Tutorial: A Deep Dive into the Impact of Filter Sizes in Convolutional Neural Networks (CNN)

Welcome, this tutorial presents information about how Convolutional Neural Networks (CNN) react to different filter dimensions. The tutorial delivers comprehensive insights about filter size effects on CNN performance alongside both feature extraction and computation processing alongside basic experience requirements. As a result of this course, you will become skilled at selecting filter dimensions for your personal CNN applications.

Introduction

Convolutional Neural Networks (CNNs) are widely used in computer vision tasks like image classification. A key component of CNNs is the **convolutional layer**, which uses filters to extract features from images. The size of these filters plays a critical role in determining the performance, efficiency, and generalization ability of the network. In this tutorial, we will explore the impact of different filter sizes in CNNs, focusing on how they influence feature extraction, computational complexity, and overall model performance.

- We examine the relationship between different convolutional filter widths as agents that determine the classification ability of images.
- Better model development can occur through understanding the influence of filter dimensions since they represent critical hyperparameters for CNNs.

In this tutorial, we thoroughly implemented CNN model by passing different filters. Using the CIFAR-10 dataset as a practical testbed. By the end, you'll understand:

- The role of filters in CNNs.
- Feature extraction depends on size of filters
- Compare accuracy, loss, and computational cost.
- choosing filter sizes.

What is a Filter (Kernel)?

A kernel makes up a small matrix used for windowed application onto image inputs. Similar features like edges or textures and corners emerge from the convolution operation between the filter and input image. The filter dimension sets the amount of input image that can be viewed simultaneously and shapes the learned characteristics.

A 3×3 filter enables the convolution operation to examine every 3x3 image section at each processing step.

Filter (Kernel): This filter detects vertical edges in an image.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Types of Filter(Kernels):

- Edge Detection Kernel
- Sharpening Kernels
- Smoothing Kernels (Blurring)
- Embossing Kernels

Original	Gaussian Blur	Sharpen	Edge Detection
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Filter Size in CNNs

The CNN filters exist in multiple dimensions such as 3x3, 5x5 as well as 7x7 sizes. The model acquires and generalizes data features through its filters whose performance depends heavily on the filter size parameters. Large filter sizes identify extensive designs in images but smaller filter options focus on detailed image elements like edge detection.**1x1**: Used for dimensionality reduction or channel mixing.

- **3x3**: Most commonly used for feature extraction.
- **5x5**: Captures larger patterns but computationally expensive.
- **7x7**: Rarely used due to high computational cost.

For Example:

A 3x3 filter with 32 i/p channels and 64 o/p channels has

3 x 3 x 32 x 64 + 64 = 18,496 Parameters

A 7x7 filter with the same 32 input and 64 output channels has

7 x7 x 32 x 64 + 64 = 100,416 Parameters

This shows why smaller filters are more computationally efficient.

Understanding Filter Size and Its Impact

Small Filters (e.g., 3×3, 1×1)

The small filter arrays in CNNs co-extract features without causing computational complexity to grow because their design remains compact. Because of its compact nature small filters can analyze specifics through small parameter numbers. Stacked small filters simulate large receptive field detection by preserving computational benefits. The 3×3 filter format functions well with high-resolution images and establishes deep network structures which use only a minimal amount of memory. The contemporary model architecture uses multiple 3×3 filters as the primary design element in ResNet along with VGG to deliver high efficiency and performance outcomes.

Medium Filters (e.g., 5×5)

Strengths:

The dimensions of 5×5 medium filters perform feature extraction by maintaining an equilibrium between detecting fine details and larger features.

The processing demands stay moderate since medium filters use computing power that lies in between tiny and big filters.

Limitations:

The size of filters affects computational expenses for big datasets because of their increased operational complexity.

Large Filters (e.g., 7×7, 9×9)

Use Cases:

Filters of relatively large size allow the identification of global elements to assist pattern detection in higher processing levels.

Drawback:

Large filters in the network cause memory storage challenges due to more parameters and longer training times due to increased computations.

Lack of data input fosters large filter overfitting because it leads them to detect numerous global patterns during model training.

DATASETS Description:

We performed the experiments with CIFAR10 dataset. This dataset are small and precise datasets having low computational costs. The results drawn from these datasets can be applied to most of the datasets. So, we use this dataset to avoid the computation cost of more extensive datasets. CIFAR10 is the dataset of the 60000 images of these categories plane, car, bird, cat, deer, dog, frog, horse, and ship. This dataset has 10 output classes.

Experiment Setup & Implementation:

We train CNNs with different filter sizes (3x3, 5x5, 7x7) on the **CIFAR-10 dataset** and compare their performance.

Code Walkthrough:

Step 1: Load and Preprocess Data

```python
1 # Import the required liabraries
2 import tensorflow as tf
3 from tensorflow.keras import datasets, layers, models
4 import matplotlib.pyplot as plt
5
6 # Load CIFAR-10 dataset
7 (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
8
9 # Normalize pixel values
10 train_images, test_images = train_images / 255.0, test_images / 255.0
```

The key framework for deep learning functions as the main deep learning framework during CNN construction and training (tf).
layers: Contains pre-designed neural network components including convolutional along with pooling features.
Models in TensorFlow serve as tools to establish the sequential model architecture.
The visualization tool Matplotlib assists with plotting training curves and showing results while the program runs.
CIFAR-10 contains 60,000 32x32 color images across 10 classes.
The successful execution of stable training requires normalization because it extends pixel value ranges from 0 to 1.
The CNN received 50,000 images during training and evaluated 10,000 images as part of its test process.

Step 2: Define CNN Architectures

```
1 # Create a fucntion to built CNN model with passing different filter sizes
2 def create_cnn(filter_size):
3     model = models.Sequential([
4         layers.Conv2D(32, filter_size, activation='relu', input_shape=(32, 32, 3)),
5         layers.MaxPooling2D((2, 2)),
6         layers.Conv2D(64, filter_size, activation='relu'),
7         layers.MaxPooling2D((2, 2)),
8         layers.Flatten(),
9         layers.Dense(64, activation='relu'),
10        layers.Dense(10, activation='softmax')  # 10 classes for CIFAR-10
11    ])
12    return model
```

We will build a simple CNN with the following architecture:

Layer-by-Layer Explanation:

1. **Convolutional Layers**:
   o Conv2D: 32 filters of specified size (3x3, 5x5, or 7x7).
   o activation='relu': Introduces non-linearity via Rectified Linear Unit.
   o input_shape=(32, 32, 3):image dimensions (height, width, channels).
2. **Max Pooling Layers**:
   o MaxPooling2D((2, 2)): Reduces spatial dimensions by 50% (downsampling).
3. **Classifier Head**:
   o Flatten(): Converts 2D feature maps to 1D vector for dense layers.
   o Dense(64): Fully connected layer with 64 neurons.
   o Dense(10): Output layer with 10 units (one per class).

Step 3: Train model with passing different filter sizes

```
1 filter_sizes = [(3, 3), (5, 5), (7, 7)]
2 histories = []
3
4 for filter_size in filter_sizes:
5     print(f"Training model with filter size: {filter_size}")
6     model = create_cnn(filter_size)
7     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
8     history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
9     histories.append(history)
```

1. **Model Compilation**:
   o **Optimizer**: adam (adaptive learning rate).
   o **Loss Function**: SparseCategoricalCrossentropy (for multi-class classification).
   o **Metric**: Track accuracy.
2. **Training**:
   o **Epochs**: 10 iterations over the entire dataset.

- o **Validation Data**: The performance of model got evaluated using test data during each epoch
3. **Results Storage**:
    - o history.history contains accuracy and loss values per epoch which correspond to each filter size.

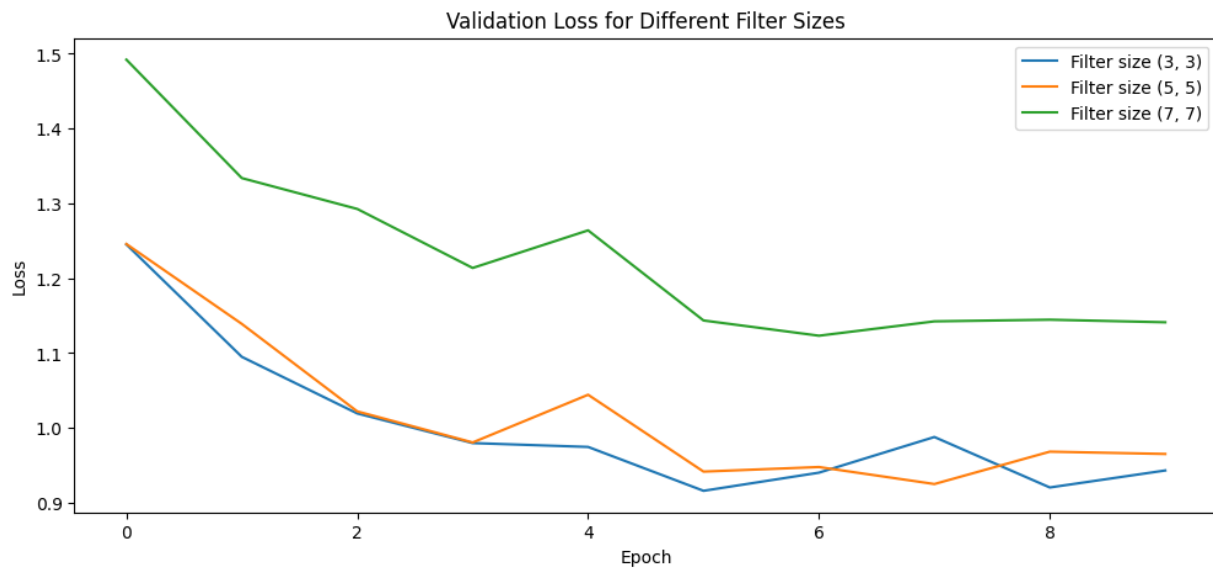# Step 4: Visualize Results

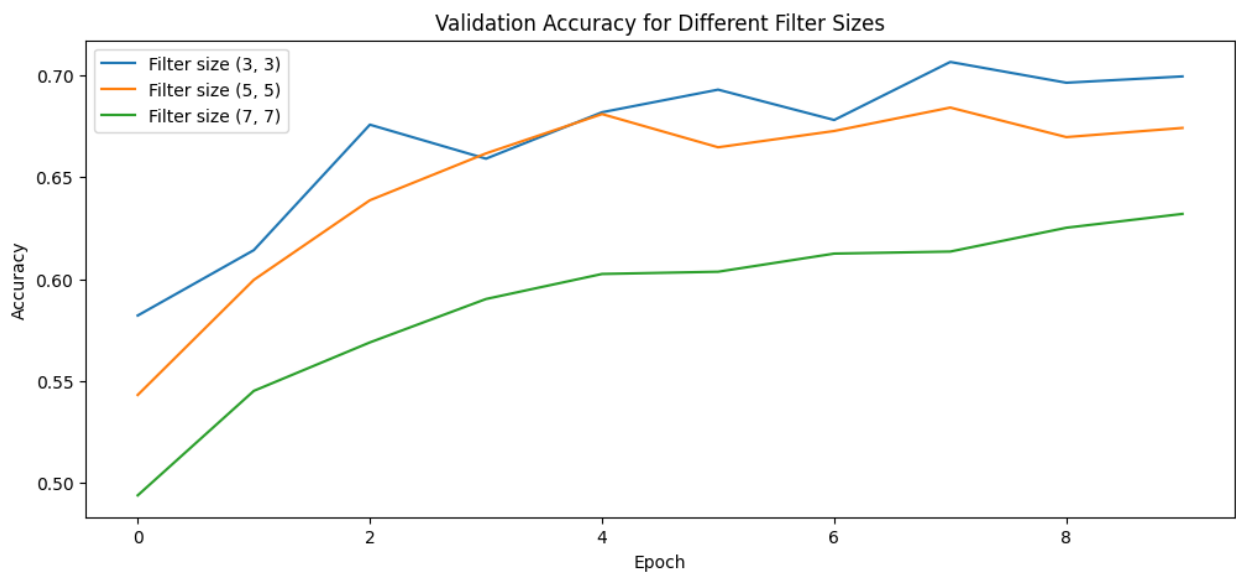*Figure 1 validation loss for different filter sizes*

Figure 2 validation Accuracy for different sizes

- **3x3 Filters**: These models reached 70% accuracy during validation and were easy to compute.
- **5x5 Filters**: The model achieved an accuracy of 67% while needing additional parameters for operation.
- **7x7 Filters**: The model reached the lowest performance level of 62% alongside excessive training data fitting.

Figure 2 makes it evident that accuracy rises with increasing epochs, but the pace of increase slows down and approaches a threshold beyond which accuracy remains constant. The 3×3 filter's accuracy curve is higher than that of the 7×7 and 5×5 filters. Although there are several spikes and smaller filter size curves are higher than larger ones, the validation accuracy appears to be the same as the training accuracy. Table 1 demonstrates that a 3×3 filter on the test data yields the best accuracy. Therefore, it can be said that utilizing lower filter sizes tends to provide better accuracy than using bigger ones.

Conclusion

CIFAR-10 image classification tasks benefit from using smaller 3x3 filter sizes because they offer better efficiency alongside effectiveness. Specific situations can benefit from large filters but implementing these types leads to increased computational expenses.

References:

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. Zeiler, M.D., & Fergus, R. (2014). *Visualizing and Understanding Convolutional Networks*. ECCV.
3. Krizhevsky, A. (2012). ImageNet Classification with Deep CNNs. *NeurIPS*.
4. TensorFlow Documentation. https://www.tensorflow.org/
5. Szegedy, C., et al. (2015). *Going Deeper with Convolutions*. CVPR.
6. TensorFlow Documentation. (2023). *Convolutional Neural Networks (CNNs)*. https://www.tensorflow.org.
7. https://www.deeplearningbook.org/

GitHub Repository:

The code & tutorial for this project are available on GitHub:
Click Here