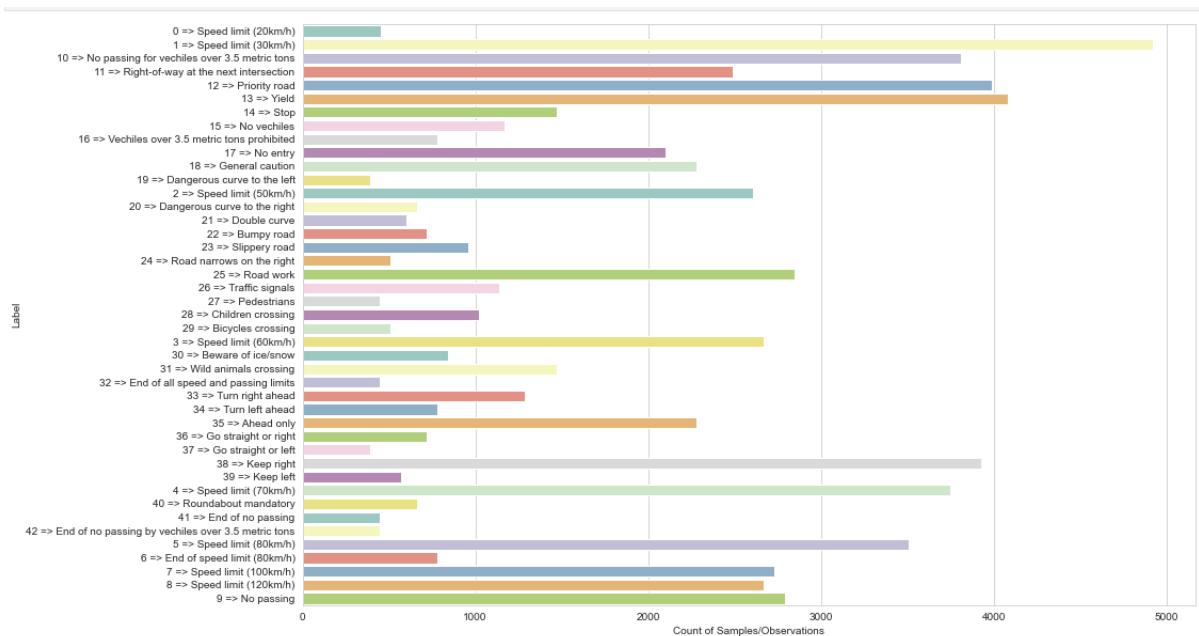
- **numpy** - biblioteka składająca się z obiektów tablic wielowymiarowych oraz zbioru procedur do przetwarzania tych tablic. Za pomocą NumPy można wykonywać matematyczne i logiczne operacje na tablicach.
- **pandas** - biblioteka Pythona służąca do pracy z zestawami danych. Posiada funkcje do analizy, czyszczenia, eksploracji i manipulacji danymi. Pandas pozwala analizować duże zbiory danych i wyciągać wnioski w oparciu o teorie statystyczne. Ponadto, potrafi oczyścić nieuporządkowane zbiory danych i uczynić je czytelnymi i przydatnymi.
- **tensorflow** - biblioteka służąca do szybkich obliczeń numerycznych, stworzona i udostępniona przez Google. Jest to biblioteka fundamentalna, która może być używana do tworzenia modeli Deep Learning bezpośrednio lub przy użyciu bibliotek wrapperów, które upraszczają ten proces, zbudowanych na bazie TensorFlow. W naszym projekcie korzystaliśmy głównie z **kerasa** - interfejsu dla sztucznych sieci neuronowych.
- **scikit-learn** - zapewnia wybór wydajnych narzędzi do uczenia maszynowego i modelowania statystycznego, w tym klasifikacji, regresji, grupowania i redukcji wymiarowości za pośrednictwem spójnego interfejsu. Opisana biblioteka została zbudowana na bazie NumPy, SciPy i Matplotlib.
- **matplotlib** - wszechstronna biblioteka do tworzenia statycznych, animowanych i interaktywnych wizualizacji w Pythonie. Użyta została do wizualizacji zdjęć i przypisany im numer etykiety oraz do wykresu przedstawiającego jak wraz z kolejnymi epokami wygląda metryka accuracy w podziale na zbiorze treningowym i testowym.
- **seaborn** - biblioteka wizualizacji danych oparta na matplotlib. Udostępnia ona wysokopoziomowy interfejs do rysowania atrakcyjnych i bogatych w informacje grafik statystycznych. Użyliśmy barplot, czyli wykres słupkowy do wizualizacji graficznej liczby próbek oraz podporządkowanym im etykiety.

5.2 Model

Istnieją dwa interfejsy API do definiowania modelu w keras:

- **sekwencyjny** - jest prosty, składa się z kolejnych warstw i grupuje się je liniowo w model `tf.keras.Model`
- **funkcjonalny** - jest w pewnym stopniu bardziej skomplikowany, ale dzięki temu można składać bardziej złożone modele z wieloma wejściami czy wyjściami, sklejać w locie czy współdzielić warstwami.

Zanim utworzy się model trzeba odpowiednie dane zaimportować, w naszym przypadku z pliku csv. Dobrą praktyką jest również wyświetlenie tych danych, żebyśmy mogli upewnić się z jaką strukturą i rozmiarem danych mamy do czynienia. Wynik tej operacji został przedstawiony na rysunku 5.1.



Rys. 5.1. Liczba próbek/obserwacji dla danych etykiet

W naszym projekcie skorzystaliśmy z sekwencyjnego interfejsu, będzie on spełniał nasze oczekiwania, co do wydajności naszego modelu. Model składa się z warstw konwolucyjnych 2D oraz Max Pooling 2D. Aby zwiększyć dokładność modelu użyliśmy funkcję aktywacji ReLU (Rectified Linear Activation). Relu stosuje do modelu potrzebną nielinowość i jest naprawdę szybka w trenowaniu.

Następnie spłaszczamy sieć w jednowymiarowy wektor. Robimy to po to, aby dopasować dane wejściowe w pełni połączonej warstwy do klasyfikacji. Zadaniem poprzedzających warstw splotowych sieci było wyodrębnienie cechy z obrazu wejściowego, a zadaniem warstwy spłaszczającej było sklasyfikowanie cech. Użyliśmy funkcji softmax do klasyfikowania tych cech, co wymagało jednowymiarowych danych wejściowych.

Przy tworzeniu takiego modelu ważne jest, aby go nie przetrenować. Jednym z bardziej popularnych metod zapobiegawczych to dodanie warstwy porzucenia (Dropout Layer). Dropout polega na losowym ustawnieniu wychodzących krawędzi ukrytych jednostek, czyli neuronów tworzących ukryte warstwy, na zero przy każdej aktualizacji fazy treningu. Metoda ta jest bardzo efektywna, ponieważ co każde przejście losowo wyłączane są połączenia. Dzięki temu sieć neuronowa nie nauczy się “na pamięć” zbyt szybko, ponieważ architektura, która przelicza, odrobinę się zmienia poprzez zerowanie losowych połączeń neuronów. Kod, który przedstawia tworzenie i trenowanie modelu ukazano na rysunku 5.2.

```
model = Sequential()

model.add(Convolution2D(filters = 128, kernel_size = (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2,2)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Convolution2D(256, (3, 3), activation='relu'))

model.add(Flatten(input_shape=(32, 32, 3)))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(43, activation = 'softmax'))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

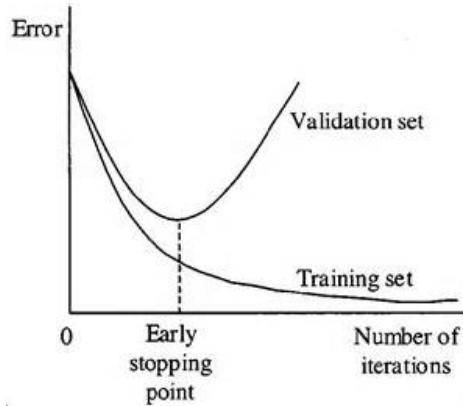
EarlyStop = EarlyStopping(monitor='val_loss',
                           patience=5,
                           verbose=1)

history = model.fit(X_train, y_train,
                     batch_size = 256,
                     epochs=10,
                     verbose=1,
                     validation_data = (X_test, y_test),
                     callbacks = [EarlyStop]
                    )

model.summary()
```

Rys. 5.2. Kod modelu sieci konwolucyjnej

Kolejny sposób do walki z przetrenowaniem polega na wykorzystaniu metody wcześniego zakończenia (Early Stopping). Wczesne zatrzymanie jest techniką optymalizacji stosowaną w celu zmniejszenia niedopasowania bez pogorszenia dokładności modelu. Głównym założeniem wcześniego zatrzymania jest przerwanie treningu, zanim model zacznie nadmiernie pasować. Proces ten przedstawia rysunek 5.3.



Rys. 5.3. Metoda wczesnego zakończenia - wizualizacja

Zbudowany przez nas model jest bardzo dobrze dobrany pod ten zbiór danych. Wskazuje na to m.in. dokładność (accuracy) tego modelu już w pierwszej epoce trenowania wynoszący 94%. Wyniki sieci neuronowej po wytrenowaniu przedstawiono na rysunku 5.4.

```

Epoch 1/10
229/229 [=====] - 51s 222ms/step - loss: 1.6922 - accuracy: 0.6082 - val_loss: 0.2674 - val_accuracy: 0.9409
Epoch 2/10
229/229 [=====] - 51s 224ms/step - loss: 0.3639 - accuracy: 0.9029 - val_loss: 0.1128 - val_accuracy: 0.9742
Epoch 3/10
229/229 [=====] - 61s 268ms/step - loss: 0.1939 - accuracy: 0.9466 - val_loss: 0.0714 - val_accuracy: 0.9826
Epoch 4/10
229/229 [=====] - 62s 272ms/step - loss: 0.1342 - accuracy: 0.9626 - val_loss: 0.0627 - val_accuracy: 0.9846
Epoch 5/10
229/229 [=====] - 60s 261ms/step - loss: 0.1043 - accuracy: 0.9711 - val_loss: 0.0592 - val_accuracy: 0.9852
Epoch 6/10
229/229 [=====] - 57s 247ms/step - loss: 0.0924 - accuracy: 0.9744 - val_loss: 0.0410 - val_accuracy: 0.9900
Epoch 7/10
229/229 [=====] - 53s 230ms/step - loss: 0.0747 - accuracy: 0.9795 - val_loss: 0.0416 - val_accuracy: 0.9889
Epoch 8/10
229/229 [=====] - 54s 237ms/step - loss: 0.0728 - accuracy: 0.9800 - val_loss: 0.0309 - val_accuracy: 0.9924
Epoch 9/10
229/229 [=====] - 54s 236ms/step - loss: 0.0655 - accuracy: 0.9819 - val_loss: 0.0395 - val_accuracy: 0.9908
Epoch 10/10
229/229 [=====] - 55s 240ms/step - loss: 0.0545 - accuracy: 0.9838 - val_loss: 0.0399 - val_accuracy: 0.9899
Model: "sequential_13"

```

Rys. 5.4. Wyniki wytrenowanej sieci neuronowej

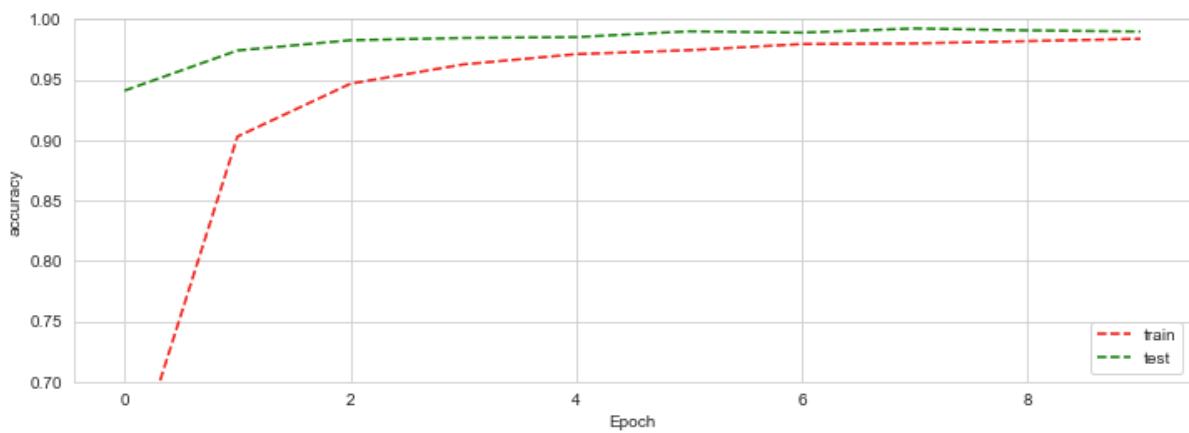
Wcześniej wytrenowaliśmy model bez trzeciej warstwy konwolucyjnej, z innymi parametrami warstwy gęstej i funkcją aktywacji typu sigmoid. Nasza sieć osiągnęła skuteczność na poziomie 60-80%, dlatego po dobraniu różnych warstw i ich parametrów oraz funkcji aktywacji wybraliśmy ten, który osiągał najlepsze wyniki dla naszego zbioru danych.

Gdy wytrenujemy model, możemy użyć funkcji **summary()** do wizualizacji wszystkich warstw, kształtu wyjściowego i liczbę parametrów wejściowych przy każdej warstwie oraz liczbę wszystkich parametrów. Wynik tej operacji przedstawiono na rysunku 5.5.

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_9 (Conv2D)	(None, 30, 30, 128)	3584
max_pooling2d_9 (MaxPooling 2D)	(None, 15, 15, 128)	0
conv2d_10 (Conv2D)	(None, 13, 13, 64)	73792
max_pooling2d_10 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_11 (Conv2D)	(None, 4, 4, 256)	147712
flatten_4 (Flatten)	(None, 4096)	0
dense_12 (Dense)	(None, 256)	1048832
dropout_8 (Dropout)	(None, 256)	0
dense_13 (Dense)	(None, 128)	32896
dropout_9 (Dropout)	(None, 128)	0
dense_14 (Dense)	(None, 43)	5547
<hr/>		
Total params:	1,312,363	
Trainable params:	1,312,363	
Non-trainable params:	0	

Rys. 5.5. Wyniki funkcji **summary()** - podsumowanie modelu sieci neuronowej

Jedna epoka oznacza przejście całego zbioru przez sieć oraz powrót. Nawet przy małej ilości tych epok zdołaliśmy otrzymać bardzo dobrą dokładność tego modelu. By lepiej zrozumieć jak uczyła się nasza sieć neuronowa narysowaliśmy wykres uczenia dla zdefiniowanej metryki oraz straty w podziale na próbkę treningową oraz testową, ukazany na rysunku 5.6. Według wykresu zbiór walidacyjny od 7 epoki już się nie polepsza, ale zbiór treningowy osiąga coraz lepszą dokładność. Dzieje się tak, dlatego że sieć neuronowa zaczęła zapamiętywać dane treningowe.



CNN Error: 1.01%

Rys. 5.6. Wykres prezentujący jak wraz z kolejnymi epokami wygląda metryka accuracy w podziale na zbiór treningowy i testowy

5.3 Wizualizacja filtrów konwolucyjnych

Po trenowaniu sieci i analizie posunęliśmy się o krok dalej, aby zobaczyć perspektywę od strony sieci neuronowej. Aby taką wizualizację osiągnąć trzeba się dowiedzieć jakie indeksy mają warstwy konwolucyjne. Można to zrobić za pomocą pętli i w sposób zautomatyzowany zapisać indeksy tych warstw. W naszym przypadku nie ma ich wiele, więc można zrobić to ręcznie. Wyodrębniamy dane wyjściowe modelu dla podanych warstw i wejść, a następnie przekazujemy je do naszego nowego modelu. Przekształcamy obrazy na tablicę, rozszerzamy wymiary i przekazujemy je jako parametry do naszej funkcji przewidywania. Wynik tego przewidywania zostanie wykorzystany do naszej wizualizacji. Wizualizację tego, co nasza sieć "widzi" przedstawiono na rysunku 5.7.

```
columns = 8
rows = 4
#from keras.preprocessing.image import img_to_array
numerLayers = [0,2,4]
outputs = [model.layers[i].output for i in numerLayers]
models = Model(inputs=model.inputs, outputs=outputs)

img = tf.keras.utils.img_to_array(X_test[0])
img = np.expand_dims(img, axis=0)
feature_output = models.predict(img)
for ftr in feature_output:
    fig=plt.figure(figsize=(12,12))
    for i in range(1, columns*rows+1):
        fig = plt.subplot(rows,columns,i)
        fig.set_xticks([])
        fig.set_yticks([])
        plt.imshow(ftr[0, :, :,i-1], cmap='gray')
    plt.show()

1/1 [=====] - 0s 61ms/step
```

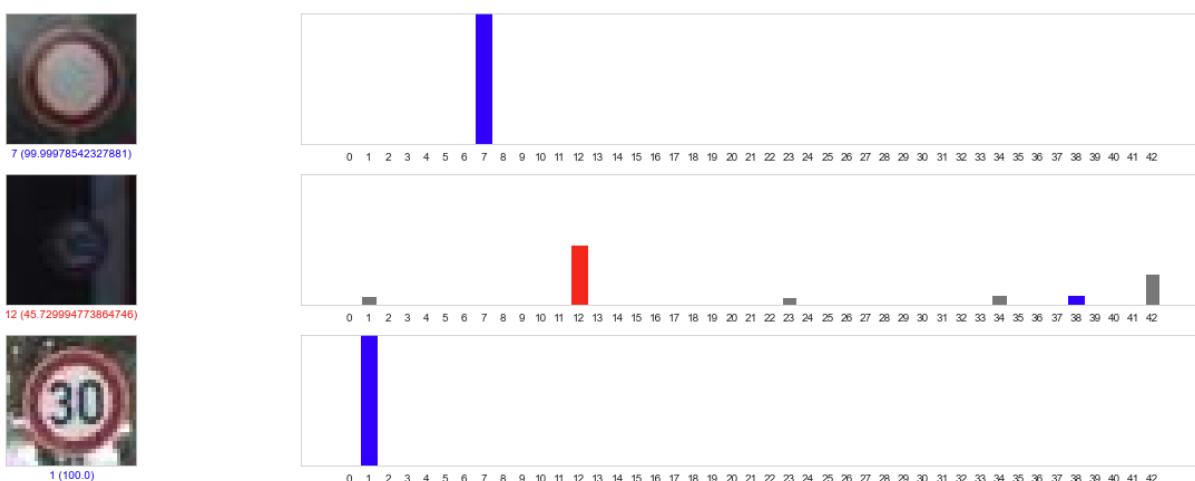


Rys. 5.7. Wizualizacja danych wyjściowych filtrów konwolucyjnych sieci neuronowej

5.4 Wartości przewidziane i prawdziwe w warstwie Softmax

Mając wytrenowany model, można go użyć do przewidywania niektórych obrazów. Prawidłowe przewidywania są zaznaczone na niebiesko, a nieprawidłowe na czerwono. Zastosowaliśmy wykres barplot, który przedstawia pewność tego modelu co do przewidzianej wartości. Wykres ten przedstawiono na rysunku 5.8.

Za pomocą takiego zestawienia można ponownie zaobserwować skuteczność modelu. Dla 100 próbek nasz model tylko raz nie był w stanie jednoznacznie przewidzieć prawidłową wartość. Zdjęcie jest bardzo rozmazane, więc pojawia się pytanie - czy istnieje inna sieć, która by sobie poradziła bez przetrenowania się? Należy pamiętać, że model może być błędny nawet przy dużej pewności.



Rys. 5.8. Obrazy i ich przewidziana wartość

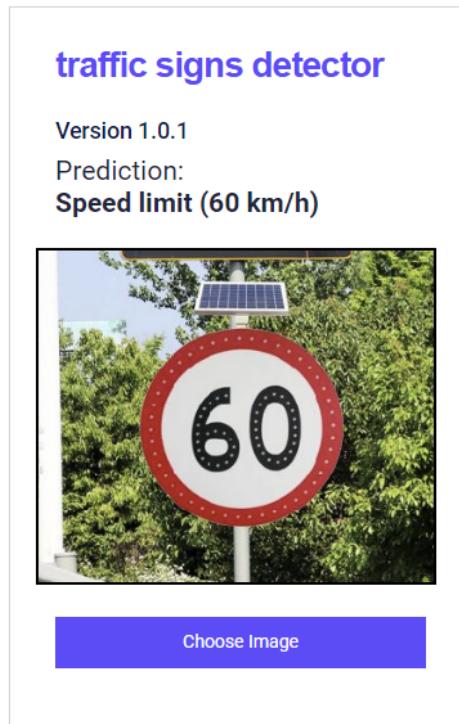
5.5 Aplikacja

Aplikacja webowa jest dopiero w etapie “proof of concept”, ale ma wiele gotowych funkcji i została wdrożona przy użyciu Dockera, Nginx i Heroku. Wykorzystaliśmy webowy framework Flask, który ułatwia cały proces połączenia modelu z aplikacją webową. Backend tej aplikacji jest zbudowany za pomocą właśnie Flaska, który wykorzystuje REST API umożliwiającą interakcję z bazą danych poprzez wykonywanie żądań HTTP. Układ naszej aplikacji przedstawiono na rysunku 5.9.

Name	Date modified	Type	Size
apps	07.06.2022 15:16	File folder	
core	07.06.2022 15:16	File folder	
media	07.06.2022 15:15	File folder	
nginx	07.06.2022 15:15	File folder	
staticfiles	07.06.2022 15:15	File folder	
.env	29.01.2022 06:50	ENV File	1 KB
.gitignore	29.01.2022 06:50	Text Document	1 KB
CHANGELOG.md	29.01.2022 06:50	Markdown Source...	2 KB
db.sqlite3	07.06.2022 15:16	SQlite3 File	128 KB
docker-compose.yml	29.01.2022 06:50	Yaml Source File	1 KB
Dockerfile	29.01.2022 06:50	File	1 KB
gunicorn-cfg.py	29.01.2022 06:50	Python Source File	1 KB
LICENSE.md	29.01.2022 06:50	Markdown Source...	1 KB
manage.py	29.01.2022 06:50	Python Source File	1 KB
package.json	29.01.2022 06:50	JSON Source File	1 KB
Procfile	29.01.2022 06:50	File	1 KB
README.md	29.01.2022 06:50	Markdown Source...	8 KB
requirements.txt	29.01.2022 06:50	Text Document	1 KB
runtime.txt	29.01.2022 06:50	Text Document	1 KB

Rys. 5.9. Układ folderów aplikacji webowej

Użytkownik może przesyłać obraz, a interfejs flaska zapyta się o predykcję za pomocą interfejsu REST API. Wynik jest zwracany i wyświetlany przez aplikację. Na razie jest to bardzo proste rozwiązanie, ale w przyszłości można je przekształcić w bardziej złożoną aplikację. Aplikację na aktualnym etapie przedstawiono na rysunku 5.10.



Rys. 5.10. Screenshot aplikacji, która wykorzystuje wytrenowany model do interpretacji odpowiedniej etykiety opisującej zdjęcie

6. Bibliografia

- Ryszard Tadeusiewicz, Maciej Szaleniec: Leksykon sieci neuronowych.
- Image Style Transfer Using Convolutional Neural Networks, Leon A. Gatys, Alexander S. Ecker, Matthias Bethge.
- Visualizing and Understanding Convolutional Networks, Zeiler, Fergus, ECCV 2014
- Neural Networks and Deep Learning, Michale A. Nielsen, Determination Press, 2015
- An Intuitive Explanation of Convolutional Neural Networks
- Convolutional Neural Networks (LeNet)
- Tianyi Liu, Shuang Sang Fang, Yuehui Zhao, Peng Wang, Jun Zhang Implementation of Training Convolutional Neural Networks
- <https://pl.wikipedia.org/wiki/RGB>
- <https://www.kaggle.com/datasets/flo2607/traffic-signs-classification>
- <https://home.agh.edu.pl/~vlsi/AI/wstep1/sieci.html>
- <https://mirosławmamczur.pl/jak-działają-konwolucyjne-sieci-neuronowe-cnn/>
- <https://datascience.eu/pl/wizja-komputerowa/konwolucyjne-sieci-neuronowe-sposob-eli5/>
- <https://bezkomputera.wmi.amu.edu.pl/ppi/chapters/computer-vision.html>
- https://pl.wikipedia.org/wiki/Rozpoznawanie_obrazów
- [https://www.logistyka.net.pl/bank-wiedzy/item/88074-problemy-rozpoznawania-i-klasyfikacji-obrazow-w-procesach-logistycznych_\(załącznik\)](https://www.logistyka.net.pl/bank-wiedzy/item/88074-problemy-rozpoznawania-i-klasyfikacji-obrazow-w-procesach-logistycznych_(załącznik))
- <https://seaborn.pydata.org/>
- <https://www.tensorflow.org/learn>
- <https://www.educative.io/edpresso/what-is-early-stopping>