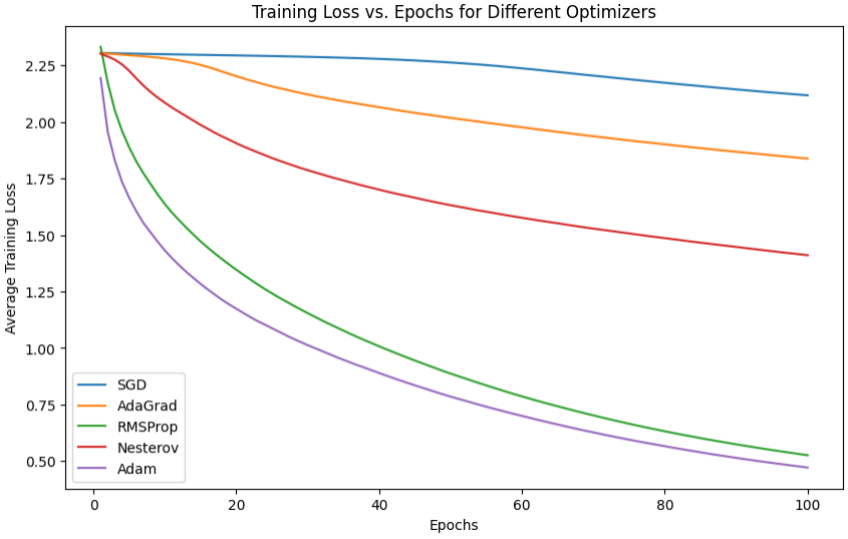
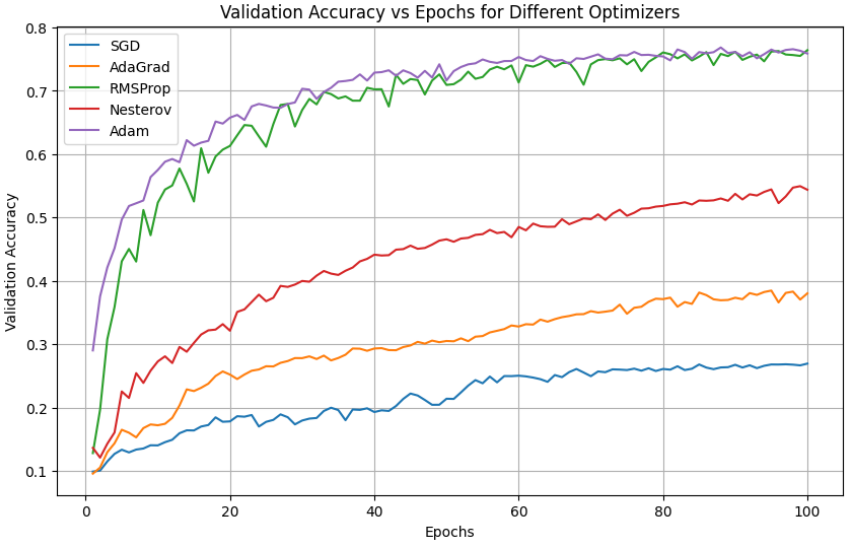
Candidate Number: 49580

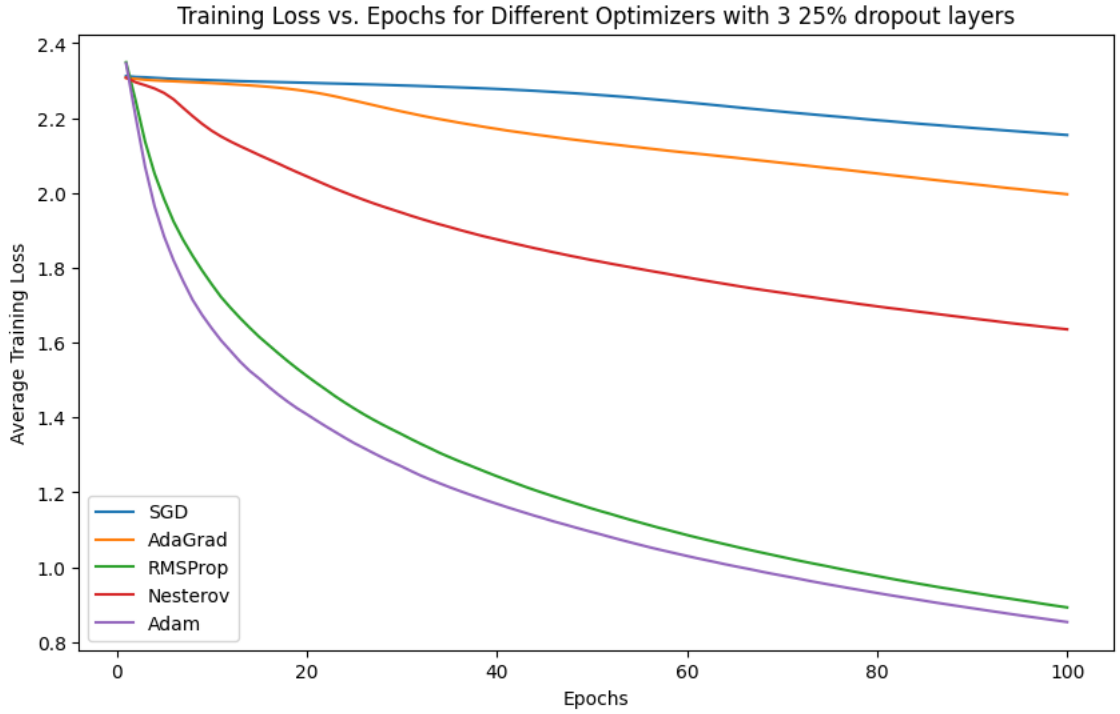
P1. CIFAR10 CNN: training using minibatch gradient descent algorithms

- The lowest training loss is observed in Adam and RMSProp
- This also tallies with those 2 optimizers having the highest validation accuracy.
- Why? Adaptive learning rates.
- Both Adam and RMSProp use per-parameter learning rates that adapt based on the history of gradients meaning they can adjust the learning rate individually for each weight in the network. In complex image datasets like CIFAR-10, different features have different importance and update frequencies. Adam and RMSProp perform better because they can make larger updates for infrequently occurring features and smaller updates for frequent features.
- Adam outperforms RMSProp as it combines the benefits of RMSProp with momentum. It keeps track of both:
    - First moment (mean of gradients) - similar to momentum
    - Second moment (uncentered variance of gradients) - similar to RMSProp

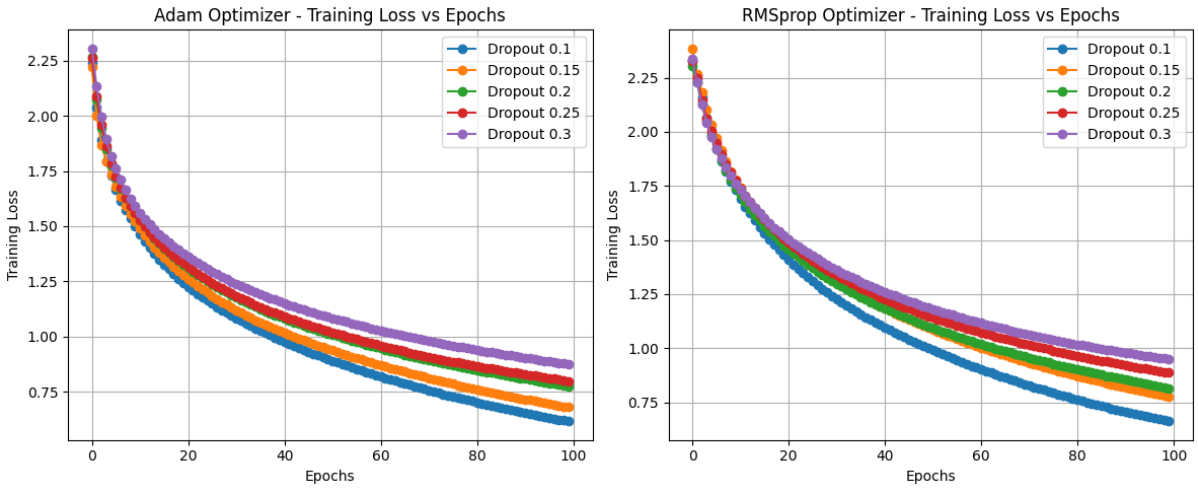




- Then we add 3 25% dropout layers each. Adding dropout increases training loss across all the optimisers, and that is a very much expected result. Dropout de-activates neurons in the training process but helps reduce overfit which should show up as higher training loss but also lower validation loss.



- Playing with different dropout rates for the best 2 optimizers does show that increasing dropout just increases training loss.



P2. CIFAR10 image classification

The following table summarises the model architectures.

Feature	Model 1 (Simple CNN)	Model 2 (Standard ResNet)	Model 3 (Custom ResNet)
Architecture	Sequential CNN with 3 convolutional blocks	ResNet9 with 2 main residual blocks	Variable-depth ResNet (depth=20) with multiple residual blocks across 3 stages
Filter Progression	32→64→128	64→128→256→512 (wider)	16→32→64 (narrower but deeper)
Pooling	MaxPooling after each block	MaxPooling for downsampling	Strided convolutions for downsampling, GlobalAveragePooling
Regularization	Dropout, BatchNormalization	L2 weight decay applied to all layers	Dropout within residual blocks, adaptive learning rate reduction
Data Augmentation	None	Basic (horizontal flips and translations)	Extensive (flips, rotations, zooms, translations)
Optimization	Standard Adam with default learning rate	Fixed learning rate with AdamW and gradient clipping	AdamW with ReduceLROnPlateau
Batch Size	32	400	128
Average Loss	0.524	0.367	0.421
Top 1 Error-rate	0.173	0.111	0.127
Top 5 Error-rate	0.010	0.004	0.005

Rationale

Why start with Model 1:

- Simplicity - It follows a straightforward architecture that's easier to implement and debug
- Clear learning path - The sequential structure makes it easier to understand how information flows through the network
- Computational efficiency - With fewer parameters and a smaller batch size (32), it requires less computing power to train

Why improve to Model 2:

- Residual connections solve vanishing gradient problems
- Wider networks (64→128→256→512) capture more complex patterns
- L2 regularization prevents overfitting more effectively

- Performance improves dramatically (error drops from 17.3% to 11.1%)

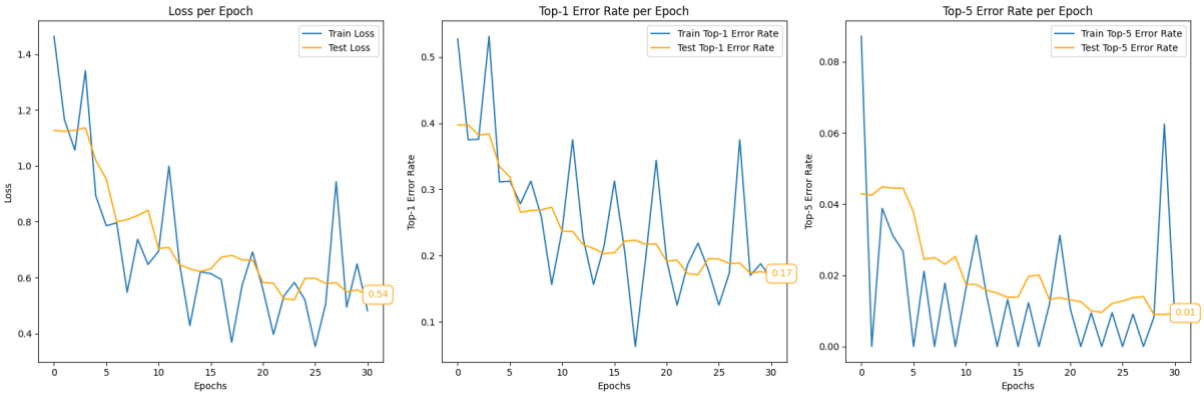
Why evolve to Model 3:

- Deeper architecture (20 layers) learns more hierarchical features
- More parameter-efficient with fewer filters per layer
- Advanced techniques (strided convolutions, adaptive learning rates)
- Robust data augmentation improves generalization
- Longer training allows complex architecture to fully converge

This progression follows a natural learning curve: establish basics, then add proven techniques for substantial gains, then refine with sophisticated approaches.

Results

Model 1 - Average Loss: 0.524, Top 1 Error-rate: 0.173, Top 5 Error-rate: 0.010

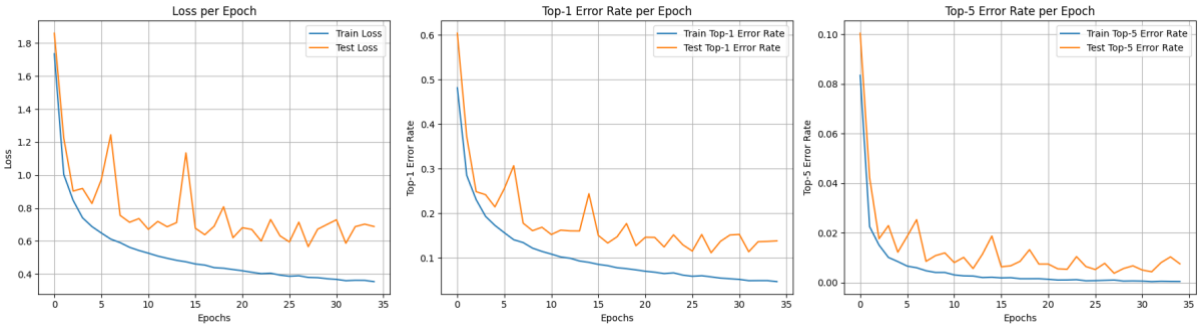


The spiky training errors in my graphs are likely due to the following factors:

1. **Small batch size**: Model 1 uses a batch size of only 32. With such small batches, each one might contain very different distributions of easy and difficult examples, causing high variance in the loss calculations from batch to batch.
2. **Insufficient smoothing**: The training metrics are likely being reported after each epoch without any smoothing mechanism.
3. **High learning rate**: The default Adam learning rate might be too aggressive for this model architecture, causing oscillations as the optimizer overshoots minima in the loss landscape.
4. **Lack of regularization**: While I have dropout, the model doesn't use more advanced regularization techniques like weight decay or data augmentation, making it more susceptible to fitting to noise in individual batches.
5. **Inconsistent data difficulty**: CIFAR-10 contains various object categories with different levels of difficulty. Batches with more challenging categories create spikes in the error rate.

Notice that the validation curves (yellow) are much smoother by comparison. This is because validation is typically run on the entire validation set at once, averaging out these variations.

Model 2 - Average Training Loss: 0.367, Top 1 Error-rate: 0.111, Top 5 Error-rate: 0.004



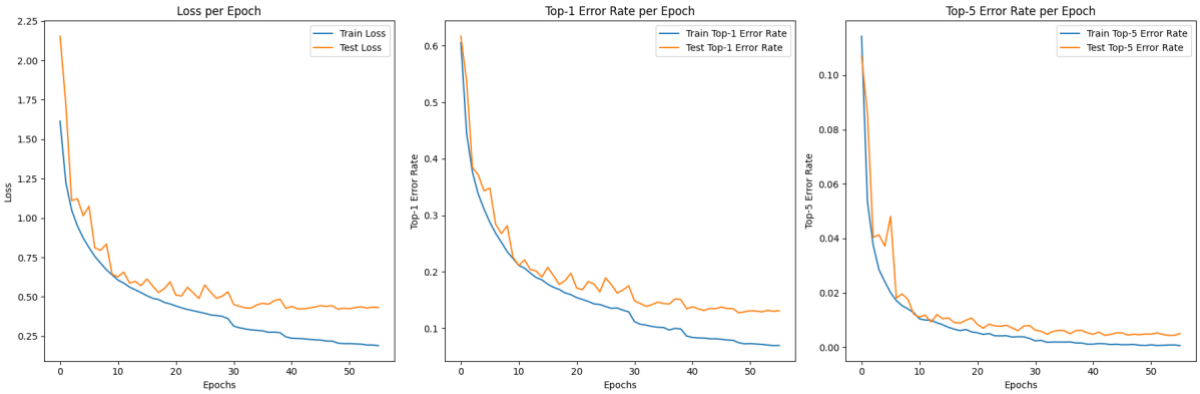
Model 2 shows clear improvements over Model 1:

- Smoother training curve (blue line) with consistent downward trajectory
- Lower overall loss (reaching ~0.3 vs ~0.45 in Model 1)
- Better validation metrics with Top-1 error rate dropping to ~0.11
- Smaller gap between training and validation performance
- More stable learning with fewer dramatic spikes in validation

ResNet specifically helps because:

- Residual connections allow information to skip layers via identity shortcuts
- These shortcuts create gradient highways during backpropagation, preventing vanishing gradients
- The network can learn optimal depth by effectively "turning off" unnecessary layers
- Error signals propagate more efficiently through the entire network
- Model learns both simple and complex features simultaneously at different depths

Model 3: Average Loss: 0.432, Top 1 Error-rate: 0.131, Top 5 Error-rate: 0.004



Note that Model 3 underperforms Model 2 on a pure validation accuracy basis. I briefly discuss this result but then also argue in favour of Model 3.

Why Model 2 shows better performance:

- The wider network approach of Model 2 captures image features more effectively

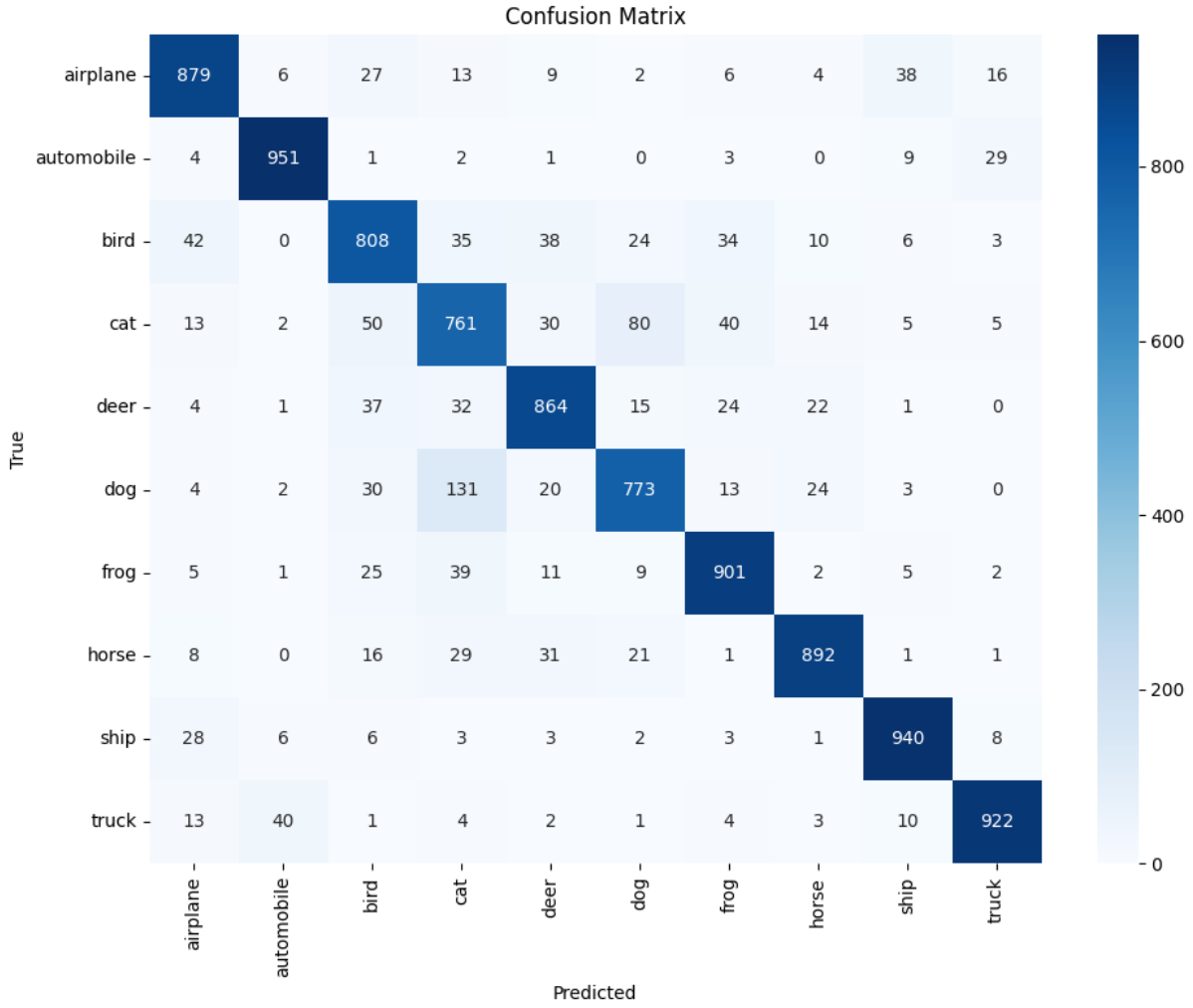
- Additional depth beyond a certain point provides diminishing returns.
- The larger filter sizes in Model 2 may better represent the relatively simple CIFAR-10 classes.
- Model 2's architecture strikes a better depth-width balance for this particular dataset

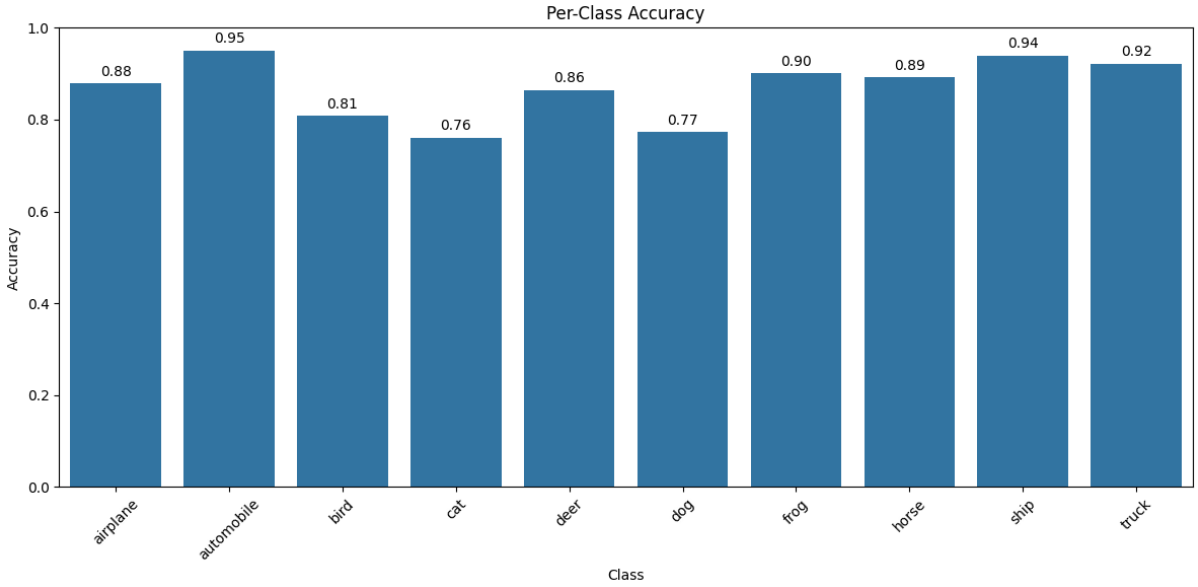
Why Model 3 might still be preferred:

- More extensive data augmentation suggests better real-world generalization
- Adaptive learning rate shows more disciplined optimization approach
- Deeper architecture may transfer better to more complex tasks
- Strided convolutions represent more principled downsampling
- More parameter-efficient design (narrower filters) for deployment scenarios
- Longer training demonstrates thorough convergence testing
- GlobalAveragePooling reduces spatial variance in final representations

The choice depends on priorities: Model 2 for pure CIFAR-10 performance, Model 3 for potential robustness in varied scenarios and as a foundation for transfer learning to more complex tasks.

Additionally from the confusion matrix and per class accuracy charts for Model 3 below we can see that the main errors in the model come from Dogs vs Cats (understandable mistake by a model which has pixelated images to deal with).





P3. Permutation equivariant neural networks

Equivariant Network Implementation

- **Purpose**: Creates a neural network layer that is invariant to permutations of set elements
- **Mathematical Principle**: Processes each set element in two ways:
  - Element-wise transformation: Applies a weight matrix (lambda_weights) to each element individually
  - Sum-based transformation: Computes the sum of all elements, then applies another weight matrix (gamma_weights) to this pooled representation
- **Implementation Details**:
  - The build method initializes two weight matrices and a bias term
  - The call method combines:
    - element_transform: Transforms each set element using lambda_weights
    - sum_transform: Computes sum across all elements, broadcasts it back to the original shape, and applies gamma_weights
    - The final output adds these transforms together with a bias term
    - Finally applies an optional activation function

Strategic Choices

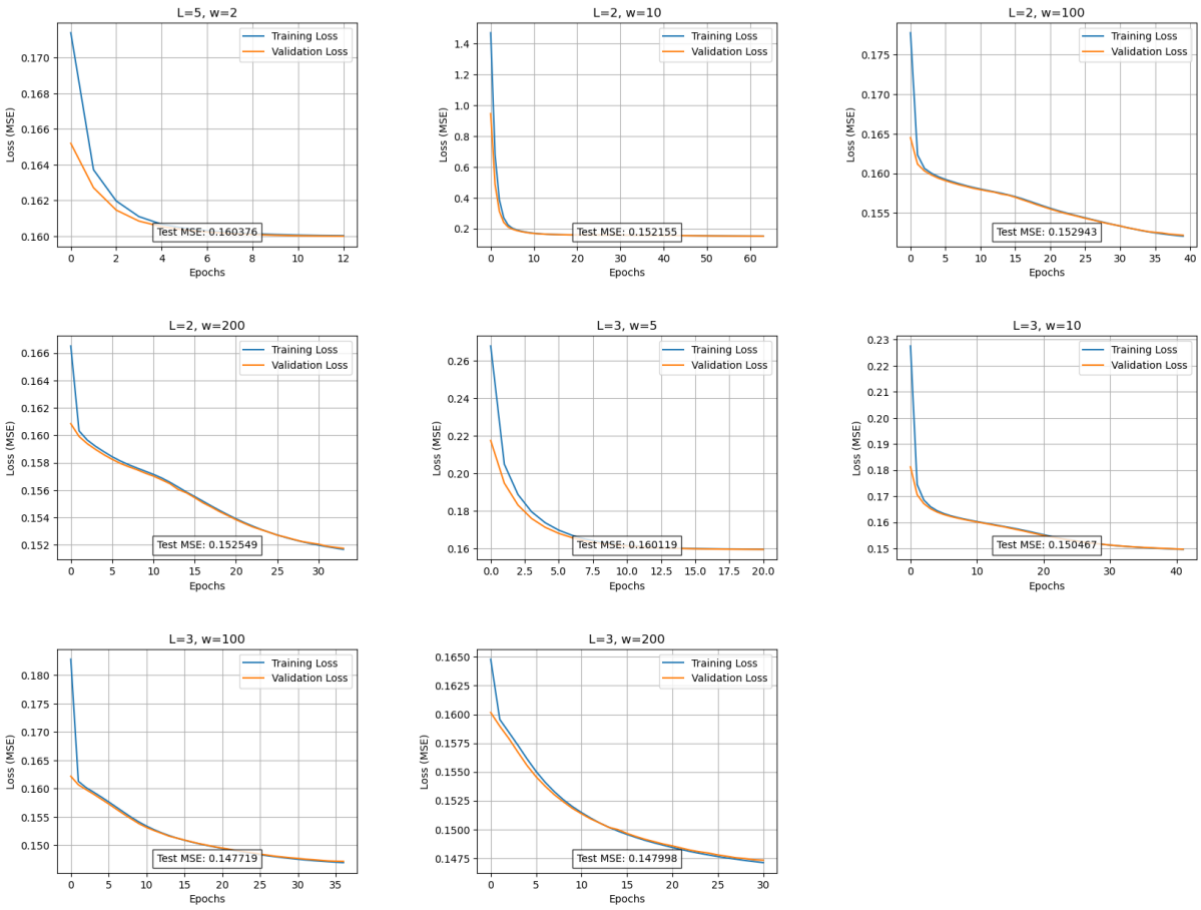
- Leveraging 1D convolutions instead of direct matrix operations for improved computational efficiency and potentially better hardware acceleration
- Separating point-wise and global operations into two distinct pathways (point_conv and pool_conv) for clearer architectural design and easier debugging
- Using mean pooling rather than sum pooling in the global pathway, which provides better numerical stability when working with sets of varying sizes
- Tile operation to efficiently broadcast the global information back to each set element, ensuring proper dimensions for integration with the point-wise representation

- Minimizing parameter count by omitting bias terms in the point-wise convolution, using kernel size 1, and sharing weights across all set elements through the convolution operation

Results

```
================================================================================================================
COMPREHENSIVE RESULTS SUMMARY
================================================================================================================
 Layers (L)  Width (w) Test MSE Final Train Loss Final Val Loss Trainable Params Training Time  Epochs
          3       100 0.147719         0.146936       0.147158         21,001        188.22s      37
          3       200 0.147998         0.147109       0.147321         82,001        324.29s      31
          3        10 0.150467         0.149646       0.149712            301         72.57s      42
          2        10 0.152155         0.151453       0.151464             91         70.44s      64
          2       200 0.152549         0.151684       0.151760          1,801        145.96s      34
          2       100 0.152943         0.152065       0.152182            901         92.31s      40
          3         5 0.160119         0.159449       0.159387            101         30.76s      21
          5         2 0.160376         0.160035       0.160009             49         21.91s      13
```

Training Results for All Model Configurations



Discussion

The best-performing model (lowest Test MSE of 0.148) has 3 layers (L=3) with a width of 100 (w=100), requiring 21,001 trainable parameters and 188 seconds to train over 37 epochs.

Model performance generally improves with:

- More layers (L=3 configurations tend to outperform L=2)
- Wider networks (w=100 or w=200 perform better than narrower options)

However, there are diminishing returns - the L=3, w=200 model has nearly 4x the parameters of L=3, w=100 but only marginally better performance, while taking 72% longer to train.

The simplest model (L=5, w=2) performs worst but trains fastest with only 49 parameters, while the L=2, w=200 configuration has the most parameters (1,801) without delivering the best results.

Early stopping triggered in all cases before reaching the maximum 100 epochs, indicating effective regularization.