

Hyperparameter Analysis of MLP and CNN: Depth vs Width vs Accuracy

Author: Shahab Zaman

Course: Advanced Machine Learning

Date: December 10, 2025

GitHub Repository: <https://github.com/shah2110291/Hyperparameter-Analysis-of-MLP-and-CNN-Depth-vs-Width-vs-Accuracy/tree/main>

1. Introduction

Artificial neural networks (ANNs) are a cornerstone of modern machine learning, capable of modeling complex non-linear relationships and achieving state-of-the-art performance in numerous domains, including image recognition, natural language processing, and financial forecasting (Goodfellow, Bengio, & Courville, 2016). Among the most widely used ANN architectures are Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs).

The hyperparameters of these networks particularly depth (number of layers) and width (number of neurons or filters per layer) have a significant impact on model performance. Depth enables networks to learn hierarchical feature representations, whereas width determines how the network's capacity to capture detailed patterns (Bengio et al., 2013). Careful tuning of these hyperparameters is essential for achieving high accuracy while avoiding overfitting, underfitting, or the unnecessary computational cost (Srivastava et al., 2014).

This tutorial aims to provide a comprehensive analysis of how depth and width influence model accuracy in MLPs and CNNs, using MNIST (handwritten digits) and CIFAR-10 (object recognition) datasets. The goal is to educate practitioners and researchers about practical considerations when designing neural networks, with a focus on balancing accuracy, efficiency, and generalization.

2. Background: MLP and CNN Architectures

Modern machine learning relies heavily on the artificial neural networks (ANNs), which are capable of approximating highly complex non-linear relationships between inputs and outputs (Goodfellow, Bengio, & Courville, 2016). Among the most widely used architectures are **Multilayer Perceptrons (MLPs)** and **Convolutional Neural Networks (CNNs)**, each suited for different types of data and applications.

Understanding these architectures and the influence of hyperparameters such as **depth** and **width** is essential for constructing efficient, accurate, and generalizable models.

2.1 Multilayer Perceptron (MLP)

The **MLP** is one of the simplest yet fundamental types of the neural networks. It is a **feedforward network** where neurons in each layer are fully connected to the neurons in the next layer. MLPs are widely used for structured or tabular data, as well as for flattened image inputs.

An MLP typically consists of:

1. Input Layer:

- Receives the data in a **flattened form**, i.e., converting multi-dimensional images into a single vector.
- For MNIST, a grayscale 28x28 image is flattened into **784 neurons**.

2. Hidden Layers:

- One or more layers that apply **non-linear transformations** using activation functions such as ReLU (Rectified Linear Unit), sigmoid, or tanh.
- Hidden layers are where the network learns internal representations of the data.
- Increasing the **number of hidden layers (depth)** allows the MLP to model more complex hierarchical features.

3. Output Layer:

- Provides final predictions.
- For MNIST digit classification, it consists of **10 neurons** corresponding to the 10 classes, typically with **softmax activation** for probability outputs.

Hyperparameters in MLPs:

• Depth (Number of Hidden Layers):

- Increasing depth enables the network to learn **more abstract and hierarchical representations** of input data (Bengio et al., 2013).
- However, very deep MLPs may suffer from **vanishing gradients**, which slows learning and can reduce accuracy (Hinton et al., 2006).

• Width (Number of Neurons per Layer):

- Wider layers allow the network to capture **more detailed patterns** in the data.
- Excessively wide layers can lead to **overfitting**, where the model memorizes the training data but fails to generalize (Bengio, 2013).
- Wide layers also increase computational complexity and memory requirements.

MLPs are highly effective for **structured data, flattened images, and classification tasks** like MNIST (LeCun et al., 1998). However, they do not exploit spatial structure in images as efficiently as CNNs.

2.2 Convolutional Neural Network (CNN)

CNNs are specialized neural networks designed for **grid-like data** such as images, where spatial relationships between pixels are important. CNNs automatically learn **spatial features** through convolutional operations, which reduces the need for manual feature engineering.

CNNs typically consist of:

1. Convolutional Layers:

- Apply **learnable filters (kernels)** that scan over the input image to detect features such as edges, textures, and shapes.
- Multiple filters per layer capture a variety of features.

2. Pooling Layers:

- Reduce the spatial dimensions of feature maps while retaining important information.
- MaxPooling selects the maximum value in a region, while AveragePooling computes the mean.
- Pooling improves computational efficiency and provides **translation invariance**.

3. Fully Connected Layers:

- After convolution and pooling, features are flattened and passed to dense layers for classification.
- The final layer typically uses softmax activation for multi-class classification.

Hyperparameters in CNNs:

• Depth (Number of Convolutional Layers):

- More layers enable the network to learn **increasingly abstract features**, from edges to object parts to high-level patterns (He et al., 2016).
 - Excessive depth may lead to **overfitting** or vanishing/exploding gradients if not properly regularized.
- **Width (Number of Filters per Layer):**
- Wider layers (more filters) capture more diverse features.
 - Very wide layers increase computational cost with diminishing returns on accuracy (Krizhevsky et al., 2012).

CNNs excel at **image classification** tasks, making them ideal for datasets such as CIFAR-10 (Simonyan & Zisserman, 2015).

3. Dataset Description and Preprocessing

The choice of dataset is crucial for demonstrating the impact of hyperparameters. For this study, we use **MNIST** and **CIFAR-10**, two widely recognized benchmarks for image classification.

3.1 MNIST Dataset

- **Size:** 70,000 grayscale images (28x28 pixels).
- **Classes:** Digits 0–9.
- **Train/Test Split:** 60,000/10,000.

Preprocessing Steps:

1. **Normalization:** Scale pixel values to the range [0,1] to improve training stability.
2. **Flattening:** Convert 28x28 images to a 784-dimensional vector for MLP input.
3. **One-hot Encoding:** Convert labels to categorical vectors for classification.

Normalization and preprocessing improve convergence and reduce the chance of vanishing/exploding gradients (Goodfellow et al., 2016).

3.2 CIFAR-10 Dataset

- **Size:** 60,000 RGB images (32x32 pixels).

- **Classes:** 10 object categories (airplane, automobile, bird, etc.).
- **Train/Test Split:** 50,000/10,000.

Preprocessing Steps:

1. **Normalization:** Scale pixel values to [0,1].
2. **One-hot Encoding:** Encode labels for classification.
3. **Data Augmentation (Optional):**
 - Random rotations, flips, and translations improve generalization and reduce overfitting.

Data preprocessing ensures **consistent scaling**, stabilizes training, and enables faster convergence (Goodfellow et al., 2016).

4. Experimental Methodology

To analyze the effects of **depth** and **width**, we systematically vary these hyperparameters and evaluate performance.

4.1 MLP Training Setup

- **Depth:** 1–5 hidden layers.
- **Width:** 32, 64, 128, 256, 512 neurons per layer.
- **Activation:** ReLU for hidden layers, softmax for output.
- **Optimizer:** Adam (Kingma & Ba, 2014).
- **Loss Function:** Categorical Crossentropy.
- **Epochs:** 5 (MNIST dataset).
- **Batch Size:** 128.
- **Random Seed:** Fixed for reproducibility.

4.2 CNN Training Setup

- **Convolutional Layers:** 1–4.

- **Filters per Layer:** 16, 32, 64, 128.
- **Fully Connected Layer:** 128 neurons.
- **Activation:** ReLU for convolutional and dense layers, softmax for output.
- **Optimizer:** Adam.
- **Loss Function:** Categorical Crossentropy.
- **Epochs:** 10 (CIFAR-10 dataset).
- **Batch Size:** 64.
- **Data Augmentation:** Optional, for improved generalization.

4.3 Evaluation Metrics

- **Accuracy:** Percentage of correctly classified samples on the test set.
- **Training vs Validation Curves:** Identify overfitting or underfitting trends.
- **3D Depth-Width Surface Plots:** Visualize optimal hyperparameter combinations.

5. Results

5.1 MLP: Depth Analysis

- Increasing depth improves accuracy up to **3 layers**.
- Depth beyond 3 layers yields marginal improvements or slight declines due to **overfitting** and **vanishing gradients**.
- Observations align with literature on deep MLP behavior (Hinton et al., 2006; Glorot & Bengio, 2010).

5.2 MLP: Width Analysis

- Accuracy improves with increasing neurons up to **256 per layer**.
- Width beyond 256 shows minimal gains, highlighting **diminishing returns** and the risk of overfitting.

5.3 CNN: Depth Analysis

- Convolutional depth improves accuracy up to **3 layers**.
- Depth=4 shows plateauing, consistent with prior studies on CIFAR-10 (Krizhevsky et al., 2012).

5.4 CNN: Width Analysis

- Increasing filters enhances feature representation.
- Optimal performance occurs around **64 filters per layer**, balancing computational efficiency and accuracy.

5.5 Combined Depth & Width Analysis

- 3D surface plots reveal that **moderate depth and width** combinations yield optimal accuracy.
- Excessive depth or width increases computational cost without meaningful accuracy gains (Srivastava et al., 2014; Zhang et al., 2017).

5.6 Overfitting and Generalization

- Training accuracy continues to rise, while validation accuracy plateaus after a few epochs.
- Regularization techniques such as **dropout** and **early stopping** help mitigate overfitting.
- This highlights the importance of balancing **network complexity** and **generalization**.

6. Discussion

- **MLP Recommendations:** Depth=3, Width=128–256 for MNIST achieves optimal accuracy.
- **CNN Recommendations:** Convolutional Layers=3, Filters=64 yield optimal CIFAR-10 performance.
- **Hyperparameter Insights:**
 1. Depth improves hierarchical feature learning.
 2. Width increases feature representation capacity.
 3. Optimal performance requires balancing depth, width, and computational cost.

- **Practical Implications:**

- Visualize hyperparameter impact using **3D surface plots**.
- Avoid excessive network sizes to prevent overfitting and reduce computational burden.

- **Educational Value:**

- Provides a visual, intuitive understanding of hyperparameter effects.
- Useful for practitioners to optimize network design.

7. Conclusion

In conclusion, the experimental analysis demonstrates that both depth and width are critical hyperparameters that profoundly influence the performance of neural networks. Specifically, increasing the depth of a network enables it to learn more abstract and hierarchical features, allowing for the modeling of complex patterns within the data, while increasing the width enhances the network's capacity to capture a diverse set of features within each layer. However, this study confirms that excessively deep or wide networks do not necessarily translate into better performance; beyond certain thresholds, additional layers or neurons lead to diminishing returns, increased computational costs, and a heightened risk of overfitting, where the network memorizes the training data but fails to generalize to unseen examples. The findings underscore the importance of carefully balancing depth and width to achieve an optimal trade-off between accuracy, efficiency, and generalization. Through systematic hyperparameter exploration and visualizations such as 3D surface plots, this tutorial provides practical insights into how these architectural choices impact learning and performance. Overall, the analysis highlights that hyperparameter tuning is not only essential for achieving high model accuracy but also crucial for designing neural networks that are computationally efficient and capable of generalizing well. By presenting these insights, this tutorial equips learners and practitioners with a deep understanding of how to make informed, evidence-based decisions when designing and optimizing both Multilayer Perceptron and Convolutional Neural Network architectures, thereby promoting best practices in neural network development and applied machine learning.

References

1. Bengio, Y. (2013). Deep Learning of Representations for Unsupervised and Transfer Learning. ICML Workshop on Unsupervised and Transfer Learning.
2. Brownlee, J. (2020). *Deep Learning for Machine Learning*. Machine Learning Mastery.

3. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
5. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *CVPR*, 770–778.
6. Hinton, G., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7), 1527–1554.
7. Krizhevsky, A., Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical Report, University of Toronto.
8. Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. *NIPS*, 1097–1105.
9. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
10. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1), 1929–1958.
11. Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *ICLR*.
12. TensorFlow (2023). TensorFlow Documentation. Available at: <https://www.tensorflow.org>
13. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
14. Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, 249–256.
15. LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361–3366.
16. Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *ICLR*.
17. Heaton, J. (2017). *Introduction to Neural Networks for Java*. Heaton Research.
18. Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *ICLR*.
19. LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database.
<http://yann.lecun.com/exdb/mnist>

20. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Chapter 6–7: Deep Feedforward Networks & Regularization. MIT Press.