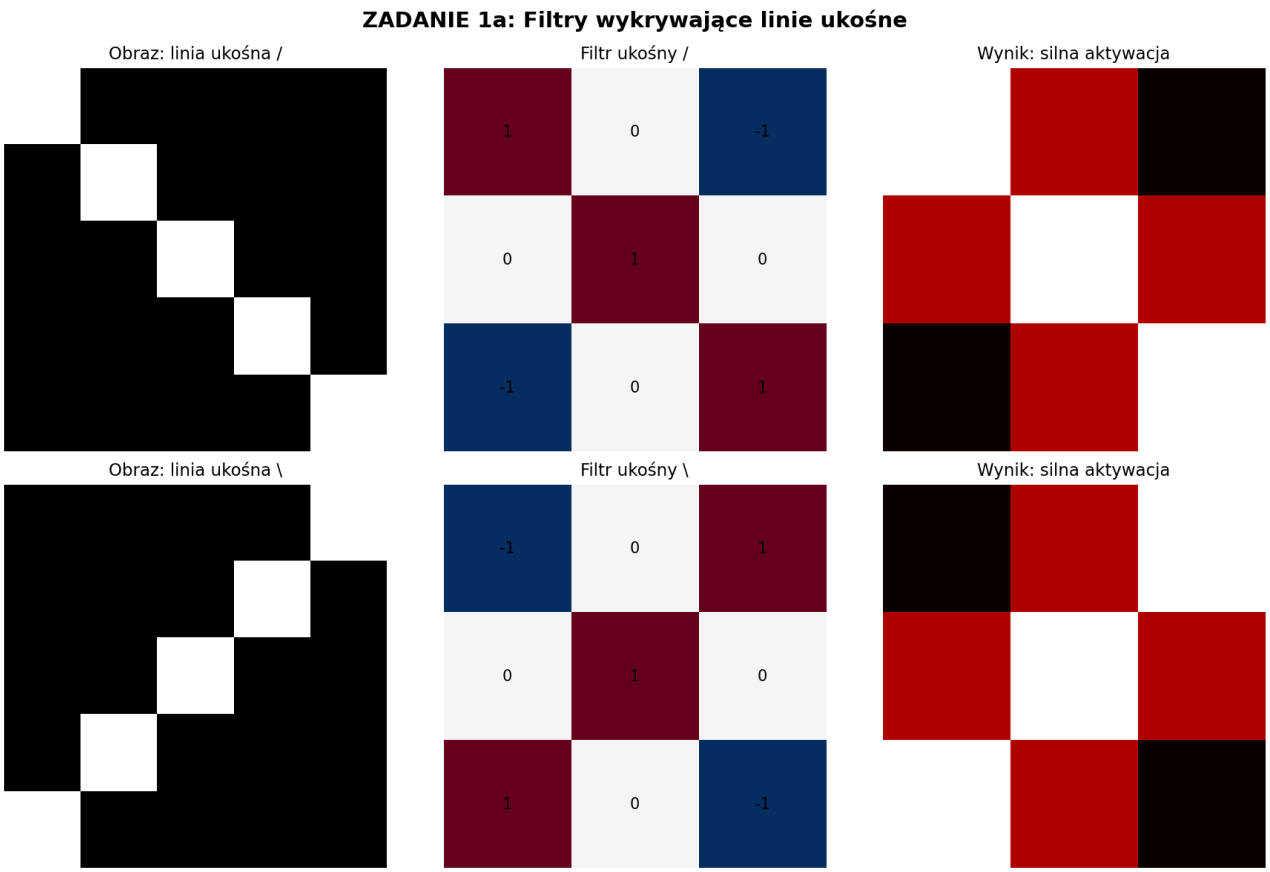
Sprawozdanie z Laboratorium CNN
Podstawy Sieci Konwolucyjnych

Kacper Łacniak
21.11.2025
Uczenie Głębokie - Laboratorium 2

Zad. 1a. - Filtr wykrywający ukośne linie

```
# Filtr wykrywający linie ukośne w prawo (/)
diagonal_filter_right = np.array([
    [1, 0, -1],
    [0, 1, 0],
    [-1, 0, 1]
])

# Filtr wykrywający linie ukośne w lewo (\)
diagonal_filter_left = np.array([
    [-1, 0, 1],
    [0, 1, 0],
    [1, 0, -1]
])

# Testowanie na obrazie z ukośną linią
diagonal_image = np.array([
    [1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 0, 0, 0, 1]
], dtype=np.float32)

result_diag_right = convolution2d(diagonal_image, diagonal_filter_right)
result_diag_left = convolution2d(diagonal_image, diagonal_filter_left)

# Wizualizacja
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

axes[0, 0].imshow(diagonal_image, cmap='gray')
axes[0, 0].set_title('Obraz: linia ukośna /', fontsize=11)
axes[0, 0].axis('off')

axes[0, 1].imshow(diagonal_filter_right, cmap='RdBu_r', vmin=-1, vmax=1)
axes[0, 1].set_title('Filtr ukośny /', fontsize=11)
axes[0, 1].axis('off')
for i in range(3):
    for j in range(3):
```

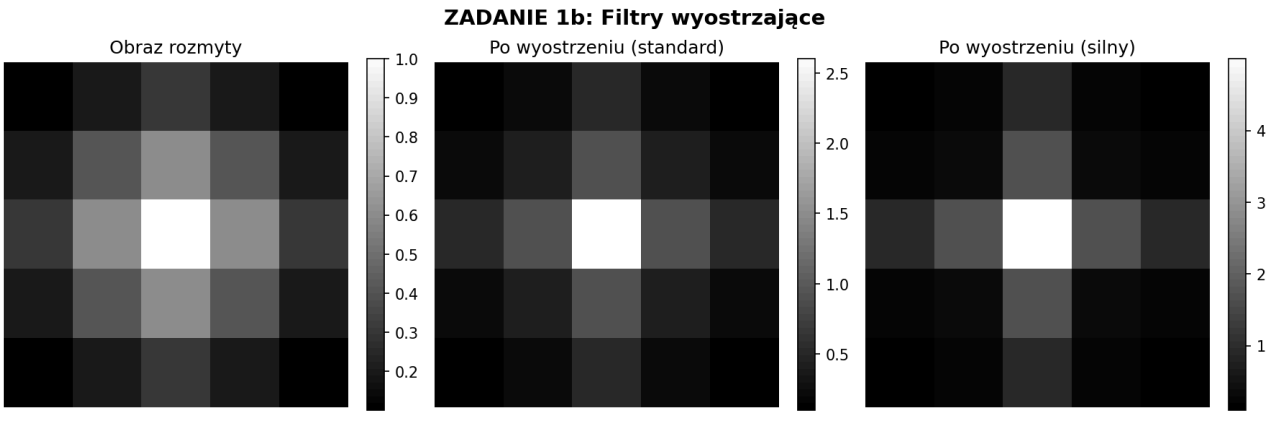
```
        axes[0, 1].text(j, i, f'{diagonal_filter_right[i,j]:.0f}',
                        ha='center', va='center', fontsize=10)

axes[0, 2].imshow(result_diag_right, cmap='hot')
axes[0, 2].set_title('Wynik: silna aktywacja', fontsize=11)
axes[0, 2].axis('off')

# Druga linia ukośna (przeciwna)
diagonal_image2 = np.array([
    [0, 0, 0, 0, 1],
    [0, 0, 0, 1, 0],
    [0, 0, 1, 0, 0],
    [0, 1, 0, 0, 0],
    [1, 0, 0, 0, 0]
], dtype=np.float32)

result_diag_left2 = convolution2d(diagonal_image2, diagonal_filter_left)

axes[1, 0].imshow(diagonal_image2, cmap='gray')
axes[1, 0].set_title('Obraz: linia ukośna \\', fontsize=11)
axes[1, 0].axis('off')

axes[1, 1].imshow(diagonal_filter_left, cmap='RdBu_r', vmin=-1, vmax=1)
axes[1, 1].set_title('Filtr ukośny \\', fontsize=11)
axes[1, 1].axis('off')
for i in range(3):
    for j in range(3):
        axes[1, 1].text(j, i, f'{diagonal_filter_left[i,j]:.0f}',
                        ha='center', va='center', fontsize=10)

axes[1, 2].imshow(result_diag_left2, cmap='hot')
axes[1, 2].set_title('Wynik: silna aktywacja', fontsize=11)
axes[1, 2].axis('off')
```



Zad. 1b - Filtr wyostrzający obraz (Sharpening)

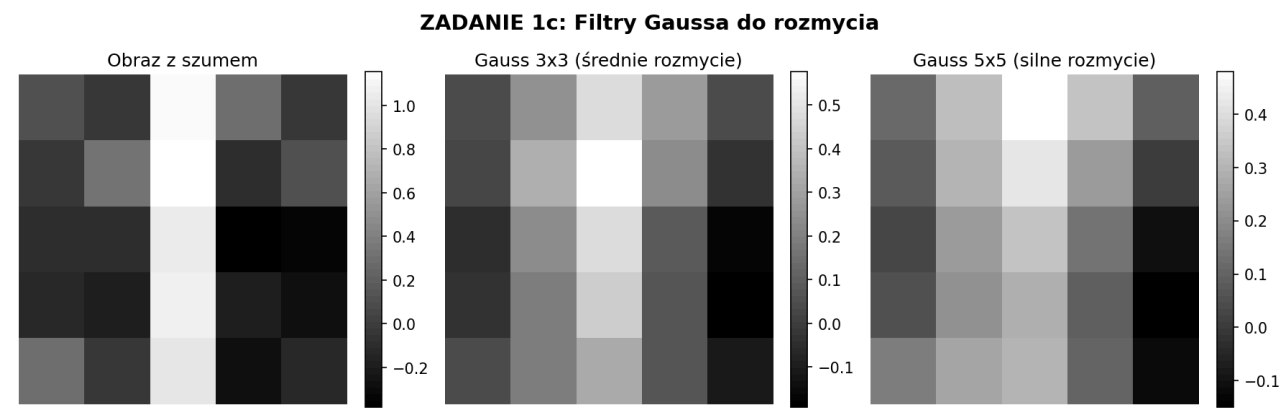
```
# Filtr wyostrzający - wzmacnia krawędzie
sharpening_filter = np.array([
    [ 0, -1,  0],
    [-1,  5, -1],
    [ 0, -1,  0]
])

# Silniejszy filtr wyostrzający
sharpening_filter_strong = np.array([
    [-1, -1, -1],
    [-1,  9, -1],
    [-1, -1, -1]
])

# Testowanie na rozmytym obrazie
blurred_image = np.array([
    [0.1, 0.2, 0.3, 0.2, 0.1],
    [0.2, 0.4, 0.6, 0.4, 0.2],
    [0.3, 0.6, 1.0, 0.6, 0.3],
    [0.2, 0.4, 0.6, 0.4, 0.2],
    [0.1, 0.2, 0.3, 0.2, 0.1]
], dtype=np.float32)

sharpened = convolution2d(blurred_image, sharpening_filter, padding=1)
sharpened_strong = convolution2d(blurred_image, sharpening_filter_strong, padding=1)

# Wizualizacja
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

im0 = axes[0].imshow(blurred_image, cmap='gray')
axes[0].set_title('Obraz rozmyty', fontsize=12)
axes[0].axis('off')
plt.colorbar(im0, ax=axes[0], fraction=0.046)

im1 = axes[1].imshow(sharpened, cmap='gray')
axes[1].set_title('Po wyostrzeniu (standard)', fontsize=12)
axes[1].axis('off')
plt.colorbar(im1, ax=axes[1], fraction=0.046)

im2 = axes[2].imshow(sharpened_strong, cmap='gray')
axes[2].set_title('Po wyostrzeniu (silny)', fontsize=12)
axes[2].axis('off')
plt.colorbar(im2, ax=axes[2], fraction=0.046)
```

Filtr skutecznie redukuje szum poprzez uśrednianie wartości pikseli z otoczeniem z wagami gaussowskimi.



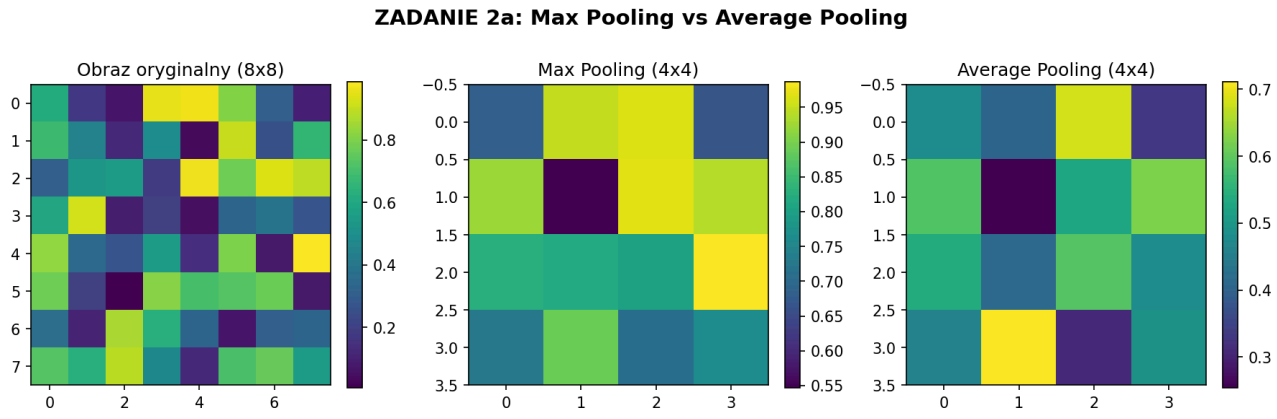
Zad. 1c - Filtr Gaussa do rozmycia

```
# Filtr Gaussa 3x3 (sigma ≈ 1)
gaussian_filter_3x3 = np.array([
    [1/16, 2/16, 1/16],
    [2/16, 4/16, 2/16],
    [1/16, 2/16, 1/16]
])

# Filtr Gaussa 5x5 (silniejsze rozmycie)
gaussian_filter_5x5 = np.array([
    [1/256,  4/256,  6/256,  4/256, 1/256],
    [4/256, 16/256, 24/256, 16/256, 4/256],
    [6/256, 24/256, 36/256, 24/256, 6/256],
    [4/256, 16/256, 24/256, 16/256, 4/256],
    [1/256,  4/256,  6/256,  4/256, 1/256]
])

# Testowanie na zaszumionym obrazie
np.random.seed(42)
noisy_image = image + np.random.normal(0, 0.2, image.shape)
blurred_3x3 = convolution2d(noisy_image.astype(np.float32), gaussian_filter_3x3, padding=1)

# Dla filtra 5x5 potrzebujemy większy padding
noisy_image_padded = np.pad(noisy_image, 2, mode='edge')
blurred_5x5 = convolution2d(noisy_image_padded.astype(np.float32), gaussian_filter_5x5)

# Wizualizacja
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

im0 = axes[0].imshow(noisy_image, cmap='gray')
axes[0].set_title('Obraz z szumem', fontsize=12)
axes[0].axis('off')
plt.colorbar(im0, ax=axes[0], fraction=0.046)

im1 = axes[1].imshow(blurred_3x3, cmap='gray')
axes[1].set_title('Gauss 3x3 (średnie rozmycie)', fontsize=12)
axes[1].axis('off')
plt.colorbar(im1, ax=axes[1], fraction=0.046)

im2 = axes[2].imshow(blurred_5x5, cmap='gray')
axes[2].set_title('Gauss 5x5 (silne rozmycie)', fontsize=12)
axes[2].axis('off')
plt.colorbar(im2, ax=axes[2], fraction=0.046)
```



Zad. 2a - Average Pooling

```
def average_pooling2d(image, pool_size=2, stride=2):
    img_height, img_width = image.shape

    # Wymiary wyjściowe
    out_height = (img_height - pool_size) // stride + 1
    out_width = (img_width - pool_size) // stride + 1

    # Inicjalizacja wyniku
    output = np.zeros((out_height, out_width))

    # Wykonaj average pooling
    for i in range(out_height):
        for j in range(out_width):
            h_start = i * stride
            h_end = h_start + pool_size
            w_start = j * stride
            w_end = w_start + pool_size

            # Oblicz średnią wartość z regionu (różnica od max pooling)
            output[i, j] = np.mean(image[h_start:h_end, w_start:w_end])

    return output

print("✅ Funkcja average_pooling2d zaimplementowana")

# Testowanie i porównanie z Max Pooling
test_image = np.random.rand(8, 8)
avg_pooled = average_pooling2d(test_image, pool_size=2, stride=2)
max_pooled = max_pooling2d(test_image, pool_size=2, stride=2)

print(f"\n📊 Wymiary po Average Pooling: {avg_pooled.shape}")
print(f"📊 Różnica między Max a Average: {np.mean(np.abs(max_pooled - avg_pooled)):.4f}")

# Wizualizacja porównania
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

im0 = axes[0].imshow(test_image, cmap='viridis')
axes[0].set_title('Obraz oryginalny (8x8)', fontsize=12)
plt.colorbar(im0, ax=axes[0], fraction=0.046)

im1 = axes[1].imshow(max_pooled, cmap='viridis')
axes[1].set_title('Max Pooling (4x4)', fontsize=12)
plt.colorbar(im1, ax=axes[1], fraction=0.046)

im2 = axes[2].imshow(avg_pooled, cmap='viridis')
axes[2].set_title('Average Pooling (4x4)', fontsize=12)
plt.colorbar(im2, ax=axes[2], fraction=0.046)
```



Wymiary po Average Pooling: (4, 4). Różnica między Max a Average: 0.3303

Zad. 3a - Analiza zmian rozmiaru i parametrów

```
# Symulacja przepływu obrazu 28x28 przez CNN
input_size = 28

# Warstwa Conv 1: 3x3, stride=1, padding=1
size_after_conv1 = (input_size - 3 + 2*1) // 1 + 1
print(f"Input: {input_size}x{input_size}")
print(f"Conv1 (3x3, s=1, p=1): {size_after_conv1}x{size_after_conv1}")

# MaxPool 1: 2x2, stride=2
size_after_pool1 = (size_after_conv1 - 2) // 2 + 1
print(f"MaxPool1 (2x2, s=2): {size_after_pool1}x{size_after_pool1}")

# Warstwa Conv 2: 3x3, stride=1, padding=0
size_after_conv2 = (size_after_pool1 - 3) // 1 + 1
print(f"Conv2 (3x3, s=1, p=0): {size_after_conv2}x{size_after_conv2}")

# MaxPool 2: 2x2, stride=2
size_after_pool2 = (size_after_conv2 - 2) // 2 + 1
print(f"MaxPool2 (2x2, s=2): {size_after_pool2}x{size_after_pool2}")
```

```
# Output:
Input: 28x28
Conv1 (3x3, s=1, p=1): 28x28
MaxPool1 (2x2, s=2): 14x14
Conv2 (3x3, s=1, p=0): 12x12
MaxPool2 (2x2, s=2): 6x6
```

Zad. 3b - Liczba parametrów dla różnych rozmiarów filtrów

```
def count_parameters(kernel_size, input_channels, output_channels):
    """Oblicza liczbę parametrów w warstwie konwolucyjnej"""
    weights = kernel_size * kernel_size * input_channels * output_channels
    biases = output_channels
    total = weights + biases
    return total

# Przykład: warstwa konwolucyjna RGB -> 64 filtrów
input_ch = 3  # RGB
output_ch = 64

print("\n=== LICZBA PARAMETRÓW ===")
print(f"Warstwa: {input_ch} kanały wejściowe → {output_ch} kanały wyjściowe\n")
for k_size in [3, 5, 7, 11]:
    params = count_parameters(k_size, input_ch, output_ch)
    print(f"Filtr {k_size}x{k_size}: {params:,} parametrów")
```

```
#Output
Warstwa: 3 kanały wejściowe → 64 kanały wyjściowe

Filtr 3x3: 1,792 parametrów
Filtr 5x5: 4,864 parametrów
Filtr 7x7: 9,472 parametrów
Filtr 11x11: 23,296 parametrów
```

Zad. 3c - Porównanie Max Pooling VS Average Pooling

```
# Test na różnych typach obrazów
test_cases = {
    'Ostre krawędzie': np.array([
        [0, 0, 1, 1],
        [0, 0, 1, 1],
        [1, 1, 0, 0],
        [1, 1, 0, 0]
    ], dtype=np.float32),

    'Gradient': np.array([
        [0.1, 0.2, 0.3, 0.4],
        [0.2, 0.3, 0.4, 0.5],
        [0.3, 0.4, 0.5, 0.6],
        [0.4, 0.5, 0.6, 0.7]
    ], dtype=np.float32),

    'Szpilki (sparsity)': np.array([
        [0, 0, 0, 9],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [8, 0, 0, 0]
    ], dtype=np.float32)
}

print("\n=== PORÓWNANIE NA RÓŻNYCH OBRAZACH ===")
for name, img in test_cases.items():
    max_p = max_pooling2d(img, pool_size=2, stride=2)
    avg_p = average_pooling2d(img, pool_size=2, stride=2)
```

```
# Output
=== PORÓWNANIE NA RÓŻNYCH OBRAZACH ===
Ostre krawędzie:
  Max Pooling: [0. 1. 1. 0.]
  Avg Pooling: [0. 1. 1. 0.]
  Różnica: 0.000

Gradient:
  Max Pooling: [0.30000001 0.5        0.5        0.69999999]
  Avg Pooling: [0.2        0.40000001 0.40000001 0.60000002]
  Różnica: 0.100
  Max Pooling: [0. 1. 1. 0.]
  Avg Pooling: [0. 1. 1. 0.]
  Różnica: 0.000

Gradient:
  Max Pooling: [0.30000001 0.5        0.5        0.69999999]
  Avg Pooling: [0.2        0.40000001 0.40000001 0.60000002]
  Różnica: 0.100
  Różnica: 0.000

Gradient:
  Max Pooling: [0.30000001 0.5        0.5        0.69999999]
  Avg Pooling: [0.2        0.40000001 0.40000001 0.60000002]
  Różnica: 0.100
Gradient:
  Max Pooling: [0.30000001 0.5        0.5        0.69999999]
  Avg Pooling: [0.2        0.40000001 0.40000001 0.60000002]
  Różnica: 0.100
  Max Pooling: [0.30000001 0.5        0.5        0.69999999]
  Avg Pooling: [0.2        0.40000001 0.40000001 0.60000002]
  Różnica: 0.100
  Różnica: 0.100

Szpilki (sparsity):
  Max Pooling: [0. 9. 8. 0.]
```

```
Szpilki (sparsity):
  Max Pooling: [0. 9. 8. 0.]
Szpilki (sparsity):
  Max Pooling: [0. 9. 8. 0.]
  Avg Pooling: [0.   2.25 2.   0.  ]
  Różnica: 3.188
  Avg Pooling: [0.   2.25 2.   0.  ]
  Różnica: 3.188
  Różnica: 3.188
```

Zad. 4a - Testowanie filtrów w różnych cyfrach

```
# Definicje różnych cyfr 6x6
digits = {
    '0': np.array([
        [0, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 1, 1, 1, 1, 0]
    ]),
    '1': np.array([
        [0, 0, 1, 0, 0, 0],
        [0, 1, 1, 0, 0, 0],
        [0, 0, 1, 0, 0, 0],
        [0, 0, 1, 0, 0, 0],
        [0, 0, 1, 0, 0, 0],
        [0, 1, 1, 1, 0, 0]
    ]),
    '2': np.array([
        [0, 1, 1, 1, 0, 0],
        [1, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 0]
    ]),
    '3': np.array([
        [1, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 1, 1, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [1, 1, 1, 1, 0, 0]
    ]),
    '7': np.array([
        [1, 1, 1, 1, 1, 1],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 1, 0, 0],
        [0, 0, 1, 0, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0]
    ]),
}


# Testowanie filtrów na różnych cyfrach
vertical_filter = np.array([[0, 1, 0], [0, 1, 0], [0, 1, 0]])

print("\n=== REAKCJA FILTRA PIONOWEGO NA CYFRY ===")
for digit_name, digit_img in digits.items():
    result = convolution2d(digit_img.astype(np.float32), vertical_filter)
    activation_strength = np.max(result)
```

```
    print(f"Cyfra {digit_name}: max aktywacja = {activation_strength:.2f}")

# Wizualizacja
fig, axes = plt.subplots(2, len(digits), figsize=(15, 6))

for idx, (digit_name, digit_img) in enumerate(digits.items()):
    # Górny wiersz - cyfry
    axes[0, idx].imshow(digit_img, cmap='gray')
    axes[0, idx].set_title(f'Cyfra {digit_name}', fontsize=11)
    axes[0, idx].axis('off')

    # Dolny wiersz - mapy aktywacji
    result = convolution2d(digit_img.astype(np.float32), vertical_filter)
    axes[1, idx].imshow(result, cmap='hot')
    axes[1, idx].set_title(f'Aktywacja:\n{np.max(result):.2f}', fontsize=10)
    axes[1, idx].axis('off')
```

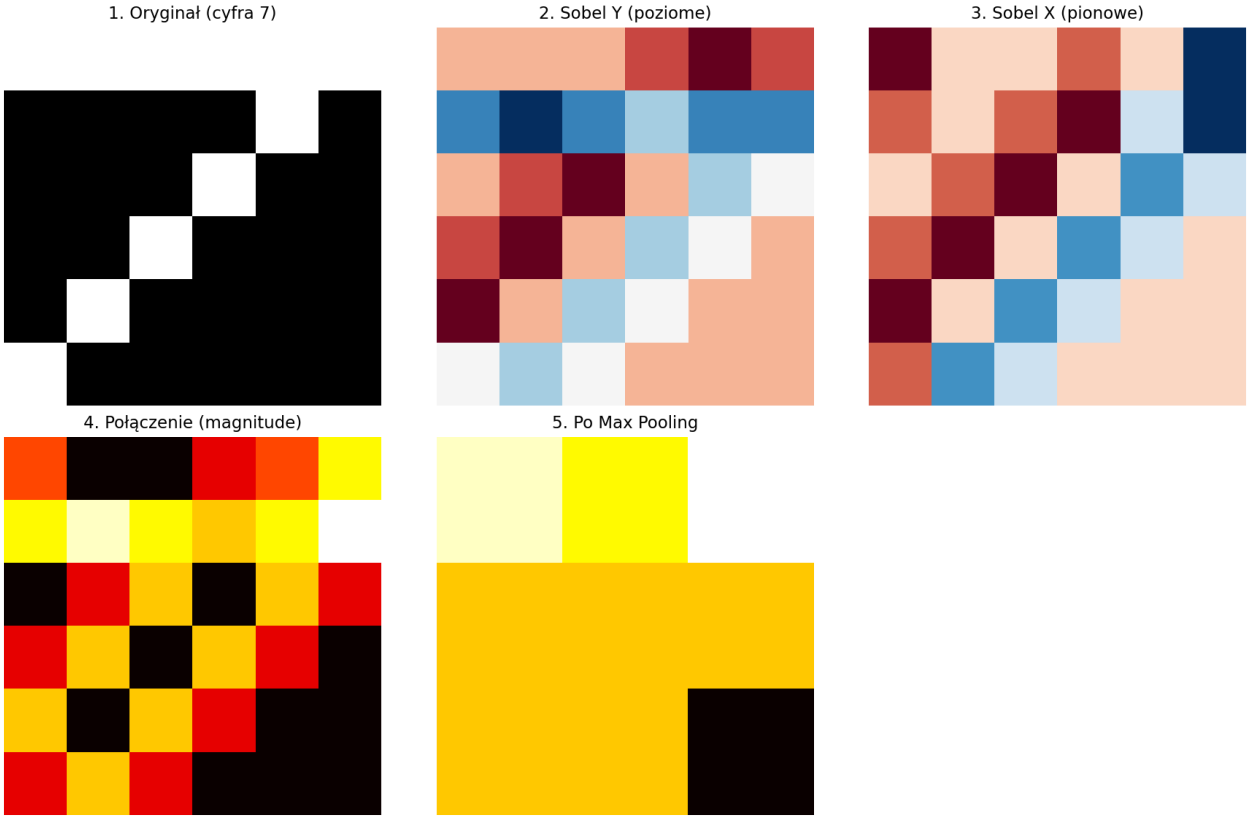
```
#output
=== REAKCJA FILTRA PIONOWEGO NA CYFRY ===
Cyfra 0: max aktywacja = 1.00
Cyfra 1: max aktywacja = 3.00
Cyfra 2: max aktywacja = 2.00
Cyfra 3: max aktywacja = 3.00
Cyfra 7: max aktywacja = 2.00
```

Zad. 4b - Sekwencyjne zastosowanie filtrów

```
# Pipeline filtrów
image_original = digits['7'].astype(np.float32)

# Krok 1: Wykrywanie krawędzi poziomych
sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
edges_h = convolution2d(image_original, sobel_y, padding=1)

# Krok 2: Wykrywanie krawędzi pionowych
sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
edges_v = convolution2d(image_original, sobel_x, padding=1)

# Krok 3: Połączenie (magnitude)
edges_combined = np.sqrt(edges_h**2 + edges_v**2)

# Krok 4: Pooling
edges_pooled = max_pooling2d(edges_combined, pool_size=2, stride=2)
```

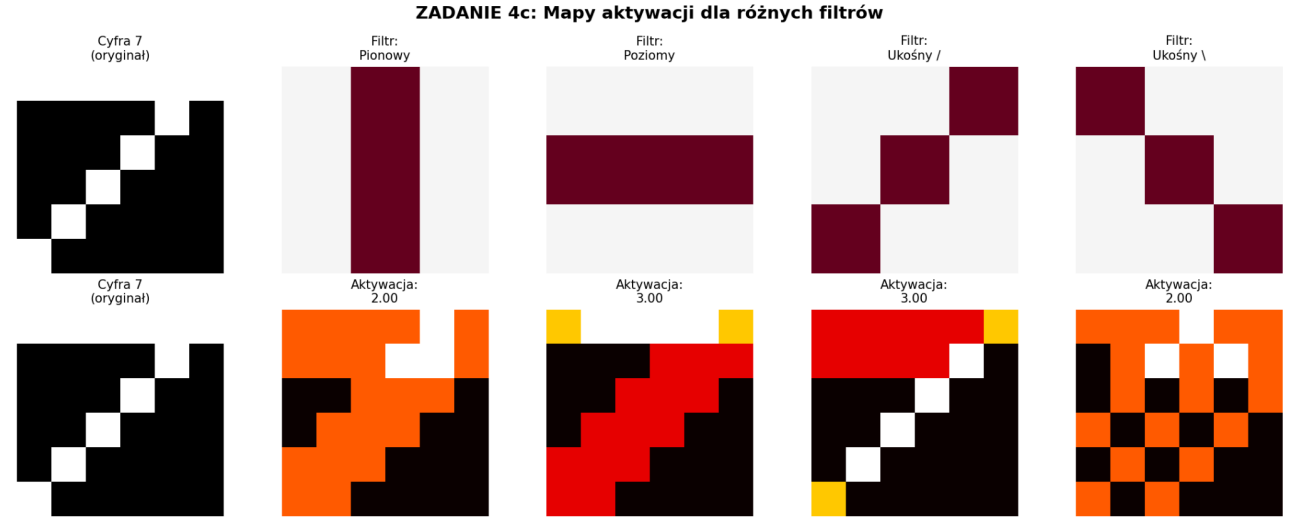
```
# Wizualizacja
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

axes[0, 0].imshow(image_original, cmap='gray')
axes[0, 0].set_title('1. Oryginał (cyfra 7)', fontsize=11)
axes[0, 0].axis('off')

axes[0, 1].imshow(edges_h, cmap='RdBu_r')
axes[0, 1].set_title('2. Sobel Y (poziome)', fontsize=11)
axes[0, 1].axis('off')

axes[0, 2].imshow(edges_v, cmap='RdBu_r')
axes[0, 2].set_title('3. Sobel X (pionowe)', fontsize=11)
axes[0, 2].axis('off')

axes[1, 0].imshow(edges_combined, cmap='hot')
axes[1, 0].set_title('4. Połączenie (magnitude)', fontsize=11)
axes[1, 0].axis('off')

axes[1, 1].imshow(edges_pooled, cmap='hot')
axes[1, 1].set_title('5. Po Max Pooling', fontsize=11)
axes[1, 1].axis('off')

axes[1, 2].axis('off')
```

ZADANIE 4b: Pipeline sekwencyjnych filtrów



Zad. 4c - Wizualizacja map aktywacji

```
# Różne filtry do testowania
filters_test = {
    'Pionowy': np.array([[0, 1, 0], [0, 1, 0], [0, 1, 0]]),
    'Poziomy': np.array([[0, 0, 0], [1, 1, 1], [0, 0, 0]]),
    'Ukośny /': np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0]]),
    'Ukośny \\': np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]),
}

# Testuj na cyfrze "7"
digit_test = digits['7'].astype(np.float32)

print("\n=== MAPY AKTYWACJI DLA CYFRY 7 ===")
for filter_name, filt in filters_test.items():
    activation_map = convolution2d(digit_test, filt, padding=1)
    print(f"\nFiltr '{filter_name}':")
    print(f"  Max aktywacja: {np.max(activation_map):.2f}")
    print(f"  Min aktywacja: {np.min(activation_map):.2f}")
    print(f"  Średnia: {np.mean(activation_map):.2f}")

# Wizualizacja wszystkich map aktywacji
fig, axes = plt.subplots(2, 5, figsize=(15, 6))

# Obraz oryginalny
axes[0, 0].imshow(digit_test, cmap='gray')
axes[0, 0].set_title('Cyfra 7\n(oryginał)', fontsize=10)
axes[0, 0].axis('off')

# Filtry
for idx, (filter_name, filt) in enumerate(filters_test.items(), 1):
    axes[0, idx].imshow(filt, cmap='RdBu_r', vmin=-1, vmax=1)
    axes[0, idx].set_title(f'Filtr:\n{filter_name}', fontsize=10)
    axes[0, idx].axis('off')

# Obraz oryginalny (powtórzenie)
axes[1, 0].imshow(digit_test, cmap='gray')
axes[1, 0].set_title('Cyfra 7\n(oryginał)', fontsize=10)
axes[1, 0].axis('off')

# Mapy aktywacji
for idx, (filter_name, filt) in enumerate(filters_test.items(), 1):
    activation_map = convolution2d(digit_test, filt, padding=1)
    axes[1, idx].imshow(activation_map, cmap='hot')
    axes[1, idx].set_title(f'Aktywacja:\n{np.max(activation_map):.2f}', fontsize=10)
    axes[1, idx].axis('off')
```



Zadanie 5 - Odpowiedzi na pytania

Dlaczego używamy małych filtrów (3x3) zamiast dużych?

Małe filtry 3x3 są znacznie bardziej efektywne obliczeniowo. Trzy warstwy z filtrami 3x3 mają razem tylko 27 parametrów (3×9), podczas gdy jeden filtr 7x7 ma 49 parametrów, a przy tym dają takie samo receptive field. Dodatkowo każda warstwa dodaje nieliniowość przez funkcję aktywacji, co pozwala sieci uczyć się bardziej złożonych wzorców. Mniejsze filtry pozwalają też budować głębsze sieci przy tej samej ilości pamięci.

Jak padding wpływa na rozmiar wyjścia?

Padding kontroluje jak bardzo zmniejsza się obraz po splotu. Bez paddingu obraz zawsze traci piksele na brzegach, np. dla filtra 3x3 obraz 28x28 staje się 26x26. Z padding=1 obraz zachowuje swój rozmiar. Dzięki paddingowi możemy precyzyjnie kontrolować wymiary na każdym etapie sieci.

Kiedy użyć stride > 1?

Stride > 1 używamy gdy chcemy zmniejszyć rozmiar mapy cech bez osobnej warstwy poolingu. Stride=2 zmniejsza obraz dwukrotnie, podobnie jak MaxPooling 2x2, ale jednocześnie wykonuje splot, co jest bardziej efektywne.

Jakie są wady Max Poolingu?

Największa wada to utrata informacji przy poolingu 2x2 tracimy 75% danych. To może być problematyczne gdy dokładna lokalizacja jest ważna (np. w segmentacji medycznej). Max Pooling jest też nieodwracalny, więc nie da się odtworzyć oryginalnego obrazu, co szkodzi w autoenkoderach. Wybiera tylko maksymalną wartość z regionu, ignorując resztę informacji, dlatego czasem lepiej działa Average Pooling który uwzględnia wszystkie wartości.