A Systematic Exploration of CNN Hyperparameter Tuning for CIFAR-10 Classification

Laura Fleig [1]

Abstract

Convolutional Neural Networks (CNNs) have become the standard approach for image classification tasks, with research typically focusing on architectural innovations to improve performance, often leaving the impact of hyperparameter selection on performance underexplored. In this paper, we present a systematic investigation of hyperparameter tuning for CNNs on the CIFAR-10 dataset, comparing two distinct architectures: a simple three-layer CNN and a deeper VGG11-style network. We examine 36 different configurations across three dimensions: activation (ReLU, Leaky ReLU, sigmoid), regularization (dropout, batch normalization, both), and optimization (SGD with momentum, Adam). Our results reveal that hyperparameter selection can have a bigger impact than model architecture, with test accuracies ranging from 10% (random guessing) to 88%. We identify interactions between hyperparameters and observe non-learning outcomes for certain configurations. Our findings suggest that hyperparameter exploration should be prioritized over architectural complexity. This study provides empirical guidance for CNN hyperparameter selection, while highlighting the sometimes overlooked importance of these choices in deep learning.

1. Introduction

Image classification represents a foundational task in computer vision and has been a driving force behind many deep learning advances over the past decade. Neural networks are able to automatically learn hierarchical features from raw image data, which has revolutionized the field and enabled unprecedented performance on complex datasets. The CIFAR-10 dataset has emerged as a standard benchmark for evaluating image classification algorithms, particularly for exploring architectural and hyperparameter choices in convolutional neural networks (CNNs).

Despite the remarkable progress in CNN architecture design, the process of selecting optimal hyperparameters remains a critical aspect of model development. A CNN's architecture does not solely determine its performance. Choices in activation functions, regularization techniques, and optimization strategies all influence performance as well. These hyperparameters interact in complex ways, creating a vast system design space that can be difficult to navigate systematically.

We conduct a comprehensive exploration of hyperparameter configurations across two distinct CNN architectures: a basic three-layer CNN and a simplified VGG11-style network. Through this investigation, we seek to answer several research questions: How do different activation functions (ReLU, Leaky ReLU, Sigmoid), regularization techniques (Dropout, Batch Normalization, combination), and optimization choices (SGD, Adam) impact model convergence and final performance across architectures of varying depth? What are the tradeoffs between model complexity, training efficiency, and accuracy?

By systematically varying these hyperparameters and analyzing their effects, we aim to provide empirically-grounded insights that can guide hyperparameter selection. This exploration serves instructive purposes and also contributes to the broader understanding of neural network behavior under different configurations.

This paper is organized as follows: Section 2 details our methodology, including dataset preparation, model architectures, and experimental setup; Section 3 presents our experimental results; Section 4 discusses the implications and limitations of these results and suggests directions for future work; and Section 5 concludes with a summary.

2. Methodology

2.1. Dataset

The CIFAR-10 dataset (Krizhevsky et al., 2009) is a widely used benchmark in computer vision and machine learning research, consisting of 60,000 32x32 color images evenly distributed across 10 mutually exclusive classes: airplane,

[1]Department of Cognitive Science, University of California San Diego, San Diego, USA.

automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is partitioned into 50,000 training images and 10,000 test images. The relatively small size of CIFAR-10 images (32x32) makes it an ideal dataset for extensive hyperparameter tuning since models can be trained to converge in reasonable time frames, while still being sufficiently complex.

For our experiments, we utilized the standard split while implementing several standard preprocessing and data augmentation techniques. Training images were randomly cropped with a padding of 4 pixels and random horizontal flipping, which are common augmentations that can expand the dataset and promote rotational invariance. All images were normalized using channel-wise mean subtraction and standard deviation scaling, which were calculated from the training set. To maintain the integrity of the evaluation process, the test set was only normalized without further augmentation.

2.2. Model Architectures

We focus on two distinct and differently complex CNN architectures: a basic CNN and a simplified VGG11. This selection lets us investigate how hyperparameter choices affect networks of varying depths and capacities.

Basic CNN Architecture. Our basic CNN represents a relatively simple three-layer convolutional network, demonstrating the capabilities of CNNs without being excessively computationally expensive. This architecture consists of:

1. Convolutional Layer 1: 32 filters with 3x3 kernels; activation function, 2x2 max pooling
2. Convolutional Layer 2: 64 filters with 3x3 kernels; activation function; 2x2 max pooling
3. Convolutional Layer 3: 128 filters with 3x3 kernels; activation function; 2x2 max pooling
4. Fully Connected Layer 1: 512 units; activation function
5. Fully Connected Layer 2: 10 units (corresponding to the number of classes)

This architecture follows a conventional pattern of increasing the number of filters with depth, and reducing spatial dimensions through pooling. With around 4.8 million trainable parameters, our basic CNN should capture the features necessary for CIFAR-10 classification while still being computationally manageable.

Simplified VGG11 Architecture. The VGG11 architecture is a deeper network inspired by the original VGG network (Simonyan & Zisserman, 2014). Our implementation is a simplified version, adapted for the smaller input size of CIFAR-10 images. This architecture consists of:

1. Eight convolutional layers: $3 \to 64 \to 128 \to 256 \to 256 \to 512 \to 512 \to 512 \to 512$. Each convolutional layer uses 3x3 kernels with padding.
2. Max pooling layers (2x2) after the 1st, 2nd, 4th, 6th, and 8th convolutional layers.
3. Two fully connected layers: 512 units, followed by 10 output units.

VGG11 uses small receptive fields with greater depth to learn more complex feature hierarchies. With around 28 million parameters, this is a substantially more complex model compared to the basic CNN above.

Both architectures were designed to flexibly configure hyperparameters, allowing for a fair comparison across the hyperparameter space while maintaining the fundamental characteristics of each architectural approach.

2.3. Hyperparameter Space

We explore three primary dimensions: activation functions, regularization techniques, and optimization options. Within each dimension, we selected options representing a variety of approaches in order to have a diverse hyperparameter space.

Activation. We investigated three activation functions with distinct characteristics:

1. ReLU (Rectified Linear Unit): Defined as $f(x) = max(0, x)$, ReLU has become the standard activation function in deep learning, due to its computational efficiency and effectiveness. However, it can suffer from the "dying ReLU" problem, where neurons can become permanently inactive (Lu et al., 2019).
2. Leaky ReLU: Defined as $f(x) = x$ if $x > 0$, else αx (where $\alpha = 0.1$ in our implementation), Leaky ReLU is a variant of ReLU. Allowing a small gradient when $x < 0$ attempts to address the dying neuron problem.
3. Sigmoid: A classical activation function defined as $f(x) = 1/(1 + e^{-x})$, which squashes inputs to the range $(0, 1)$. Sigmoid activations are known to suffer from vanishing gradients and saturation issues in deep networks (Roodschild et al., 2020).

Regularization. To combat overfitting and improve generalization, we explored several regularization configurations:

1. No explicit regularization: Training without dropout or batch normalization serves as our baseline.
2. Dropout ($p = 0.5$): Applied after the activation function in the final convolutional layer and the first fully

connected layer, we randomly zero 50% of activations during training to prevent co-adaptation of neurons.

3. Dropout ($p = 0.5$) + Batch Normalization: Combining dropout with batch normalization is applied after each convolutional and fully connected layer, before activation. This configuration examines potential interactions between these regularization techniques.

Optimization. We compared two widely used optimization algorithms:

1. SGD (Stochastic Gradient Descent) with momentum: Using a learning rate of 0.01 and momentum of 0.9, SGD typically achieves good generalization performance but might require longer training times.
2. Adam (Adaptive Moment Estimation): Using a learning rate of 0.001 and default β parameters (0.9, 0.999), Adam adapts learning rates for each parameter based on first and second moments of gradients, often converging faster.

This hyperparameter space results in 18 distinct configurations (3 activation functions $\times$ 3 regularization settings $\times$ 2 optimizers) for each architecture, creating a total of 36 experimental conditions.

2.4. Experimental Setup

We implemented our models using PyTorch, a popular deep learning framework. Each model configuration was trained for 10 epochs using mini-batch training with a batch size of 128. The relatively small number of epochs is justified by the size of CIFAR-10 and the efficiency of modern optimizers.

During training, we recorded loss and accuracy metrics after each epoch. Training progress was monitored using a cross-entropy training loss and accuracy computed over the entire training set.

Models were evaluated based on several metrics: 1) test accuracy, which measures the proportion of correctly classified test set images; 2) cross-entropy test loss; 3) how learning curves evolve over epochs; 4) training time, to give practical considerations of training efficiency. Overall, holistically looking at these metrics allows us to consider which configurations offer the best trade-offs between accuracy and efficiency.

3. Results

3.1. Overall Comparison

Our experiments revealed considerable variation in performance across the 36 model configurations, with test accuracies ranging from 0.1 to 0.88 (Fig.1). This large spread highlights the critical importance of hyperparameter selection in CNN design and training. The VGG11 architecture demonstrated better classification capability overall, with the top eight performing configurations all being VGG11 variants. The best-performing model was a VGG11 network with ReLU activation, Adam optimizer, and both dropout and batch normalization, achieving a test accuracy of 0.8801. For comparison, the best basic CNN configuration (also using ReLU, Adam, dropout, and batch normalization) reached 0.8109 accuracy.

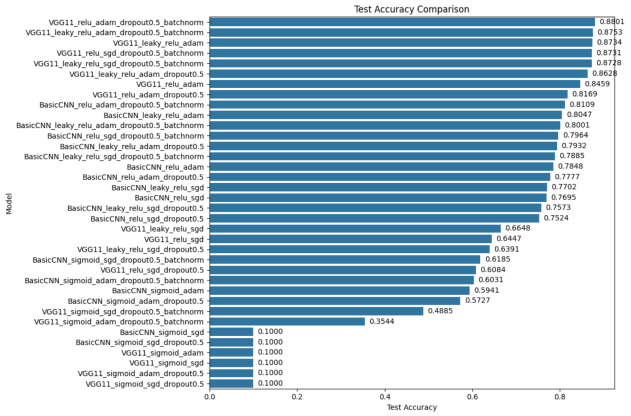

Figure 1. A comparison of test accuracy across all 36 model configurations.

However, architectural complexity alone did not guarantee superior performance. Notably, among the worst-performing models (those achieving only 0.1 test accuracy, equivalent to random guessing), four were VGG11 variants and two were basic CNNs. This finding emphasizes that deeper networks, while offering greater representational capacity, are also more sensitive to suboptimal hyperparameter configurations.

The training time analysis showed expected differences between architectures, with VGG11 models requiring substantially more computational resources (296-342 seconds) compared to basic CNN models (186-198 seconds). Models implementing batch normalization generally exhibited longer training times, reflecting the additional computational overhead of this regularization technique. This presents a clear trade-off between performance and computational efficiency.

See Fig.2 and Fig.3 for hyperparameter heatmaps for each of the model architectures. See Appendix A for training accuracy and loss graphs, along with a graph comparing training time.

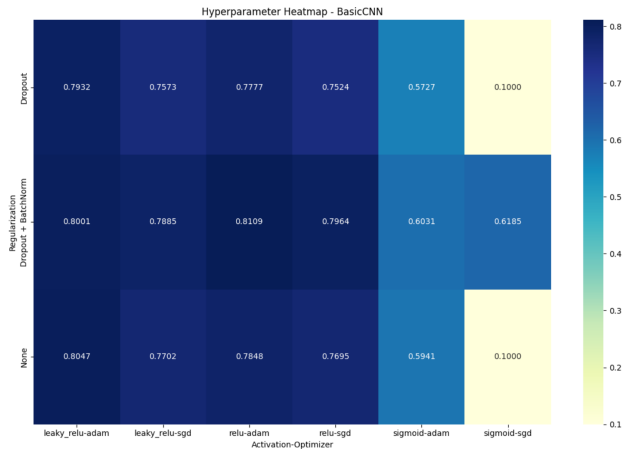

Figure 2. A heatmap of all 18 hyperparameter configurations for the basic CNN.

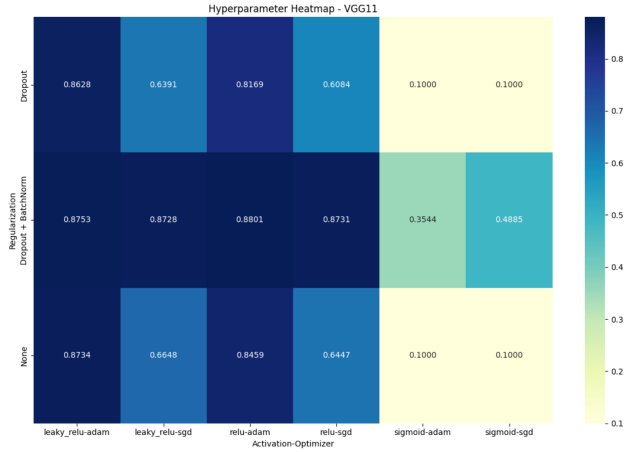

Figure 3. A heatmap of all 18 hyperparameter configurations for the VGG11 network.

3.2. Effect of Activation Functions

Activation function choice was a critical determinant of performance and had notable differences across architectures. Sigmoid activation consistently underperformed relative to ReLU and Leaky ReLU across both architectures, particularly in the VGG11 network where sigmoid activation appeared in all of the worst-performing configurations. This can be attributed to the vanishing gradient problem, which becomes increasingly severe in deeper architectures due to the saturation characteristics of the sigmoid function.

The relative performance of ReLU vs. Leaky ReLU varied by architecture. For VGG11 models, Leaky ReLU outperformed standard ReLU. Meanwhile, the basic CNN models showed mixed results between these two activation functions, with some ReLU configurations outperforming their Leaky ReLU counterparts and vice versa. This finding indicates that the benefits of Leaky ReLU's non-zero negative gradient may be more significant in deeper networks where gradient propagation is more challenging.

3.3. Effect of Regularization Techniques

Our experiments confirmed the effectiveness of regularization, with combined dropout and batch normalization yielding the highest performance in both architectures. The top-performing models for both the basic CNN and VGG11 used the combined approach (resulting in test accuracies of 0.81 and 0.88, respectively).

The interaction between regularization and activation functions also showed interesting results. Models using sigmoid activation showed limited benefit from regularization, suggesting that the primary limitation in these configurations was the activation function's constraints on gradient flow, rather than overfitting. In contrast, ReLU and Leaky ReLU activations showed substantial improvements with regularization, particularly in VGG11 configurations.

3.4. Effect of Optimization Techniques

Adam consistently outperformed SGD with momentum across both architectures. Interestingly, optimization choice interacted with activation function choice, particularly for the basic CNN. For both settings with no regularization or just dropout, the basic CNN sigmoid-adam models performed significantly better than the sigmoid-sgd models (around 0.58 accuracy, compared to 0.1 accuracy). This highlights the importance of considering hyperparameters holistically rather than individually, as the interdependencies can greatly influence performance.

3.5. Architecture-Specific Effects

Our experiments displayed distinct patterns in how hyperparameters impact performance across different architecture complexities. The basic CNN architecture, having a simpler structure and fewer parameters, demonstrated greater robustness to hyperparameter choices, having reasonable performance across a wider range of configurations. In contrast, the VGG11 architecture showed higher variance in performance, achieving both the highest and some of the lowest accuracies. This difference in sensitivity was particularly evident with sigmoid activation, which severely impaired VGG11 performance, while allowing the basic CNN to achieve moderate (but suboptimal) accuracy. The deeper network presumably amplifies the vanishing gradient problem inherent to sigmoid activation, effectively preventing meaningful learning.

4. Discussion

Our experimental results show several insights with broader implications for deep learning practice, not only confirming some established heuristics but also challenging some assumptions about hyperparameter selection.

The consistently poor performance of sigmoid activation empirically confirms the field's existing concerns about vanishing gradients. ReLU and its variants likely pose a better option.

Leaky ReLU outperformed standard ReLU in VGG11 models, compared to mixed results in the shallower networks. This suggests that the dying ReLU problem, where neurons can become inactive during training, may be more prevalent in deeper networks with more complex feature spaces. As networks grow deeper, it becomes more likely that some neurons will only receive negative inputs throughout training, rendering them useless with standard ReLU activation. The small gradient allowed by Leaky ReLU for negative inputs seems to provide a good way to maintain activity in deeper networks.

We found that Adam tended to outperform SGD, which does not align with some recent findings, e.g. (Gupta et al., 2021). However, one consideration is that our experiments only used 10 epochs and a moderate-sized dataset, and SGD typically shows slower but steadier improvement over longer training periods.

The interaction between regularization techniques also yielded interesting results. We achieved the best performance when combining dropout and batch normalization, contrary to some concerns about potential interference between these methods (Li et al., 2019). However, our results align with other studies that emphasize the benefit of batch normalization over dropout (Garbin et al., 2020), as the addition of batch normalization drastically improved performance for many VGG11 models.

4.1. Analysis of Non-Learning Models

Arguably the most notable observation is the range of performance outcomes (10% to 88% accuracy) achieved simply by varying hyperparameters while keeping the underlying architectures fixed. Our findings highlight how critical hyperparameter selection is in neural network design, potentially more important than the base architecture itself. Potentially most intriguingly, certain hyperparameter combinations resulted in models performing no better than random guessing (10% accuracy) throughout the entire training process. These optimization failures occurred more frequently in the deeper architecture and were exclusively associated with sigmoid activation functions. Adding batch normalization without changing anything else helped drive model performance up to 35%, again highlighting the importance of normalization for learning to occur at all. Without appropriate normalization techniques, sigmoid-activated networks (especially deeper ones) might never begin to learn, regardless of training duration or data quality.

Additionally, the "non-learning" configurations may also be partially explained by unlucky weight initialization, placing the network in regions with virtually flat gradients from which optimization algorithms cannot escape. We emphasize the importance of proper initialization strategies.

4.2. Analysis of Computational Efficiency

Beyond performance, computational efficiency is also a consideration. While VGG11 did achieve higher peak performance, the basic CNN reached 81% accuracy with around 40% less training time. For applications with limited computational resources, simpler architectures might be more practical, particularly if the performance gap is less of a concern.

4.3. Limitations and Future Work

A limitation of our study is the fixed training duration of 10 epochs, which may not allow all configurations to fully learn. Future work could extend our analysis by training to convergence or implementing early stopping based on validation performance. Furthermore, exploring a wider range of learning rates, momentum values, and dropout rates could provide more insights into the interactions between hyperparameters and architectures.

5. Conclusion

We have demonstrated that hyperparameter selection can be incredibly important, perhaps more important than architectural differences themselves. The wide range in accuracy achieved solely through different hyperparameter configurations of the same base architectures underscores an often underappreciated aspect of deep learning research. A well-tuned "basic" architecture can potentially outperform a poorly tuned "advanced" architecture.

Our exploration also revealed that optimal configurations are likely highly dependent on context, dataset, and task. The complex interactions between activation, regularization, and optimization suggest that hyperparameter recommendations should not be universally applied. Instead, we recommend conducting a thorough exploration tailored to one's application context.

Developing more efficient methods for hyperparameter search remains an important challenge, as the number of possible configurations can be impractical for larger hyperparameter spaces. The deep learning field as a whole would greatly benefit from more in-depth research on improved

prediction of hyperparameters based on dataset and model architecture in order to leverage the full potential of neural networks across diverse applications.

Supplementary Materials

We provide our code as a Jupyter notebook at https://github.com/laurafleig/cogs181-final.

References

Garbin, C., Zhu, X., and Marques, O. Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia tools and applications*, 79(19): 12777–12815, 2020.

Gupta, A., Ramanath, R., Shi, J., and Keerthi, S. S. Adam vs. sgd: Closing the generalization gap on image classification. In *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*, pp. 1–7, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Li, X., Chen, S., Hu, X., and Yang, J. Understanding the disharmony between dropout and batch normalization by variance shift. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2682–2690, 2019.

Lu, L., Shin, Y., Su, Y., and Karniadakis, G. E. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

Roodschild, M., Gotay Sardiñas, J., and Will, A. A new approach for the vanishing gradient problem on sigmoid activation. *Progress in Artificial Intelligence*, 9(4):351–360, 2020.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

A. Appendix

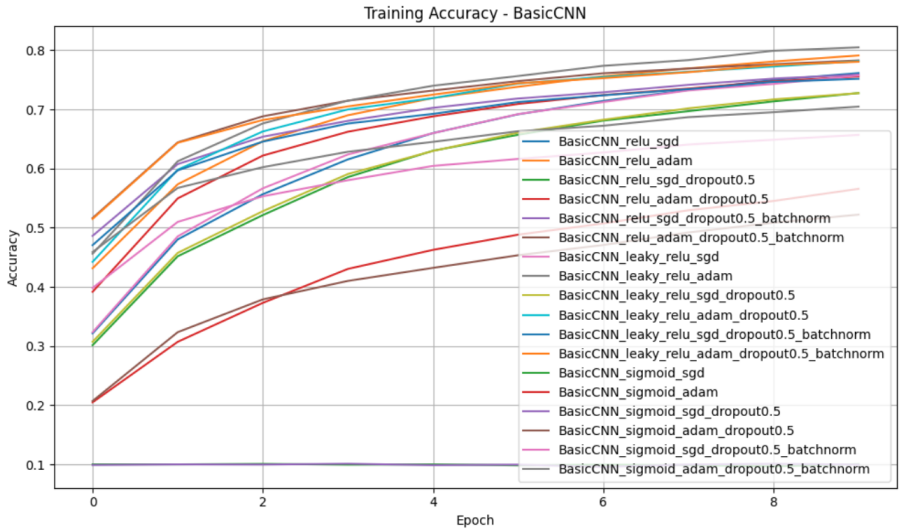

Figure 4. Training accuracy for the basic CNN configurations.

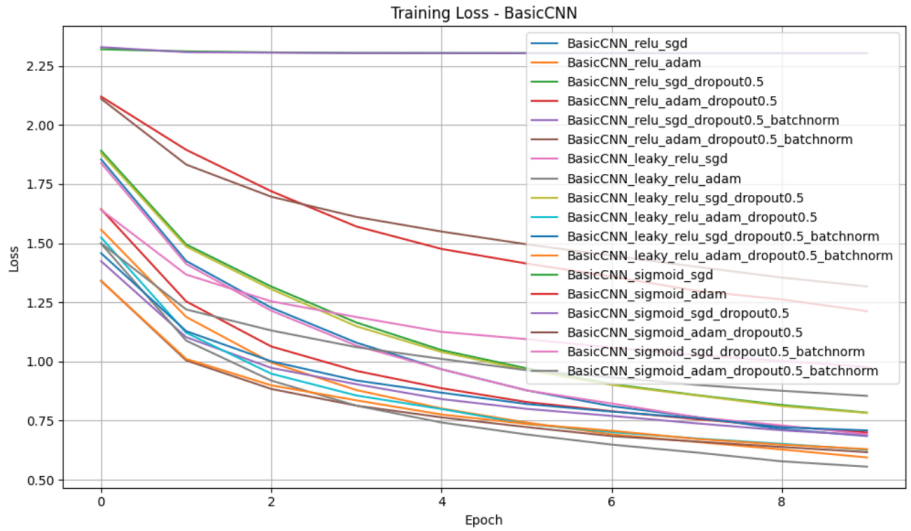

Figure 5. Training loss for the basic CNN configurations.

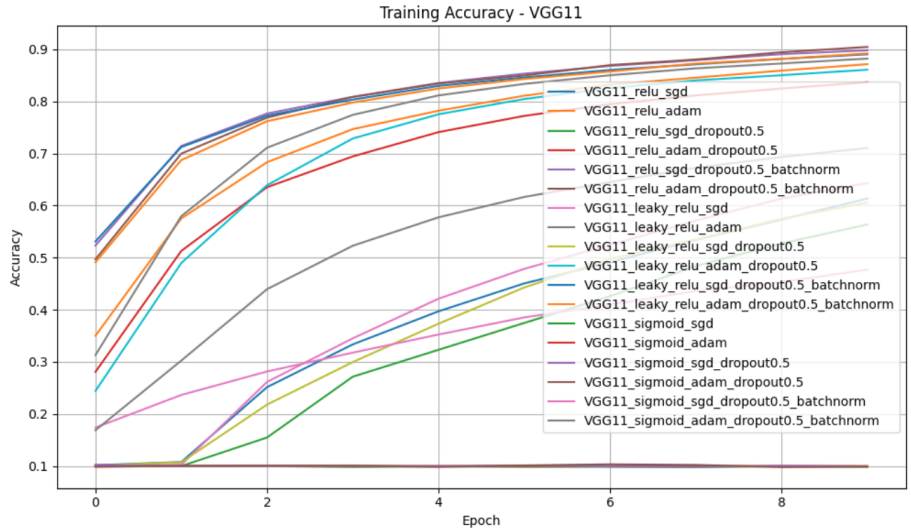

Figure 6. Training accuracy for the VGG11 configurations.

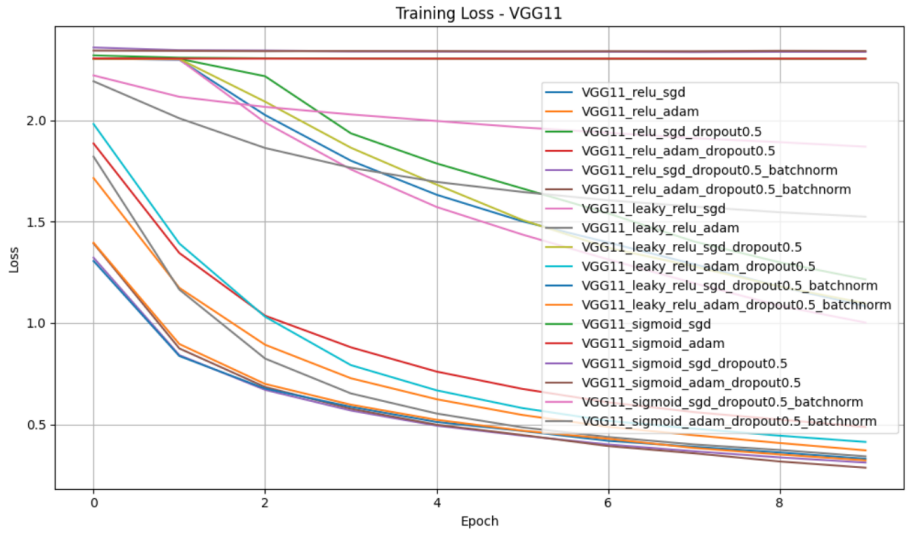

Figure 7. Training loss for the VGG11 configurations.

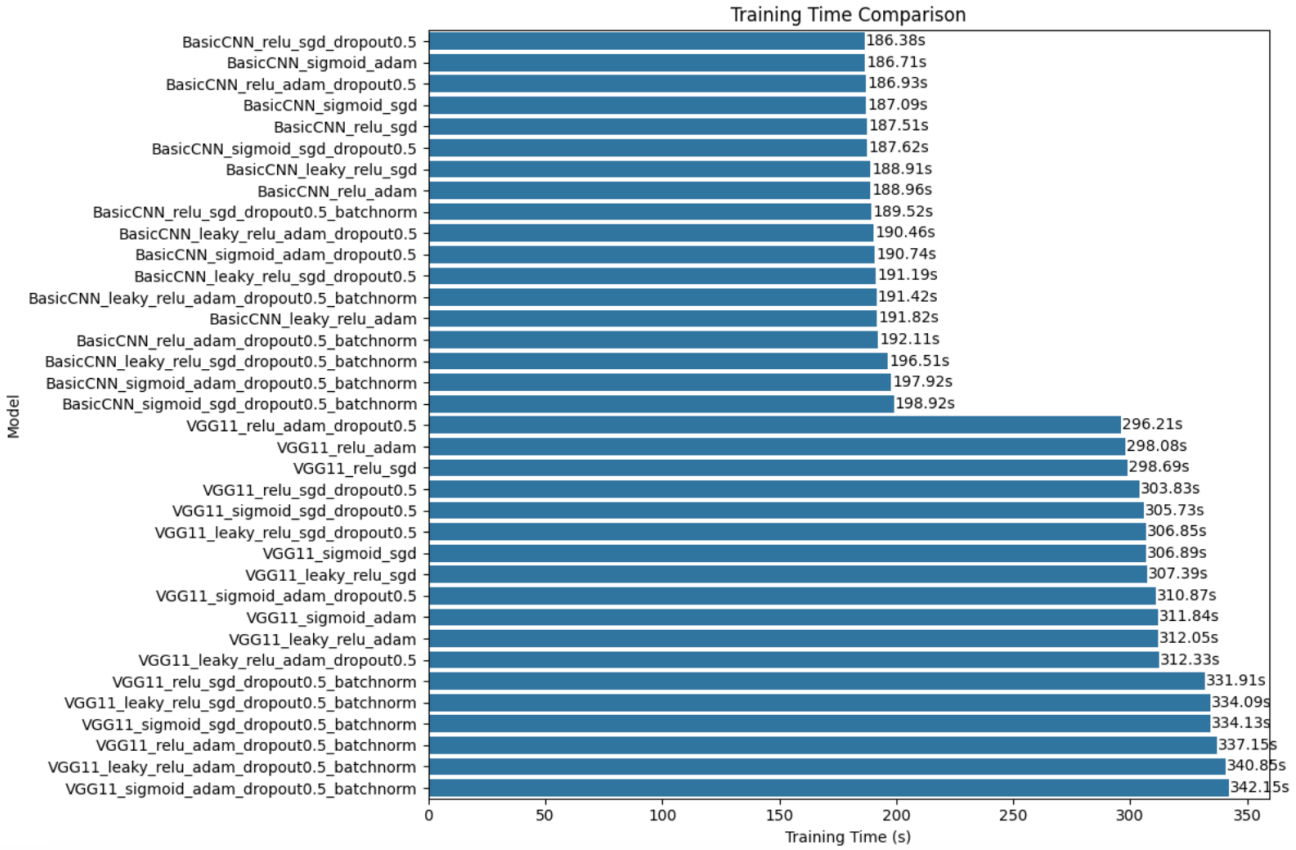

Figure 8. A comparison of training time across all 36 model configurations.