



# Дослідження і проектування інтелектуальних систем

Лекція 2: Згорткові мережі

Кочура Юрій Петрович  
[iuriy.kochura@gmail.com](mailto:iuriy.kochura@gmail.com)  
[@y\\_kochura](https://twitter.com/y_kochura)

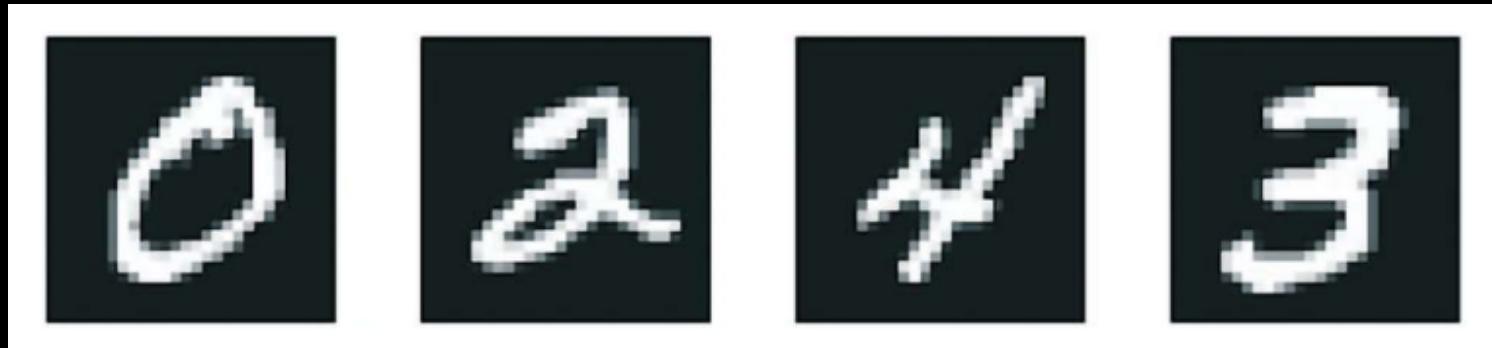
# Сьогодні

Розуміння згорткових нейронних мереж (convnets або CNNs):

- Повнозв'язна vs згорткова мережа
- Операція згортки
- Крок згортки
- Ефект доповнення (padding)
- Операція агрегації (pooling)

# Повнозв'язна мережа

# MNIST: приклади



**Примітка!** У машинному навчанні для задачі класифікації категорія даних називається **класом**. Кожна одиниця даних називається **прикладом**. Клас, який пов'язаний із певним прикладом називається **міткою (label)**.

# Імпортування набору даних MNIST у Keras

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.lo
```

- `train_images` та `train_labels` – навчальний набір даних (дані на яких модель буде навчатись)
- `test_images` та `test_labels` – тестовий набір (дані на яких буде оцінено продуктивність моделі)

# Навчальний набір

```
train_images.shape
```

```
(60000, 28, 28)
```

```
len(train_labels)
```

```
60000
```

```
train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

# Тестовий набір

```
test_images.shape
```

```
(10000, 28, 28)
```

```
len(test_labels)
```

```
10000
```

```
test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

# Архітектура мережі

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Робочий процес буде таким: спочатку ми передамо нейронній мережі навчальні дані `train_images` та `train_labels`. Таким чином мережа навчиться пов'язувати зображення з мітками. Потім ми попросимо мережу створити прогнози для `test_images` та перевіримо, чи відповідають ці прогнози міткам з `test_labels`.

# Готуємо мережу до навчання

Щоб підготувати мережу до навчання, нам потрібно визначити на етапі компіляції:

- Оптимізатор – алгоритм за допомогою якого модель оновлюватиметься на основі навчальних даних, які надаються моделі для покращення свої продуктивності.
- Функція втрат – спосіб виміру втрат моделі. Оптимізатор намагається мінімізувати втрати моделі.
- Метрики для моніторингу під час навчання та тестування – у цій задачі ми слідкуватимо за точністю (відсоток зображень, які були правильно класифіковані).

# Компіляція моделі

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=[ "accuracy" ])
```

# Підготовка даних

```
train_images = train_images.reshape(60000, 28 * 28)
train_images = train_images.astype("float32") / 255

test_images = test_images.reshape(10000, 28 * 28)
test_images = test_images.astype("float32") / 255
```

Раніше наші навчальні зображення зберігалися в масиві  $(60000, 28, 28)$  типу `uint8` зі значеннями в інтервалі  $[0, 255]$ . Ми перетворюємо його на масив `float32` форми  $(60000, 28 * 28)$  зі значеннями від `0` до `1`. Тестовий набір даних перетворюємо аналогічним чином.

# Навчання моделі

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5  
469/469 [=====] - 3s 5ms/step - loss: 0.2  
Epoch 2/5  
469/469 [=====] - 2s 5ms/step - loss: 0.10  
Epoch 3/5  
469/469 [=====] - 2s 5ms/step - loss: 0.01  
Epoch 4/5  
469/469 [=====] - 3s 6ms/step - loss: 0.00  
Epoch 5/5  
469/469 [=====] - 3s 6ms/step - loss: 0.01  
  
<keras.src.callbacks.History at 0x7e81cba25e70>
```

# Виконання прогнозу

```
test_digits = test_images[0:10]
prediction = model.predict(test_digits)
prediction[0]
```

```
array([3.9224375e-08, 4.9862869e-09, 9.4487259e-06, 8.6249776e-05,
       2.9801717e-11, 3.3959207e-08, 4.4558798e-13, 9.9990362e-01,
       1.0565784e-07, 3.3271664e-07], dtype=float32)
```

Кожне індекс  $i$  в цьому масиві відповідає ймовірності того, що наше тестове зображення `test_digits[0]` належить до класу  $i$ .

# Виконання прогнозу

Перше тестове зображення має найбільшу ймовірність (`0.99990362`, майже 1) для індекса масива `7`, тому відповідно до цього прогнозу моделі це має бути `7`:

```
prediction[0].argmax()
```

7

```
prediction[0][7]
```

0.9999036

Перевіримо на відповідність тестовій мітці:

```
test_labels[0]
```

7

# Оцінка моделі на нових даних

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [=====] - 1s 2ms/step - loss: 0.0
```

```
print(f"test accuracy: {test_acc}")
```

```
test accuracy: 0.9817000031471252
```

# Згорткові мережі

# Створюємо згорткову мережу

```
from tensorflow import keras
from tensorflow.keras import layers
input = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu") (input)
x = layers.MaxPool2D(pool_size=2) (x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu") (x)
x = layers.MaxPool2D(pool_size=2) (x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu") (x)
x = layers.Flatten() (x)
output = layers.Dense(10, activation="softmax") (x)
model = keras.Model(inputs=input, outputs=output)
```

Важливо, що convnet приймає на вхід тензори форми (`image_height`, `image_width`, `image_channels`), не включаючи розмірність пакету. У цьому випадку ми налаштуємо convnet для обробки вхідних даних розміром `(28, 28, 1)`, що відповідає формату зображень MNIST.

```
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
<hr/>		
input_2 (InputLayer)	[ (None, 28, 28, 1) ]	0
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 128)	73856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 10)	11530
<hr/>		

Total params: 104202 (407.04 KB)

Trainable params: 104202 (407.04 KB)

Non-trainable params: 0 (0.00 Byte)

# Підготовка даних, компіляція та навчання моделі

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape(60000, 28, 28, 1)
train_images = train_images.astype("float32") / 255

test_images = test_images.reshape(10000, 28, 28, 1)
test_images = test_images.astype("float32") / 255

model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

# Оцінка згорткової моделі на нових даних

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [=====] - 2s 6ms/step - loss: 0.01
```

```
print(f"test accuracy: {test_acc}")
```

```
test accuracy: 0.991100013256073
```

# Будівельні блоки

# Тензор (**tensor**)

масив чисел, розташованих у сітці зі змінною кількістю осей

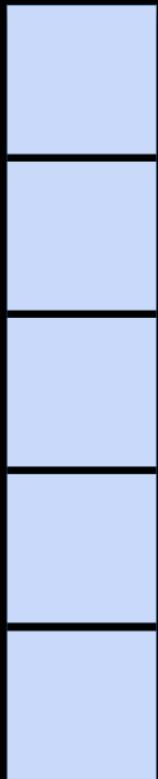




0D tensor

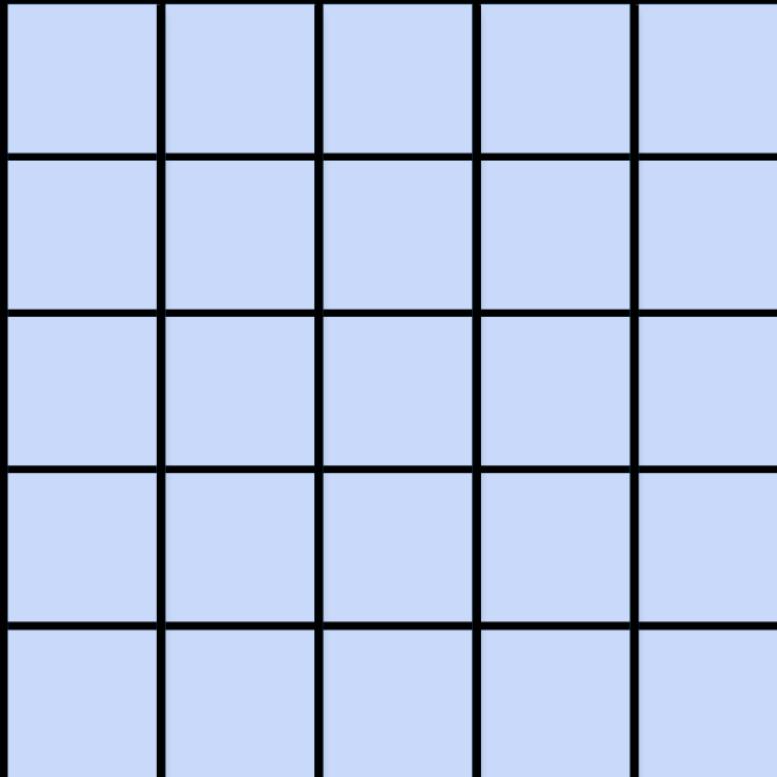


$\text{rows} \times 1$

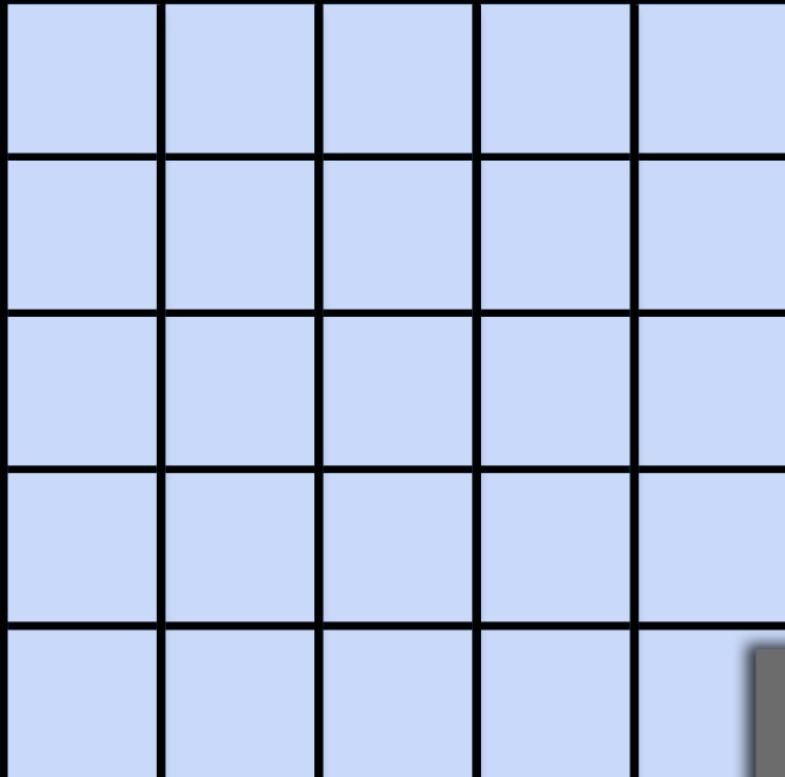


1D tensor

rows × 1

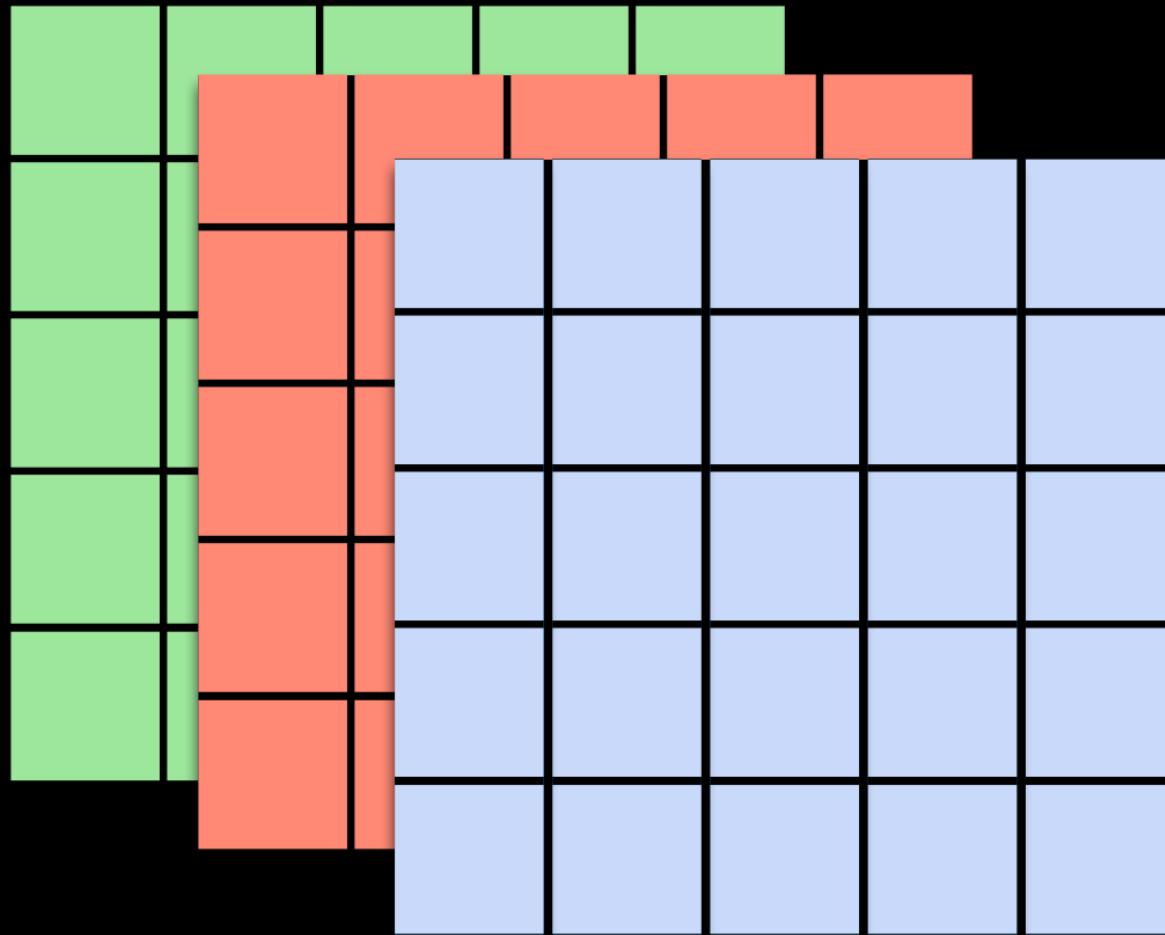


rows × cols

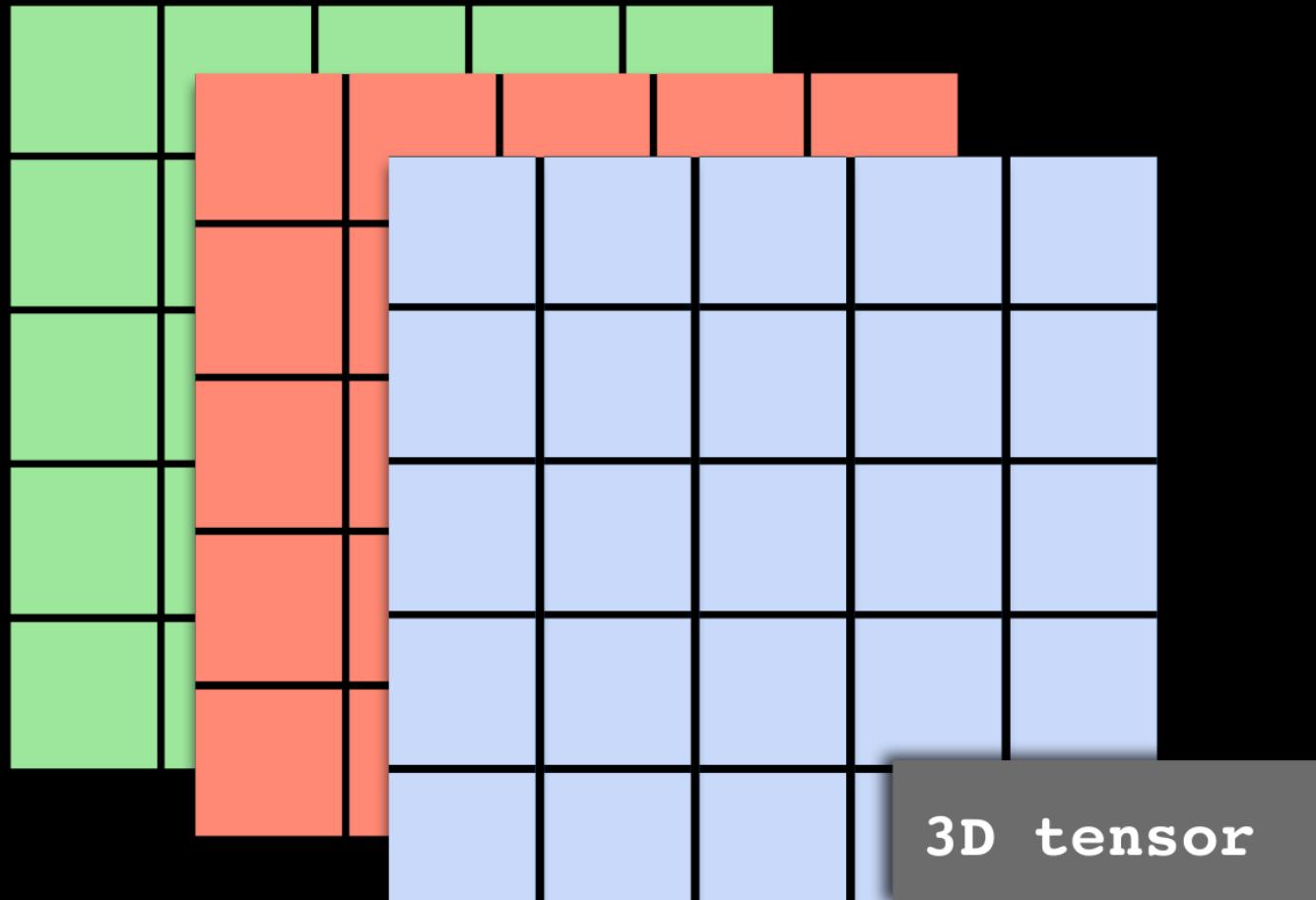


**2D tensor**

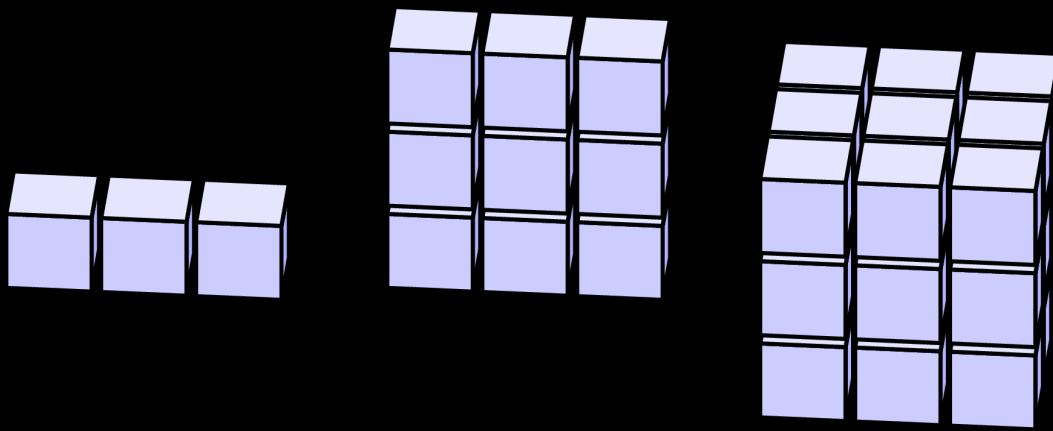
rows × cols



$\text{rows} \times \text{cols} \times \text{channels}$

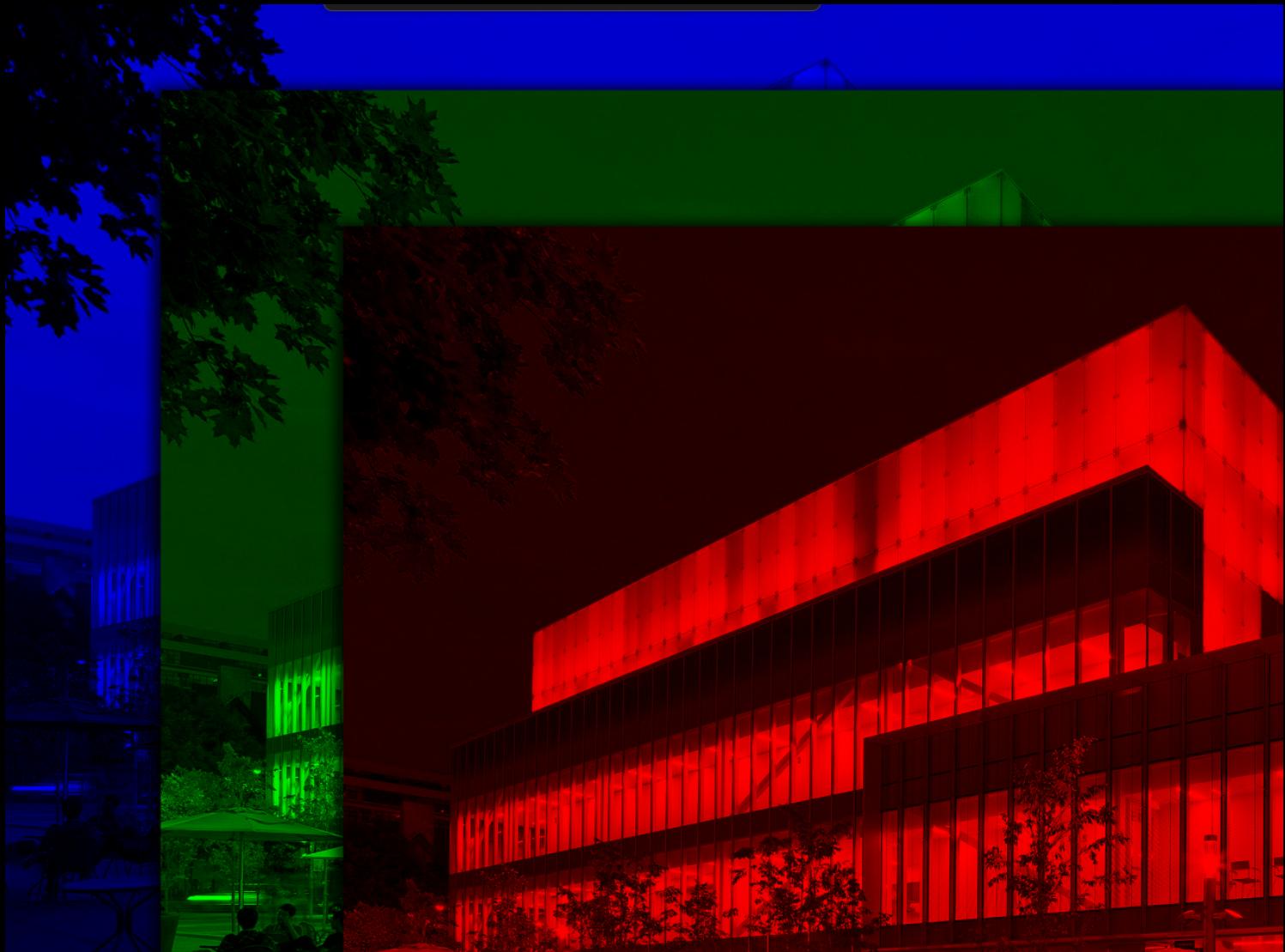


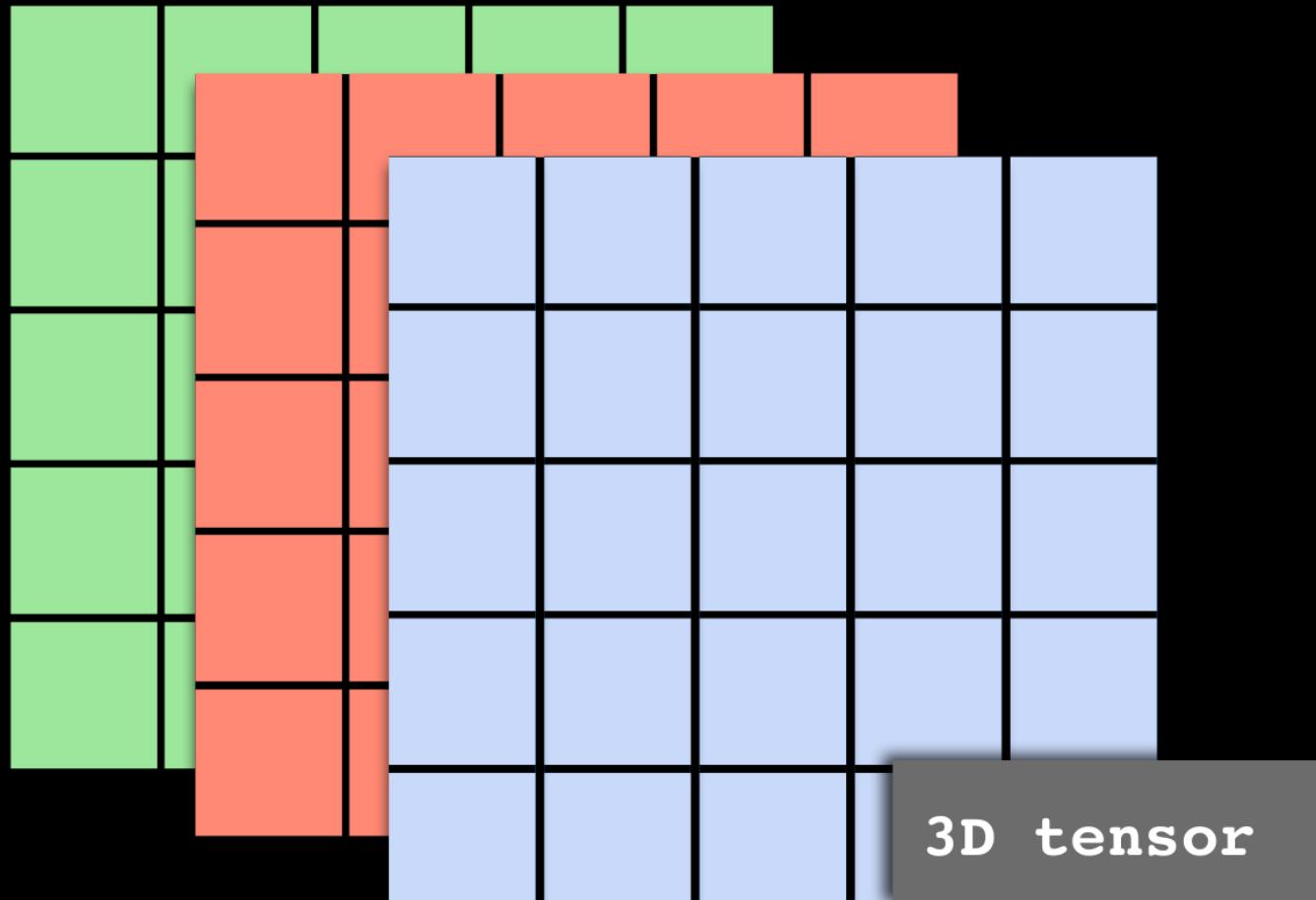
$\text{rows} \times \text{cols} \times \text{channels}$



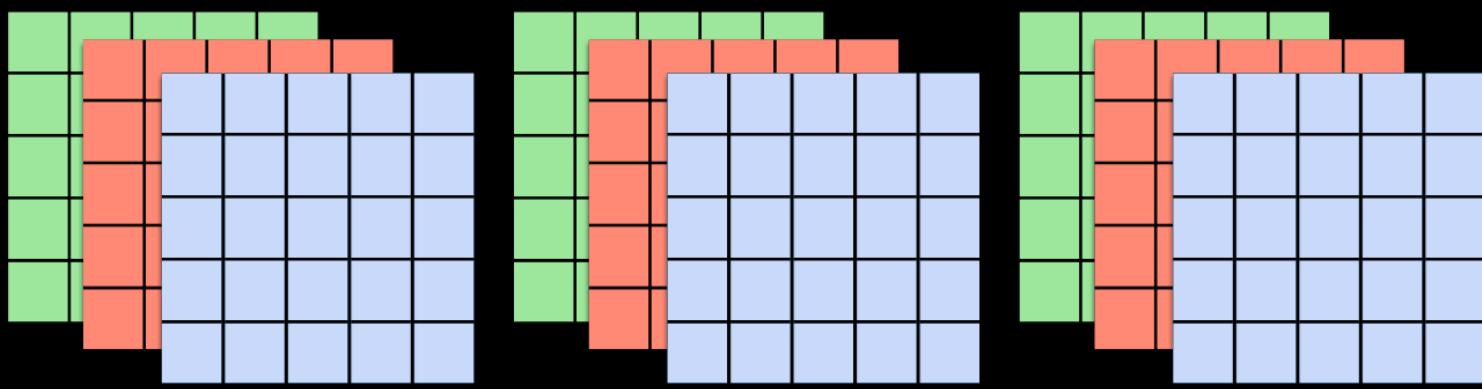


$768 \times 1024 \times 3$

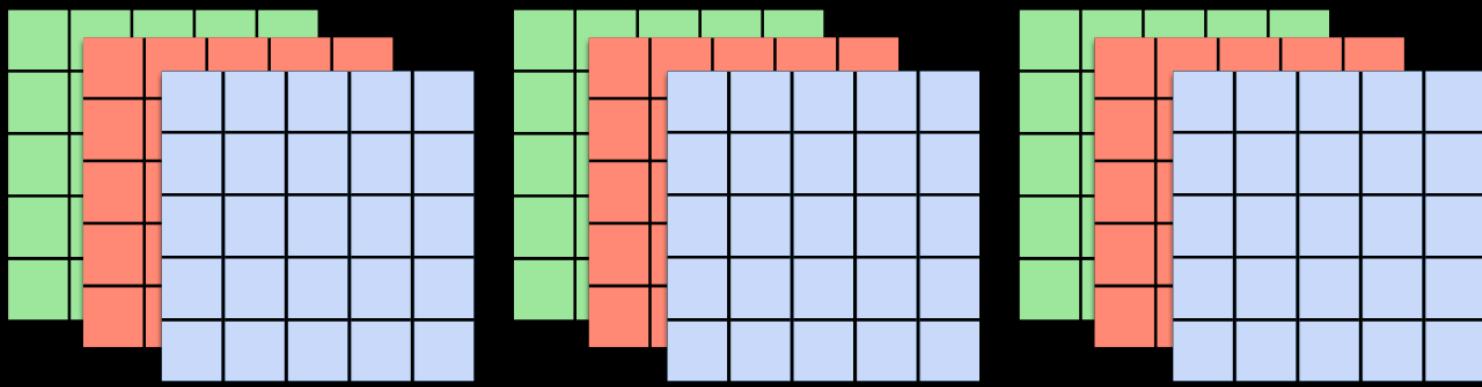




$\text{rows} \times \text{cols} \times \text{channels}$



$\text{rows} \times \text{cols} \times \text{channels} \times t$



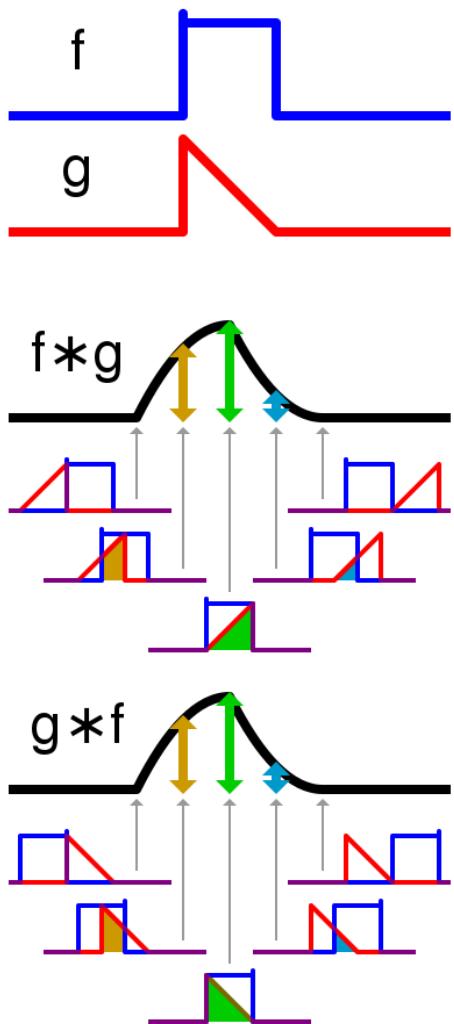
$\text{rows} \times \text{cols} \times \text{channels} \times t$

**4D tensor**

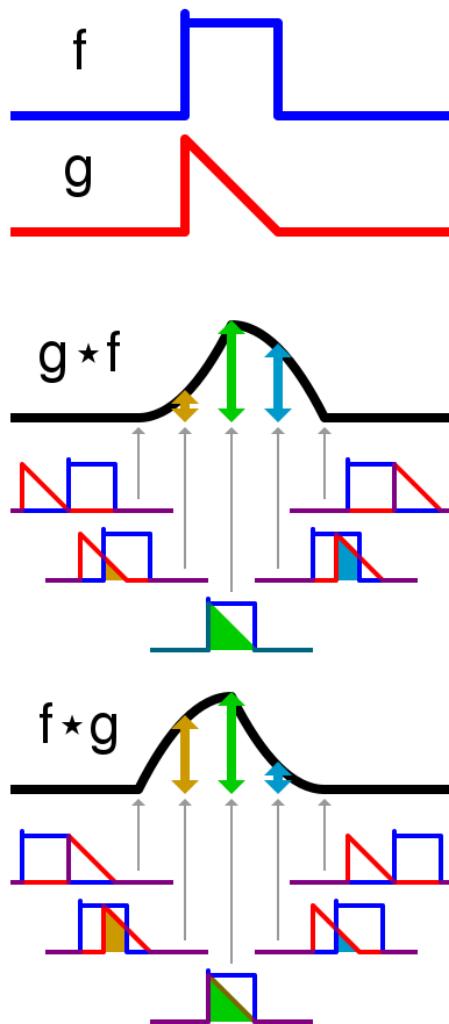
# Згортка (convolution)

$$(f * g)(x) = \int f(z)g(x - z) dz$$

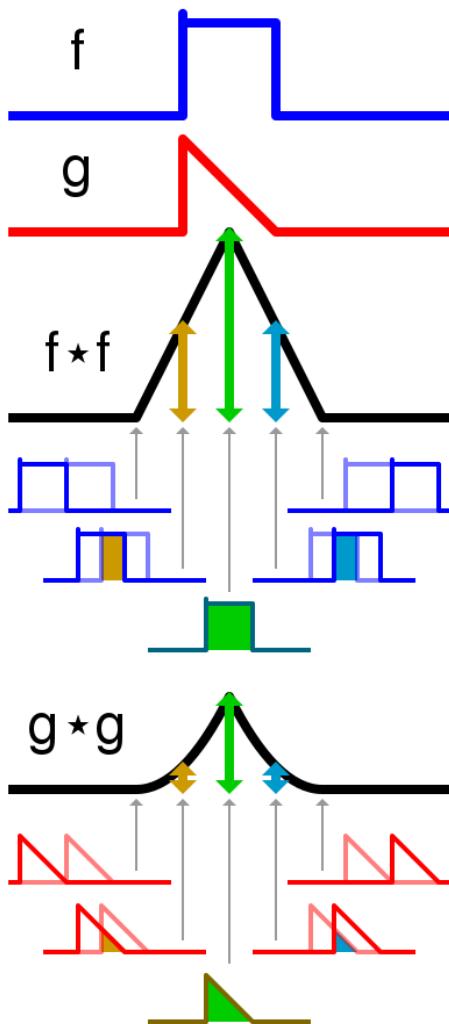
## Convolution



## Cross-correlation



## Autocorrelation



# convolution



# correlation

# Взаємна кореляція

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Вхід

0	1	2
3	4	5
6	7	8

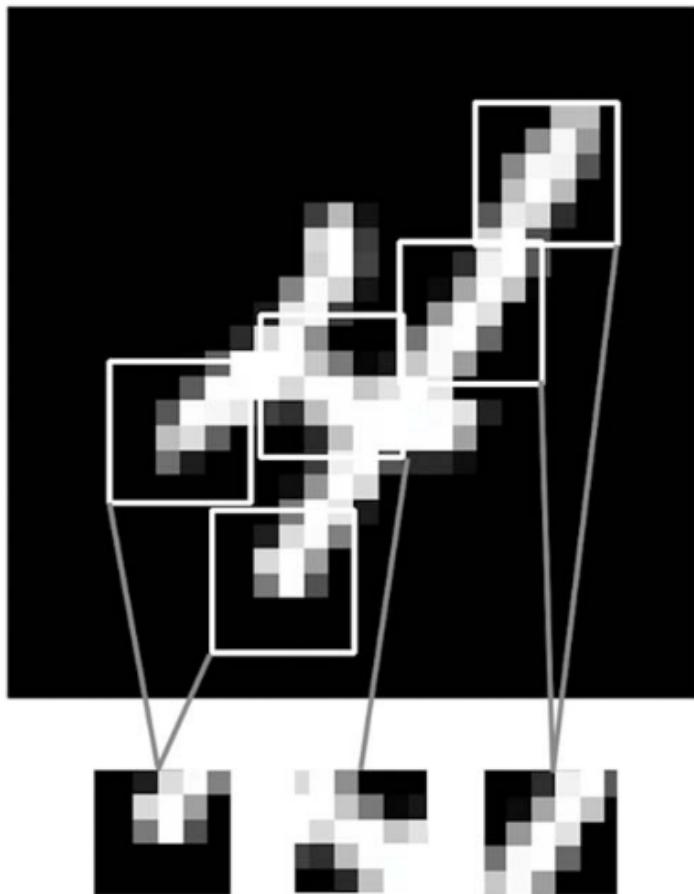
.

Фільтр

258	294
357	438

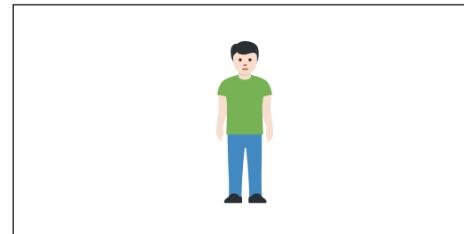
Вихід

# Згортковий шар

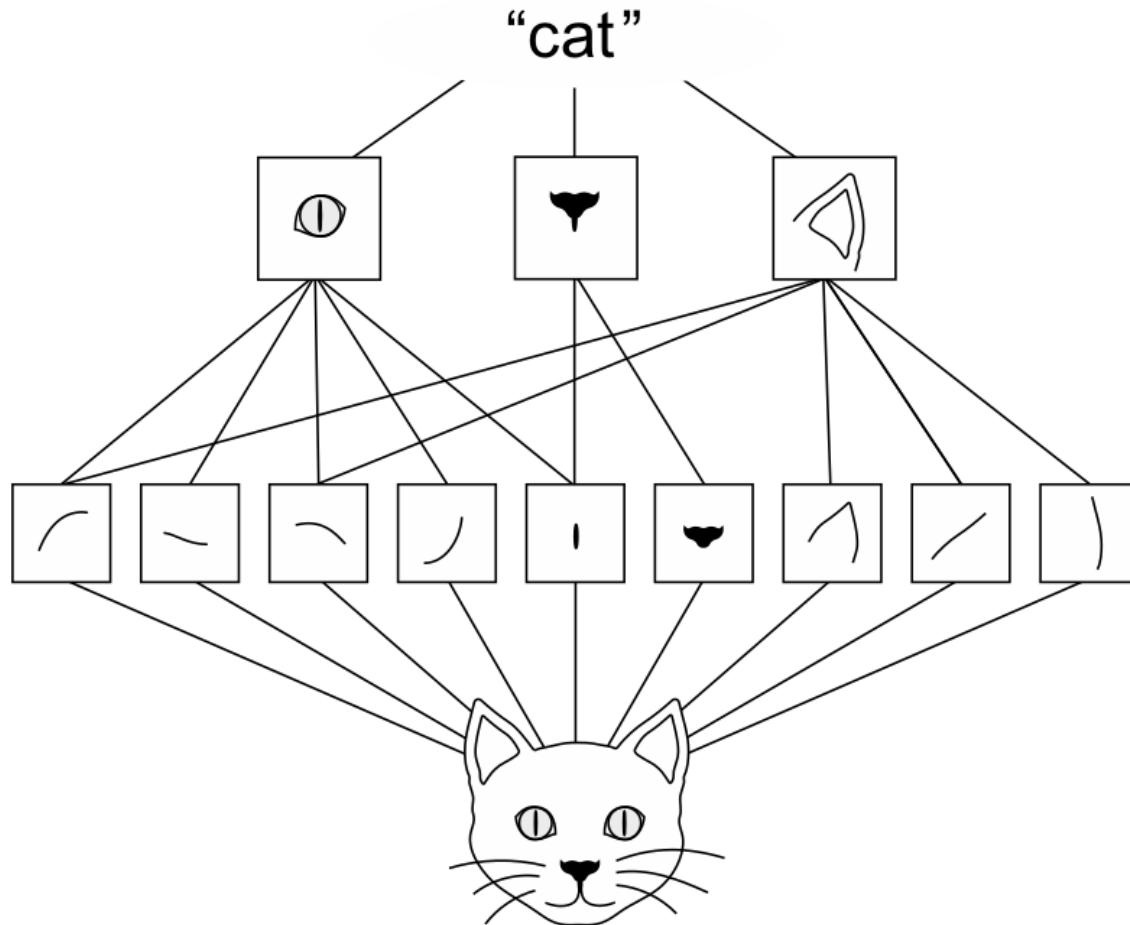


# Інваріантнти відносно зміщень

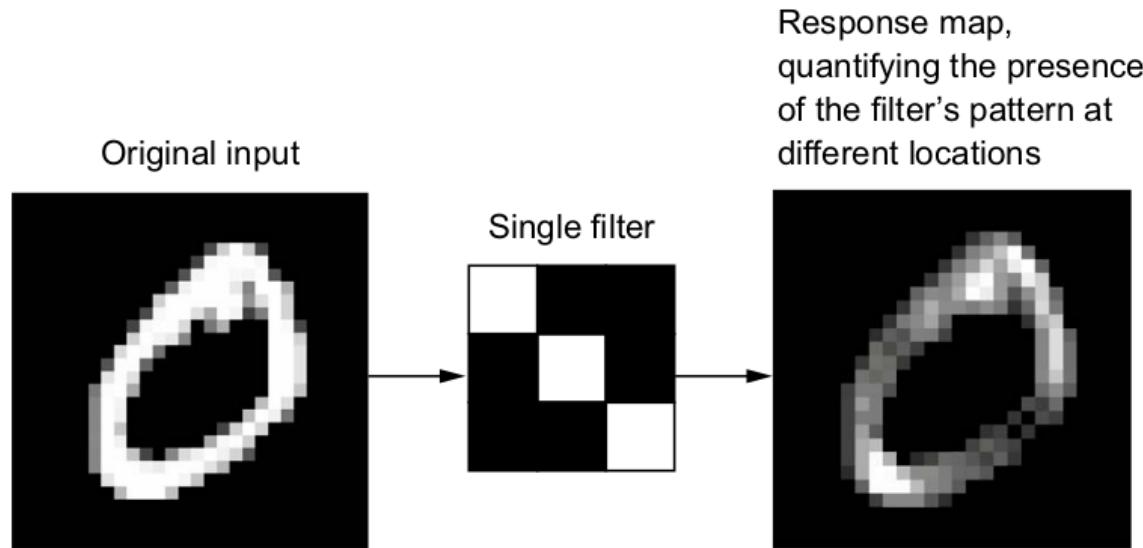
CNN інваріантнти відносно зміщень



# Вивчають просторову ієрархію шаблонів

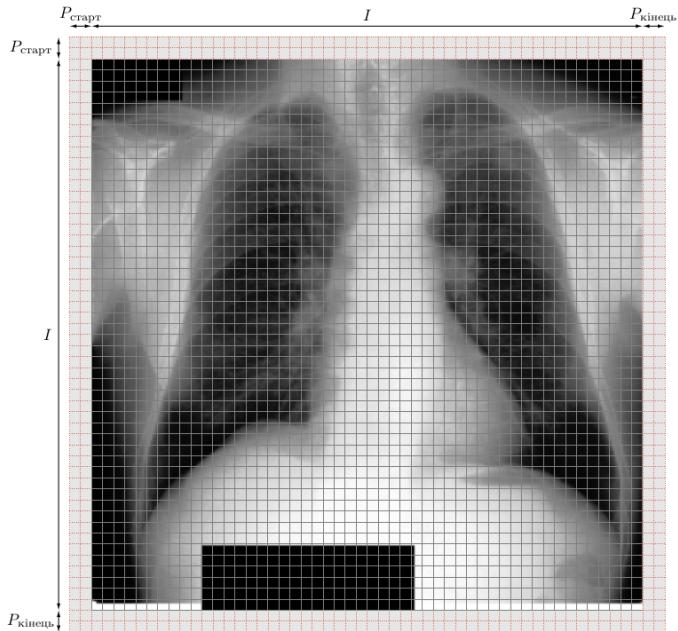


Вихідна карта ознак: двовимірна карта присутності візерунка в різних місцях вхідного тезора

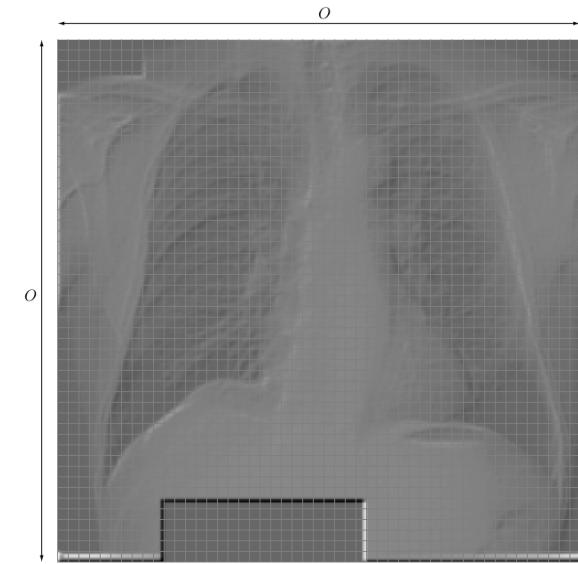
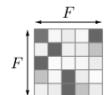


У прикладі MNIST перший шар згортки приймає карту ознак розміром **(28, 28, 1)** і виводить карту функцій розміром **(26, 26, 32)**: він обчислює 32 фільтри над своїм входом. Кожен із цих 32 вихідних каналів містить сітку значень  $26 \times 26$ , яка є картою відгуку фільтра над входом, що вказує на відповідь цього шаблону фільтра в різних місцях входу (див. малюнок вище).

# Розмір виходу згортки



Вхід



Фільтр  
Вихід

$$O = \frac{I - F + 2P}{S} + 1$$

# Демо

## Як працює згортка?

Згортки визначаються двома ключовими параметрами:

- **Розмір патчів, отриманих із вхідних даних** – зазвичай це  $3 \times 3$  або  $5 \times 5$ . У прикладі вони були  $3 \times 3$ , що є звичайним вибором.
- **Глибина вихідної карти ознак** – це кількість фільтрів, обчислених згорткою. Приклад почався з глибини 32 і закінчився глибиною 128.

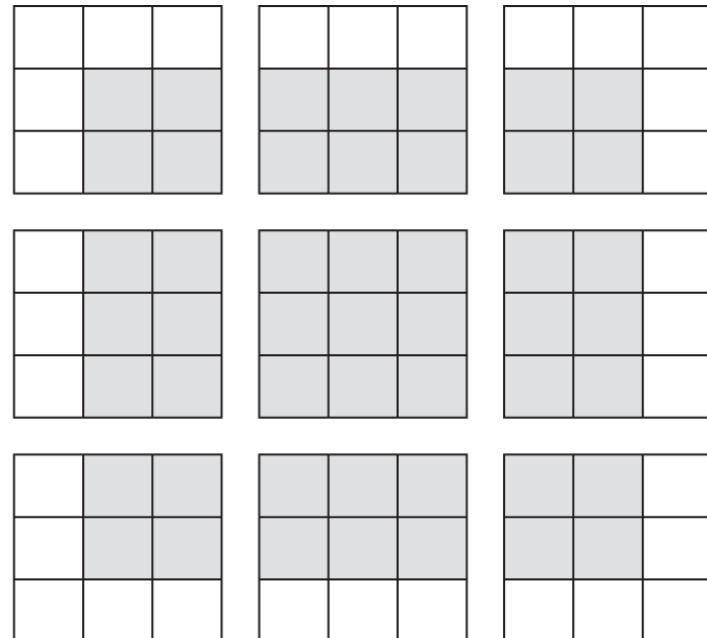
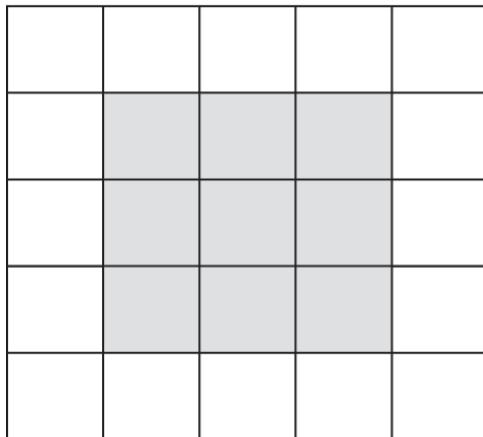
```
from tensorflow import keras
from tensorflow.keras import layers
input = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu") (input)
x = layers.MaxPool2D(pool_size=2) (x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu") (x)
x = layers.MaxPool2D(pool_size=2) (x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu") (x)
x = layers.Flatten() (x)
output = layers.Dense(10, activation="softmax") (x)
model = keras.Model(inputs=input, outputs=output)
```

У Keras шар Conv2D має наступні параметри ([Глибина вихідної карти ознак, Розмір патчів, отриманих із вхідних даних](#)):

```
Conv2D (output_depth, (window_height, window_width))
```

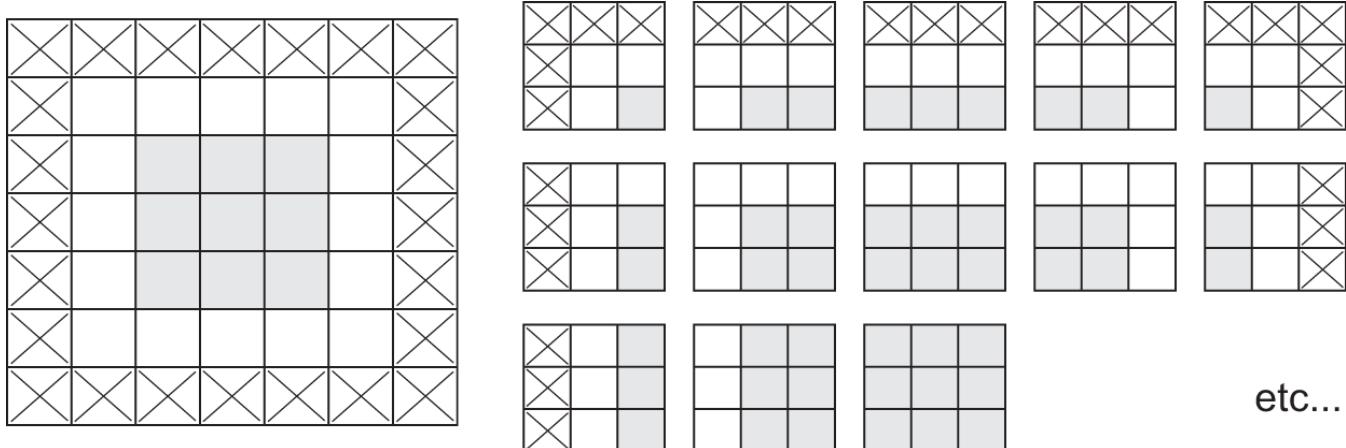
# Розуміння ефекту додавання (padding)

Ядро розміром  $3 \times 3$ , вхідна карта ознак  $5 \times 5$



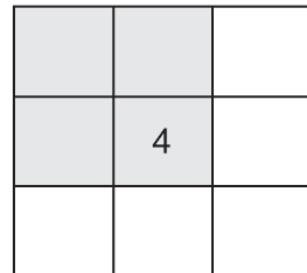
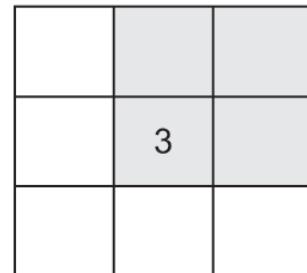
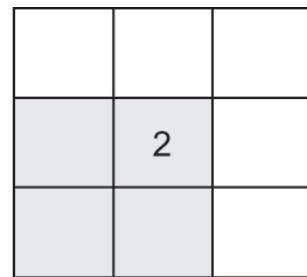
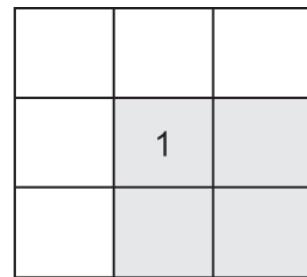
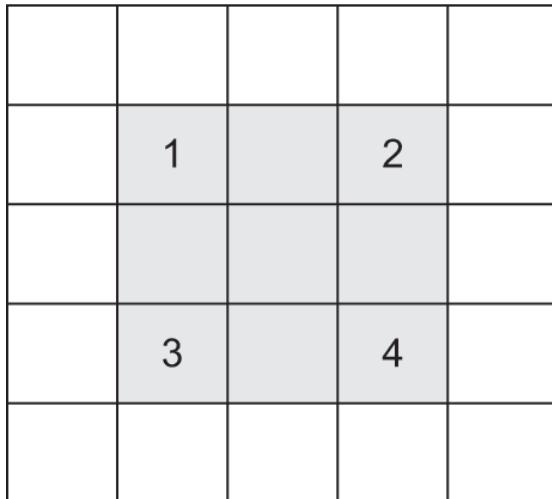
# Розуміння ефекту доповнення

Доповнююємо вхідну карту ознак  $5 \times 5$  для того, щоб можна було витягнути **25** патчів розміром  $3 \times 3$



# Розуміння кроку згортки

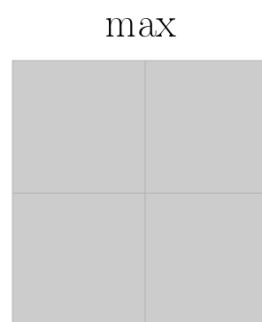
**3×3 ядро з кроком 2×2**



# Операція максимізаційного агрегування (max-pooling)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Вхід



max

=

5	7
13	15

Вікно

Вихід

## Операція максимізаційного агрегування (max-pooling)

```
from tensorflow import keras
from tensorflow.keras import layers
input = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu") (input)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu") (x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu") (x)
x = layers.Flatten() (x)
output = layers.Dense(10, activation="softmax") (x)
model_without_max_pool = keras.Model(inputs=input, outputs=output)
```

```
model_without_max_pool.summary()
```

Model: "model\_1"

Layer (type)	Output Shape	Param #
<hr/>		
input_2 (InputLayer)	[None, 28, 28, 1]	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
conv2d_4 (Conv2D)	(None, 24, 24, 64)	18496
conv2d_5 (Conv2D)	(None, 22, 22, 128)	73856
flatten_1 (Flatten)	(None, 61952)	0
dense_1 (Dense)	(None, 10)	619530
<hr/>		

Total params: 712202 (2.72 MB)

Trainable params: 712202 (2.72 MB)

Non-trainable params: 0 (0.00 Byte)

# Кінець