

## ***Lab 4 - Wstęp do TensorFlow i klasyfikacja obrazów z użyciem konwolucyjnych sieci neuronowych***

### **1. TensorFlow**

TensorFlow to biblioteka do uczenia maszynowego i sztucznej inteligencji. Jest wykorzystywana do różnych zadań, ale głównie służy do trenowania i wnioskowania sieci neuronowych. Jest jedną z najpopularniejszych frameworków do głębokiego uczenia, obok PyTorch. Nazwa TensorFlow odnosi się do przepływu i przetwarzania danych w reprezentowanych w formie tensorów.

**Tensor** jest to matematyczne uogólnienie skalar, wektora i macierzy, które pozwala na reprezentowanie danych o dowolnej liczbie wymiarów:

- Tensor 0-wymiarowy - skalar,
- Tensor 1-wymiarowy - wektor, tablica jednowymiarowa,
- Tensor 2-wymiarowy - macierz, tablica dwuwymiarowa,
- Tensor wielowymiarowy.

### **2. Instalacja TensorFlow**

Tensorflow instaluje się poprzez pip:

```
pip install tensorflow
```

Zaleca się użycie wirtualnego środowiska.

Można sprawdzić poprawność instalacji:

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
✓ 0.0s
TensorFlow version: 2.16.2
```

### 3. Przykładowy model w TensorFlow

Poniżej znajduje się przykład wykorzystania TensorFlow do implementacji prostego MLP z dwiema warstwami ukrytymi na zbiorze MNIST.

- Przygotowanie danych - przekształcanie na wektory, normalizacja oraz kodowane etykiety do postaci one-hot

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28 * 28) / 255.0
x_test = x_test.reshape(-1, 28 * 28) / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

✓ 0.4s

- Zdefiniowanie architektury modelu - dwie warstwy ukryte z 128 i 64 neuronami wykorzystując funkcję aktywacji ReLU do wprowadzenia nieliniowości. Warstwa wyjściowa z 10 neuronami (po jednej dla każdej klasy) używa softmax, aby uzyskać prawdopodobieństwo klas.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(28 * 28,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

✓ 0.0s

- Kompilacja - zdefiniowanie optymalizatora i funkcji straty. Optymalizator (najczęściej [Adam](#) lub SGD), kontroluje, jak model aktualizuje swoje wagi na podstawie gradientów, aby zminimalizować stratę.

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

✓ 0.0s

- Trenowanie - Model trenuje przez 5 epok z podziałem na partie (batch\_size=32) i wykorzystuje 20% danych treningowych do walidacji.

```
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
✓ 13.0s

Epoch 1/5
1500/1500 ————— 3s 2ms/step - accuracy: 0.8705 - loss: 0.4513 - val_accuracy: 0.9565 - val_loss: 0.1444
Epoch 2/5
1500/1500 ————— 2s 2ms/step - accuracy: 0.9651 - loss: 0.1171 - val_accuracy: 0.9657 - val_loss: 0.1148
Epoch 3/5
1500/1500 ————— 2s 2ms/step - accuracy: 0.9758 - loss: 0.0780 - val_accuracy: 0.9653 - val_loss: 0.1140
Epoch 4/5
1500/1500 ————— 2s 1ms/step - accuracy: 0.9829 - loss: 0.0561 - val_accuracy: 0.9723 - val_loss: 0.0971
Epoch 5/5
1500/1500 ————— 2s 2ms/step - accuracy: 0.9876 - loss: 0.0401 - val_accuracy: 0.9750 - val_loss: 0.0876
```

- Ewaluacja - ocena modelu na zbiorze testowym.

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy:.2f}")
✓ 0.6s

313/313 ————— 1s 988us/step - accuracy: 0.9694 - loss: 0.0968
Test accuracy: 0.98
```

- Predykcja - zwrócenie prawdopodobieństwa i na ich podstawie wyznaczenie klas.

```
predictions = model.predict(x_test[:5])
print("Predicted classes:", predictions.argmax(axis=1))
print("True classes:", y_test[:5].argmax(axis=1))
✓ 0.1s

1/1 ————— 0s 54ms/step
Predicted classes: [7 2 1 0 4]
True classes: [7 2 1 0 4]
```

#### 4. Konwolucyjne sieci neuronowe

**Konwolucyjne sieci neuronowe** (ang. *CNN - Convolutional neural network*) to rodzaj hierarchicznych sieci neuronowych zaprojektowanych do przetwarzania danych o strukturze przestrzennej, takich jak obrazy (lub często szeregi czasowe). Wykorzystują operację konwolucji, która pozwala na automatyczne wykrywanie lokalnych wzorców w danych (np. krawędzie, tekstury) przy jednoczesnym zachowaniu ich struktury przestrzennej.

#### 5. Konwolucja

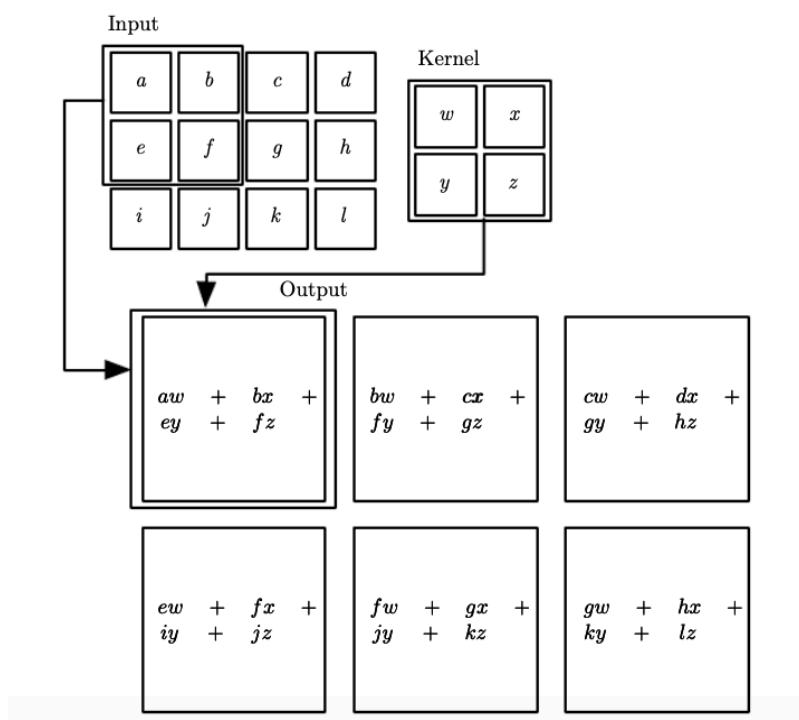
**Konwolucja** polega na “przesuwaniu” małej macierzy, zwanej **filtrem** lub **jądrem (kernel)**, po wejściowej macierzy danych i obliczaniu wartości przez mnożenie elementów filtra z odpowiadającymi im elementami macierzy wejściowej, a następnie sumowaniu wyników. Wynik operacji konwolucji jest często określany “mapą cech” (ang. *feature map*).

Przykład działania kernela

Input		Kernel		Output																	
<table border="1" style="border-collapse: collapse;"> <tr><td>0</td><td>1</td><td>2</td></tr> <tr><td>3</td><td>4</td><td>5</td></tr> <tr><td>6</td><td>7</td><td>8</td></tr> </table>	0	1	2	3	4	5	6	7	8	*	<table border="1" style="border-collapse: collapse;"> <tr><td>0</td><td>1</td></tr> <tr><td>2</td><td>3</td></tr> </table>	0	1	2	3	=	<table border="1" style="border-collapse: collapse;"> <tr><td>19</td><td>25</td></tr> <tr><td>37</td><td>43</td></tr> </table>	19	25	37	43
0	1	2																			
3	4	5																			
6	7	8																			
0	1																				
2	3																				
19	25																				
37	43																				

Liczba 19 została uzyskana przez operację  $0*0 + 1*1 + 3*2 + 4*3$ .

Można konwolucję przedstawić jako:

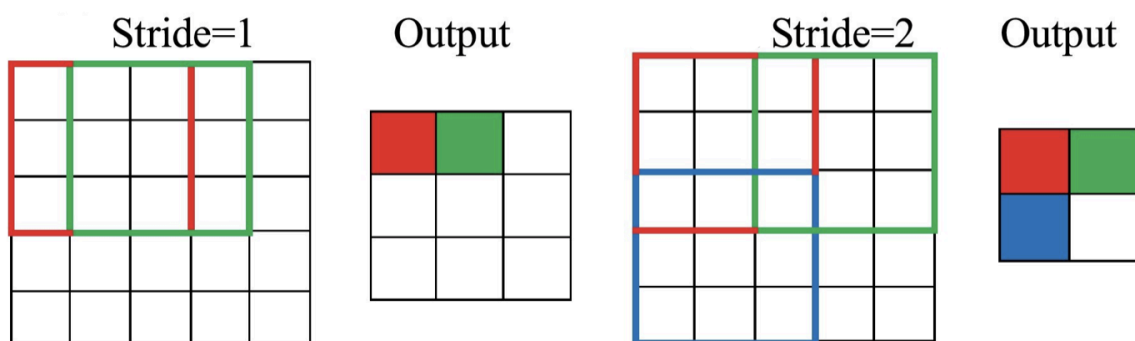


Konwolucja opiera się na trzech kluczowych ideach:

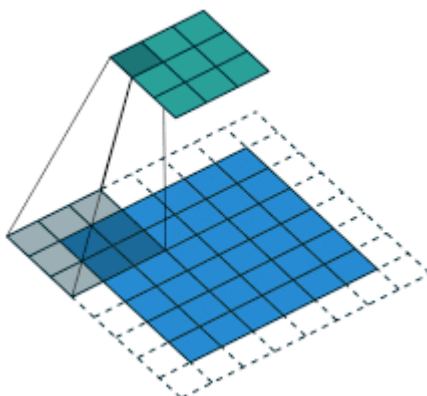
- **Rzadkie interakcje** (ang. *sparse interactions*) - w warstwach konwolucyjnych każdy neuron jest połączony tylko z lokalnym regionem wejścia. Dzięki temu sieć może efektywnie wykrywać lokalne cechy, takie jak krawędzie czy tekstury, jednocześnie redukując liczbę obliczeń i parametrów.
- **Dzielenie parametrów** (ang. *parameter sharing*) - jeden filtr jest stosowany na całym obrazie, co oznacza, że te same wagi są używane w wielu miejscach obrazu.
- **Reprezentacja ekwiwariantna** (ang. *equivariant representation*) - konwolucja jest ekwiwariantna względem translacji, co oznacza, że przesunięcie wzorca w obrazie (np. krawędzi) prowadzi do odpowiedniego przesunięcia w reprezentacji wyjściowej.

Dodatkowo, często stosuje się dwie operacje w warstwach konwolucyjnych:

- **Stride** - zmiana sposobu przesuwania się filtra. Stride=1 analizuje każdy punkt macierzy wejściowej, Stride=2 "przeskakuje" co drugi punkt, co daje możliwość redukcji rozdzielczości wyjścia.



- **Padding** - dodawanie dodatkowych elementów (zwykle zer) wokół krawędzi danych wejściowych. Celem jest umożliwienie filtrowi pełnego przetwarzania danych na krawędziach macierzy.



W przypadku stosowania stride i padding rozmiar macierzy wyjściowej można określić jako:

$$\text{rozmiar\_wyj} = \frac{\text{rozmiar\_wej} + 2 * \text{padding} - \text{rozmiar\_filtr}}{\text{stride}} + 1$$

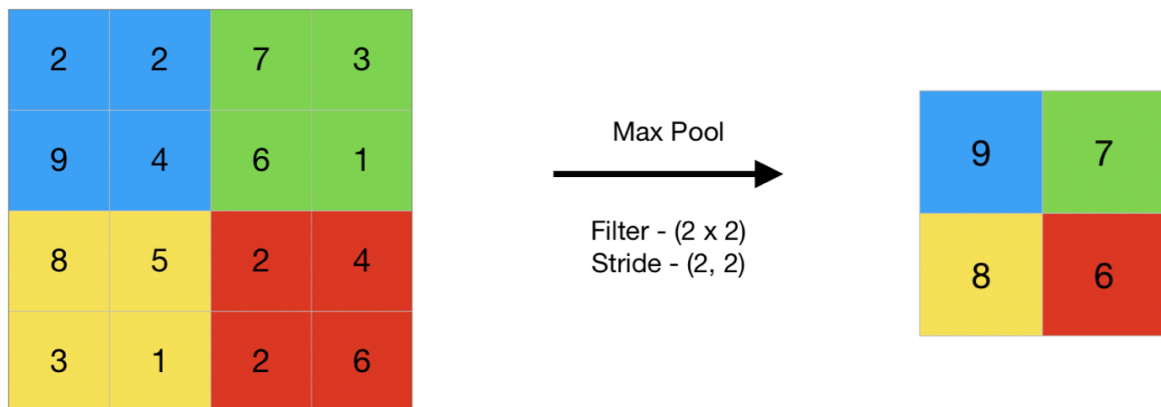
## 6. Pooling

**Pooling** to operacja stosowana w celu zmniejszenia wymiarów danych przy jednoczesnym zachowaniu najważniejszych informacji. Pooling redukuje rozmiary i zapobiegania przeuczeniu.

Pooling dzieli dane wejściowe na małe regiony (np.  $2 \times 2$ ) i stosuje określoną operację na każdym z tych regionów, zastępując go pojedynczą wartością. W poolingu nie stosujemy żadnego filtru do danych wejściowych, a jedynie upraszczamy informacje.

Najczęściej stosowane operacje:

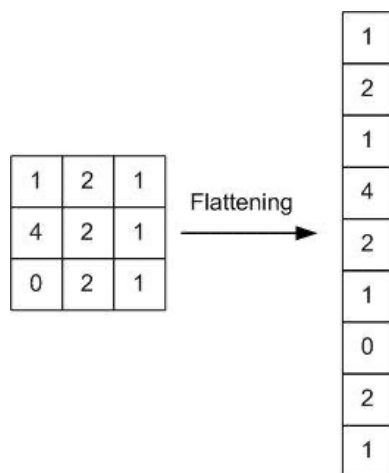
- **Max pooling** - Wybiera największą wartość w każdym regionie - zachowuje najważniejsze cechy w obszarze (np. krawędzie lub tekstury)



- **Avg pooling** - Oblicza średnią wartość z każdego regionu - zachowuje więcej szczegółów w danych, ale z mniejszą selektywnością niż max pooling.

## 7. Flattening

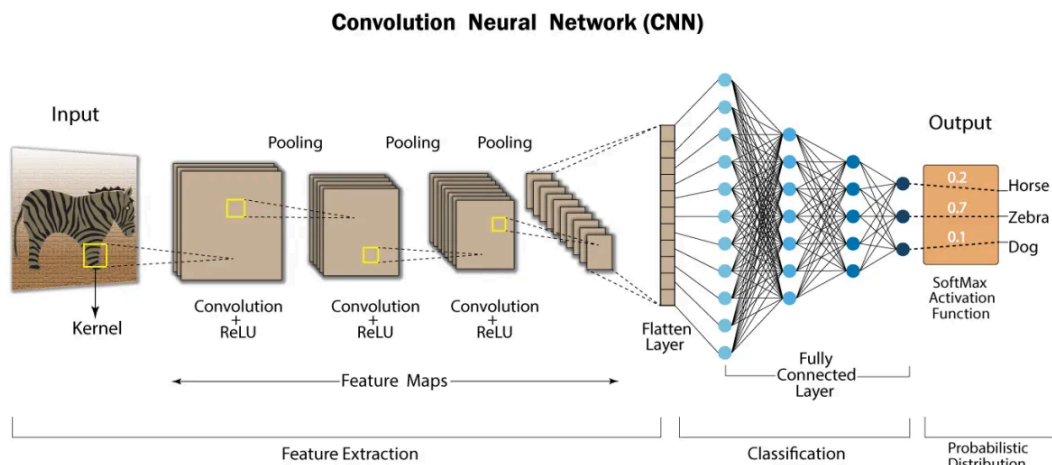
**Warstwa spłaszczająca** (Flattening) przekształca dane wielowymiarowe (np. tensor  $4 \times 4 \times 3$ ) w jednowymiarowy wektor. Ta warstwa nie wprowadza dodatkowych parametrów ani operacji uczących – jej jedyną funkcją jest przygotowanie danych z warstw konwolucyjnych lub poolingowych do wejścia w warstwy w pełni połączone (ang. *fully connected layers*), które wymagają danych w formie wektorowej.



Flattening layer pełni rolę "mostu" między częścią konwolucyjną (ekstrakcja cech) a klasyfikacyjną (decyzje modelu).

## 8. Fully-connected layer

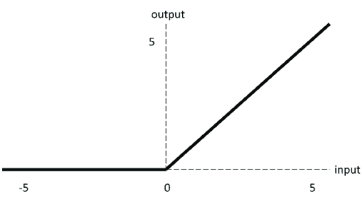
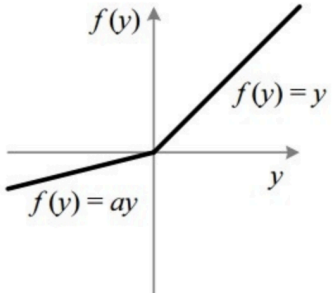
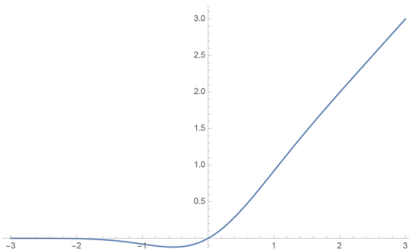
**Warstwy w pełni połączone** (ang. *fully-connected layer*) takich warstwach każdy neuron jest połączony z każdym neuronem z poprzedniej warstwy, co umożliwia modelowi reprezentowanie złożonych relacji między wejściem a wyjściem. Warstwa generuje jednowymiarowy wektor wyjściowy, który można interpretować np. jako predykcję klasy (klasyfikacja). Są częścią składową CNN i mają taką samą architekturę jak MLP (MLP jest strukturą sieci neuronowej, która składa się wyłącznie z warstw w pełni połączonych).



## 9. Funkcje aktywacyjne w CNN

Funkcje aktywacyjne występują w warstwach konwolucyjnych oraz w warstwach pełni połączonych. Warstwy pooling i spłaszczające NIE mają funkcji aktywacji.

Najczęściej stosowane funkcje aktywacyjne (poza sigmoid i tanh - omówione na Lab3):

funkcja	wykres	postać
ReLU		$\text{ReLU}(x) = x^+ = \max(0, x)$
Leaky ReLU		$f(x) = \begin{cases} x & x > 0, \\ \alpha x & x \leq 0 \end{cases}$
Gelu		$\text{GELU}(x) = xP(X \leq x) = x\Phi(x)$

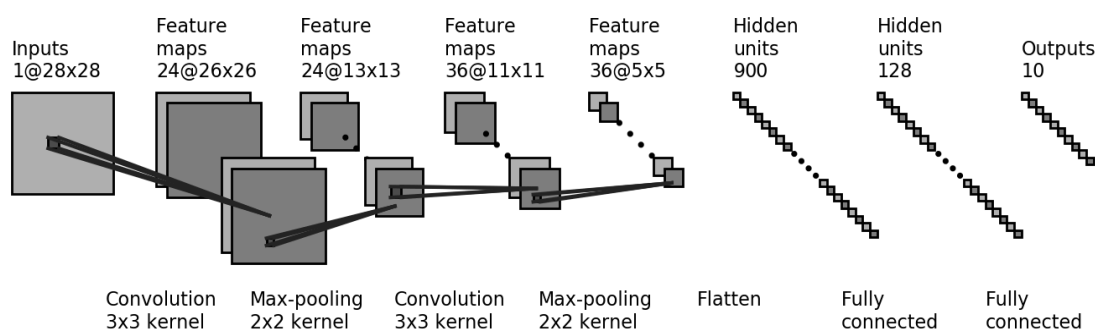
Inne często stosowane to **softmax** i **softplus**.



## 10. Zadanie

Zaimplementuj konwolucyjną sieć neuronową do **klasyfikacji obrazów cyfr** z zestawu **MNIST**, używając architektury przedstawionej na schemacie:

- dwie warstwy konwolucyjne (24 filtry 3x3 i 36 filtrów 3x3) przeplatane warstwami max-pooling (2x2),
- warstwa spłaszczająca,
- dwie w pełni połączone warstwy (900 i 128 neuronów)
- warstwa wyjściowa z 10 neuronami.



Skompiluj model z optymalizatorem Adam, a następnie wytrenuj go na zbiorze treningowym przez 10 epok z walidacją. Oceń dokładność na zbiorze testowy.

### Protips:

- Zbiór MNIST znajduje się w bibliotece tensorflow `mnist = tf.keras.datasets.mnist`;
- Przygotuj dane przed budową sieci (przekształcanie na wektory, normalizacja oraz kodowane etykiety);
- Do budowy modelu użyj metod z TensorFlow (`tf.keras.layers.Conv2D`, `tf.keras.layers.MaxPool2D`, `tf.keras.layers.Flatten`, `tf.keras.layers.Dense`). Użyj funkcji aktywacyjnych ReLU, a dla warstwy wyjściowej softmax;
- Użyj metody `compile` do kompilacji modelu, zdefiniuj funkcję straty, metrykę i optymalizator (np. Adam);
- Zdefiniuj `batch_size`, `validation_split`, liczbę epok oraz rozmiar zbioru ewaluacyjnego w metodzie `fit`;
- Użyj metody `evaluate` do oceny dokładności;

**Zadania dodatkowe:**

- Spróbuj użyć kilku optymalizatorów (np. SGD, RMSprop, Adam) i porównaj wyniki. Zmieniaj współczynniki uczenia i sprawdź, jak wpływają na wydajność modelu.
- Zmodyfikuj architekturę modelu, zmieniając liczbę filtrów w warstwach konwolucyjnych (np. 32, 64 zamiast 24, 36) lub dodaj więcej warstw konwolucyjnych.
- Zmień funkcję aktywacji w warstwach ukrytych z ReLU na inne funkcje aktywacji (np. LeakyReLU lub ELU).
- Dodaj regularizację L2 lub Dropout. Sprawdź wpływ różnych wartości współczynnika regularizacji (np. 0.01, 0.001);
- Zastosuj augmentację danych (np. obracanie, przesunięcie, skalowanie) na zbiorze treningowym. Sprawdź, czy poprawia to dokładność modelu.