Tutorial: Effect of Kernel Size and Network Depth on CNN Performance

1. Introduction

Convolutional Neural Networks (CNNs) are a class of neural networks designed to process grid-like data such as images (Goodfellow et al., 2016). They exploit spatial structure through local connectivity and weight sharing, making them particularly effective for computer vision tasks. In this tutorial, the focus is placed on kernel size and network depth because these architectural choices play a central role in explaining why convolutional neural networks outperform fully connected models on image data. By examining these elements directly, the objective is to understand how these choices affect feature extraction, generalisation, and overfitting, using CIFAR-10 as a benchmark dataset.

The tutorial focuses on two key architectural choices in CNN design:

Kernel size: the spatial dimensions of convolutional filters (e.g. 3×3 vs 5×5)

Network depth: the number of stacked convolutional layers

2. Background Concepts

This tutorial draws on a small number of well-established research papers and authoritative sources to motivate and explain key concepts. The aim is to condense key ideas that are directly relevant to understanding kernel size and depth in CNNs. Classic CNN architectures such as LeNet-5 demonstrated the effectiveness of small convolutional kernels for digit recognition (LeCun et al., 1998), while later work such as VGGNet showed that stacking many 3×3 convolutions can outperform shallower networks with larger kernels while using fewer parameters (Simonyan and Zisserman, 2014). These insights motivate the focus of this tutorial.

2.1 Convolutional Kernels: A convolutional kernel (or filter) is a small matrix that slides across an image to extract local features such as edges, corners, or textures. Smaller kernels (e.g. 3×3) capture fine-grained local patterns, while larger kernels (e.g. 5×5) have a wider receptive field but introduce more parameters.

2.2 Network Depth: refers to the number of convolutional layers stacked in a CNN. Increasing depth allows the network to learn hierarchical representations, where early layers capture simple features and deeper layers combine them into more abstract concepts. However, deeper networks are more prone to overfitting and harder to train without appropriate regularisation.

3. Dataset Selection: CIFAR-10

It is a standard image classification benchmark consisting of 60,000 colour images of size 32×32, split into 10 object classes (Krizhevsky, 2009). Although CIFAR-10 has been widely used in the literature, it was deliberately chosen for this tutorial because it is significantly more challenging than MNIST, which often saturates quickly, architectural differences between CNNs become more visible and it remains computationally manageable for coursework settings. Rather than proposing CIFAR-10 as a novel dataset, this tutorial uses it as a controlled benchmark to clearly illustrate how kernel size and depth influence learning behaviour.

4. Step-by-Step Experiments and Code

This section walks through the experimental process used to investigate kernel size and network depth. Each step builds on the previous one, mirroring how a practitioner might approach CNN design in practice.

Step 1: Preparing the Data, the CIFAR-10 dataset is loaded and normalised to ensure stable training. A tf.data pipeline is used to batch, shuffle, and prefetch data efficiently. This step ensures that model performance reflects architectural choices rather than data-loading bottlenecks.

Step 2: Defining a Configurable CNN, a CNN model is defined using a modular builder function. This function allows kernel size and depth to be varied while keeping other hyperparameters fixed. By controlling these variables explicitly, the effect of each design choice can be isolated.

```
x = layers.Conv2D(filters, kernel_size, padding="same")(x)

x = layers.BatchNormalization()(x)

x = layers.Activation("relu")(x)
```

Repeating this structure allows the network depth to be increased systematically, while the kernel_size parameter is adjusted across experiments to study its effect. Each convolutional block follows a consistent design, combining convolution, batch normalisation, and a ReLU nonlinearity. Max pooling and dropout are introduced at regular intervals to progressively reduce spatial resolution and to mitigate overfitting.

Step 3: Training Strategy

The model is trained using Adam optimisation with learning-rate scheduling and early stopping. These techniques help stabilise training and prevent overfitting, allowing meaningful comparisons between architectures. It is illustrated in code below:

EarlyStopping(monitor="val_loss", patience=4, restore_best_weights=True)

ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2)

Step 4: Monitoring Training and Validation

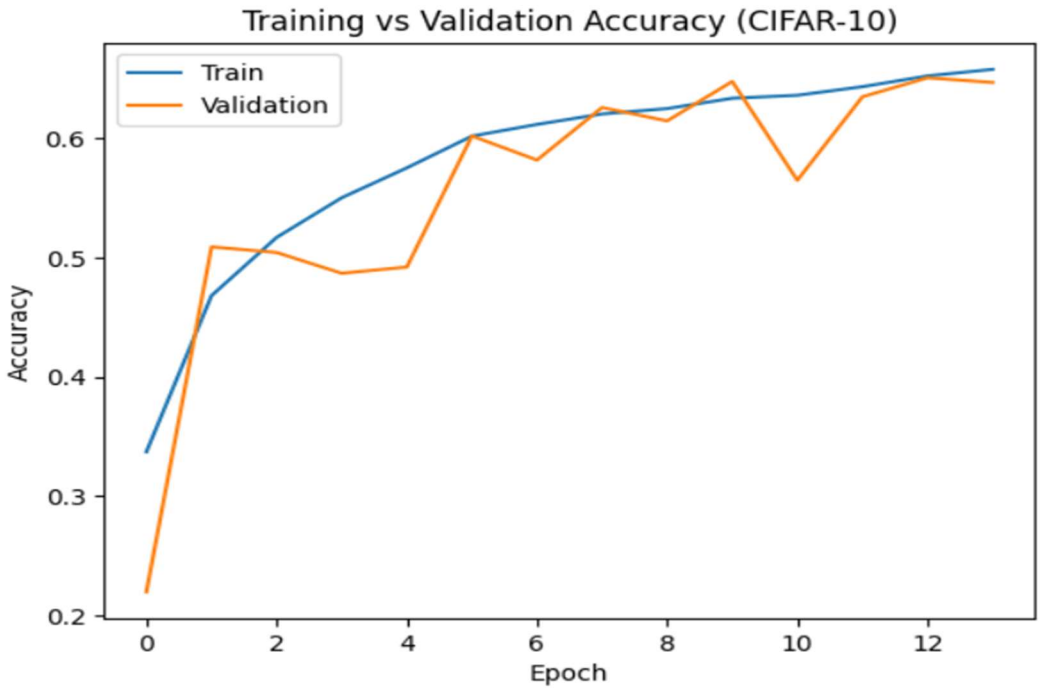
Training and validation accuracy and loss are plotted after training. These plots provide immediate feedback on whether a model is underfitting, overfitting, or generalising well.

5. Results and Analysis

Accessibility note

All plots use clear line styles and labels rather than colour alone to distinguish training and validation curves. This ensures that the figures remain interpretable for readers with colour vision deficiencies and are suitable for grayscale printing.

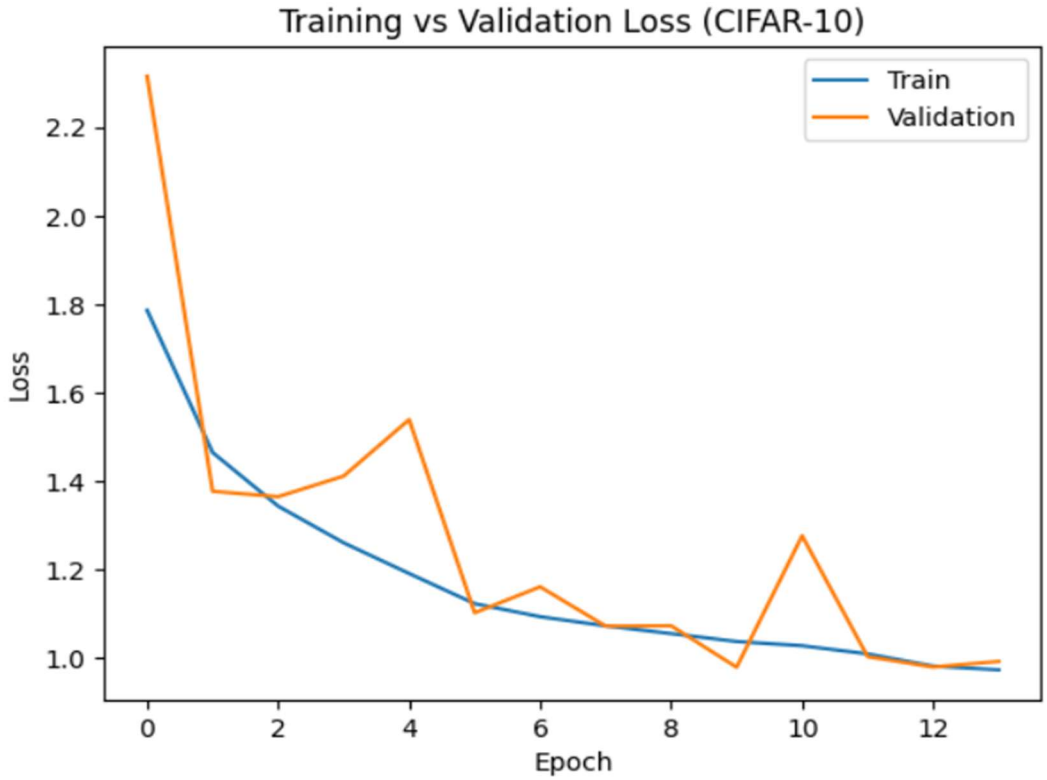
5.1 Training vs Validation Accuracy



Training and validation accuracy curves provide a high-level view of how well a model generalises beyond the training data. Training accuracy reflects how effectively the model fits the data it has seen, while validation accuracy measures performance on unseen examples.

When both curves increase together and remain close, this indicates that the model is learning features that transfer well to new data. In contrast, when training accuracy continues to improve while validation accuracy plateaus or decreases, the model is likely overfitting by memorising training-specific patterns rather than learning generalisable representations.

5.2 Training vs Validation Loss



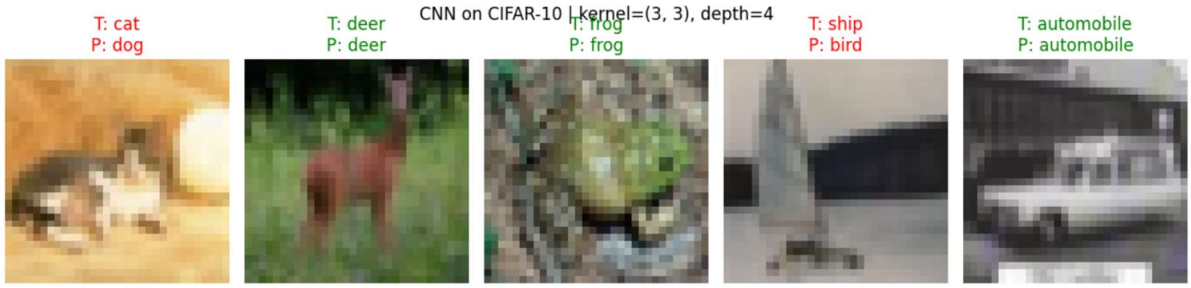
Loss curves provide a finer-grained perspective on the training process than accuracy alone. Whereas accuracy simply reports the proportion of correctly classified examples, the loss function also captures the model's confidence in its predictions. For this reason, changes in loss can highlight optimisation or generalisation issues that may not be immediately apparent from accuracy curves. Under effective training conditions, both training and validation loss decrease over time. When training loss continues to fall while validation loss begins to rise, this indicates overfitting, as the model is adapting too closely to the training data at the expense of generalisation.

5.3 Effect of Kernel Size

Smaller kernels (3×3) tend to perform well when stacked deeply, as multiple small kernels can approximate a larger receptive field while using fewer parameters, an idea popularised by very deep CNN architectures (Simonyan and Zisserman, 2014). Larger kernels (5×5) may learn broader features earlier but increase computational cost and the risk of overfitting. In practice,

differences between kernel sizes become clearer on CIFAR-10 than on simpler datasets such as MNIST, where performance often saturates quickly.

5.4 Qualitative Predictions



Qualitative evaluation complements numerical metrics by providing insight into how and why a model makes particular predictions. In this tutorial, a small set of CIFAR-10 test images is visualised alongside their predicted labels to illustrate both correct classifications and typical failure modes.

Correct predictions often correspond to images with strong, distinctive structural cues. Objects such as automobiles tend to exhibit rigid geometry and consistent shapes, which are easier for convolutional filters to capture reliably across different images. Misclassifications reveal more subtle limitations of the learned representations. For example, confusion between cats and dogs commonly arises due to similar textures, poses, and backgrounds, especially at the low spatial resolution of CIFAR-10. Other errors, such as ships misclassified as birds, are often caused by ambiguous silhouettes, motion blur, or background context, where the object's shape resembles features learned for another class. These examples demonstrate that even when overall accuracy is reasonable, CNNs may struggle with fine-grained distinctions or ambiguous visual cues. Qualitative inspection therefore plays an important role in diagnosing model behaviour and understanding the practical implications of architectural choices such as kernel size and network depth.

6. Conclusion

This tutorial examined how kernel size and network depth affect CNN performance on CIFAR-10. We showed that smaller kernels combined with greater depth often yield better generalisation, increasing depth improves representational power but increases overfitting risk. Proper training strategies (early stopping, data augmentation, learning-rate scheduling) are essential for stable CNN training. By focusing on a small number of controlled architectural choices, this tutorial aims to provide an accessible and practical understanding of CNN design decisions that readers can apply in their own image classification tasks.

References

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) *Gradient-based learning applied to document recognition.*

Simonyan, K. and Zisserman, A. (2014) *Very Deep Convolutional Networks for Large-Scale Image Recognition.*

Krizhevsky, A. (2009) *Learning multiple layers of features from tiny images.*

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning.*

Link to GitHub:

https://www.github.com/hassan310120/cnn-kernel-size-depth-tutorial