

Wstęp

Poniższy raport przedstawia zagadnienie konwolucyjnych sieci neuronowych (CNN), które są metodą należącą do technik uczenia maszynowego oraz deep learning. W raporcie można znaleźć opis najważniejszych pojęć związanych z konwolucyjnymi sieciami neuronowymi, opis działania CNN, przedstawienie najpopularniejszego frameworku pozwalającego na łatwą implementację konwolucyjnych sieci neuronowych, przykładową implementację algorytmu na bazie zbioru Fashion Mnist, oraz przykładowe zastosowania sieci.

Uczenie maszynowe

Uczenie maszynowe to nauka interdyscyplinarna ze szczególnym uwzględnieniem takich dziedzin jak informatyka, robotyka i statystyka. Głównym celem jest praktyczne zastosowanie dokonań w dziedzinie sztucznej inteligencji do stworzenia automatycznego systemu potrafiącego doskonalić się przy pomocy zgromadzonego doświadczenia (czyli danych) i nabywania na tej podstawie nowej wiedzy.

Uczenie głębokie

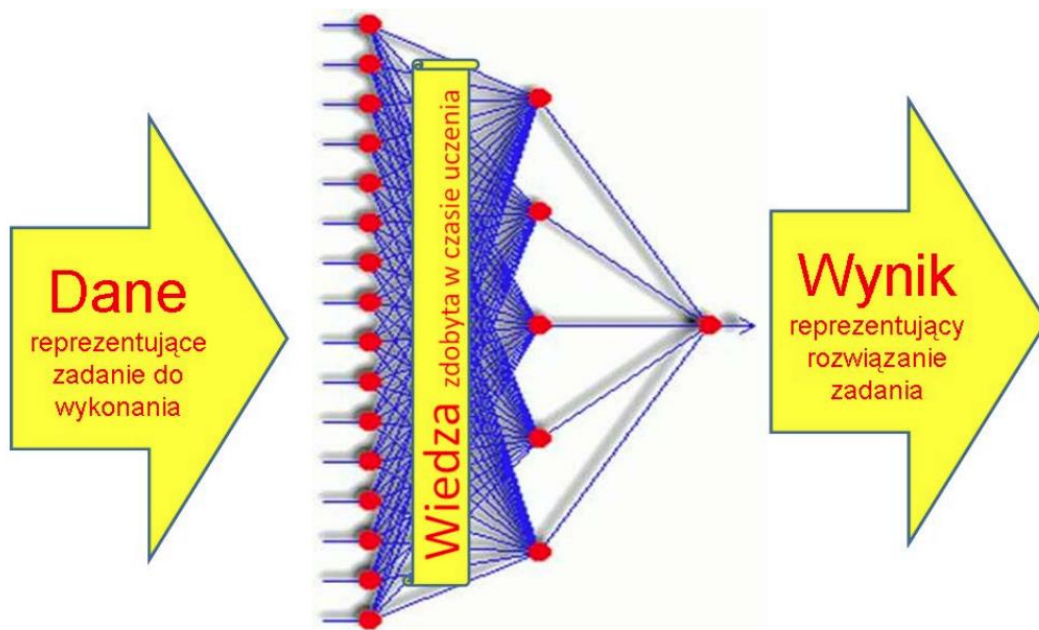
Uczenie głębokie (deep learning) jest częścią szerszej rodziny metod uczenia maszynowego opartych na sztucznych sieciach neuronowych z uczeniem reprezentacyjnym. Nauka może być nadzorowana, częściowo nadzorowana lub nienadzorowana.

Sieć neuronowa

Z tego względu, że sieci konwolucyjne należą do klasy głębokich sieci neuronowych, niezbędne jest sprecyzowanie tego terminu.

Sieć neuronowa to system przeznaczony do przetwarzania informacji, którego budowa i zasada działania są w pewnym stopniu wzorowane na funkcjonowaniu fragmentów rzeczywistego (biologicznego) systemu nerwowego. Na przesłankach biologicznych oparte są schematy sztucznych neuronów wchodzących w skład sieci oraz (w pewnym stopniu) jej struktura. Jednak schematy połączeń neuronów w sieci neuronowej są wybierane arbitralnie, a nie stanowią modelu rzeczywistych struktur nerwowych. Wyróżniającą cechą sieci neuronowej jako narzędzia informatycznego jest możliwość komputerowego rozwiązywania przy jej pomocy praktycznych problemów bez ich uprzedniej matematycznej formalizacji. Dalszą zaletą jest brak konieczności odwoływania się przy stosowaniu sieci do jakichkolwiek teoretycznych założeń na temat rozwiązywanego problemu - założenie o przyczynowo-skutkowych zależnościach między wejściem a wyjściem nie musi być egzekwowane. Najbardziej znaną cechą sieci neuronowej jest jej zdolność uczenia się na podstawie przykładów i możliwość automatycznego uogólniania zdobytej wiedzy (generalizacja). Rysunek przedstawia najbardziej typowy schemat praktycznego użycia sieci neuronowej (Ryszard Tadeusiewicz, Maciej Szaleniec, 2015, s.94).

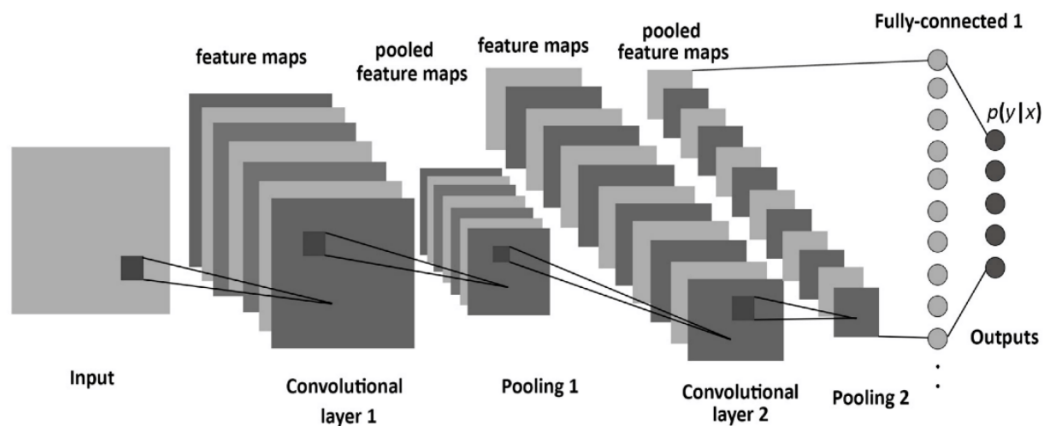
Rysunek 1. Najbardziej typowy schemat praktycznego użycia sieci neuronowej



Sieć konwolucyjna

W celu zoptymalizowania parametrów sieci, powstała specjalna architektura, zwana siecią konwolucyjną (convolutional neural network, CNN). Konwolucyjne sieci neuronowe składają się z wielu warstw, a każda z nich ma swoje specyficzne zastosowanie. Neurony znajdujące się w warstwie wejściowej zostają podzielone na obszary o identycznym rozmiarze, a neurony z tego samego obszaru połączone z neuronami z warstwy wyżej. Powoduje to utworzenie filtra, który w trakcie uczenia sieci jest odpowiedzialny za wykrywanie konkretnej cechy obrazu. Aby dana cecha była w taki sam sposób wykrywana w całości analizowanego obrazu, konieczne jest, aby filtr o takiej samej strukturze połączeń oraz identycznych wagach występował we wszystkich obszarach. Wszystkie filtry wykrywające niskopoziomowe cechy obrazów tworzą warstwę konwolucyjną, a jej neurony połączone są z neuronami z funkcją aktywacji, na przykład ReLU. Kolejnym elementem sieci jest warstwa pooling, która wspomaga wydobywanie najważniejszych cech z poprzednich warstw. Wszystkie trzy warstwy odpowiadają za wykrycie złożonych wzorców w obrazie. Poniżej przedstawiona została typowa architektura sieci konwolucyjnej.

Rysunek 2. Przykładowa architektura sieci konwolucyjnych



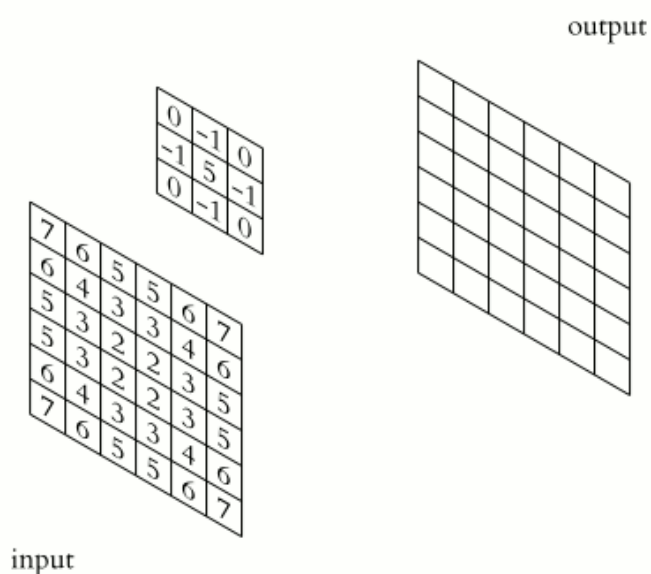
Warstwą wspierającą ostateczny proces klasyfikacji obiektu jest tzw. warstwa gęsta (fully connected layer). Każdy neuron jest połączony z każdym neuronem z poprzedniej warstwy, a każde połączenie ma swoją własną wagę, czyli wyuczony parametr. Warstwa gęsta może występować w sieci wielokrotnie, na różnych poziomach i w różnych konfiguracjach (Dzierżak, Grądz, 2018, s. 59-60).

Jak wcześniej wspomniano, klasyczna architektura sieci konwolucyjnej sprowadza się do trzech specyficznych, następujących po sobie warstw:

1) Warstwa konwolucyjna


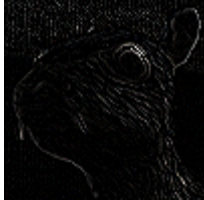


Jest to podstawowa warstwa głębokiej sieci neuronowej. Konwolucja wykorzystuje „jądro” (filtr to zamienna nazwa używana dla sieci konwolucyjnych) do wyodrębnienia pewnych „cech” z obrazu wejściowego. Jądro to macierz, która jest przesuwana po obrazie i mnożona przez dane wejściowe tak, że wynik jest ulepszany w określony pożądany sposób. Istnieje wiele różnych jąder konwolucji, które mają za zadanie wyodrębnić z obrazka ukryte cechy. Poniżej przedstawione jest działanie filtru,

Rysunek 3. Przykładowe działanie warstwy konwolucyjnej



który pozwala wyostrzyć obraz.

Tabela 1. Przykłady dostępnych filtrów

Operacja	Jądro konwolucji	Wynik działania
Wykrywanie krawędzi	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Wykrywanie krawędzi	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Rozmycie	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Wyostrzenie	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Niezbędne jest odpowiednie wybranie filtrów, aby wydobyć z obrazka najważniejsze informacje.

Na tym właśnie polega uczenie sieci konwolucyjnych. Zamiast klasycznych wag wykorzystuje się jądra konwolucji (zwane mapami własności - feature maps), które są optymalizowane w procesie uczenia.

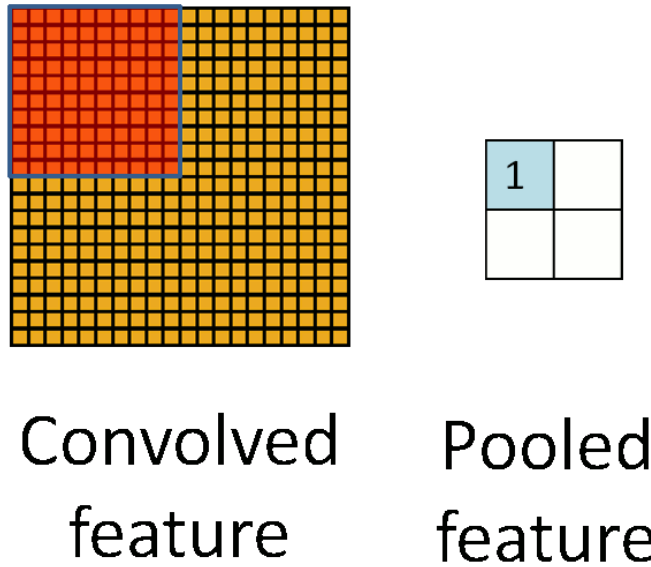
2) Wprowadzenie nieliniowości

Konwolucja jest ze swojej definicji operacją liniową, w rzeczywistości relacje pomiędzy elementami otaczającej nas rzeczywistości są nieliniowe. Z sieci neuronowych korzystamy przede wszystkim w celu ich wychwytywania, przez co aby poprawić jakość aproksymacji konieczne jest przemnożenie wyników otrzymanych na warstwie konwolucyjnej przez nieliniową funkcję aktywacji (np. ReLU).

3) Pooling

Pooling możemy traktować jako "przycinanie" danych wychodzących z poprzedniej warstwy.

Rysunek 4. Przykład działania warstwy poolingowej



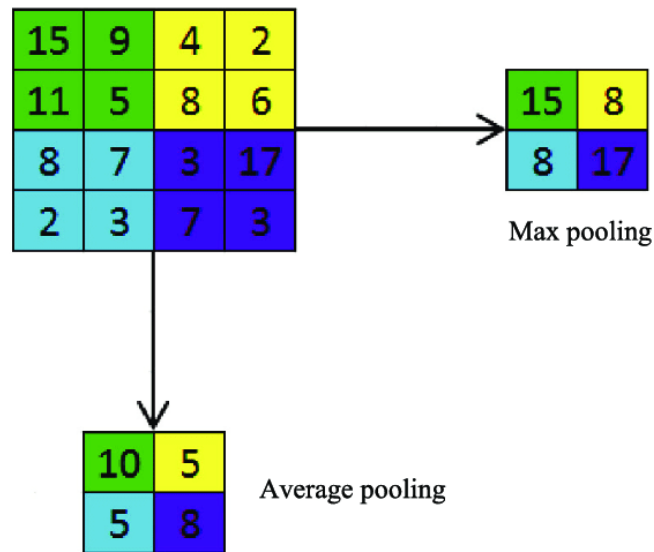
Pooling wykorzystujemy z kilku powodów:

- zmniejsza on rozmiar danych na każdej kolejnej warstwie, dzięki temu możemy zbudować głębsze modele, które będą się efektywniej uczyły.
- Pooling pogarsza jakość obrazów (zmniejsza ich rozmiar). Dzięki temu obrazy są bardziej odporne na przeuczenie.
- Pooling wiąże się też z założeniem o "stacjonarności" własności obrazu - zakładamy, że własność wykryta na jednym obszarze obrazu jest równie ważna na innych.
- Dzięki poolingowi aproksymacja jest możliwa mimo nieznaczących przekształceń danych wejściowych i skali obrazu.

Wyróżniamy kilka podstawowych rodzajów poolingu:

- max-pooling
- mean-pooling
- min-pooling

Rysunek 5. Przykład działania warstwy poolingowej



Opcjonalnie rozważa się czwartą warstwę „dropout”, która skutecznie walczy z overfittingiem. Zaproponował ją Geoffrey E. Hinton et al. w pracy „Improving neural networks by preventing co-adaptation of feature detectors”. Jest to relatywnie prosta, ale zarazem bardzo skuteczna technika przeciwdziałania overfittingowi. Polega na losowym usuwaniu z sieci (z warstw wewnętrznych, czasami również wejściowych) pojedynczych neuronów w trakcie uczenia. Ponieważ skomplikowane sieci (a takie niewątpliwie są głębokie sieci neuronowe), szczególnie dysponujące relatywnie niewielkimi ilościami danych uczących, mają tendencję do dokładnego dopasowywania się do danych, to taki sposób deregulacji zmusza je do uczenia w sposób bardziej zgeneralizowany.

Zastosowanie dropout w praktyce prowadzi do sytuacji, w której architektura sieci zmienia się dynamicznie i otrzymujemy model, w którym jeden zbiór danych został wykorzystany do nauczania wielu sieci o różniących się architekturach, a następnie został przetestowany na zbiorze testowym z uśrednionymi wartościami wag.

TensorFlow

Istnieje wiele bibliotek, które umożliwiają łatwą implementację metod uczenia maszynowego. Najpopularniejszą z nich jest TensorFlow, który został opracowany przez firmę Google. Biblioteka ta w prosty sposób pozwala opracowywać i implementować m.in. konwolucyjne sieci neuronowe w takich językach programowania jak C++, Python, Java czy Java Script. Co jest bardzo ważne TensorFlow biblioteką open source, co oznacza, że kod źródłowy jest wydawany na podstawie licencji, na mocy której właściciel praw autorskich przyznaje użytkownikom prawa do badania, zmiany i rozpowszechniania oprogramowania w ramach licencji wolnego oprogramowania.

Zalety używania biblioteki TensorFlow

1) Narzędzie Tensorboard

TensorBoard to narzędzie służące do wizualizacji danych. Pozwala prezentować w czytelny sposób m.in. grafy obliczeniowe sieci neuronowych (ang. computational graph), dane wejściowe, dane

Rysunek 6. Przykładowy schemat stworzony w TensorBoard



Wiele frameworków wymusza podanie pełnego kodu modelu w celu załadowania jego parametrów. Natomiast TensorFlow wymaga jedynie pliku zawierającego wagi oraz nazw warstw potrzebnych do inferencji (warstwa wejściowa jest najważniejsza, ponieważ bez niej nie możemy uruchomić grafu obliczeniowego).

3) Obsługa wielu języków

Ze względu na dużą popularność, społeczność TensorFlow utworzyła wiązania (ang. bindings), aby korzystać z framework'a w innych językach, takich jak C # czy Ruby.

Wady używania biblioteki TensorFlow

- 1) Konieczny jest dodatkowy kod

Ilość kodu potrzebnego do dodania funkcjonalności nie jest zbyt duża. Niemniej jednak, konwencja nazewnictwa może być niespójna, a złożoność modułów przytłaczająca.

Każde obliczenie musi być wywołane z poziomu obsługi sesji. Stwarza to sytuację, w której korzystanie z TensorFlow jest jak używanie języka wewnątrz innego języka. Można zapomnieć o pisaniu czystego kodu pythonowego. Nawet coś tak prostego jak pętla for, wymaga nazwania przy użyciu odpowiednika TensorFlow.

Czasem dokumentacja nawet "zapomina" poinformować, że trzeba zawrzeć dodatkowe instrukcje w swoim kodzie, aby pewne rzeczy działały

Moduły nie zawsze są ze sobą idealnie połączone, co prowadzi do braku komunikacji między nimi.

2) Częste wydania nowej wersji

Dla niektórych to może brzmieć jak zaleta. Jednak w rzeczywistości w środowisku produkcyjnym lepiej unikać nowych wersji wydawanych co 1-2 miesiące, zwłaszcza jeśli mogą wpłynąć na kompatybilność wsteczną.

Jest to wyjątkowo szkodliwe podczas stosowania powiązań z innymi językami, jak TensorSharp w języku C#.

To zrozumiałe, że pewne zmiany w tak szybko rozwijającym się frameworku są nieuniknione, ale być może społeczność skorzystałaby bardziej, gdyby nowe wersje były rzadsze, a więcej uwagi poświęcono na spójność framework'a.

Pomimo wad, TensorFlow jest jednym z najczęściej stosowanych frameworków do projektów związanych z uczeniem maszynowym, używanym przez gigantów takich jak IBM, Twitter, 9GAG, Ocado i Google.

TensorFlow może być trochę przytłaczający dla początkujących użytkowników, ponieważ często zapewnia wiele implementacji tych samych funkcjonalności. Może to być mylące dla osób bez doświadczenia lub podstawowej wiedzy na temat różnic między sugerowanymi implementacjami. Osoby początkujące być może powinny zainteresować się prostszymi alternatywami.

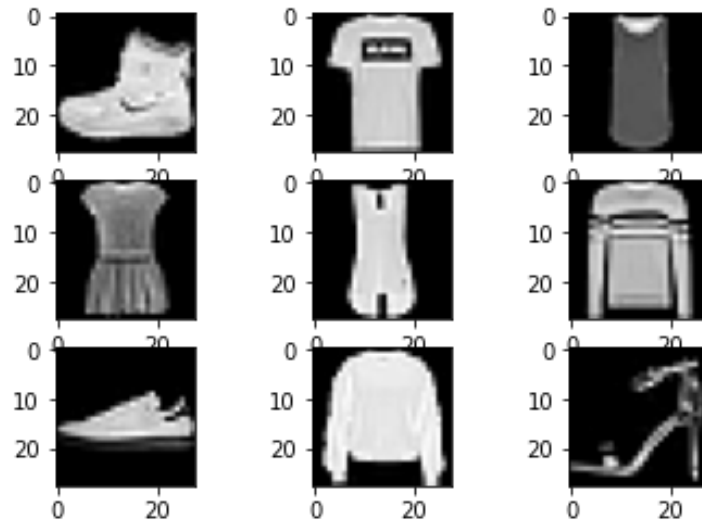
Przykładowa implementacja sieci konwolucyjnych

Celem sieci konwolucyjnych jest klasyfikacja obiektu do danej klasy. Proponujemy wykorzystanie architektury sieci konwolucyjnych do klasyfikacji zdjęć ubrań z serwisu odzieżowego Zalando. Zbiór danych zawiera obserwacje, które możemy zaklasyfikować do następujących klas:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker

- 8 Bag
- 9 Ankle boot

Rysunek 7. Przykładowe zdjęcia ubrań występujące w zbiorze



Zbiór danych zawiera 70tys obserwacji, z czego 60tys obserwacji przypisano do zbioru treningowego, a 10tys obserwacji do zbioru testowego. Każde zdjęcie jest wymiaru 28x28 pikseli. Każdy piksel oznacza pewną wartość określającą jasność – od 0 do 255. Przykładowo 0 oznacza kolor czarny, a 255 kolor biały. Im niższa wartość tym ciemniejszy jest piksel.

Zastosowano następującą strukturę sieci konwolucyjnych:

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 26, 26, 32)	320
conv2d_17 (Conv2D)	(None, 24, 24, 32)	9248
max_pooling2d_12 (MaxPooling)	(None, 12, 12, 32)	0
dropout_9 (Dropout)	(None, 12, 12, 32)	0
conv2d_18 (Conv2D)	(None, 10, 10, 64)	18496
conv2d_19 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_13 (MaxPooling)	(None, 4, 4, 64)	0
dropout_10 (Dropout)	(None, 4, 4, 64)	0
flatten_6 (Flatten)	(None, 1024)	0

dense_6 (Dense)	(None, 512)	524800
dropout_11 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 10)	5130
=====		

Po warstwie wejściowej zastosowano dwie warstwy konwolucyjne z jądrem wielkości 3x3, 32 filtrami i funkcją aktywacyjną ReLU. Następnie zastosowano max-pooling, która pomniejszyła obraz dwukrotnie, a później warstwę dropout, która z prawdopodobieństwem 50% określa czy neuron zostanie usunięty z sieci. Następnie po raz kolejny powtarzamy schemat, tylko tym razem w warstwach konwolucyjnych stosujemy 64 filtry, a w warstwie dropout usuwamy każdy neuron z prawdopodobieństwem 25%. Warstwa flatten przekształca obrazek w wektor. W dalszej kolejności dodano warstwę ukrytą z 512 neuronami z funkcją aktywacyjną ReLU. Na koniec ponownie występuje warstwa dropout z prawdopodobieństwem wykluczenia neuronu 50% oraz warstwa wyjściowa z liczbą neuronów równą liczbie klas, oraz funkcją aktywacji softmax.

Przedstawiona sieć uczona była przez 15 epok z funkcją straty categorical crossentropy i optymalizatorem ADAM.

Wartość accuracy wyliczona na zbiorze walidacyjnym wyniosła 91,9%, co jest jak najbardziej zadowalającym wynikiem.

Cały kod projektu sieci konwolucyjnych wykorzystanych do predykcji klas ubrań można znaleźć pod linkiem <https://github.com/kacperk77/Fashion-MNIST/blob/main/Fashion%20MNIST.ipynb>.