Project Title

An Empirical Study of Data Augmentation Effects on Average Pooling Behavior in Convolutional Neural Networks

Project Objective

The objective of this project is to systematically analyze how common data augmentation techniques affect the output of average pooling in Convolutional Neural Networks (CNNs). The study aims to evaluate whether average pooling preserves consistency under geometric, photometric, and content-based transformations such as rotation, mirroring, scaling, cropping, and brightness variation. By comparing the pooled outputs of augmented images with the original image, the project highlights the limitations of average pooling and emphasizes the role of data augmentation in improving CNN robustness.

Question 1: What is Data Augmentation?

Answer: Data augmentation is a technique used in machine learning, especially in computer vision, where additional training data is generated by applying transformations to existing data. These transformations include operations such as rotation, flipping, scaling, cropping, and brightness adjustment. The goal of data augmentation is to increase the diversity of the training dataset without collecting new data.

Question 2: How does Data Augmentation help in CNNs?

Answer: Data augmentation helps Convolutional Neural Networks (CNNs) by exposing the model to different variations of the same image during training. This allows the CNN to learn more robust and generalized features instead of memorizing specific patterns. As a result, data augmentation reduces overfitting, improves model generalization, and increases accuracy on unseen data.

Question 3: Practical Applications of Data Augmentation

Answer: Data augmentation is widely used in real-world applications where collecting large labeled datasets is difficult or expensive. Common applications include medical image analysis (X-rays, MRI scans), face recognition systems, autonomous driving (traffic signs and road scenes), satellite image analysis, and handwritten character recognition. In these applications, data augmentation

improves model reliability under varying real-world conditions.

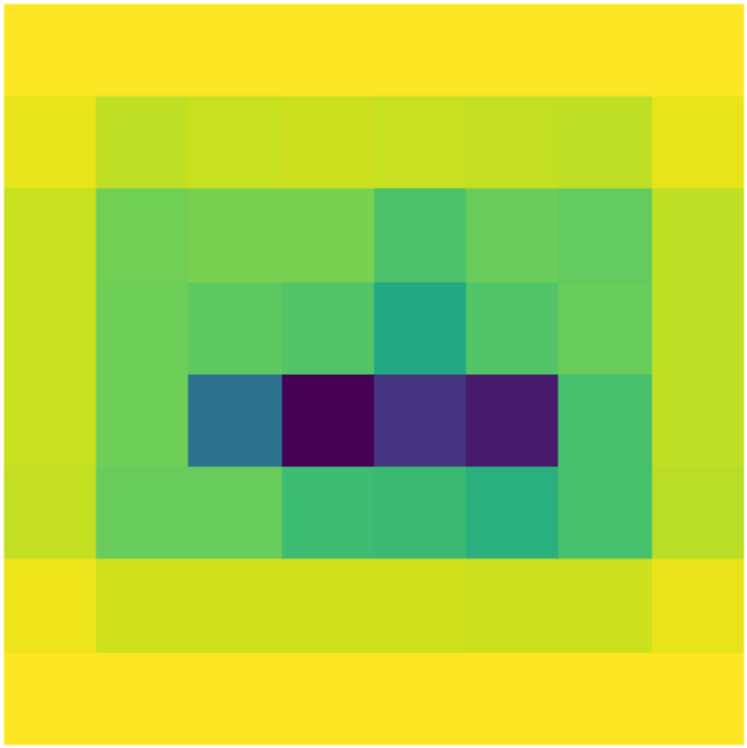
Question 4: Advantages and Limitations of Data Augmentation

Advantages Increases training data size without additional data collection Reduces overfitting and improves generalization Makes CNN models more robust to real-world variations Improves performance on unseen or noisy data Limitations Cannot replace genuinely diverse real-world data Excessive or unrealistic augmentation may degrade model performance Increases training time and computational cost Some transformations may alter important semantic information

In [10]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read image (change path if needed)
img = cv2.imread('Fresh cherries on turquoise backdrop.png')

# Convert BGR to RGB for correct display
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Convert to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Resize image to 8x8
img_8x8 = cv2.resize(img_gray, (8, 8), interpolation=cv2.INTER_AREA)

# Store 8x8 matrix in a variable
image_matrix_8x8 = img_8x8

# Display original image
plt.imshow(img_rgb)
plt.title("Original Image")
plt.axis("off")
plt.show()

# Display 8x8 grayscale image
plt.imshow(image_matrix_8x8)
plt.title("8x8 Grayscale Image")
plt.axis("off")
plt.show()

# Print the 8x8 matrix
print("8x8 Image Matrix:\n", image_matrix_8x8)
```

Original Image

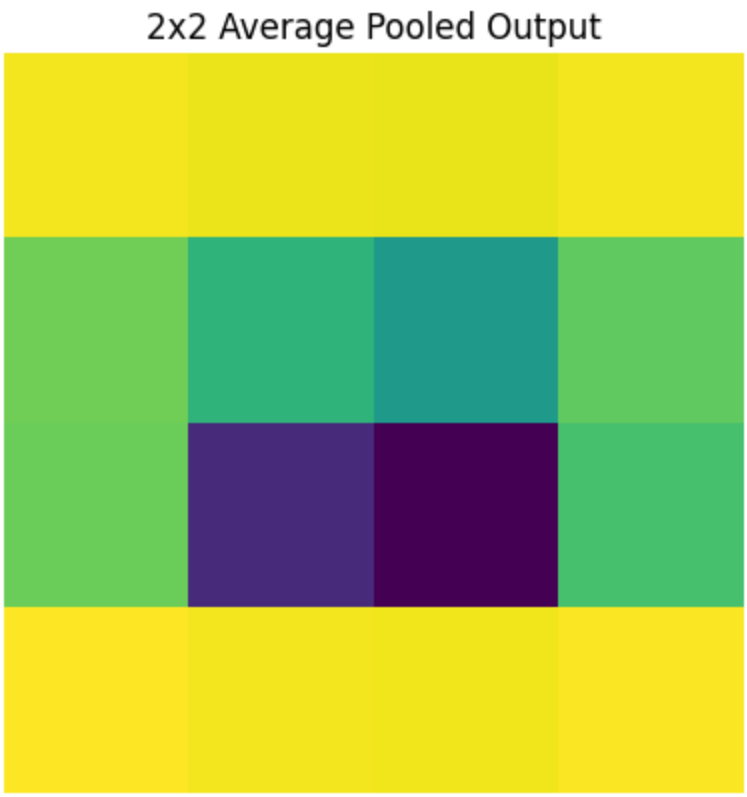
8x8 Grayscale Image



```
8x8 Image Matrix:
 [[254 254 254 254 254 254 254 254]
 [248 239 241 242 241 240 239 248]
 [241 221 223 223 211 219 217 239]
 [241 220 216 212 193 212 218 239]
 [241 220 158 100 124 111 209 239]
 [240 218 218 206 205 198 209 238]
 [250 243 243 243 243 242 242 249]
 [254 254 254 254 254 254 254 254]]
```

In [9]:

```python
# Average Pooling function
def average_pooling(matrix, pool_size=2, stride=2):
    h, w = matrix.shape
    pooled_h = h // stride
    pooled_w = w // stride

    pooled = np.zeros((pooled_h, pooled_w))

    for i in range(0, h, stride):
        for j in range(0, w, stride):
            window = matrix[i:i+pool_size, j:j+pool_size]
            pooled[i//stride, j//stride] = np.mean(window)

    return pooled

# Apply average pooling
avg_pooled_matrix = average_pooling(image_matrix_8x8)

# Display pooled result
plt.imshow(avg_pooled_matrix)
plt.title("2x2 Average Pooled Output")
plt.axis("off")
plt.show()

# Print pooled matrix
print("Average Pooled Matrix:\n", avg_pooled_matrix)
```

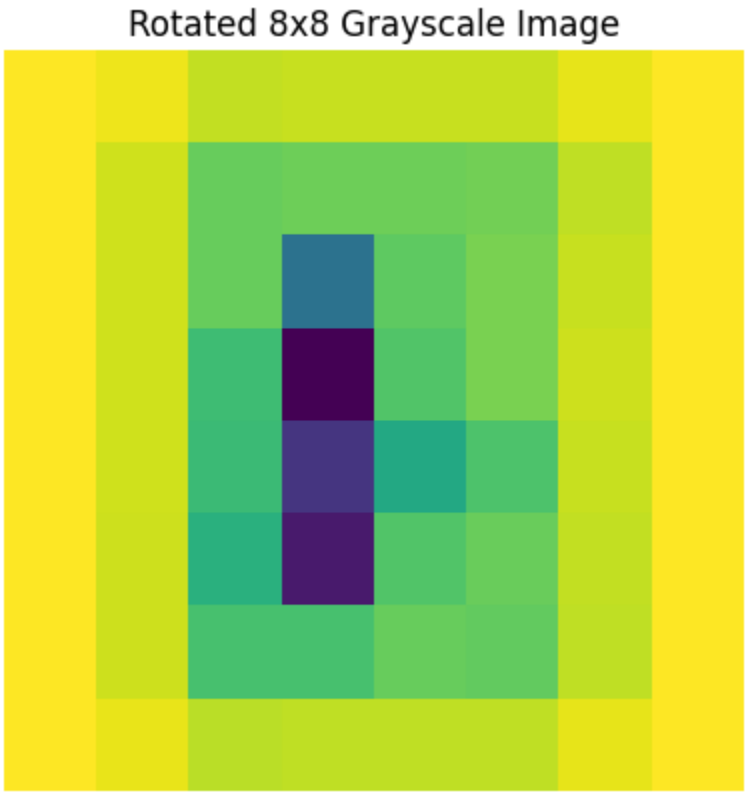


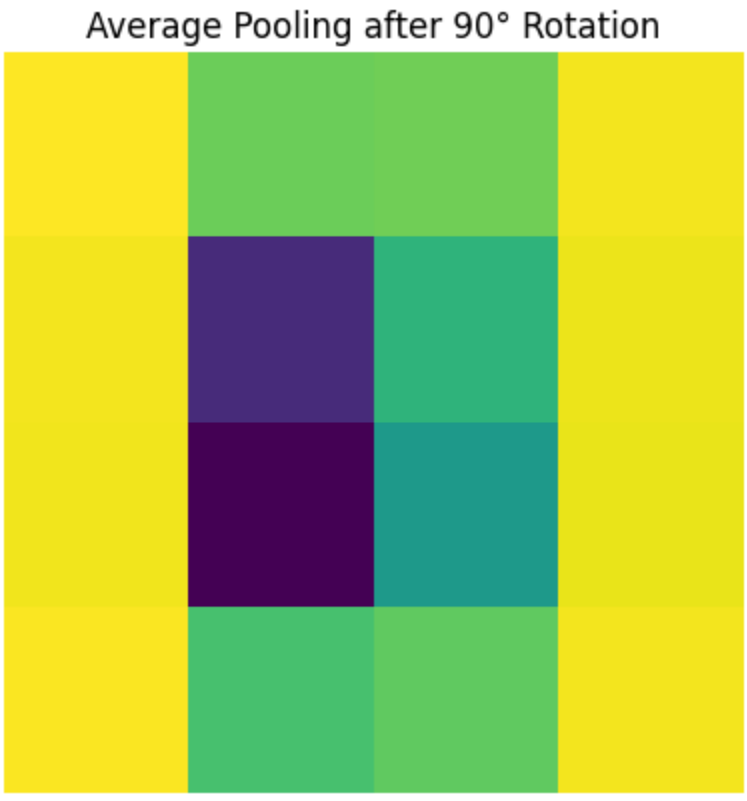

```
Average Pooled Matrix:
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]
```

In [8]:

```python
# Rotate original grayscale image by 90 degrees clockwise
rotated_img = cv2.rotate(img_gray, cv2.ROTATE_90_CLOCKWISE)

# Resize rotated image to 8x8
rotated_8x8 = cv2.resize(rotated_img, (8, 8), interpolation=cv2.INTER_AREA)

# Store rotated 8x8 matrix
rotated_image_matrix_8x8 = rotated_8x8

# Apply average pooling on rotated image
rotated_avg_pooled_matrix = average_pooling(rotated_image_matrix_8x8)

# Display rotated 8x8 image
plt.imshow(rotated_image_matrix_8x8)
plt.title("Rotated 8x8 Grayscale Image")
plt.axis("off")
plt.show()

# Display pooled output
plt.imshow(rotated_avg_pooled_matrix)
plt.title("Average Pooling after 90° Rotation")
plt.axis("off")
plt.show()
```

```
# Print matrices
print("Rotated 8x8 Matrix:\n", rotated_image_matrix_8x8)
print("\nAverage Pooled Matrix (Rotated):\n", rotated_avg_pooled_matrix)
```





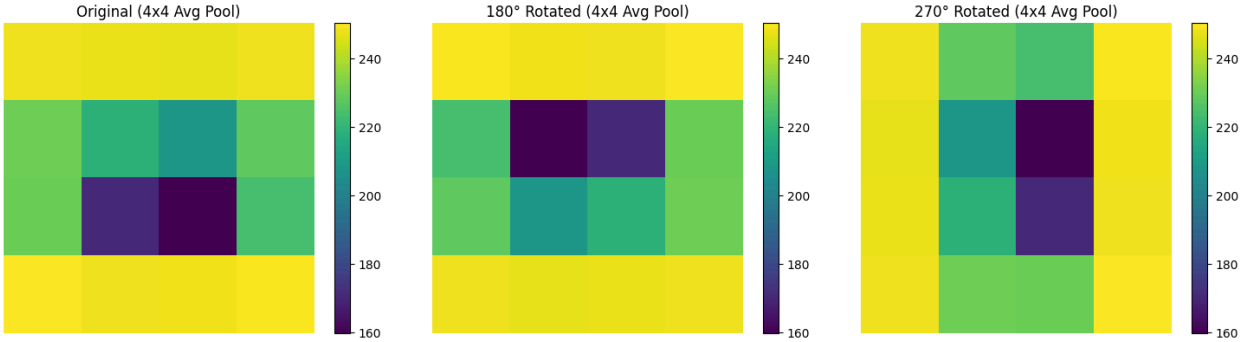
```
Rotated 8x8 Matrix:
 [[254 250 240 241 241 241 248 254]
 [254 243 218 220 220 221 239 254]
 [254 243 218 158 216 223 241 254]
 [254 243 206 100 212 223 242 254]
 [254 243 205 124 193 211 241 254]
 [254 242 198 111 212 219 240 254]
 [254 242 209 209 218 217 239 254]
 [254 249 238 239 239 239 248 254]]

Average Pooled Matrix (Rotated):
 [[250.25 229.75 230.75 248.75]
 [248.5  170.5  218.5  247.75]
 [248.25 159.5  208.75 247.25]
 [249.75 223.75 228.25 248.75]]
```

In [21]:
```python
# Rotate original grayscale image by 180 degrees
rotated_180_img = cv2.rotate(img_gray, cv2.ROTATE_180)

# Resize to 8x8
rotated_180_8x8 = cv2.resize(rotated_180_img, (8, 8), interpolation=cv2.INTER_

# Store matrix
rotated_180_matrix_8x8 = rotated_180_8x8

# Apply average pooling
rotated_180_avg_pooled_matrix = average_pooling(rotated_180_matrix_8x8)
```

In [22]:
```python
# Rotate original grayscale image by 270 degrees clockwise
rotated_270_img = cv2.rotate(img_gray, cv2.ROTATE_90_COUNTERCLOCKWISE)

# Resize to 8x8
rotated_270_8x8 = cv2.resize(rotated_270_img, (8, 8), interpolation=cv2.INTER_

# Store matrix
rotated_270_matrix_8x8 = rotated_270_8x8

# Apply average pooling
rotated_270_avg_pooled_matrix = average_pooling(rotated_270_matrix_8x8)
```

In [23]:
```python
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(15, 4))

# Original pooled (Cell 2)
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original (4x4 Avg Pool)")
plt.colorbar()
plt.axis("off")

# 180° pooled
```

```
plt.subplot(1, 3, 2)
plt.imshow(rotated_180_avg_pooled_matrix, cmap="viridis")
plt.title("180° Rotated (4x4 Avg Pool)")
plt.colorbar()
plt.axis("off")

# 270° pooled
plt.subplot(1, 3, 3)
plt.imshow(rotated_270_avg_pooled_matrix, cmap="viridis")
plt.title("270° Rotated (4x4 Avg Pool)")
plt.colorbar()
plt.axis("off")

plt.tight_layout()
plt.show()

# Numerical verification
print("Original Pooled Matrix (Cell 2):\n", avg_pooled_matrix)
print("\n180° Rotated Pooled Matrix:\n", rotated_180_avg_pooled_matrix)
print("\n270° Rotated Pooled Matrix:\n", rotated_270_avg_pooled_matrix)
```



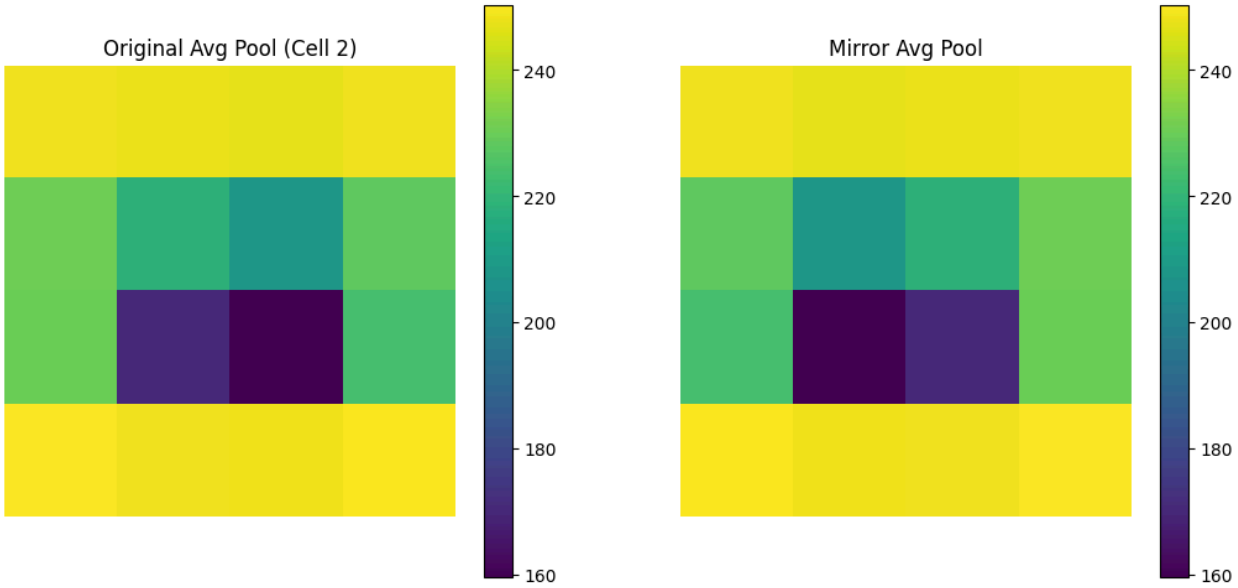

```
Original Pooled Matrix (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

180° Rotated Pooled Matrix:
 [[249.75 248.25 248.5  250.25]
 [223.75 159.5  170.5  229.75]
 [228.25 208.75 218.5  230.75]
 [248.75 247.25 247.75 248.75]]

270° Rotated Pooled Matrix:
 [[248.75 228.25 223.75 249.75]
 [247.25 208.75 159.5  248.25]
 [247.75 218.5  170.5  248.5 ]
 [248.75 230.75 229.75 250.25]]
```

In [31]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```python
# Create mirror image (horizontal flip)
mirror_img = cv2.flip(img_gray, 1)   # 1 = horizontal flip

# Resize to 8x8
mirror_8x8 = cv2.resize(mirror_img, (8, 8), interpolation=cv2.INTER_AREA)

# Apply average pooling
mirror_avg_pooled_matrix = average_pooling(mirror_8x8)

# ---------- Visualization ----------
plt.figure(figsize=(20, 6))

# Cell 2 output
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original Avg Pool (Cell 2)")
plt.colorbar()
plt.axis("off")

# Mirror pooled output
plt.subplot(1, 3, 2)
plt.imshow(mirror_avg_pooled_matrix, cmap="viridis")
plt.title("Mirror Avg Pool")
plt.colorbar()
plt.axis("off")

# ---------- Numerical verification ----------
print("Original Avg Pool (Cell 2):\n", avg_pooled_matrix)
print("\nMirror Avg Pool:\n", mirror_avg_pooled_matrix)
```

```
Original Avg Pool (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

Mirror Avg Pool:
 [[248.75 247.25 247.75 248.75]
 [228.25 208.75 218.5  230.75]
 [223.75 159.5  170.5  229.75]
 [249.75 248.25 248.5  250.25]]
```

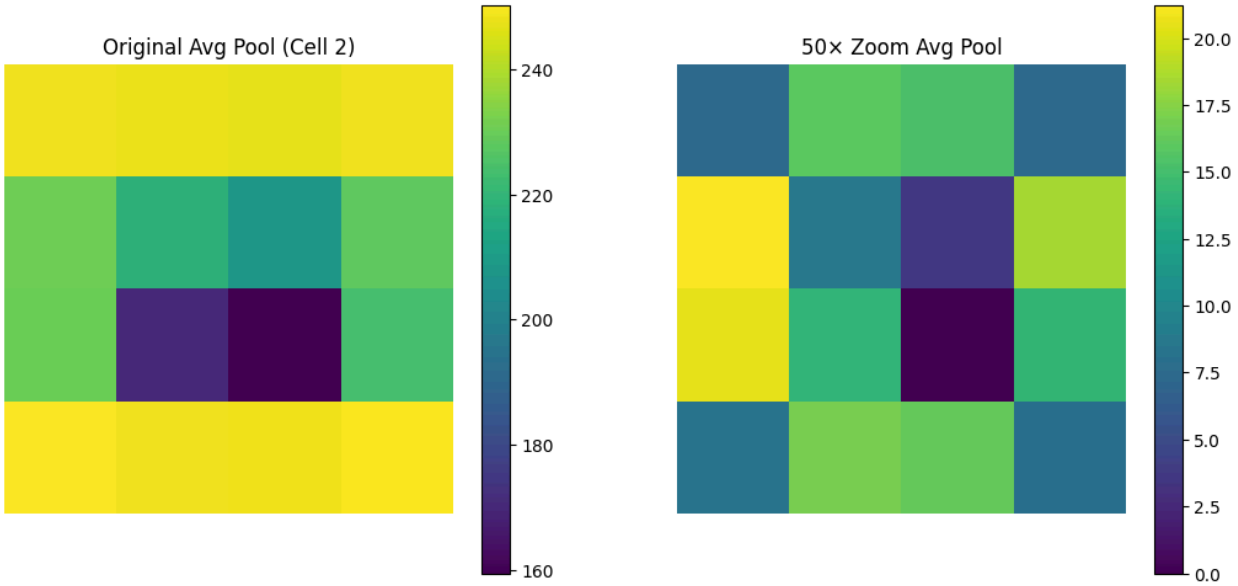


In [34]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# -------- Zoom image by 50x --------
zoom_factor = 50

h, w = img_gray.shape
zoomed_img = cv2.resize(
    img_gray,
    (w * zoom_factor, h * zoom_factor),
    interpolation=cv2.INTER_CUBIC
)

# Resize zoomed image back to 8x8
zoomed_8x8 = cv2.resize(zoomed_img, (8, 8), interpolation=cv2.INTER_AREA)

# Apply average pooling
zoomed_avg_pooled_matrix = average_pooling(zoomed_8x8)

# -------- Visualization --------
plt.figure(figsize=(20, 6))

# Cell 2 output
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original Avg Pool (Cell 2)")
plt.colorbar()
plt.axis("off")

# Zoomed pooled output
plt.subplot(1, 3, 2)
plt.imshow(zoomed_avg_pooled_matrix, cmap="viridis")
plt.title("50× Zoom Avg Pool")
plt.colorbar()
```

```
plt.axis("off")

# -------- Numerical verification --------
print("Original Avg Pool (Cell 2):\n", avg_pooled_matrix)
print("\n50× Zoom Avg Pool:\n", zoomed_avg_pooled_matrix)
```

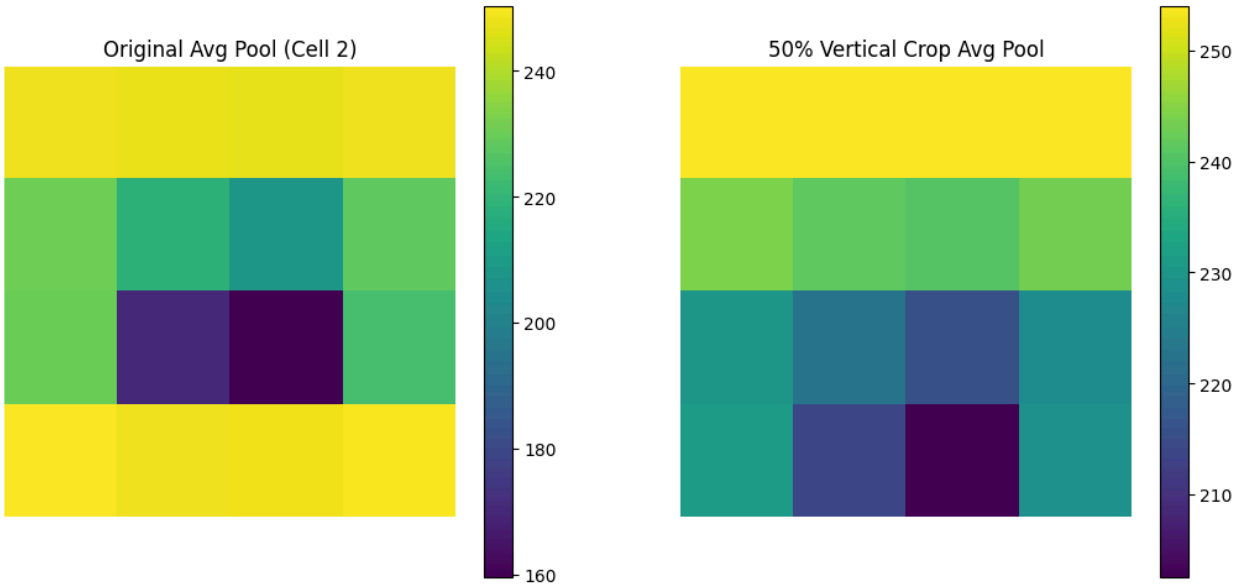
```
Original Avg Pool (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

50× Zoom Avg Pool:
 [[ 7.5  16.   15.25  7.25]
 [21.25  8.75  3.5  18.5 ]
 [20.5  14.    0.   14.25]
 [ 8.25 17.   16.25  8.  ]]
```



In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# -------- Crop 50% vertically (top half) --------
h, w = img_gray.shape
cropped_img = img_gray[:h//2, :]   # top 50%

# Resize cropped image to 8x8
cropped_8x8 = cv2.resize(cropped_img, (8, 8), interpolation=cv2.INTER_AREA)

# Apply average pooling
cropped_avg_pooled_matrix = average_pooling(cropped_8x8)

# -------- Visualization --------
plt.figure(figsize=(20, 6))

# Cell 2 output
```

```
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original Avg Pool (Cell 2)")
plt.colorbar()
plt.axis("off")

# Cropped pooled output
plt.subplot(1, 3, 2)
plt.imshow(cropped_avg_pooled_matrix, cmap="viridis")
plt.title("50% Vertical Crop Avg Pool")
plt.colorbar()
plt.axis("off")


# -------- Numerical verification --------
print("Original Avg Pool (Cell 2):\n", avg_pooled_matrix)
print("\n50% Vertical Crop Avg Pool:\n", cropped_avg_pooled_matrix)
```

```
Original Avg Pool (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

50% Vertical Crop Avg Pool:
 [[254.   254.   254.   254.  ]
 [244.   241.5  240.5  243.5 ]
 [230.5  222.75 215.5  228.  ]
 [231.   214.   202.5  228.5 ]]
```

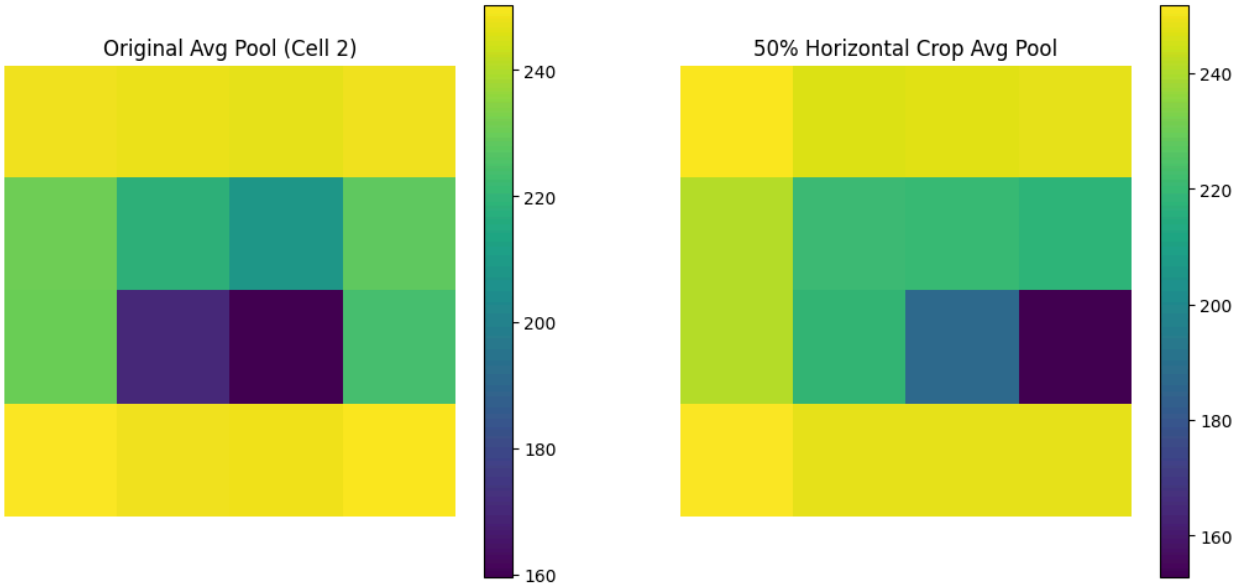


In [41]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# -------- Crop 50% horizontally (left half) --------
h, w = img_gray.shape
```

```
cropped_horiz_img = img_gray[:, :w//2]   # left 50%

# Resize cropped image to 8x8
cropped_horiz_8x8 = cv2.resize(cropped_horiz_img, (8, 8), interpolation=cv2.IN

# Apply average pooling
cropped_horiz_avg_pooled_matrix = average_pooling(cropped_horiz_8x8)

# -------- Visualization --------
plt.figure(figsize=(20, 6))

# Cell 2 output
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original Avg Pool (Cell 2)")
plt.colorbar()
plt.axis("off")

# Horizontally cropped pooled output
plt.subplot(1, 3, 2)
plt.imshow(cropped_horiz_avg_pooled_matrix, cmap="viridis")
plt.title("50% Horizontal Crop Avg Pool")
plt.colorbar()
plt.axis("off")


# -------- Numerical verification
print("Original Avg Pool (Cell 2):\n", avg_pooled_matrix)
print("\n50% Horizontal Crop Avg Pool:\n", cropped_horiz_avg_pooled_matrix)
```

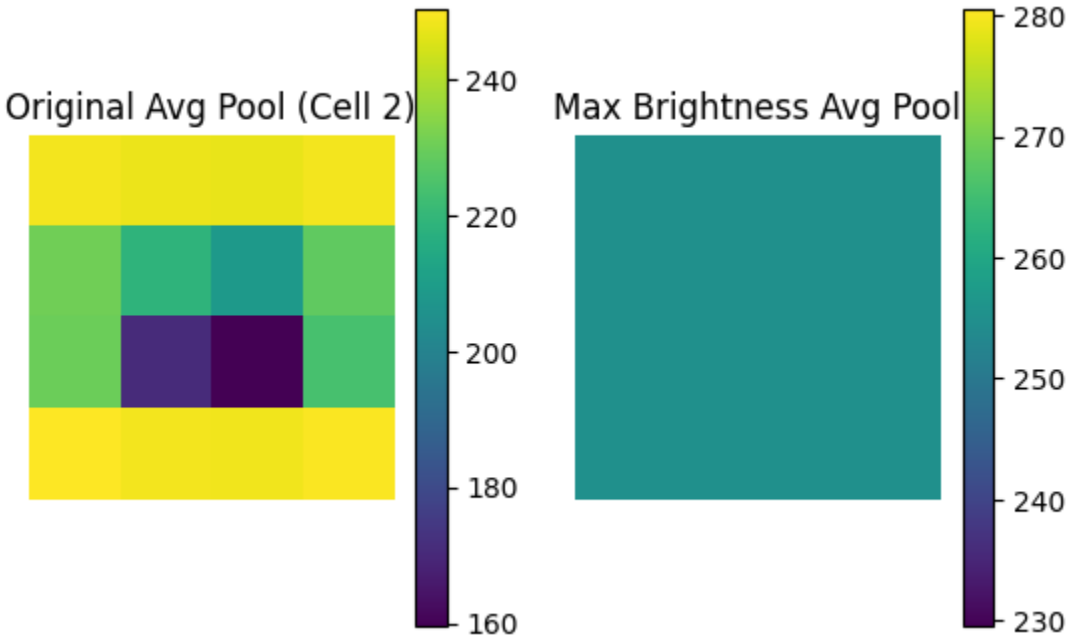
```
Original Avg Pool (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

50% Horizontal Crop Avg Pool:
 [[251.25 246.75 247.75 248.  ]
 [241.25 220.5  219.75 217.5 ]
 [240.75 219.25 188.   152.75]
 [251.75 248.5  248.5  248.5 ]]
```



In [45]:

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2

# -------- Convert image to maximum brightness --------
bright_img = np.full_like(img_gray, 255)  # all pixels = max intensity

# Resize to 8x8
bright_8x8 = cv2.resize(bright_img, (8, 8), interpolation=cv2.INTER_AREA)

# Apply average pooling
bright_avg_pooled_matrix = average_pooling(bright_8x8)

# -------- Visualization --------
plt.figure(figsize=(10, 4))

# Cell 2 output
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original Avg Pool (Cell 2)")
plt.colorbar()
plt.axis("off")

# Max brightness pooled output
plt.subplot(1, 3, 2)
plt.imshow(bright_avg_pooled_matrix, cmap="viridis")
plt.title("Max Brightness Avg Pool")
plt.colorbar()
plt.axis("off")

# -------- Numerical verification --------
print("Original Avg Pool (Cell 2):\n", avg_pooled_matrix)
print("\nMax Brightness Avg Pool:\n", bright_avg_pooled_matrix)
```

```
Original Avg Pool (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

Max Brightness Avg Pool:
 [[255. 255. 255. 255.]
 [255. 255. 255. 255.]
 [255. 255. 255. 255.]
 [255. 255. 255. 255.]]
```

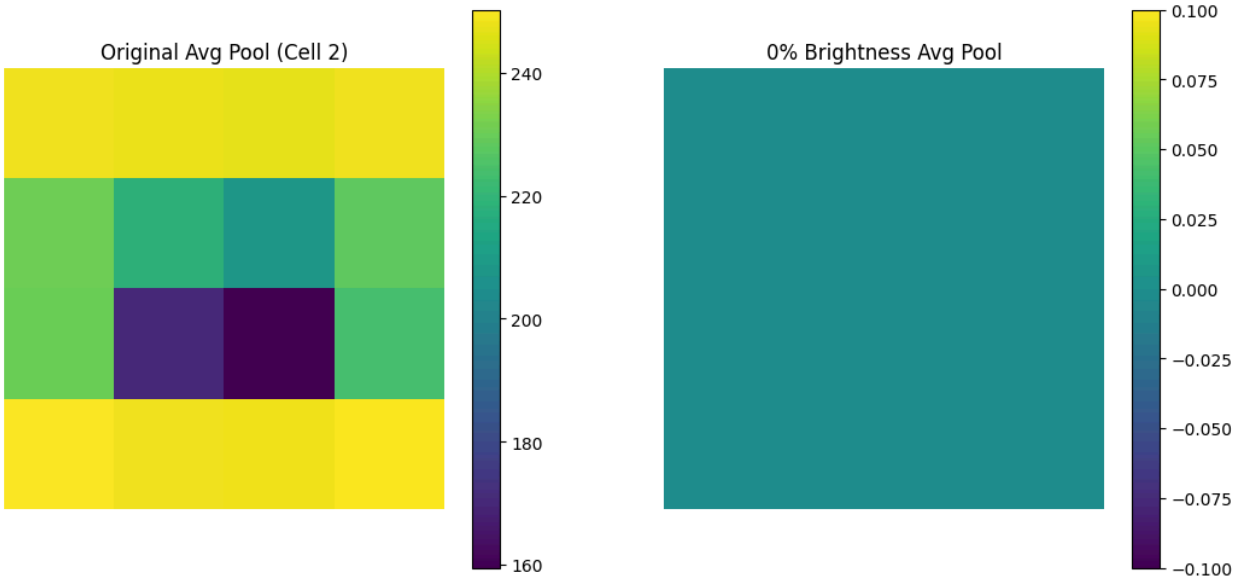


In [44]:

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2

# -------- Set brightness to 0% --------
# 0% brightness = all pixel values become 0
dark_img = np.zeros_like(img_gray)

# Resize to 8x8
dark_8x8 = cv2.resize(dark_img, (8, 8), interpolation=cv2.INTER_AREA)

# Apply average pooling
dark_avg_pooled_matrix = average_pooling(dark_8x8)

# -------- Visualization --------
plt.figure(figsize=(20, 6))

# Cell 2 output
plt.subplot(1, 3, 1)
plt.imshow(avg_pooled_matrix, cmap="viridis")
plt.title("Original Avg Pool (Cell 2)")
plt.colorbar()
```

```
plt.axis("off")

# 0% brightness pooled output
plt.subplot(1, 3, 2)
plt.imshow(dark_avg_pooled_matrix, cmap="viridis")
plt.title("0% Brightness Avg Pool")
plt.colorbar()
plt.axis("off")

# -------- Numerical verification --------
print("Original Avg Pool (Cell 2):\n", avg_pooled_matrix)
print("\n0% Brightness Avg Pool:\n", dark_avg_pooled_matrix)
```

```
Original Avg Pool (Cell 2):
 [[248.75 247.75 247.25 248.75]
 [230.75 218.5  208.75 228.25]
 [229.75 170.5  159.5  223.75]
 [250.25 248.5  248.25 249.75]]

0% Brightness Avg Pool:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```



Deep Insights

Parameter (Cell)	Difference between changed image and comparison with original image (Avg Pooling Insight)
90° Rotation	The pooled output differs in spatial arrangement because 2×2 pooling windows now aggregate pixels from different original neighborhoods. Average values may be similar in range, but their **positions change**, proving average pooling is **not rotation invariant**.

Parameter (Cell)	Difference between changed image and comparison with original image (Avg Pooling Insight)
180° Rotation	Pooling values correspond to inverted spatial regions. While local intensity statistics are preserved, their **global placement is reversed**, so the pooled matrix does not match the original. Pooling preserves *local means*, not orientation.
270° Rotation	Similar to 90°, pooling windows cover different semantic regions. The pooled output shows **rearranged and mismatched spatial averages**, confirming lack of rotational robustness.
Horizontal Flip (Mirror)	Column-wise inversion occurs. Pooled values remain numerically comparable but appear in **mirrored positions**, meaning average pooling is **not reflection invariant**.
Vertical Flip (Mirror)	Row-wise inversion of pooled values is observed. The operation preserves intensity statistics but loses absolute spatial correspondence.
50× Zoom (Scale up → resize → pool)	Interpolation during zoom smooths the image, and resizing back to 8×8 removes high-frequency details. Pooling output becomes **more uniform** compared to the original. This confirms average pooling is **not scale invariant**.
Vertical Crop (50%)	Half of the image content is permanently removed. After resizing, pooled values represent **only partial scene statistics**, leading to significant deviation from the original pooled output. Pooling is **not robust to occlusion**.
Horizontal Crop (50%)	Similar to vertical cropping, but bias appears along columns. The pooled output reflects **region-specific intensity bias**, not the full image distribution.
Maximum Brightness (255)	All pixels become identical. Average pooling outputs a **uniform matrix** where every value is the same. All spatial information is lost, proving pooling alone cannot handle extreme illumination.
Minimum Brightness (0%)	All pooled values collapse to zero. This confirms average pooling has **no internal normalization** and depends entirely on input intensity distribution.

Project Conclusion

This project demonstrates that average pooling is not inherently invariant to common image transformations such as rotation, reflection, scaling, cropping, or extreme illumination changes. Although average pooling smooths local pixel intensities and reduces spatial resolution, it remains highly sensitive to changes in image orientation, scale, and content distribution. The experimental comparisons confirm that transformations significantly alter pooled outputs, even when resized back to a fixed dimension. These findings reinforce the understanding that data augmentation improves CNN accuracy not because pooling operations are invariant, but because convolutional filters learn robust features by being exposed

to diverse variations during training. Therefore, average pooling should be viewed as a dimensionality reduction and noise-smoothing operation rather than a mechanism for achieving transformation invariance. Robust CNN performance ultimately depends on learned representations, normalization techniques, and diverse augmented training data. **bold text**

In [ ]: