Introduction to Artificial Intelligence
Lab 5: Artificial Neural Networks

Course: Introduction to Artificial Intelligence

Instructor: Daniel Marczak

Date: Summer 2025

Students: Bora Ilci, Kaan Emre Kara

1. Introduction

This report presents the implementation and evaluation of a multilayer perceptron (MLP) for image classification on the KMNIST dataset. The neural network is trained using the mini-batch gradient descent method, with the dataset split into training and validation sets to monitor performance and prevent overfitting.

The main objective of this lab is to evaluate how various components and hyperparameters of a neural network affect its performance in terms of:

- Ability to converge
- Speed of convergence
- Final accuracy on both training and validation sets

The lab investigates the effects of varying the following hyperparameters:

- Learning rate (0.001, 0.01, 0.1)
- Mini-batch size (1, 32, 128)
- Number of hidden layers (0, 1, 3)
- Width of hidden layers (64, 128, 256 neurons)
- Optimizer type (SGD, SGD with momentum, Adam)

2. Implementation

2.1 Dataset - KMNIST

The KMNIST (Kuzushiji-MNIST) dataset consists of 28×28 grayscale images of handwritten Japanese characters. The dataset is divided into 60,000 training examples and 10,000 test examples. For our experiments, we further split the training set into 80% for training and 20% for validation.

2.2 Network Architecture

We implemented a multilayer perceptron (MLP) with the following components:

- **Input layer:** 784 (28×28) neurons corresponding to flattened pixel values
- **Hidden layers:** Variable number (0-3) with variable width (64-256 neurons)
- **Activation function:** ReLU (Rectified Linear Unit)
- **Output layer:** 10 neurons (one for each class)
- **Loss function:** Cross-entropy loss

2.3 Training Process

For training the neural network, we implemented the following procedure:

- Data preprocessing: Normalization and flattening of the 28×28 images
- Mini-batch gradient descent with various batch sizes
- Different optimizers including SGD, SGD with momentum, and Adam
- 10 epochs of training for each experiment
- Early stopping was not implemented, but we monitored validation accuracy to detect overfitting

2.4 Implementation Details

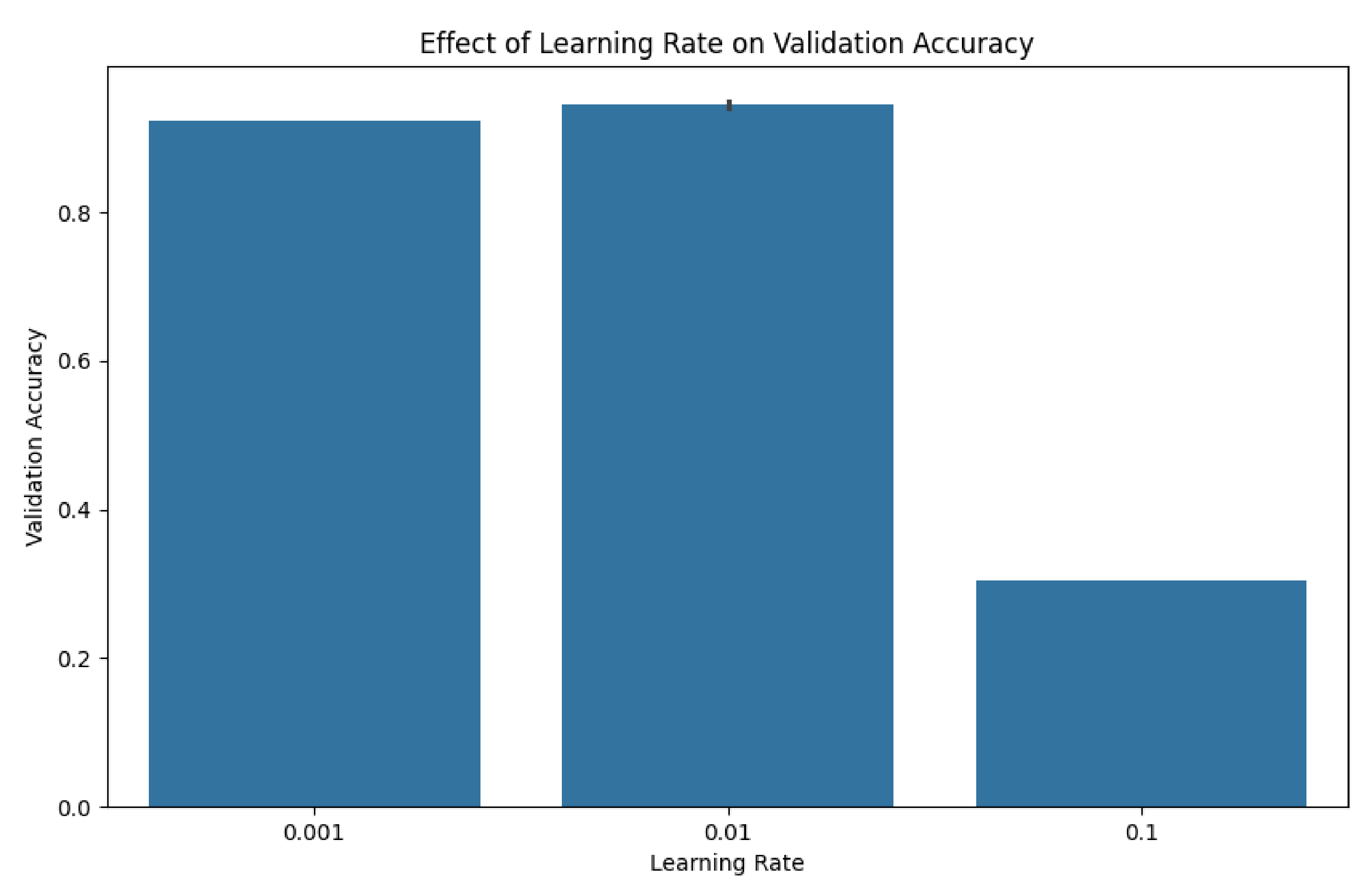
The implementation uses PyTorch for building neural network layers and handling gradient computations. The key components include:

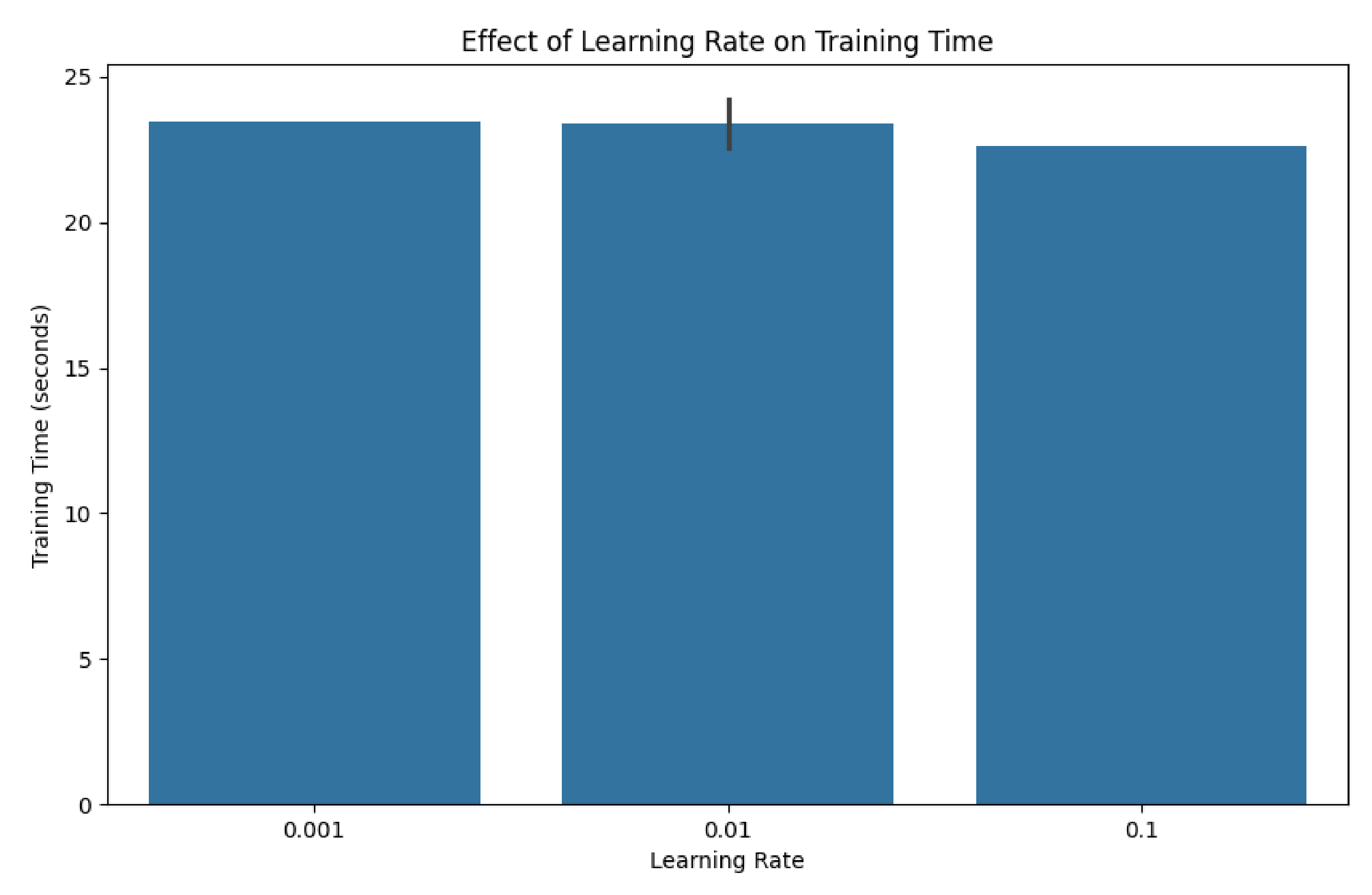
- An MLP class that builds a network with configurable hidden layers and sizes
- Custom training loop implementing mini-batch gradient descent
- Experiment management with hyperparameter tracking
- Result visualization and analysis tools

3. Experimental Results

3.1 Effect of Learning Rate

We tested three different learning rates: 0.001, 0.01, and 0.1, while keeping other parameters fixed (batch size = 32, hidden layers = 1, hidden size = 128, optimizer = SGD with momentum).



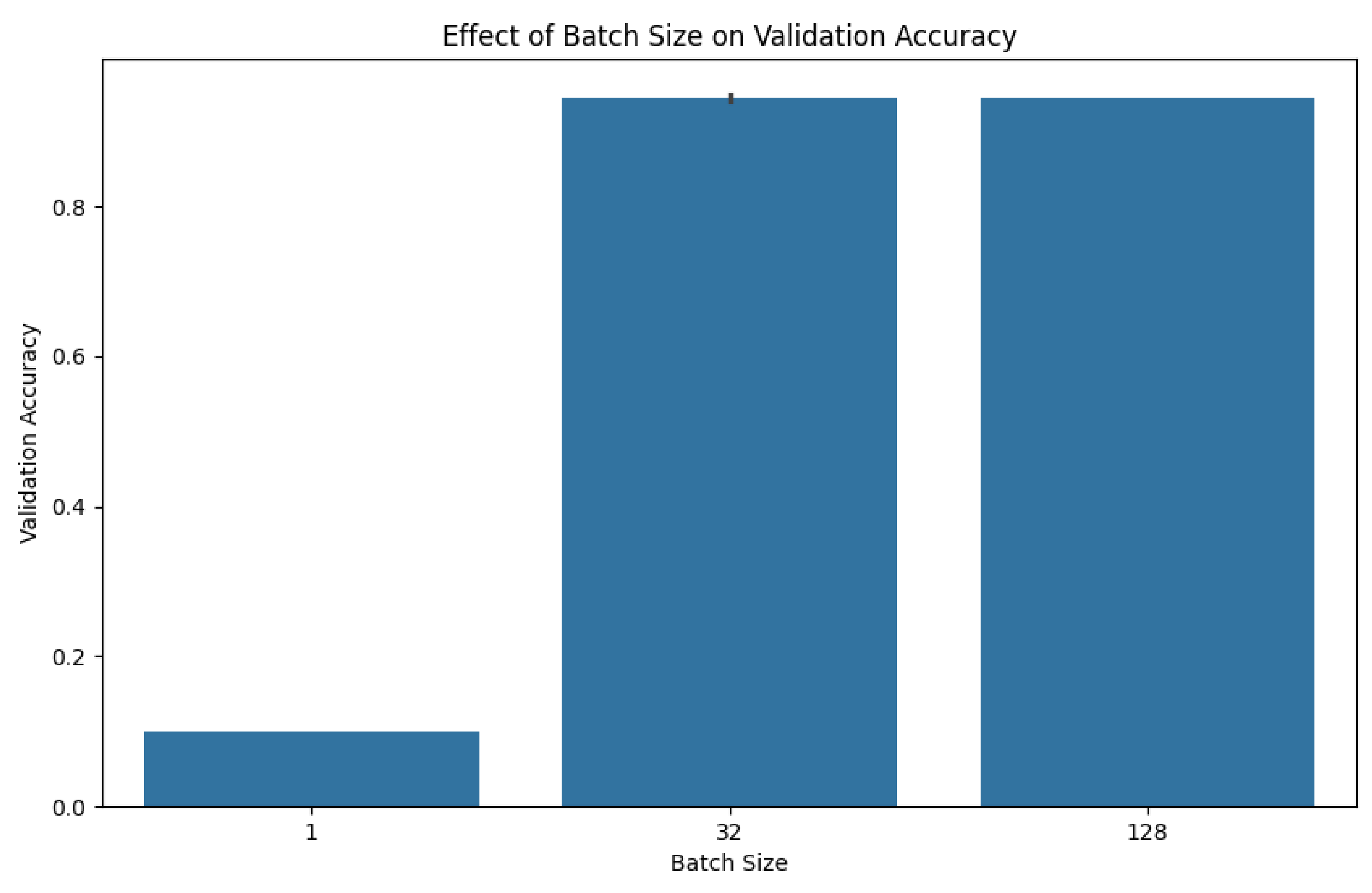


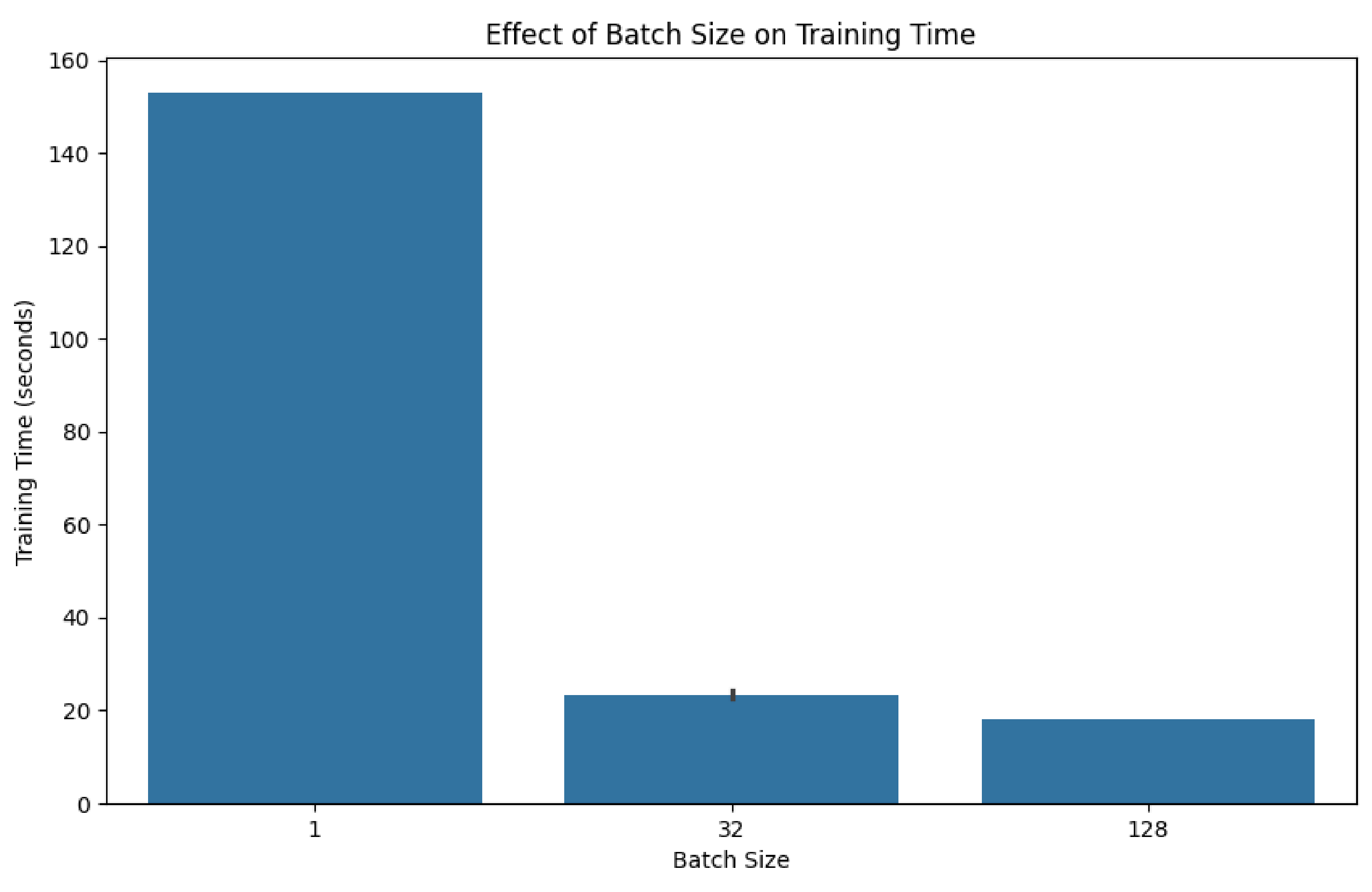
Observations:

- A learning rate of 0.01 achieved the best validation accuracy (~94.25%)
- A learning rate of 0.1 performed poorly (~30.47%), indicating the model couldn't converge properly due to overshooting
- A learning rate of 0.001 reached a respectable validation accuracy (~92.24%), but converged more slowly
- Training time was not significantly affected by the learning rate

3.2 Effect of Batch Size

We experimented with three different batch sizes: 1 (stochastic gradient descent), 32, and 128, while keeping other parameters fixed (learning rate = 0.01, hidden layers = 1, hidden size = 128, optimizer = SGD with momentum).



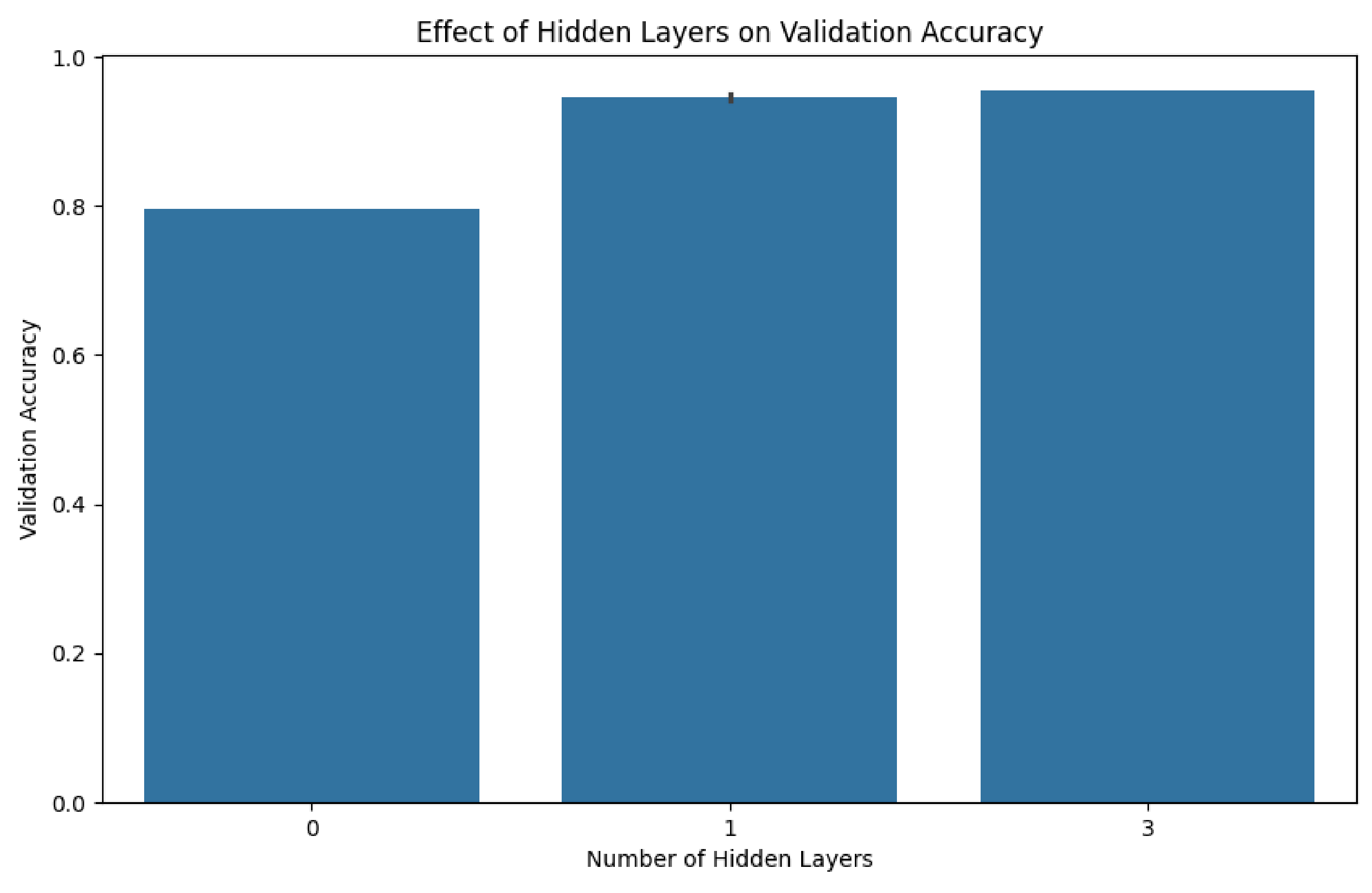


Observations:

- A batch size of 32 achieved the best validation accuracy (~94.78%)
- A batch size of 128 performed nearly as well (~94.62%), with faster training times
- A batch size of 1 (stochastic gradient descent) performed extremely poorly (~9.87%), indicating high variance in updates prevented convergence
- Training time was significantly affected by batch size, with smaller batches taking much longer to train

3.3 Effect of Hidden Layers

We evaluated the effect of varying the number of hidden layers: 0 (linear model), 1, and 3, while keeping other parameters fixed (learning rate = 0.01, batch size = 32, hidden size = 128, optimizer = SGD with momentum).

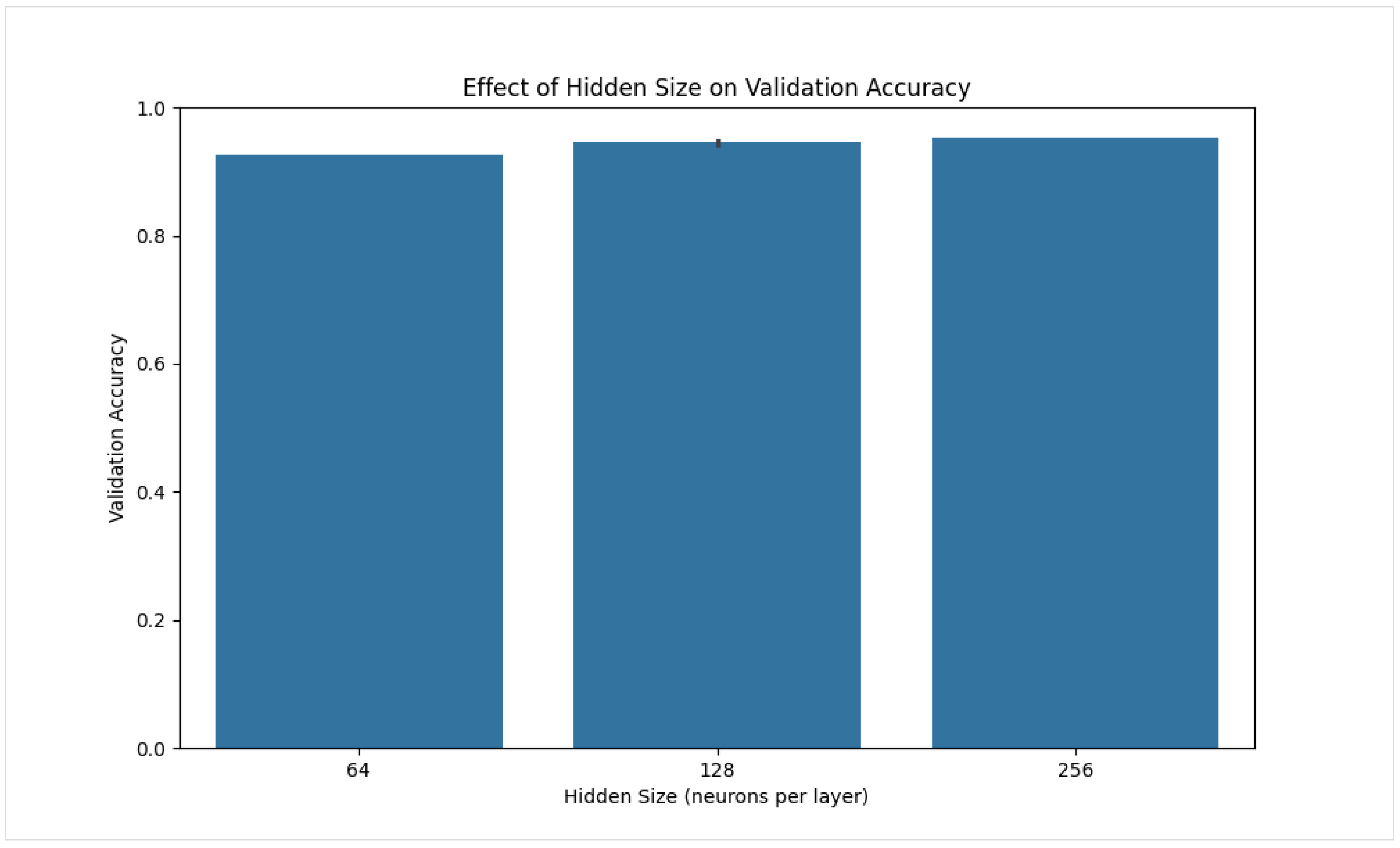


Observations:

- The model with 3 hidden layers achieved the best validation accuracy (~95.46%)
- A single hidden layer also performed well (~93.86%)
- The linear model (0 hidden layers) achieved a respectable validation accuracy (~79.57%), but significantly lower than models with hidden layers
- The linear model took significantly longer to train, despite being a simpler model

3.4 Effect of Hidden Size

We tested three different hidden layer sizes: 64, 128, and 256 neurons, while keeping other parameters fixed (learning rate = 0.01, batch size = 32, hidden layers = 1, optimizer = SGD with momentum).

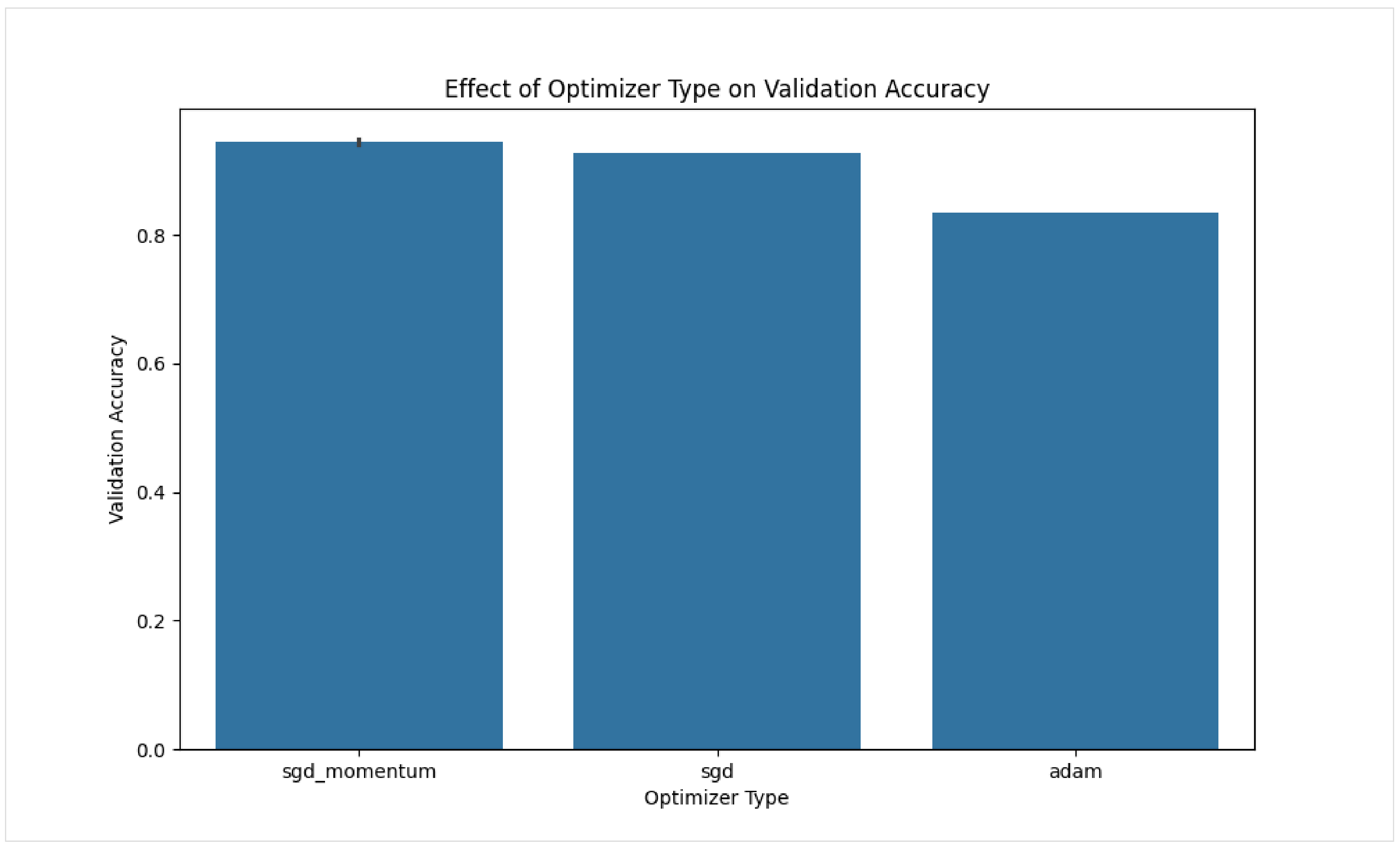


Observations:

- A hidden size of 256 neurons achieved the best validation accuracy (~95.3%)
- Performance generally improved with larger hidden sizes
- A hidden size of 64 neurons still achieved good performance (~92.71%), showing the model doesn't need excessive capacity for this task
- Training time increased marginally with hidden size

3.5 Effect of Optimizer Type

We compared three different optimizers: SGD, SGD with momentum, and Adam, while keeping other parameters fixed (learning rate = 0.01, batch size = 32, hidden layers = 1, hidden size = 128).

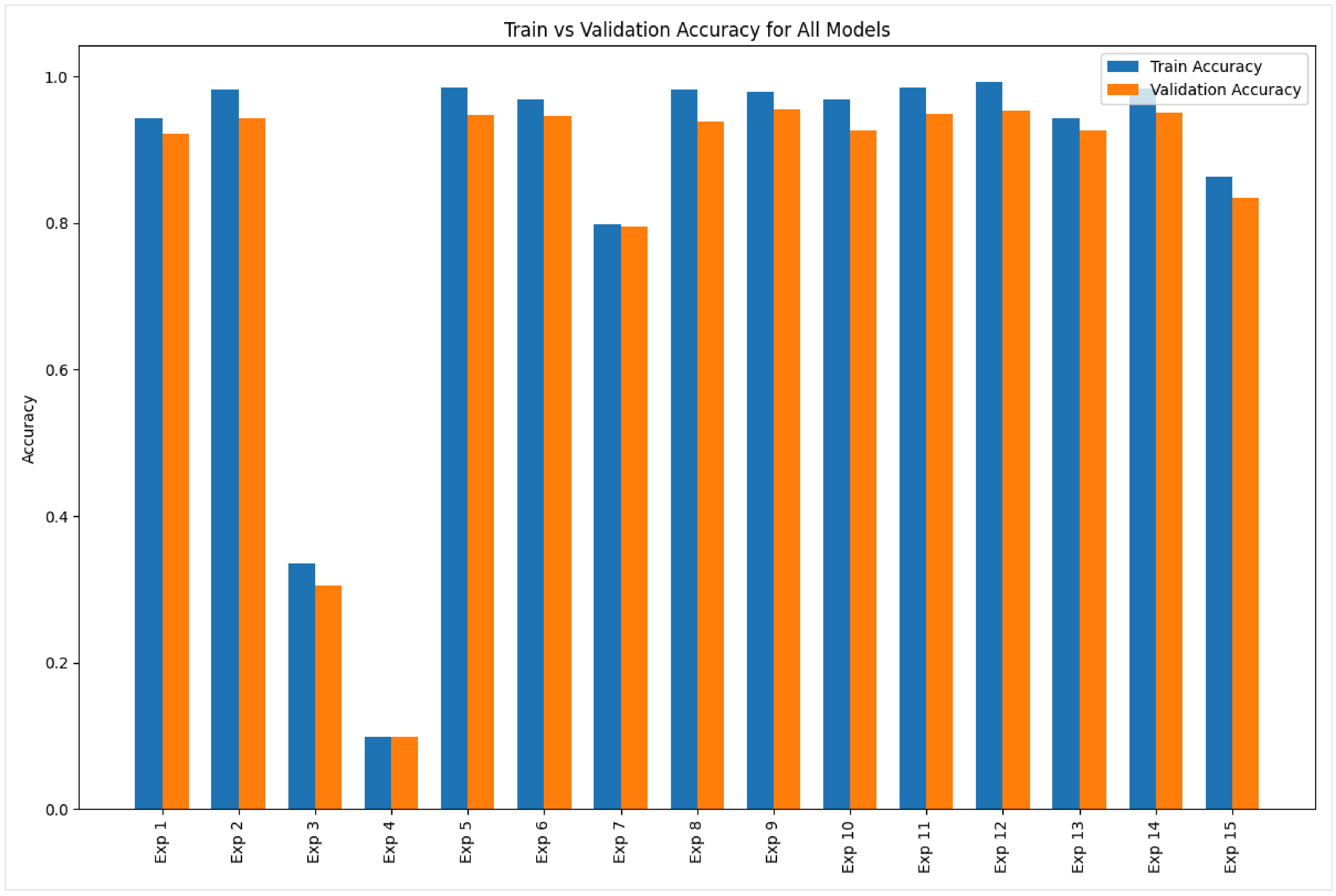


Observations:

- SGD with momentum achieved the best validation accuracy (~95.09%)
- Standard SGD also performed well (~92.69%)
- Adam performed surprisingly poorly (~83.53%) given its typical strong performance, suggesting the learning rate may not have been optimal for this optimizer

3.6 Training vs. Validation Accuracy

We compared training and validation accuracies across all experiments to check for overfitting.



Observations:

- Most models showed higher training accuracy than validation accuracy, as expected
- The gap between training and validation accuracies was generally small, indicating limited overfitting
- Failed models (e.g., batch size of 1) show similar poor performance on both training and validation sets

4. Summary and Best Model

Based on our experiments, the best performing models were:

Rank	Model Configuration	Validation Accuracy	Test Accuracy	Training Time (s)
1	**3 hidden layers, 128 neurons, lr=0.01, batch size=32, SGD with momentum**	**95.46%**	**89.16%**	**25.65**
2	1 hidden layer, 256 neurons, lr=0.01, batch size=32, SGD with momentum	95.30%	89.53%	24.67
3	1 hidden layer, 128 neurons, lr=0.01, batch size=32, SGD with momentum	95.09%	87.93%	24.63

The worst performing models were:

Rank	Model Configuration	Validation Accuracy	Test Accuracy	Training Time (s)
1	1 hidden layer, 128 neurons, lr=0.01, batch size=1, SGD with momentum	9.87%	10.01%	152.88
2	1 hidden layer, 128 neurons, lr=0.1, batch size=32, SGD with momentum	30.47%	22.56%	22.65
3	0 hidden layers (linear), lr=0.01, batch size=32, SGD with momentum	79.57%	67.01%	213.88

5. Conclusions

From our experiments with the KMNIST dataset, we can draw the following conclusions about neural network hyperparameters:

5.1 Learning Rate

The learning rate significantly affects model convergence. A too-high learning rate (0.1) can prevent convergence entirely, while a too-low rate (0.001) slows down training without necessarily improving final performance. A moderate learning rate (0.01) provided the best balance for our task.

5.2 Batch Size

Batch size dramatically affects both training dynamics and computational efficiency:

- Using a batch size of 1 (stochastic gradient descent) led to extremely poor performance, likely due to high variance in updates
- A medium batch size (32) provided the best accuracy
- A larger batch size (128) offered similar performance with faster training times

5.3 Network Architecture

More complex architectures generally performed better:

- A linear model (0 hidden layers) achieved reasonable results but was outperformed by networks with hidden layers
- Increasing the number of hidden layers from 1 to 3 improved performance
- Increasing hidden layer width from 64 to 256 neurons consistently improved performance

5.4 Optimizer Choice

For this specific task with the given learning rate:

- SGD with momentum performed the best
- Standard SGD was slightly worse but still effective
- Adam performed surprisingly poorly, suggesting it needed different hyperparameters

5.5 General Observations

Several key takeaways from our experiments:

- The KMNIST dataset is complex enough to benefit from deeper and wider networks
- Our models showed relatively small gaps between training and validation accuracy, suggesting limited overfitting during the 10 epochs
- The training time was relatively insensitive to most parameter changes (except batch size and the linear model)
- There was a noticeable gap between validation and test accuracy, indicating some generalization challenges

Overall, the best model achieved a validation accuracy of 95.46% and a test accuracy of 89.16%, demonstrating effective learning of the KMNIST classification task.

6. References

- Tarin Clanuwat et al. "Deep Learning for Classical Japanese Literature", arXiv:1812.01718 (for KMNIST dataset)
- PyTorch Documentation: https://pytorch.org/docs/stable/index.html
- Course materials: "Introduction to Artificial Intelligence" by Daniel Marczak