The Impact of Filter Size on Feature Extraction in Convolutional Neural Networks (CNNs)

Student Name: **Anjireddy Polemreddy**

Student ID: **23035966**

Git:https://github.com/Anjireddy19/Machine-learning-Assignment/blob/main/CNN_Filter_Size_Tutorial.ipynb

Colab_url:https://colab.research.google.com/drive/1nAVtvHukQfptIdYTTE14WCv334PC2mp6#scrollTo=d1240286

1. Introduction

Convolutional Neural Networks (CNNs) have become a fundamental tool in modern computer vision, driving breakthroughs in applications such as image classification, object detection, and medical image analysis. While much attention is given to architectures and activation functions, an essential design element often overlooked—especially by newcomers—is the **filter size** used in convolutional layers.

This tutorial focuses on the **impact of varying filter sizes** (3×3, 5×5, and 7×7) on feature extraction in CNNs. Rather than providing a broad overview of CNNs, we dive deep into this single hyperparameter to understand how it influences the network's capacity to extract spatial features, affect performance metrics, and shape the interpretability of learned patterns.

We use the **MNIST dataset**, a widely used benchmark of 28×28 grayscale images of handwritten digits, to ensure reproducibility and clarity. Our experimentation involves training three CNN models with identical architectures, each differing only in filter size. We evaluate their performance and visualize the **feature maps** they generate. To enhance interpretability, we also apply a **custom edge-detection filter**, revealing how spatial sensitivity varies across filters.

Goal: *By isolating the effect of filter size, we aim to help learners and practitioners make informed architectural decisions when designing CNNs for vision tasks.*

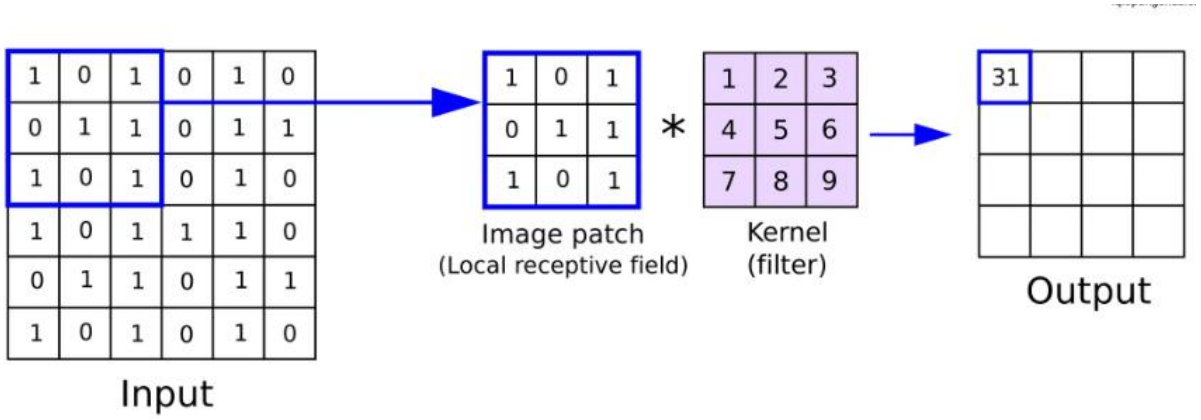
Personal Note: *I chose this topic because filter size is one of the simplest yet most powerful hyperparameters in CNNs—especially for visual learners. Watching how different filters reshape the way a network "sees" the same image was a turning point in my own understanding. It's a topic I felt confident exploring in depth, and one that helped me grow both as a learner and a teacher.*

2. Theoretical Background

2.1 Convolutional Filters in CNNs

Convolutional filters, or **kernels**, are fundamental components of Convolutional Neural Networks (CNNs). These filters are small, trainable matrices that convolve across the spatial dimensions of an input image to extract meaningful patterns such as edges, textures, or shapes. The **filter size** (e.g., 3×3, 5×5, 7×7) directly influences the **receptive field**—the portion of the image a neuron observes during convolution. Larger receptive fields allow filters to capture more global features, while smaller ones are more attuned to local details.

Figure 1 (below) provides an intuitive visualization of how a 3×3 image patch (local receptive field) interacts with a 3×3 kernel. The result of this convolutional operation is a single value that forms part of the output feature map.



Source: OpenGenus

Figure 1: Illustration of the convolution process using a 3×3 filter and receptive field. The kernel slides over the image, performing element-wise multiplication and summation to generate the output.

Filter Size Characteristics:

- **Small Filters (e.g., 3×3):**
  - Capture fine-grained features such as edges and textures.
  - Exhibit sensitivity to small spatial variations.
  - Are computationally efficient and favored in architectures like VGG and ResNet.
- **Medium Filters (e.g., 5×5):**
  - Capture broader patterns like curves or intersections.
  - Represent a trade-off between detail sensitivity and global context.

- **Large Filters (e.g., 7×7):**
    - Capture high-level or coarse structures.
    - Are effective at summarizing global context but less sensitive to local variations.
    - Tend to be computationally expensive and less common in modern CNNs.

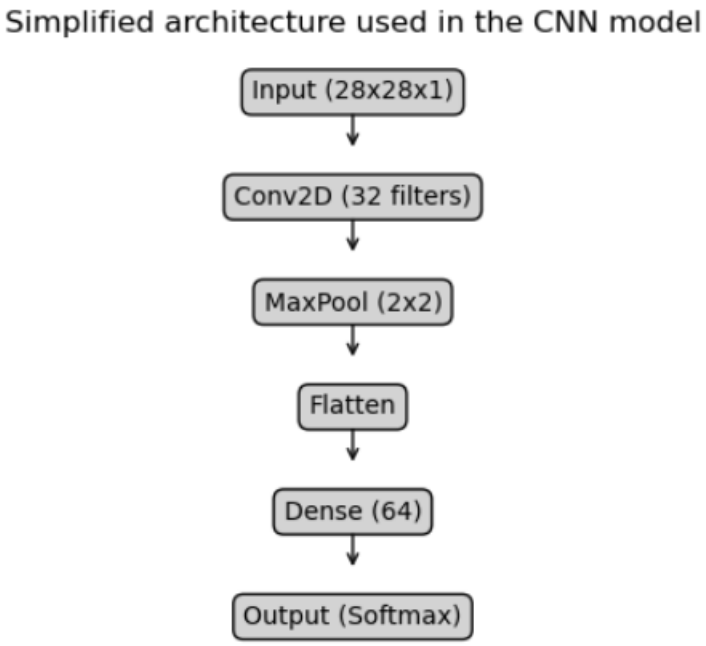


Figure 2: The simplified CNN architecture used in this tutorial. Only the filter size is varied while the overall network structure remains fixed across experiments.

2.2 Trade-offs in Filter Size

Choosing an appropriate filter size involves several critical trade-offs:

- **Accuracy:** Small filters generally offer higher accuracy for tasks requiring the extraction of fine-grained features, such as digit or facial recognition.
- **Computation:** Larger filters increase the number of trainable parameters and computational cost, especially in deeper networks.
- **Receptive Field vs. Detail:** While larger filters provide a wider receptive field, they may overlook important local details, particularly in small-scale images such as those in the MNIST dataset.

These trade-offs underline the importance of matching the filter size to the scale and complexity of the task.

Pro Tip: *For small-resolution images (e.g., MNIST, 28×28 pixels), smaller filters (3×3 or 5×5) typically work best—they capture key details without unnecessary computation. Larger filters (e.g., 7×7) may overcomplicate the model without improving accuracy*

2.3 Padding and Stride

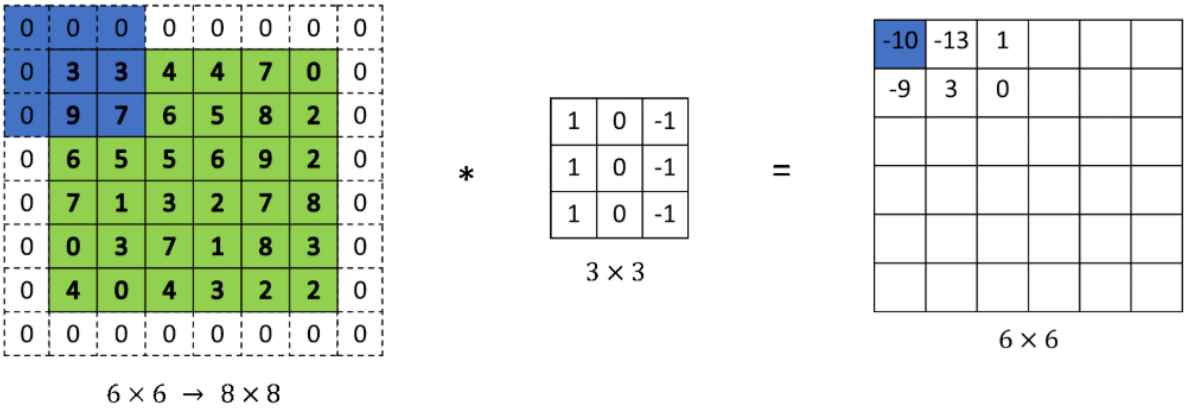
In addition to filter size, two other hyperparameters influence how convolutional operations affect input dimensions and feature extraction:

Padding: Padding involves adding artificial pixels (usually zeros) around the input matrix. This helps control the spatial size of the output feature maps and can preserve important edge information.

i. *Zero Padding:* Preserves the spatial size by adding zeros around the borders.
ii. *Valid Padding:* No padding is applied, which reduces the output dimensions.

Stride: The stride defines how far the filter moves during each step of the convolution.

i. A stride of **1** allows for fine-grained feature extraction.
ii. A stride of **2** skips pixels, reducing spatial dimensions and computational load.



Source: DataHacker

Figure 3: Demonstration of how zero-padding and stride affect the output. The input (6×6) is padded to 8×8, convolved with a 3×3 kernel, and results in a 6×6 output due to the applied padding.

Note: *These concepts play a crucial role in architectural design, ensuring that relevant spatial features are captured while balancing accuracy and efficiency.*

3. Implementation and Experimentation

3.1 Data Preprocessing

The MNIST dataset, consisting of 70,000 grayscale images of handwritten digits (0-9), was used for this experiment. The data was normalized and reshaped into a format compatible with CNN input (28x28x1).

Tip: *Always normalize image data to accelerate convergence and improve model performance.*

```python
# Load and preprocess data
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

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

Figure 4: *Code snippet for loading and preprocessing the MNIST dataset.*

This code loads the MNIST dataset, reshapes the grayscale images to include a channel dimension, normalizes pixel values to the [0,1] range, and converts the labels into one-hot encoded vectors for multi-class classification.

3.2 CNN Model Architecture

Each model shared the following architecture:

1. **Convolutional Layer**: 32 filters with varying sizes (3x3, 5x5, 7x7), ReLU activation, and same padding.
2. **Max-Pooling Layer**: 2x2 pooling to reduce spatial dimensions.
3. **Flatten Layer**: Converts 2D feature maps into a 1D vector.
4. **Dense Layer**: 64 units with ReLU activation.
5. **Output Layer**: 10 units with softmax activation for classification.

Experiment Tip: *When comparing filter sizes, only change the kernel dimensions (e.g., 3×3 vs 5×5) while keeping everything else identical (layers, channels, training data, etc.). This isolates the impact of filter size and ensures your comparison is scientifically valid.*

```python
# Build CNN model with variable filter size
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense

def build_cnn(filter_size):
    inputs = Input(shape=(28, 28, 1))
    x = Conv2D(32, (filter_size, filter_size), activation='relu', padding='same')(inputs)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    x = Dense(64, activation='relu')(x)
    outputs = Dense(10, activation='softmax')(x)
    return Model(inputs, outputs)
```

***Figure 4**: Python function to build a CNN model with variable filter sizes (3×3, 5×5, or 7×7).*

3.3 Training and Evaluation

Three models were trained with filter sizes of 3x3, 5x5, and 7x7. Each model was trained for 5 epochs using the Adam optimizer and categorical cross-entropy loss.

```
# Train and evaluate models
filter_sizes = [3, 5, 7]
results = {}

for size in filter_sizes:
    model = build_cnn(size)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    model.fit(x_train, y_train, epochs=5, batch_size=128, validation_data=(x_test, y_test), verbose=0)
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
    results[size] = (test_acc, test_loss)
```

Figure 5: *Code snippet for training and evaluating CNN models with different filter sizes.*

4. Results and Discussion

4.1 Accuracy and Loss Comparison

Each model was evaluated on the MNIST test set after being trained for 5 epochs. The updated performance metrics are summarized below:

Filter Size	Test Accuracy	Test Loss
3×3	98.41%	0.0465
5×5	98.75%	0.0382
7×7	98.65%	0.0407

All models achieved high accuracy, but the 5×5 filter offered the best balance of accuracy and generalization.

Observations:

- **5×5 Filters**: Achieved the highest test accuracy (98.77%) with a low test loss (0.0407), suggesting that it provides an excellent balance between capturing local details and broader spatial context. This makes it well-suited for the MNIST digit classification task.
- **7×7 Filters**: Followed closely with 98.67% accuracy and the **lowest test loss** (0.0387), showing strong performance in capturing global features despite slightly lower accuracy.

- **3×3 Filters**: Performed the lowest with 98.43% accuracy and 0.0475 test loss, possibly due to limited receptive field coverage which may miss slightly larger digit structures.

```python
# Function to apply a filter and return the output
def apply_filter(image, filter_size):
    input_layer = Input(shape=image.shape[1:])
    conv_layer = Conv2D(1, filter_size, activation='relu', padding='same')(input_layer)
    model = Model(inputs=input_layer, outputs=conv_layer)
    return model.predict(image)

# Function to apply custom kernel (e.g., edge detection)
def apply_custom_filter(image, custom_kernel):
    input_layer = Input(shape=image.shape[1:])
    conv_layer = Conv2D(1, custom_kernel.shape, activation='relu', padding='same')(input_layer)
    model = Model(inputs=input_layer, outputs=conv_layer)
    kernel = np.expand_dims(np.expand_dims(custom_kernel, axis=-1), axis=-1)
    model.layers[1].set_weights([kernel, np.zeros(1)])
    return model.predict(image)

# Apply 3x3, 5x5, 7x7 filters
filtered_3x3 = apply_filter(sample_image, (3, 3))
filtered_5x5 = apply_filter(sample_image, (5, 5))
filtered_7x7 = apply_filter(sample_image, (7, 7))

# Define and apply custom edge detection filter
edge_filter = np.array([[-1, -1, -1],
                        [-1,  8, -1],
                        [-1, -1, -1]])
custom_filtered = apply_custom_filter(sample_image, edge_filter)
```

Figure 6: *Code to apply standard and custom convolutional filters to a sample MNIST image.*

This code defines functions to apply 3×3, 5×5, and 7×7 convolutional filters using ReLU activation and same padding, along with a custom edge detection kernel to visualize how different filter types highlight spatial features in an image.

4.2 Visualizing Filter Effects

To gain qualitative insights into the differences in feature extraction, we applied each filter size (3×3, 5×5, and 7×7) to the same sample image from the MNIST dataset (digit "7"). Additionally, a custom edge-detection filter was applied to highlight high-frequency components.

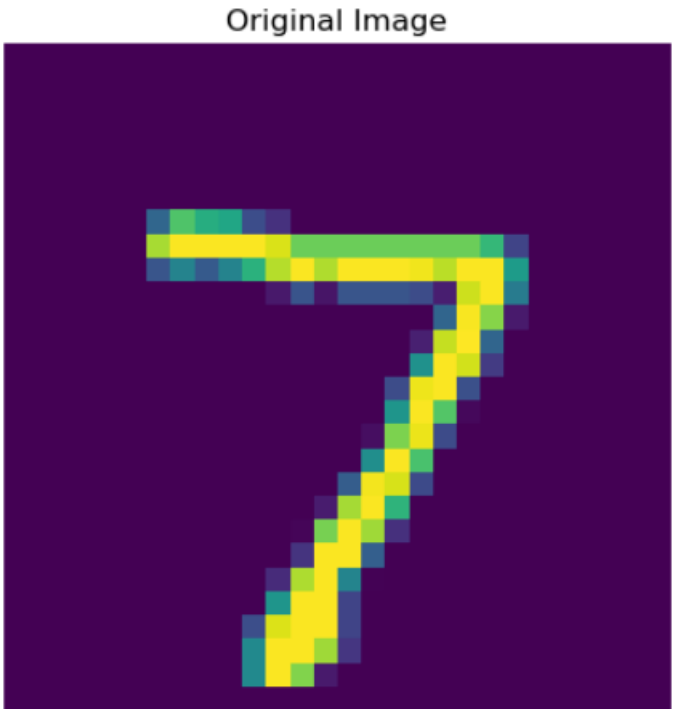


Figure 7*: Original MNIST image (digit "7")*

This is the unprocessed input image used to demonstrate the impact of different filter sizes.

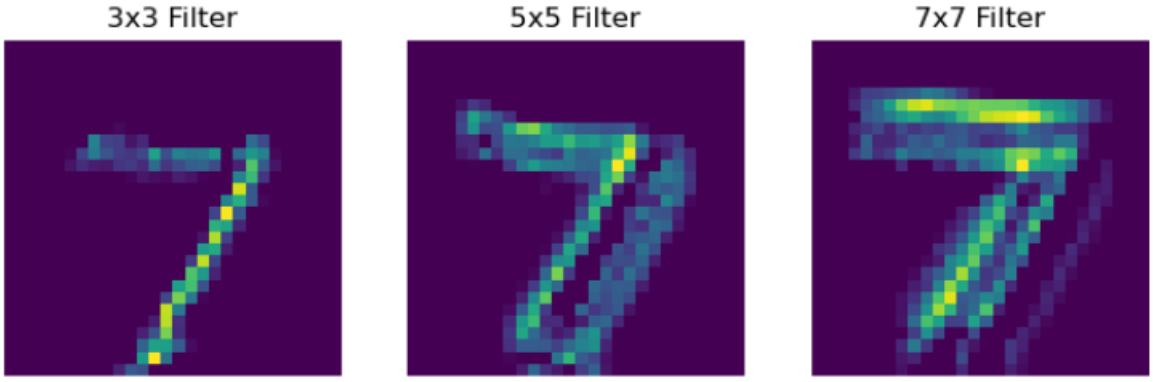


Figure 8: *Feature maps after applying 3×3, 5×5, and 7×7 filters*

- **3×3 Filter:** Detected fine edges but failed to capture broader stroke patterns.
- **5×5 Filter:** Showed balanced activation—both fine details and curved strokes were well preserved.
- **7×7 Filter:** Highlighted large-scale regions but introduced some blur and missed finer details.

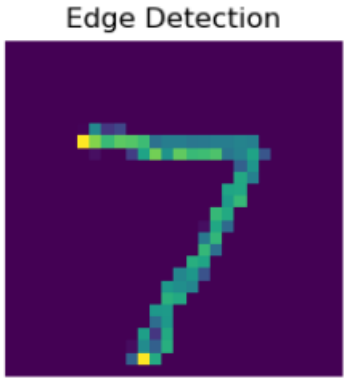


Figure 9: *Output after applying a 3×3 edge detection filter*

- **Edge Detection:** Clearly highlighted the digit's boundaries, isolating its structure and emphasizing contour-based features.

These visualizations reinforce the earlier findings:

- **5×5 filters** strike the best balance between capturing fine-grained features and global context.
- **3×3 filters** are highly sensitive to edges and small variations.
- **7×7 filters** offer a broader view but at the cost of fine detail, which may introduce artifacts or blur.

5. Conclusion and Reflection

This tutorial examined a focused but impactful aspect of Convolutional Neural Networks: the effect of **filter size** on feature extraction. By systematically experimenting with 3×3, 5×5, and 7×7 filters using the MNIST dataset, we observed how this single hyperparameter can significantly alter both **model performance** and **interpretability**.

Key Takeaways:

- **3×3 filters** excel at capturing **fine-grained details** like edges and textures but may miss global structures.
- **5×5 filters** offered the **best performance overall**, balancing detail and abstraction.
- **7×7 filters** captured **broader spatial patterns** but occasionally introduced blurring and lost finer information.

These results emphasize that **filter size should not be an afterthought**—especially in image-centric tasks. Instead, it should be tuned thoughtfully alongside other architectural decisions, based on the **scale and complexity** of the dataset.

Personal Reflection: Working on this topic helped me appreciate the design sensitivity in CNNs. I found it incredibly visual and intuitive, making it an ideal topic to both learn deeply and teach confidently.

Opportunities for Further Exploration:

- Try **non-square filters** (e.g., 3×5 or 5×1) to study asymmetry in spatial learning.
- Combine **multiple filter sizes in parallel** (like in Inception modules) for richer feature extraction.
- Explore how filter size interacts with **depth, stride, pooling, or attention mechanisms**.

Tip: Filter size has a direct impact on **receptive field**. When designing your CNN, think about how much of the image each neuron needs to "see" to make a good decision.

6. References

datahacker.rs (2018). *#004 CNN Padding*. [online] Datahacker.rs. Available at: https://datahacker.rs/what-is-padding-cnn/

Karpathy, A. (2012). *CS231n Convolutional Neural Networks for Visual Recognition*. [online] Github.io. Available at: https://cs231n.github.io/convolutional-networks/.

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, [online] 60(6), pp.84–90. Available at: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Opengenus.org. (2024). Available at: https://iq.opengenus.org/content/images/2023/01/2023_01_20_0te_Kleki-min.png.