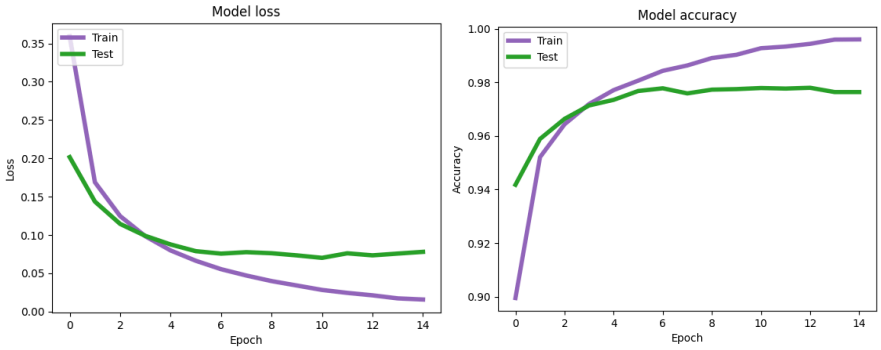
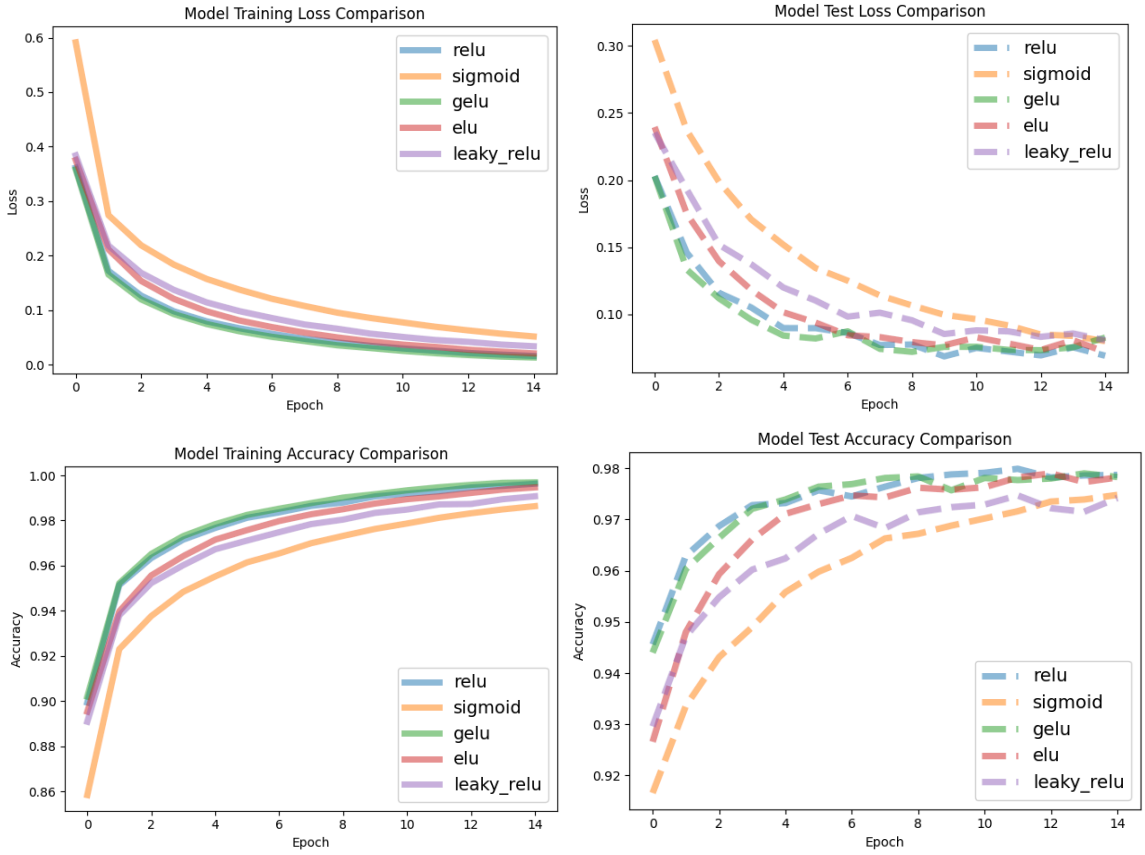
Assignment 0: Training a Neural Network on MNIST with Keras

All the implementations have been evaluated by the NVIDIA RTX 4090. Default implementation has a basic model with ReLU activation, and has been trained by Adam optimizer with 0.001 learning rate. We use 128 batches for training and testing the model. Starting from training loss 0.6221, It achieves 0.078 validation loss and 97.64% validation accuracy



1. **Effect of Activation Function**

   We change the activation (sigmoid, gelu, elu, leaky_relu) and compare it to the default implementation. Sigmoid performs worst likely due to vanishing gradients. Leaky relu[1] and gelu[2] closely match relu while leaky relu showing marginally better training stability. Elu[3] underperforms slightly compared to Relu, suggesting its exponential component may not benefit MNIST's simple patterns.



[1] Leaky ReLU: An activation function where the negative selection allows a small gradient instead of being completely zero.
[2] GeLU: Gaussian Error Linear Unit
[3] ELU: Exponential Linear Unit

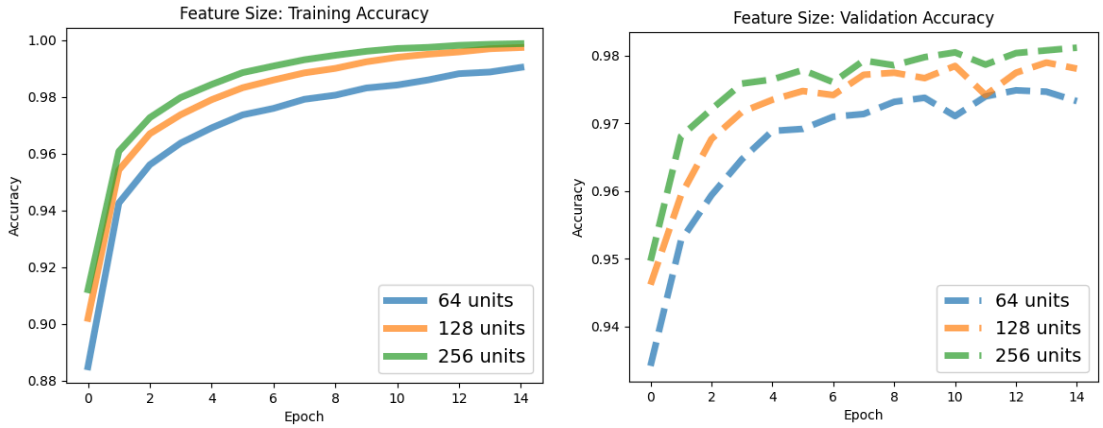
	Relu	Sigmoid	Gelu	Elu	Leaky Relu
Accuracy	97.87%	97.48%	97.83%	97.82%	97.42%

2. Effect of Model Architecture

We change the model architecture while keeping the activation function to Relu which has the highest accuracy.

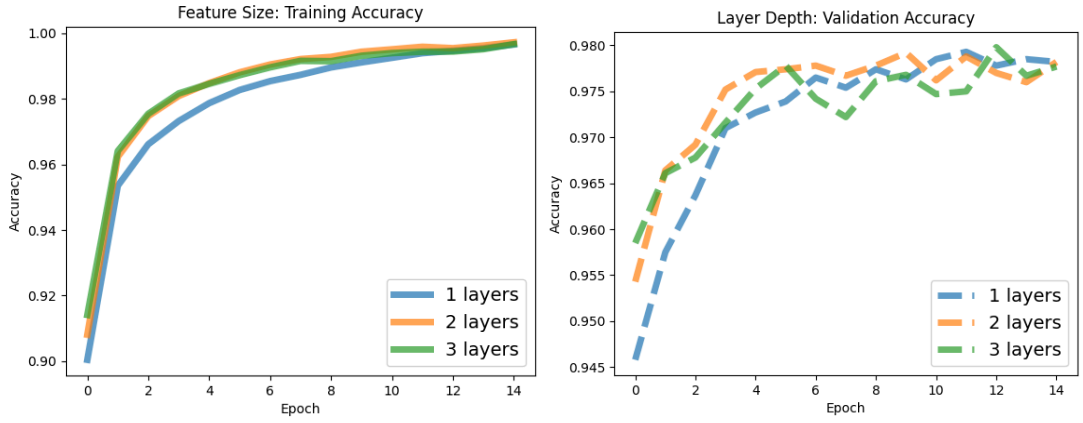
a. Changing feature size

Larger feature size improve the performance from 97.38% accuracy with 64 units to 98.12% accuracy with 256 units.



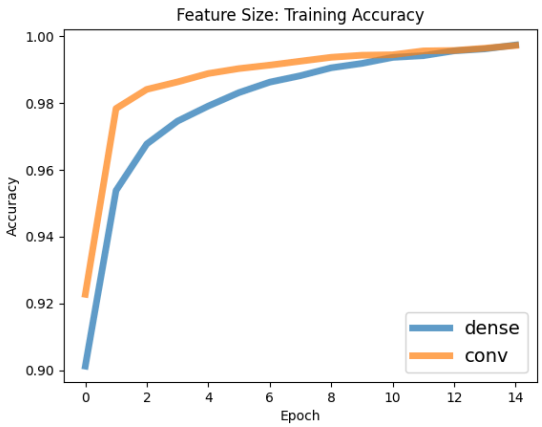
b. Changing layer depth

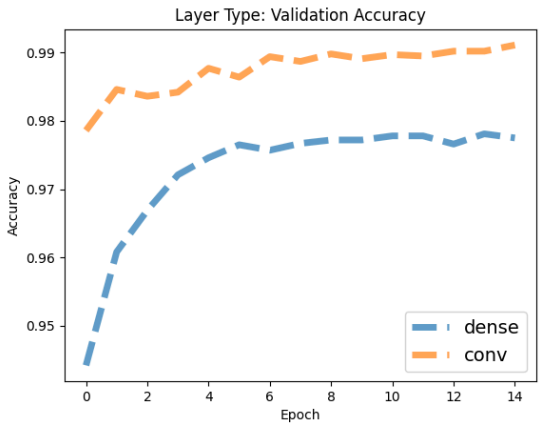
Deeper networks show minimal improvement, or even degradation from 97.86% accuracy with single-layer network to 97.74% of accuracy with three layers. Overly deep models may introduce redundancy for simple MNIST 28x28 pixel inputs



c. Changing layer type

CNNs capture better spatial relationships in image data, even for simple datasets like MNIST. As a result, convolutional layers outperform dense layer by improving accuracy from 97.90% to 98.85%.



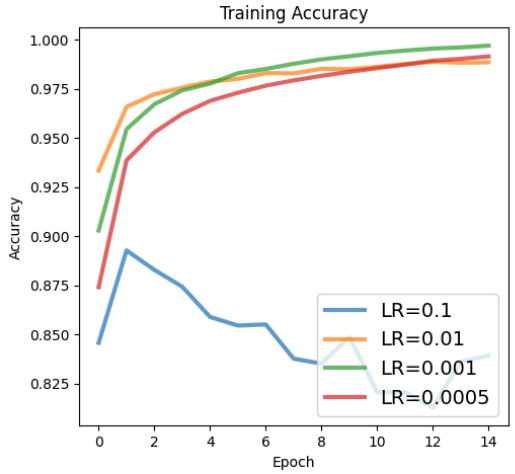


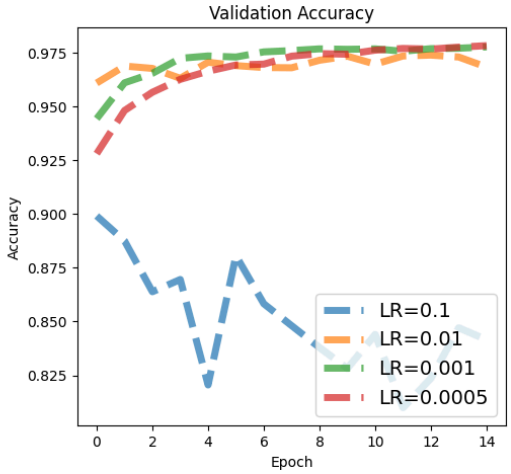
3. Effect of Training Hyperparameter

We change the hyperparameter for training while keeping the model architecture same as default implementation.

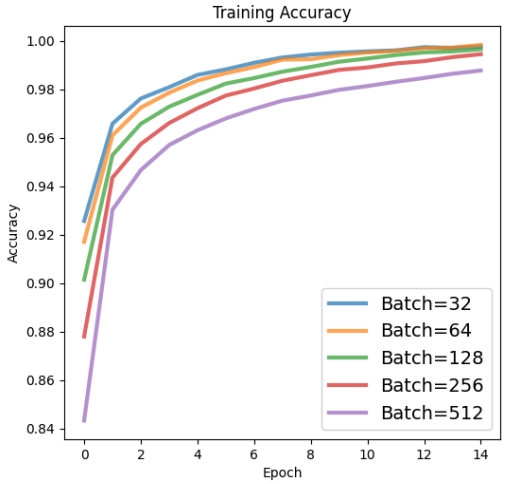
a. Learning rate

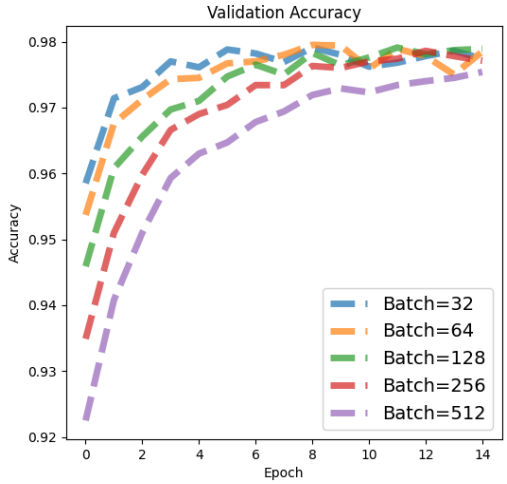
Optimal learning rate is 0.0001. Overly high learning rate (0.1) makes unstable training, resulting in low 76.48% accuracy. On the other hand, low learning rate (0.0005) shows slow convergence with smooth slope in training accuracy graph. By having an optimal learning rate, model can achieve 97.89% accuracy.





b. Batch size

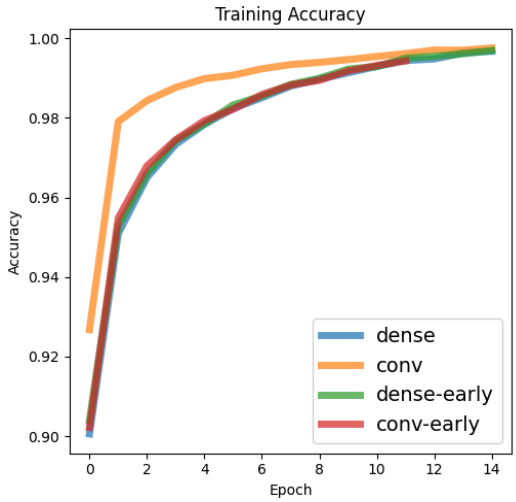


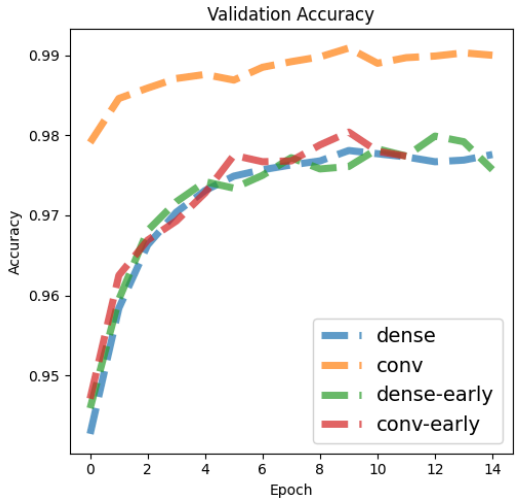


Smaller batch sizes generalize better by providing more frequent weight updates, improving the gradient estimation. For example, training with 32 batches achieves 97.69% accuracy while training with 512 batches reaches a lower accuracy of 97.62%. Optimal batch size is 64, showing the highest accuracy of 97.89%.

4. Effect of Overfitting

Convolution network with early stopping stops training earlier epoch in 13 while keeping the model parameter with the best performance in epoch 9. Even if it stops the training earlier than full-epoch, it achieves comparable performance.





	Dense	Dense-early	Conv	Conv-early
Accuracy	97.78%	97.79%	98.84%	97.79%