Final assignment
When Less is More (Until it Isn't)
- A Double Descent Story in Machine Learning
GRA4157

Jakob Sverre Alexandersen

ja.al@fsncapital.com

December 4, 2025

Contents

1 Introduction **2**

2 Methodology **2**

2.1 Data 2

2.2 Model Architectures 4

2.2.1 Simple CNN (MNIST) 4

2.2.2 FlexibleCNN 4

2.3 Design Choices 4

3 Results **5**

4 Summary **6**

1 Introduction

Click me to view the full code on Github!

This final project builds (conceptually) on my second project. A short recap of that was to go in-depth on how CNNs (Convolutional Neural Networks) are actually built. To demonstrate this, I built a relatively simple classifier (found in the `simple_bootleg_cnn_classifier.py` file). Furthermore, I ran it on the well-known MNIST dataset, which is a balanced dataset of $70,000$ images of the digits 0 to 9. It achieved both an $F1$ and accuracy score of 99%, which is considered industry standard. Since the problem is considered solved, it inspires very little motivation to do further research on the subject.

In my previous project, I very briefly mentioned Belkin et. al's theory of *double descent* [BHMM19], which challenges the traditional bias-variance tradeoff. Subsequent empirical work by Nakkiran et. al at OpenAI demonstrated this phenomenon across various model architectures including CNNs, ResNets and Transformers [NKB$^+$19]. The theory posits that test error exhibits non-monotonic relationship with model complexity and training time: performance initially worsens after the interpolation threshold (where the model perfectly fits training data), but surprisingly improves again with further increases in complexity or training epochs. However, MNIST's simplicity, with near-perfect accuracy already achieved, provides limited opportunity to empirically observe this phenomenon.

To meaningfully test double descent, I require a more challenging dataset where my model has substantial room for improvement. Therefore I will apply my CNN architecture to the CIFAR-10 dataset, which consists of $60,000$ 32×32 color images across 10 classes (airplane, car, bird, cat, deer, dog, frog, horse, ship and truck). CIFAR-10 presents significantly greater complexity due to color channels, natural image variations, and inter-class similarity, typically yielding baseline accuracies of $70 - 85\%$ without advanced techniques.

The objective of this project therefore to investigate whether double descent can be empirically observed by varying model width, using methodology accessible with a single consumer GPU. While these results have been established with large-scale experiments requiring thousands of training epochs and significant computational resources, demonstrating that the core n is observable with more modest setups, a simple two-block CNN trained for only 200 epochs on a single GPU, has practical value for "us folks at home" without access to extensive compute and large research teams

2 Methodology

The experimental design is a modified version of the previously mentioned simple CNN. Rather than optimizing hyperparameters to achieve the best possible performance on a single model, we systematically vary model width across a fixed range and evaluate test error for each configuration. This approach allows us to observe the complete double descent curve, the relationship between model capacity and generalization error, rather than selecting a single optimal point. The test error is reported across all width configurations for scientific observation of the phenomenon. This differs from standard hyperparameter tuning where test data would be reserved solely for final evaluation after validation-based model selection. This is important to state as we are not actually conducting statistical malpractice by allowing data leakage.

As mentioned, we use the CIFAR-10 dataset, which originally consists of 60,000 images in total. The dataset was obtained via PyTorch's torchvision library. The original training set of 50,000 (which includes validation data) images is first split into training (35k samples) and validation (15k samples) sets. The test set of 10,000 is reserved for final evaluation for each model. For the width-sweep experiments, we further subsample the training set to 10k samples. This reduction serves a methodological purpose: with fewer training examples, models reach the interpolation threshold at lower parameter counts, allowing us to observe the complete double descent curve within our range of width multipliers. Importantly, the validation and test sets remain at their full sizes, ensuring reliable evaluation metrics. The validation set is used during training to monitor loss and accuracy, while the test set is used exclusively for final evaluation to construct the double descent curve.

2.1 Data

Before we dive into the architectures of the different models and other techinicals, it is important to achieve an understanding of the data itself. Therefore we start off by seeing how the distribution of the original data looks like, to ensure that the data is indeed balanced.

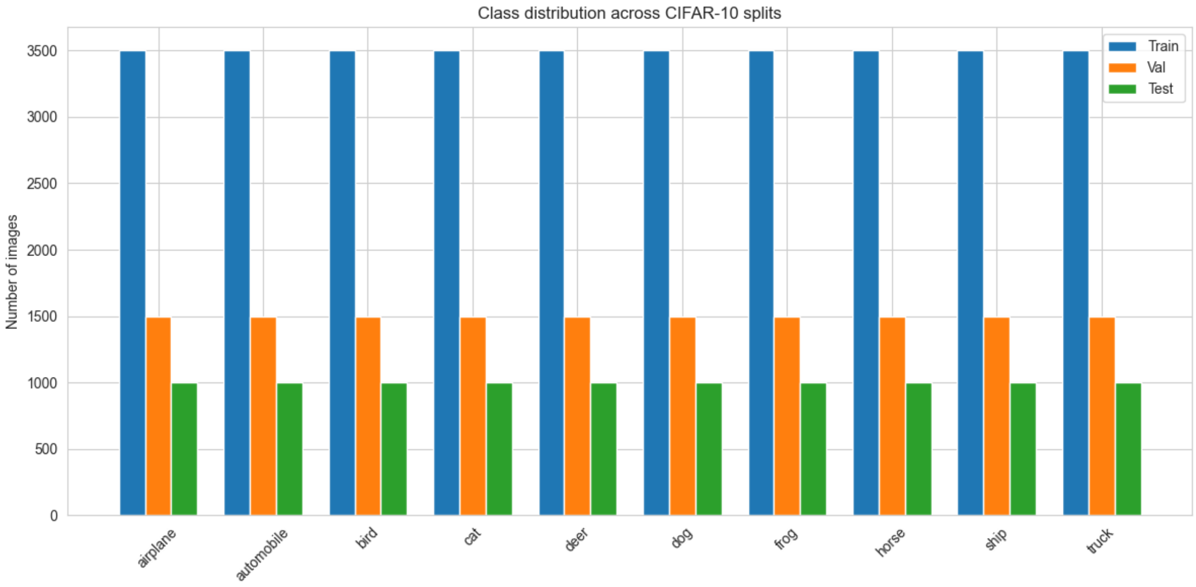

Figure 1: Class distribution across CIFAR-10 splits. *Note: This is the original dataset, not the subsampled dataset.*

The original data is balanced (this also implies that the subsampled training data also is rougly balanced), which means that we will not accidentally train a biased model that performs poorly on underrepresented classes. If that were the case, the model may have struggled to correctly identify the classes with fewer examples, potentially resulting in lower accuracy and recall for those classes.

If we run a tSNE analysis on the full training set, we get this plot:

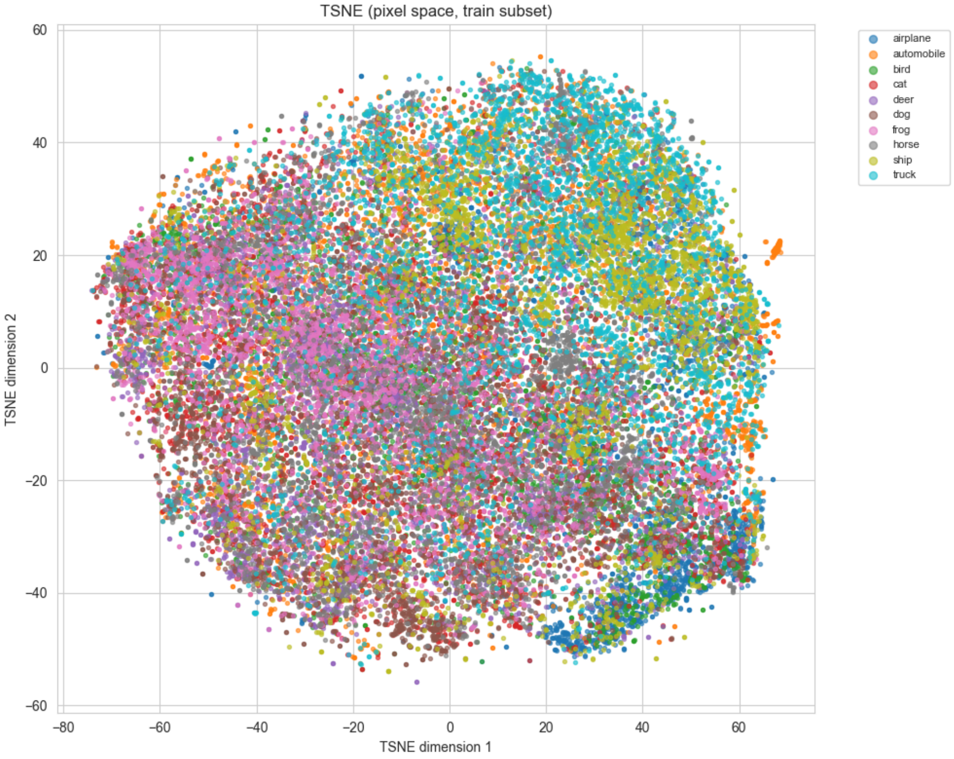

Figure 2: tSNE plot of the classes with two components and perplexity=30, with automatic learning rate

We can already tell from here that the models themselves are not going spark impressive feedback, as the classes are very mixed into each other, and there is no clear separation between clusters. This, however, is not really a problem as the goal we're trying to achieve is not the best model.

2.2 Model Architectures

Previously I mentioned that I initially started with the CNN in `simple_bootleg_cnn_classifier.py` (we will call this the simple CNN from now on). The CNN used for this experiment ended up being quite different as I had to make adjustments to make it work with the CIFAR-10 dataset.

2.2.1 Simple CNN (MNIST)

The simple CNN begins with a convolutional layer that accepts one input channel (grayscale) and produces 16 output channels by applying 16 different 3×3 filters. Each filter learns to detect different low-level features such as edges or corners. A ReLU activation introduces non-linearity, and max pooling reduces the spatial dimensions from 28×28 to 14×14.

The second convolutional layer takes these 16 feature maps and produces 32 channels, learning to combine low-level features into higher-level patterns. After another pooling operation, the dimensions are reduced to 7×7, yielding a $32 \times 7 \times 7$ tensor.

The fully connected section flattens this into 1,568 values, maps them through a 64-unit hidden layer with dropout regularization, and finally outputs probabilities for the ten digit classes. This hierarchical structure—simple patterns in early layers combining into complex features in later layers—is what makes CNNs effective for image classification.

2.2.2 FlexibleCNN

The FlexibleCNN used for the width-sweep experiments follows the same architectural pattern as the simple CNN, but introduces key modifications to enable systematic variation of model capacity. Instead of fixed channel counts, the FlexibleCNN parametrizes its width using a multiplier k. The first convolutional layer accepts three input channels (since we are working with RGB images instead of grayscale images) and produces 8k output channels, while the second layer expands to 16k channels. Similarly, the fully connected hidden layer scales to 32k units. This means that when $k = 1$, the model has relatively few parameters, but as k increases to values like 64 or 128, the model grows to millions of parameters, specifically around 555 million for the latter k

This scaling approach differs fundamentally from the simple CNN's fixed architecture. Where the simple CNN uses 16 and 32 filters in its convolutional layers and 64 hidden units, totalling around 100k parameters. In contrast, the FlexibleCNN spans a much wider range. At $k = 1$ it has roughly 35,000 parameters, placing in the underparameterized regime where it lacks sufficient capacity to memorize the training data. As mentioned, at $k = 128$ it has roughly half a billion parameters, firmly in the overparameterized regime where it has far more capacity than needed to fit the training set.

Another difference is the removal of dropout regularization. The simple CNN applies 25% dropout on its fully connected layer to prevent overfitting and improve generalization. However, for observing double descent, dropout is set to zero in the FlexibleCNN. Regularization techniques like dropout smooth out the sharp transition at the interpolation threshold, which is the point where the model has just barely enough capacity to perfectly fit the training data. By removing this regularization, the characteristic peak in test error at the interpolation threshold becomes visible, allowing empirical observation of the double descent phenomenon.

The input dimensions also change to accomodate CIFAR-10's 32×32 color images compared to MNIST's 28×28 grayscale images. After two convolutional blocks with max pooling, the spatial dimensions reduce to 8×8 (rather than 7×7 in the simple CNN), resulting in a flattened size of 1024k values before the fully connected layers. This larger input combined with the scaled layer widths allows the model to handle the increased complexity of natural color images while providing the fine-grained control over model capacity necessary for probing different regimes of the bias-variance-capacity landscape.

2.3 Design Choices

A critical design choice in these experiments is the introduction of 15% label noise to the training data. This means that 1,500 of the 10,000 training images have their true labels randomly replaced with incorrect class labels. This deliberate corruption serves an important methodological purpose, which is to make the interpolation threshold more visible and the double descent peak more pronounced.

To understand why label noise is necessary, consider what happens at the interpolation threshold without it. When all training labels are correct, a model that perfectly fits the training data has essentially learned the true underlying patterns in the images. The transition from underfitting to interpolation is relatively smooth, and the test error peak may be subtle or difficult to observe. However, when a fraction of the labels are corrupted, a model at the interpolation threshold must memorize not only the true patterns but also the arbitrary noise in the mislabeled examples. This memorization of random label assignments cannot possibly generalize to the test set, where labels are correct. The result is a sharper, more visible spike in test error at precisely the point where the model gains enough capacity to fit the corrupted training data.

Convolutional neural networks were chosen over alternative such as fully-connected networks due to their ability to exploit spatial structure in images. The Adam optimizer with learning rate 0.001 was used for all experiments, selected for its adaptive learning rate properties and stable convergence across all model widths. The choice to vary model width rather than depth follows the methodology of Nakkiran et al., as width scaling provides fine-grained control over parameter count while keeping architectural complexity constant.

3 Results

The experimental results reveal a clear manifestation of the double descent phenomenon. As shown in Figure 3, test error exhibits non-monotonic behavior as model width increases. In the underparameterized regime, $k < 10$, test error initially decreases as the model gains sufficient capacity to learn meaningful representations. The lowest test error of 45% occurs at $k = 10$, after which the error begins to rise as the model approaches the interpolation threshold.

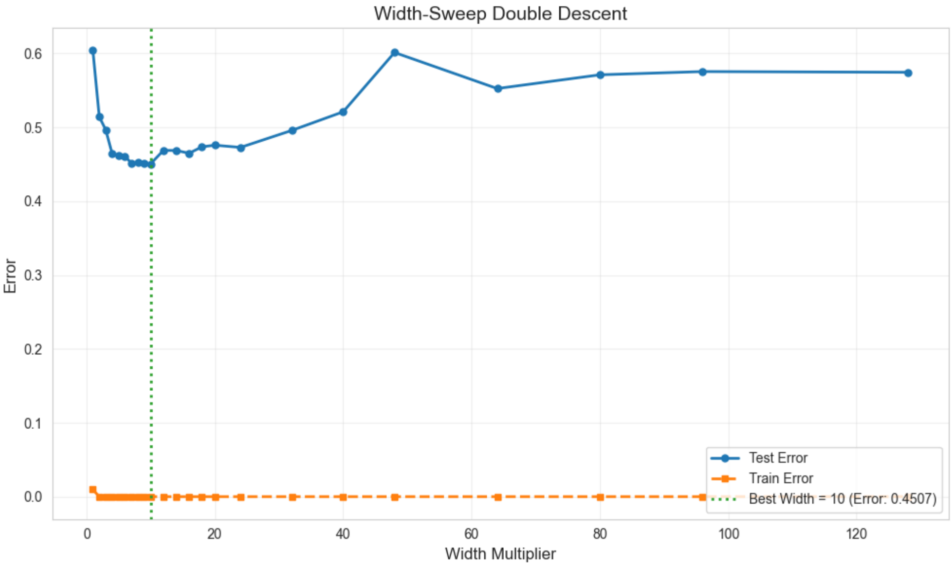

Figure 3: Clear manifestation of the double descent phenomenon

The characteristic peak in test error occurs around k=48, where test error reaches approximately 60%. This corresponds to the interpolation threshold, where models can memorize the training set (note the ≈ 0 train error on almost all models) but generalize poorly to unseen data. At this critical capacity, models can effectively memorize the training set including the 15% artificially corrupted labels, but this memorization comes at the cost of generalization to unseen data. The gap between train and test error is widest at this point, representing maximum overfitting.

Beyond the interpolation threshold, in the overparameterized regime, test error decreases again despite the model having far more capacity than necessary to fit the training data. This counterintuitive recovery is the hallmark of double descent. At $k = 128$, with over 555 million parameters ($1/3$ of the original GPT-2 size), the test error stabilizes around 57%. While this still is higher than the optimal point at $k = 10$, it demonstrates the characteristic recovery predicted by double descent theory: extremely overparameterized models can generalize better than models at the interpolation threshold.

It is important to note that these experiments were conducted with 200 training epochs per model. According to Nakkiran et. al [NKB+19], the double descent peak becomes substantially more pronounced with additional training epochs. Longer training allows model near the interpolation threshold to more completely memorize the training data, sharpening the distinction between the underparameterized and overparameterized regimes. The original experiments demonstrating model-wise double descent used up to 4,000 epochs for similar architectures, and the resulting curves showed dramatically sharper peaks and more pronounced recovery in the overparameterized regime.

4 Summary

Despite having an RTX5090 at my disposal, extending these experiments to 4,000 epochs would require either substantial additional computational resources or time (none of which I have). Training 23 models of varying widths for 4,000 epochs each, with the largest model exceeding 555 million parameters, would represent a significant investment in compute time, likely requiring several days continuous GPU utilization. The results presented here therefore represent a computationally tractable demonstration of the phenomenon; the double descent curve would likely be even more striking given additional training epochs, with a sharper peak at the interpolation threshold and more complete recovery in the overparameterized regime.

The practical implications of these findings are noteworthy. Traditional machine learning wisdom, rooted in the bias-variance tradeoff, suggests that models should be sized to balance underfitting and overfitting. The double descent phenomenon challenges this intuition; if computational resources permit, practitioners may benefit from using substantially overparameterized models rather than carefully tuned “just right” architectures. However, as our experiments demonstrate, the optimal choice depends heavily on available compute, one of the smallest models actually achieved the best test error in our 200-epoch setting, suggesting that for limited training budgets, smaller models near the first descent minimum may be preferable to massive overparameterized networks that require extensive training to realize their generalization benefits.

References

[BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, July 2019.

[NKB+19] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019.