Implementacja sztucznej konwolucyjnej sieci neuronowej od zera

MATEUSZ KRAWCZYK

Promotor: Marcin Kowalik

27.01.2025

SPIS TREŚCI

CEL PRACY

Motywacja

CNN odgrywają coraz ważniejszą rolę w wielu dziedzinach, takich jak:

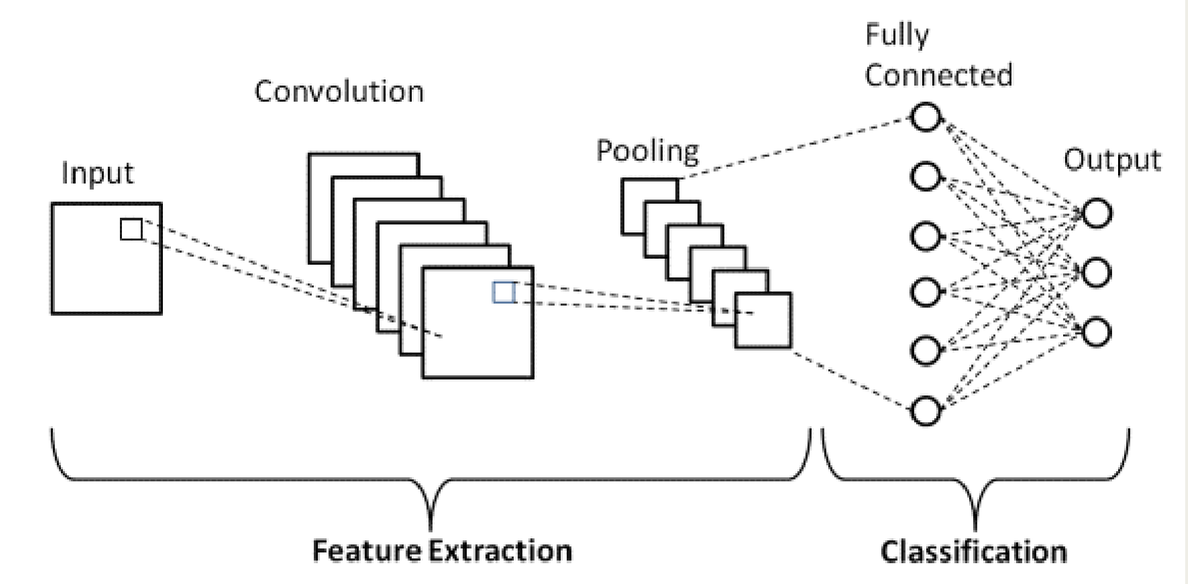
- **rozpoznawanie obrazów**
- **analiza danych medycznych**
- **autonomiczne pojazdy**

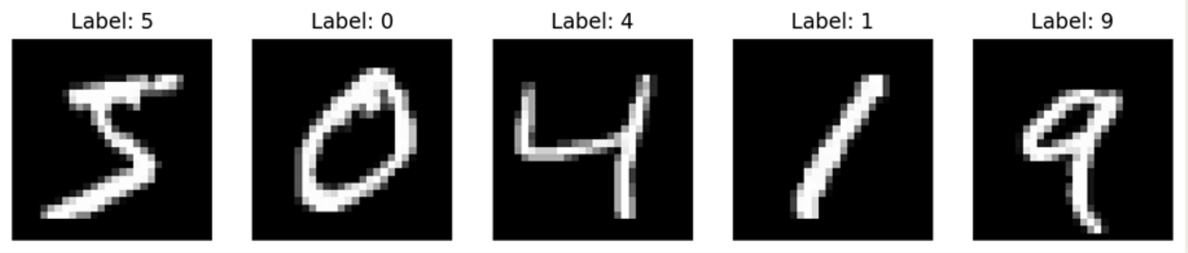
Cel pracy

- implementacja CNN from scratch
- szczegółowe omówienie architektury CNN oraz kluczowych operacji
- zrozumienie mechanizmów działania poszczególnych warstw oraz algorytmu propagacji wstecznej
- porównanie z istniejącymi rozwiązaniami

Dlaczego to ważne?

- rosnące zapotrzebowanie na tworzenie **bardziej zoptymalizowanych i zrozumiałych modeli AI**
- Ważne jest, aby modele te były **zrozumiałe dla twórców** i **umożliwiały kontrolę** nad poszczególnymi etapami uczenia

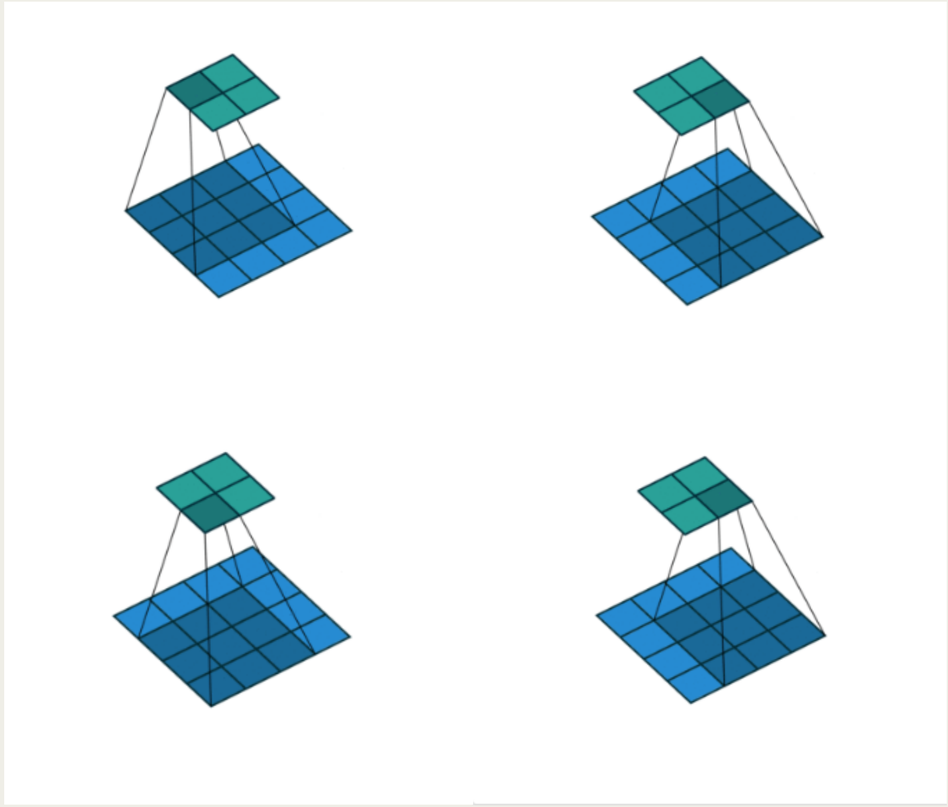




CNN I JEJ OPERACJE

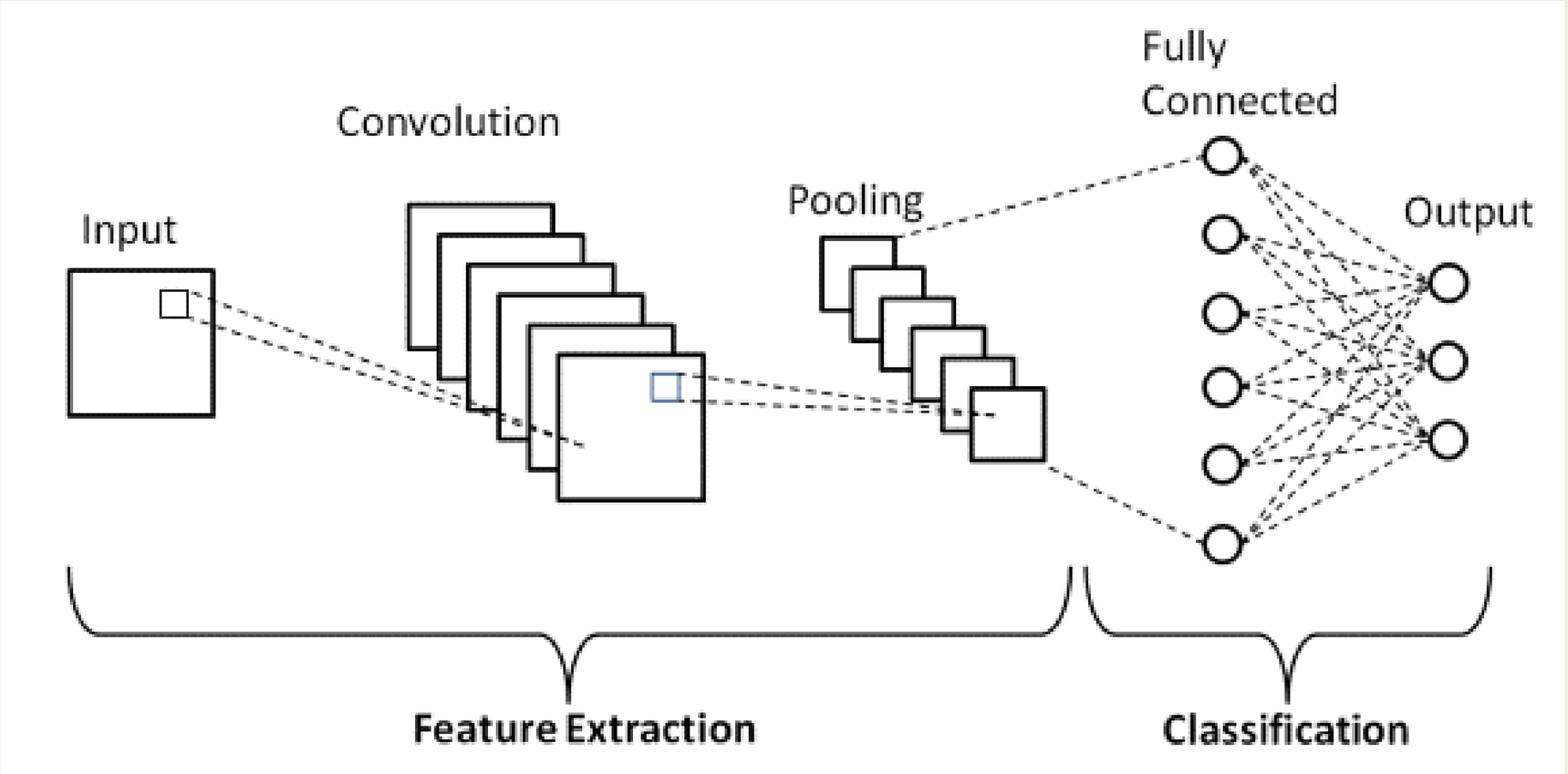
Konwolucyjne sieci neuronowe (CNN):

- szczególnie skuteczne w zadaniach związanych z przetwarzaniem obrazów.
- potrafią **automatycznie uczyć się hierarchii cech**, od prostych (np. krawędzi) do złożonych (np. kształtów obiektów)



Konwolucja:
Operacja matematyczna, która tworzy **mapę cech** poprzez przesuwanie filtru po obrazie wejściowym. Filtr ten mnoży swoje wagi z wartościami pikseli w danym obszarze obrazu, tworząc pojedynczy punkt na mapie cech

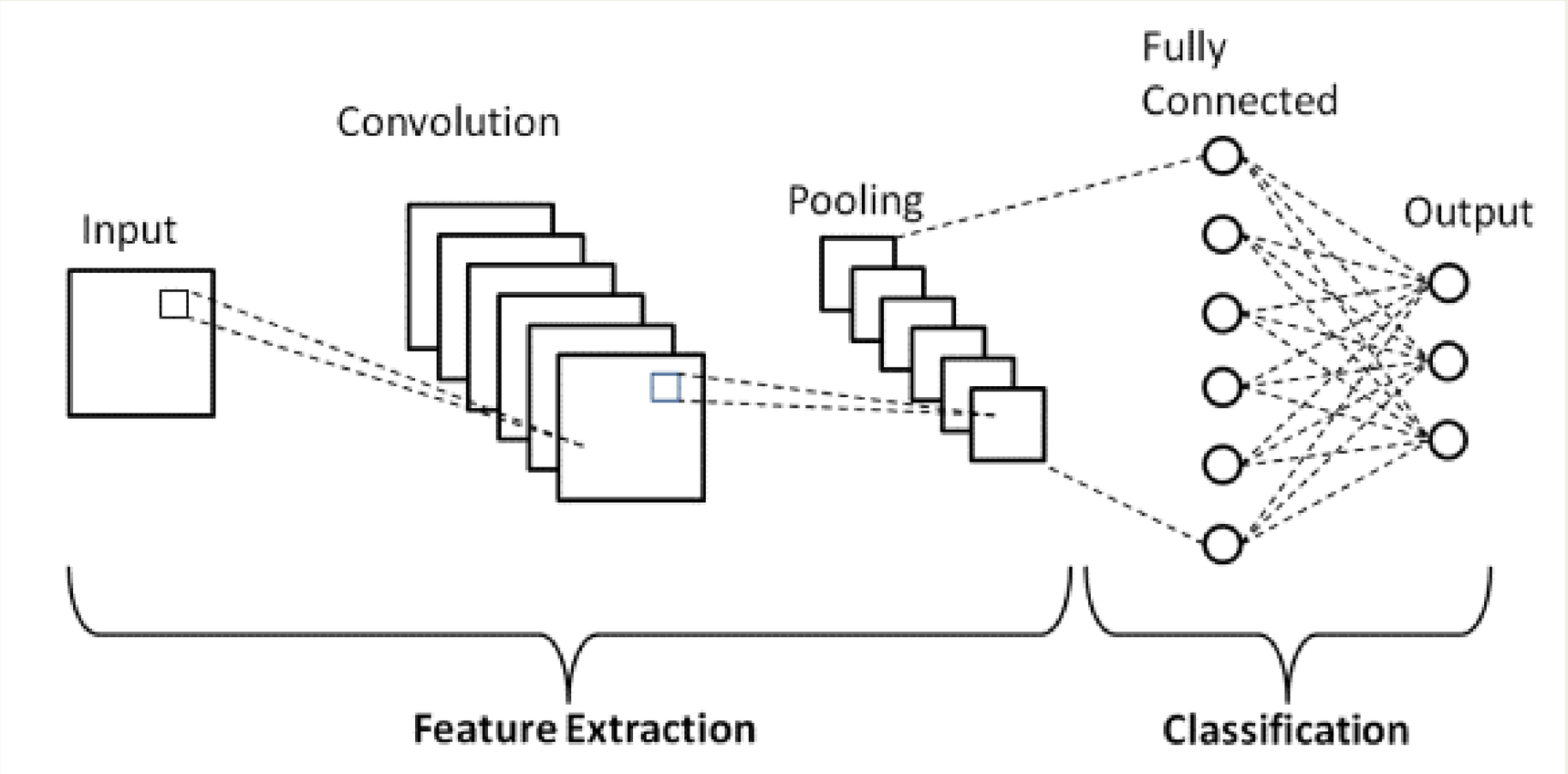
EKSTRAKCJA CECH (FEATURE EXTRACTION)



Input

Na początku sieć otrzymuje obraz, który jest reprezentowany jako macierz pikseli.

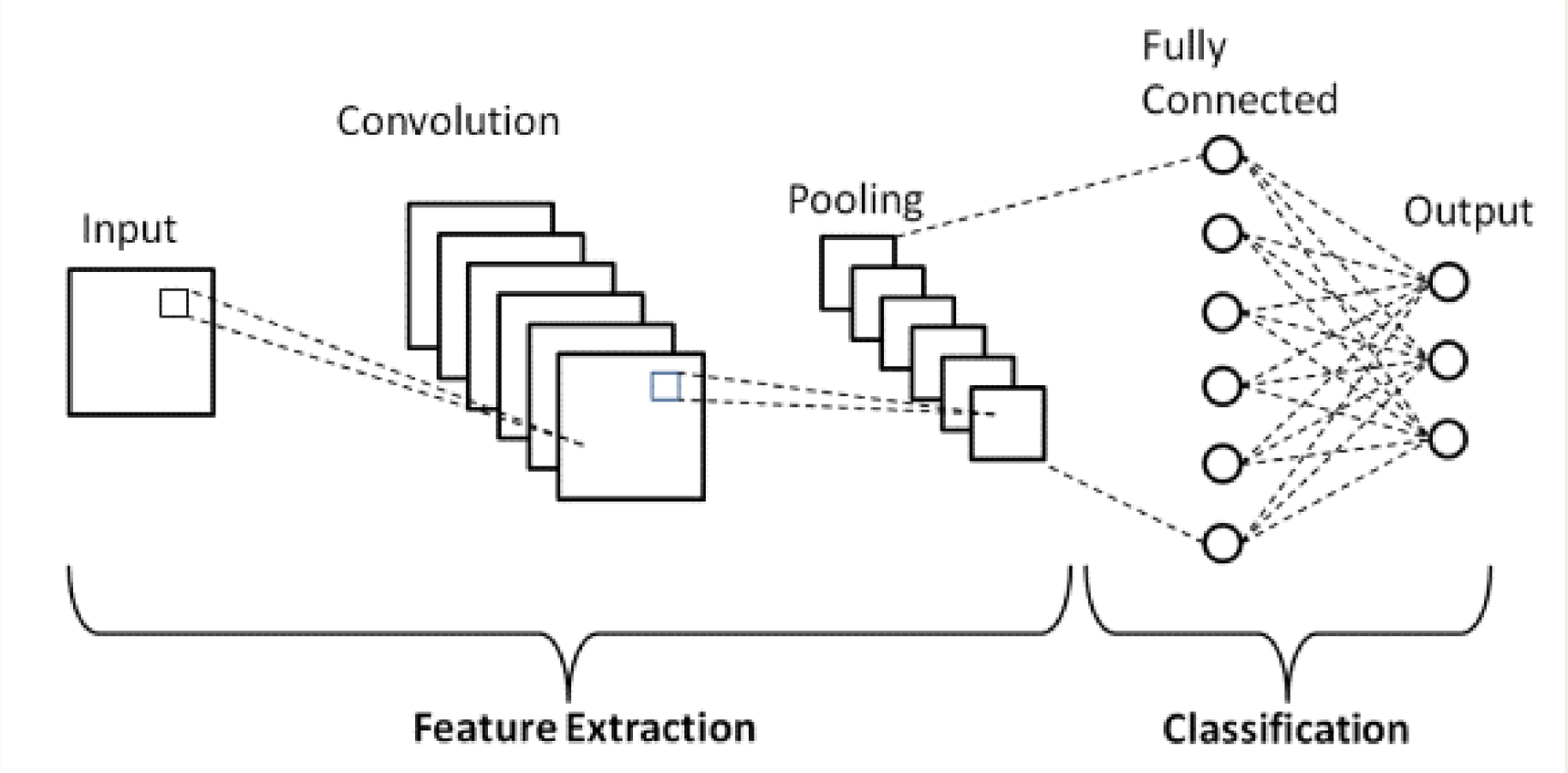
EKSTRAKCJA CECH (FEATURE EXTRACTION)



Convolution

Następnie na obraz nakładany jest filtr (jądro), który przesuwa się po obrazie, wykonując operację konwolucji. To pozwala wydobyć cechy, takie jak krawędzie, kolory czy tekstury.

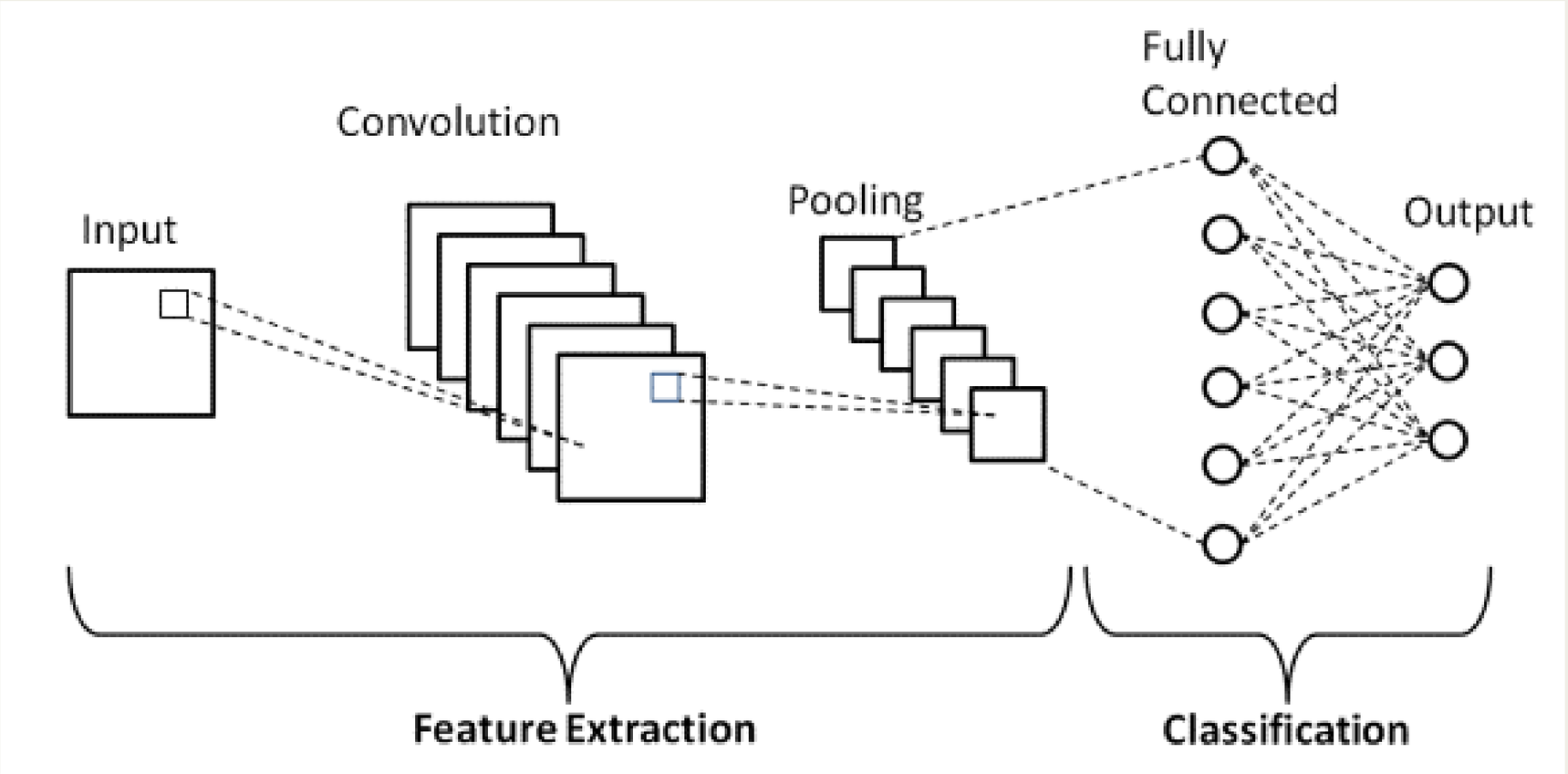
EKSTRAKCJA CECH (FEATURE EXTRACTION)



Pooling

Po konwolucji następuje warstwa pooling (average pooling), która zmniejsza rozmiar mapy cech, zachowując najważniejsze informacje. To redukuje liczbę parametrów i obliczeń, a także pomaga w zwiększeniu odporności na zakłócenia.

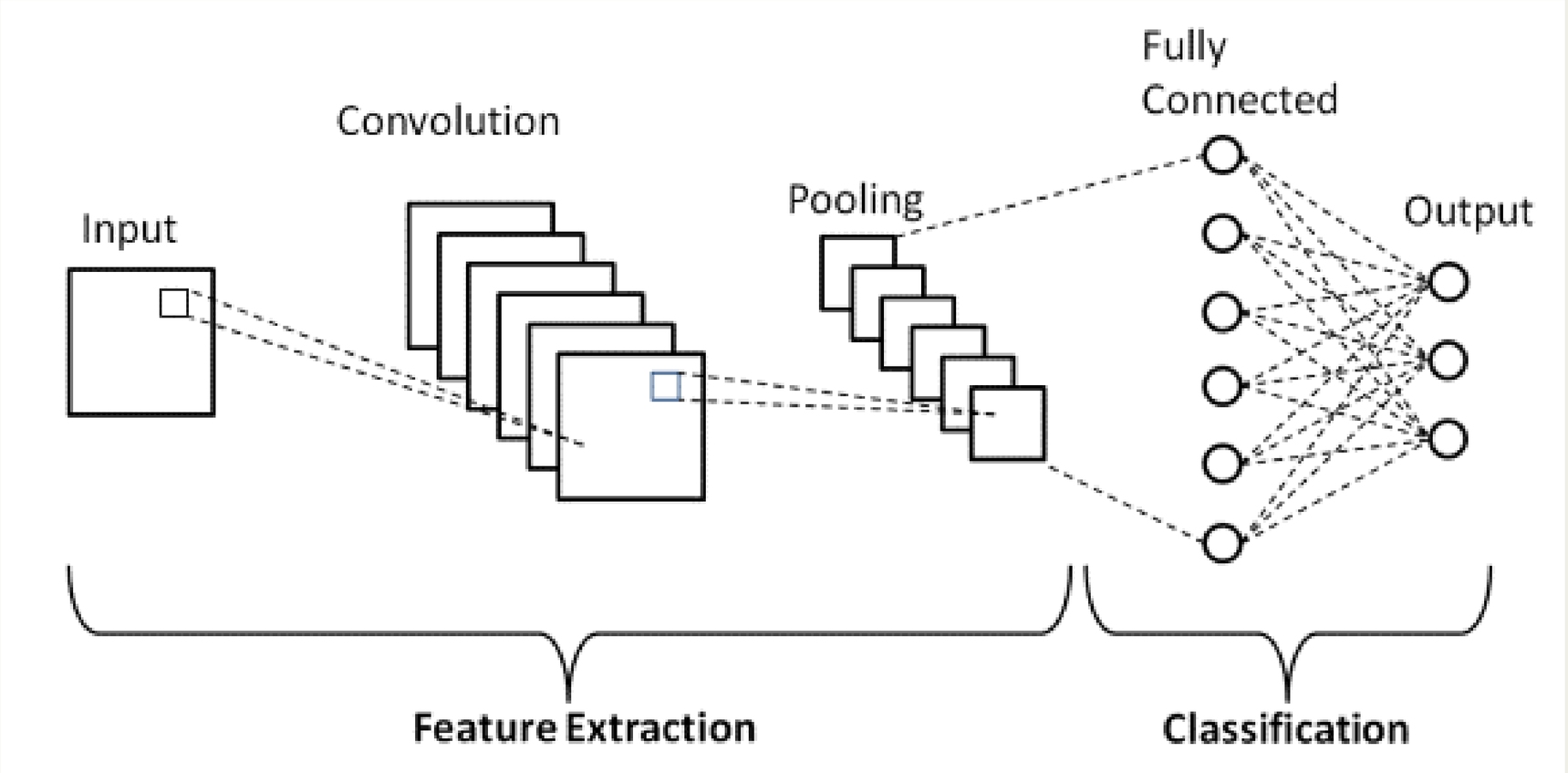
K L A S Y F I K A C J A



Fully Connected

Po ekstrakcji cech, uzyskane dane są spłaszczane i przesyłane do warstwy w pełni połączonej (fully connected), która łączy wszystkie neurony. Ta warstwa uczy się, jak łączyć cechy w celu klasyfikacji.

K L A S Y F I K A C J A



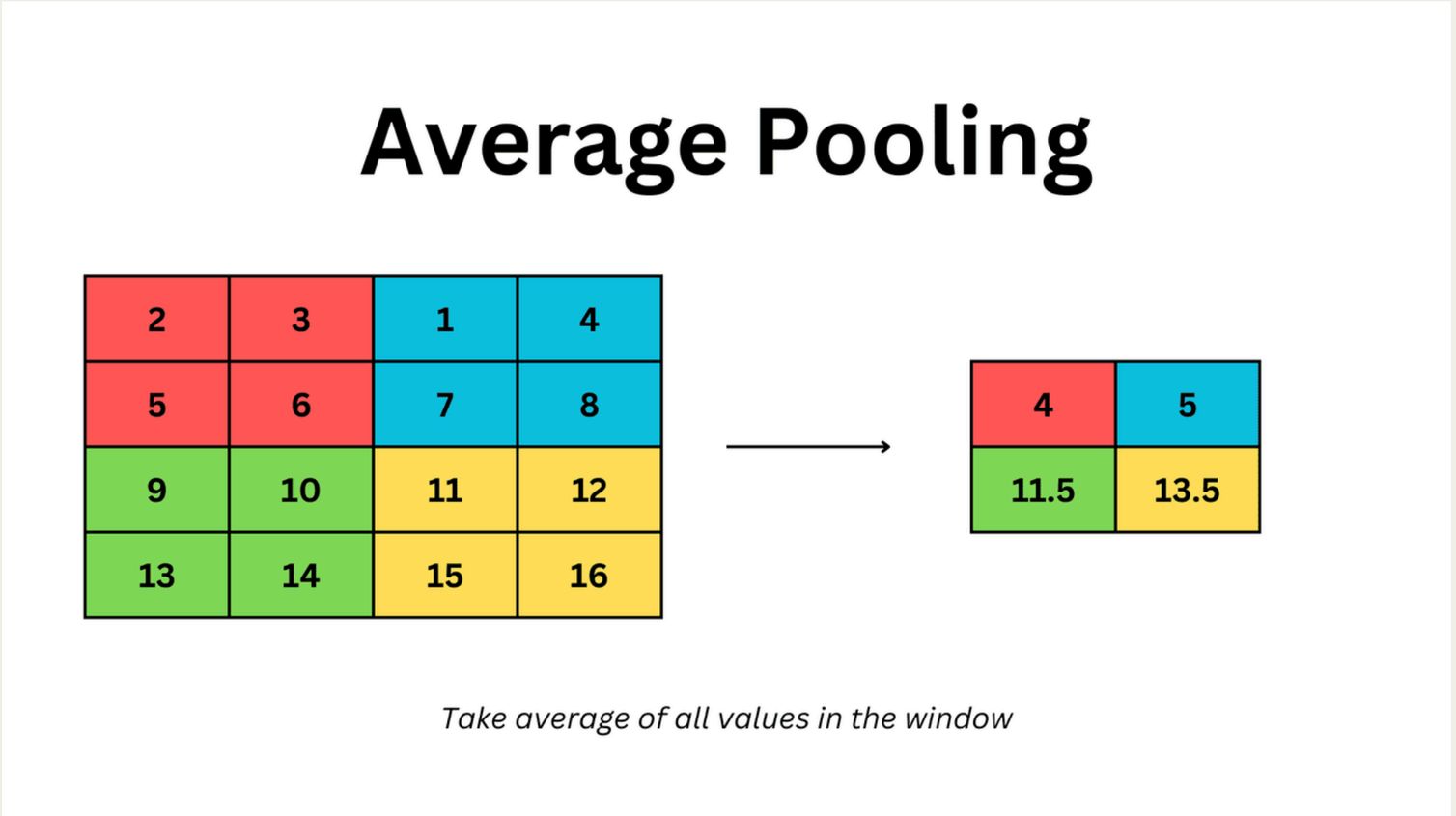
Output

Na końcu znajduje się warstwa wyjściowa (output), która generuje prawdopodobieństwa dla różnych klas, wskazując, do której kategorii należy analizowany obraz.

CNN I JEJ OPERACJE

Pooling (AVERAGE POOLING):

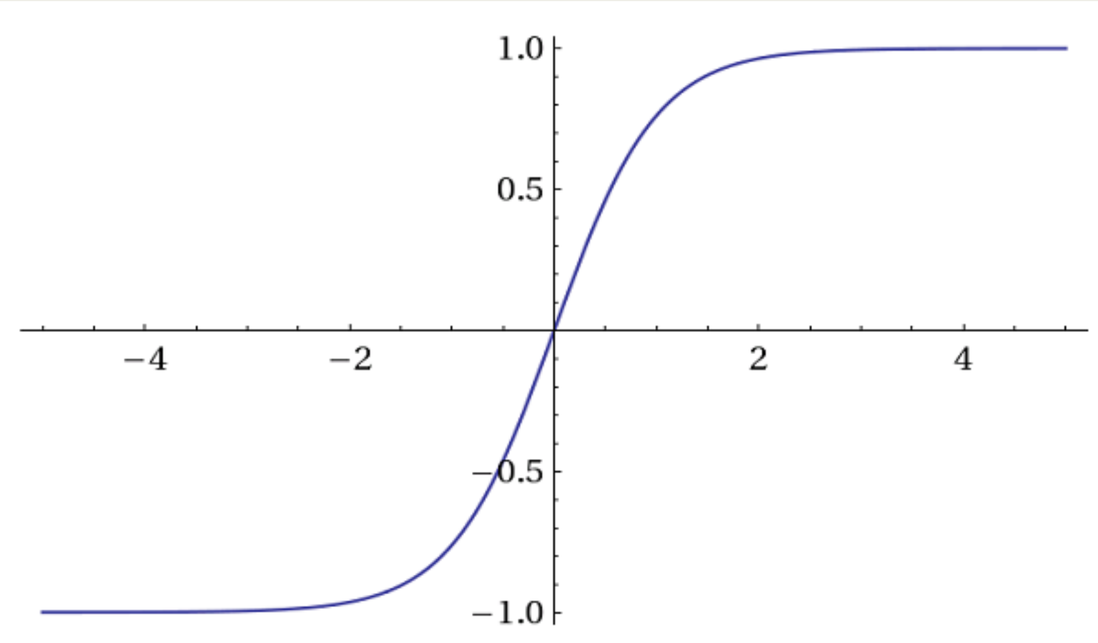
AveragePooling redukuje rozmiar map cech poprzez obliczenie średniej wartości w oknie. Pomaga w uogólnieniu cech i zmniejszeniu czułości na drobne zakłócenia w danych wejściowych. Sprawdza się szczególnie w danych o jednorodnym tle, takich jak MNIST



TANH - NIELINIOWA FUNKCJA AKTYWACJI

Dlaczego używa się funkcji aktywacji?

- Tanh pozwala przenieść wartości na przedział od -1 do 1.
- Dzięki tej właściwości lepiej sprawdza się w modelach, gdzie istotne są zarówno dodatnie, jak i ujemne wartości aktywacji.
- Tanh znajduje zastosowanie w sieciach o mniejszej liczbie warstw ukrytych, gdzie stabilność propagacji gradientu jest wystarczająca dla skutecznego uczenia



IMPLEMENTACJA CNN OD ZERA

Opis implementacji od podstaw

- W mojej pracy stworzyłem konwolucyjną sieć neuronową od podstaw, implementując kluczowe operacje: konwolucję i pooling, a także funkcje aktywacji.
- Wykorzystałem bibliotekę NumPy, co pozwoliło na efektywne operacje na tablicach bez użycia złożonych pętli.

Kluczowe elementy

- Konwolucja
- Pooling (AVERAGEPOOL)
- Funkcje aktywacji- Tanh
- Optymalizacja (ograniczanie operacji for na rzecz NumPy)

```
# ===== FORWARD =====
self.Conv1.forward(M, 0, 1)
self.T[0].forward(self.Conv1.output)
self.AP1.forward(self.T[0].output, 2, 2)

self.Conv2.forward(self.AP1.output, 0, 1)
self.T[1].forward(self.Conv2.output)
self.AP2.forward(self.T[1].output, 2, 2)

self.Conv3.forward(self.AP2.output, 2, 3)
self.T[2].forward(self.Conv3.output)

self.F.forward(self.T[2].output)
x = self.F.output

self.dense1.forward(x)
self.T[3].forward(self.dense1.output)
self.dense2.forward(self.T[3].output)

loss = loss_activation.forward(self.dense2.output, C)
predictions = np.argmax(loss_activation.output, axis=1)
if C.ndim == 2:
    C = np.argmax(C, axis=1)
accuracy = np.mean(predictions == C)

# ===== BACKWARD =====
loss_activation.backward(loss_activation.output, C)
self.dense2.backward(loss_activation.dinputs)
self.T[3].backward(self.dense2.dinputs)
self.dense1.backward(self.T[3].dinputs)

self.F.backward(self.dense1.dinputs)
self.T[2].backward(self.F.dinputs)
self.Conv3.backward(self.T[2].dinputs)

self.AP2.backward(self.Conv3.dinputs)
self.T[1].backward(self.AP2.dinputs)
self.Conv2.backward(self.T[1].dinputs)

self.AP1.backward(self.Conv2.dinputs)
self.T[0].backward(self.AP1.dinputs)
self.Conv1.backward(self.T[0].dinputs)
```

IMPLEMENTACJA CNN OD ZERA

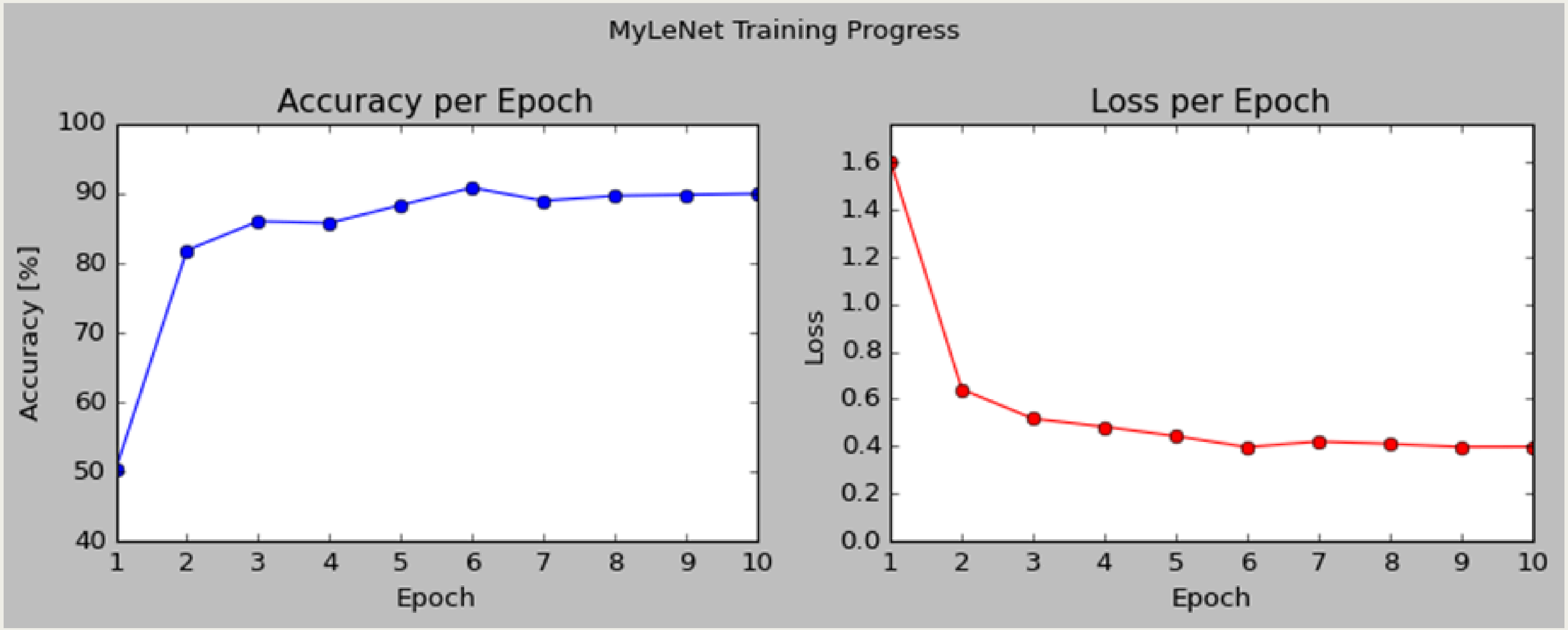
Backpropagation

Algorytm ten pozwala na obliczenie gradientów funkcji błędu względem wag i biasów, co umożliwia ich aktualizację w celu zmniejszenia błędu.

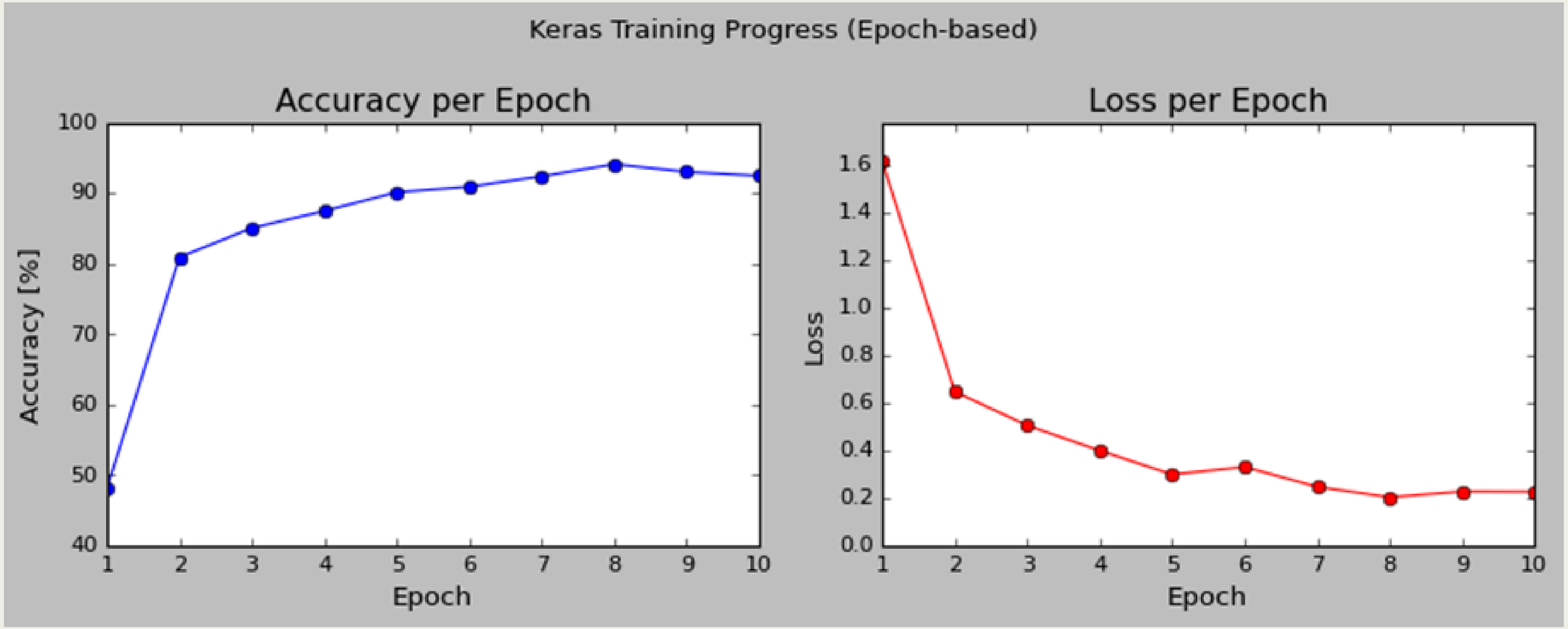
Kluczowe elementy:

- Propagacja gradientów (od warstwy wyjsciowej w tył)
- Obliczanie gradientów (funkcji aktywacji i straty, dzięki czemu dostosowałem wagi)
- Dostosowanie wag i biasów
- Wykorzystanie bibliotek (NumPy- operacje na tablicach)

MYLENET - WYKRES DOKŁADNOŚCI ORAZ FUNKCJI STRATY



KERAS - WYKRES DOKŁADNOŚCI ORAZ FUNKCJI STRATY



Proces optymalizacji w Keras przebiega regularnie i dynamicznie, a wartości straty zmniejszają się równomiernie, co wynika z lepszego zarządzania współczynnikiem uczenia i efektywniejszego obliczania operacji macierzowych, efektywniejszego użycia pamięci

MYLENET - PRZYKŁADOWE PREDYKCJE SIECI

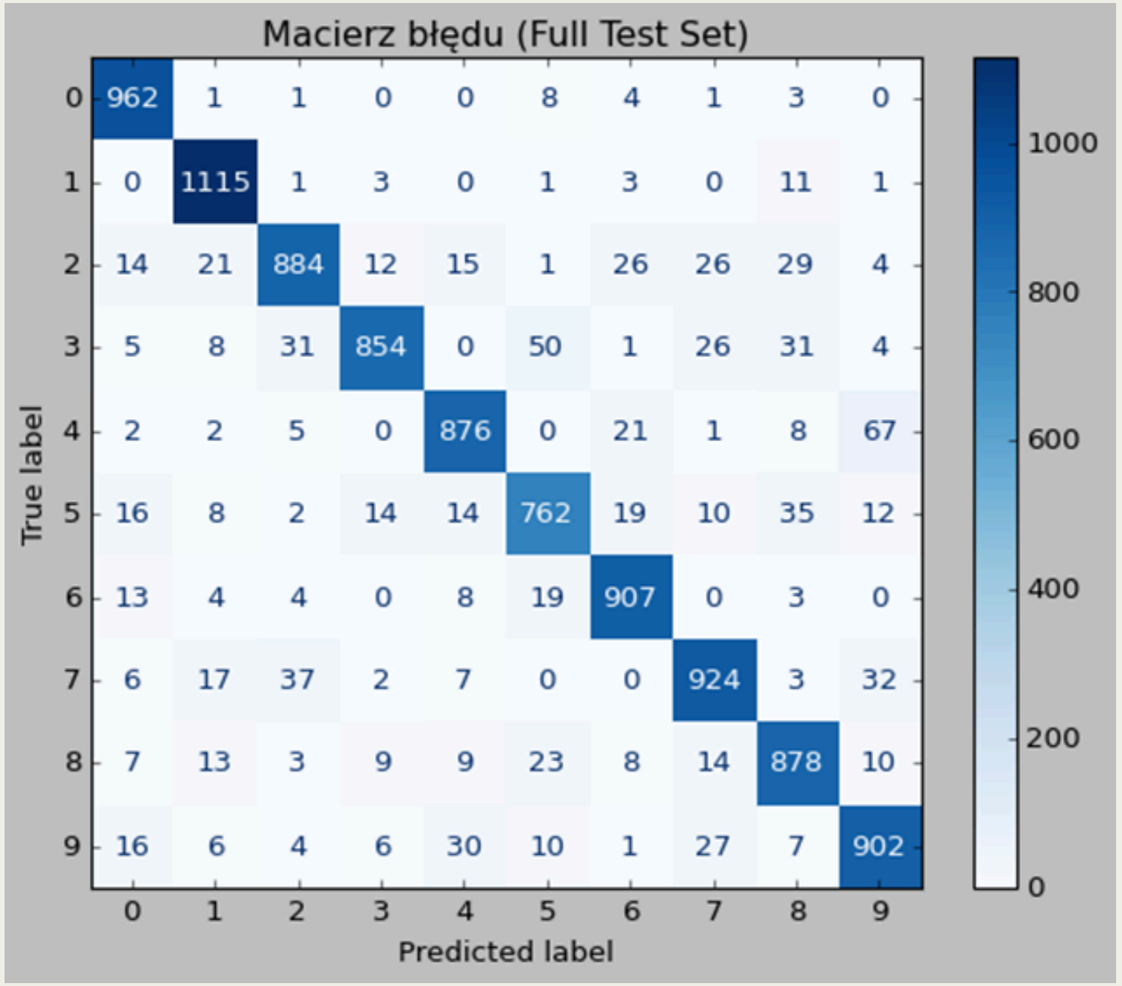
Pomyłki w obszarze cyfr o różnych kształtach, takich jak 6 czy 4. Niektóre z tych błędów wynikają ze specyfiki pisma ręcznego: w zbiorze MNIST cyfry bywają „zamazane” albo zapisane w nietypowy sposób

KERAS - PRZYKŁADOWE PREDYKCJE SIECI

Zdarzają się wciąż wpadki przy trudno czytelnych przykładach, ale sieć Keras wydaje się bardziej odporna na kształty cyfr mylonych przez MyLeNet.

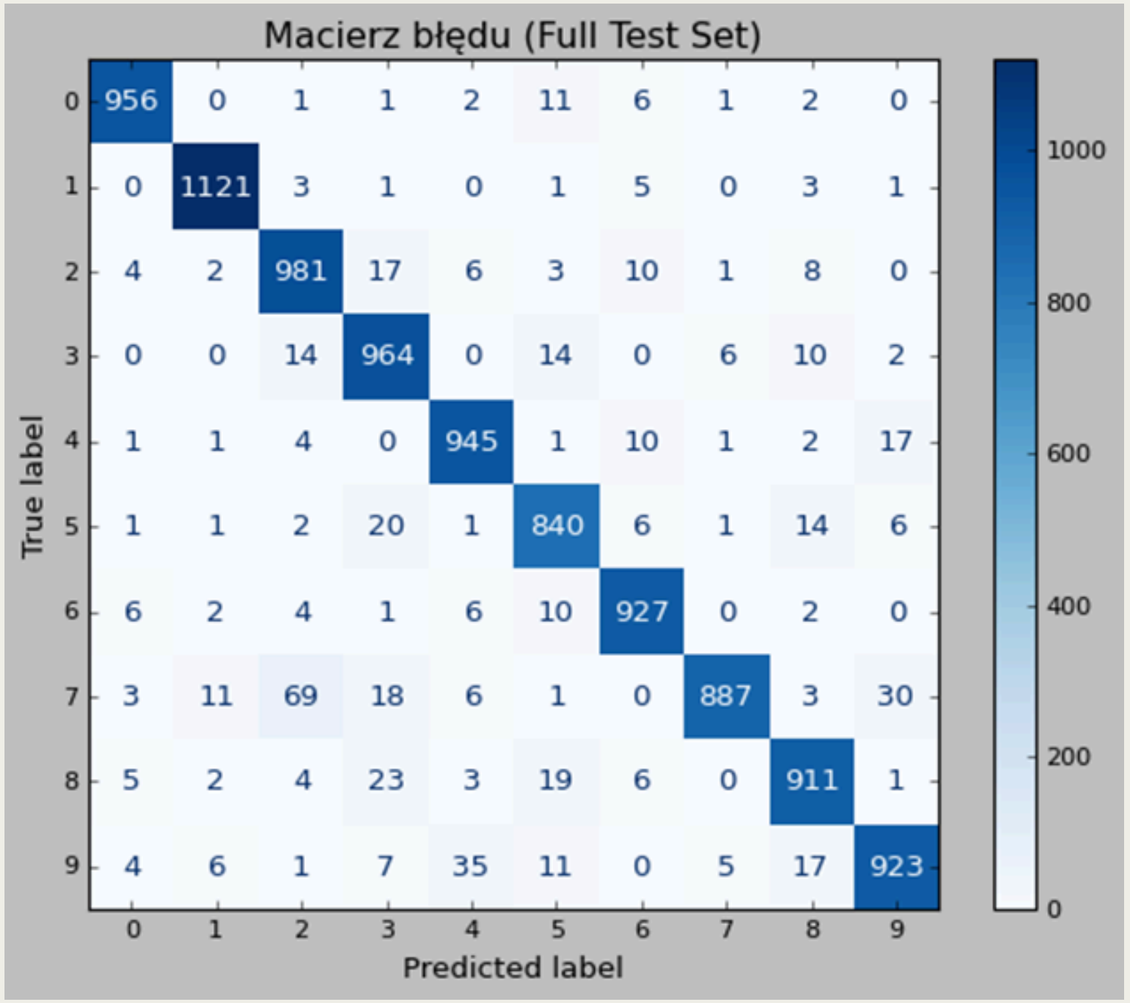
MYLENET - MACIERZ BŁĘDU

Dobrze rozpoznaje cyfry 0, 1 i 7, co widać po wysokich wartościach po przekątnej.
Największe trudności model ma z cyframi 5, 4 i 8, które częściej są błędnie klasyfikowane.
Błędy między cyframi często wynikają z podobieństw w ich kształtach.



KERAS - MACIERZ BŁĘDU

Model najlepiej rozpoznaje cyfry 1, 2 i 3.
Największe trudności sprawia cyfra 5.
Cyfra 7 również sprawia pewne problemy, zwłaszcza w przypadku mylenia jej z cyfrą 2 (69 przypadków). To częstszy problem w Keras niż w MyLeNet, gdzie takich błędów było tylko 37.



PORÓWNANIE WYNIKÓW SIECI MYLENET I KERAS

Parametr	MyLeNet	Keras
Test Accuracy	0.906	0.945
Test Loss	0.369	0.190
Czas treningu	168.84 sec	6.14 sec

Trening przy:

- epochs = 10
- batch_size = 128
- iteracje/ steps_per_epoch = 10

Różnice w końcowych wynikach mogą wynikać z bardziej udoskonalonego algorytmu optymalizacyjnego w Keras, który skuteczniej zarządza gradientami oraz stabilnością aktualizacji.

BIBLIOTEKI (KERAS)

Ręczna implementacja

- Wymagała szczegółowej implementacji kluczowych operacji.
- Pozwoliła na **głębsze zrozumienie działania** poszczególnych komponentów sieci neuronowej.
- Była **czasochłonna** i wymagała wielu testów i poprawek.

Keras

- Dostęp do zaawansowanych funkcji
- Optymalizacja obliczeń
- Łatwość implementacji

PODSUMOWANIE

Wnioski

- MyLeNet osiągnął dokładność **90,6%**, co jest dobrym wynikiem przy implementacji od podstaw.
- Sieć oparta na Keras osiągnęła **94,5%**, co wynika z zaawansowanej optymalizacji w bibliotekach
- Ręczna implementacja pozwoliła na dogłębne zrozumienie mechanizmów działania CNN.

Propozycje usprawnień

- **Batch Normalization** (Przyspiesza uczenie poprzez stabilizację wartości gradientów.)
- **Dropout** (Wymusza, aby sieć nauczyła się bardziej uniwersalnych cech, zamiast polegać na kilku wybranych neuronach)
- Przeniesienie na **GPU** z użyciem np. CuPy

Dziękuję za uwagę!

MATEUSZ KRAWCZYK