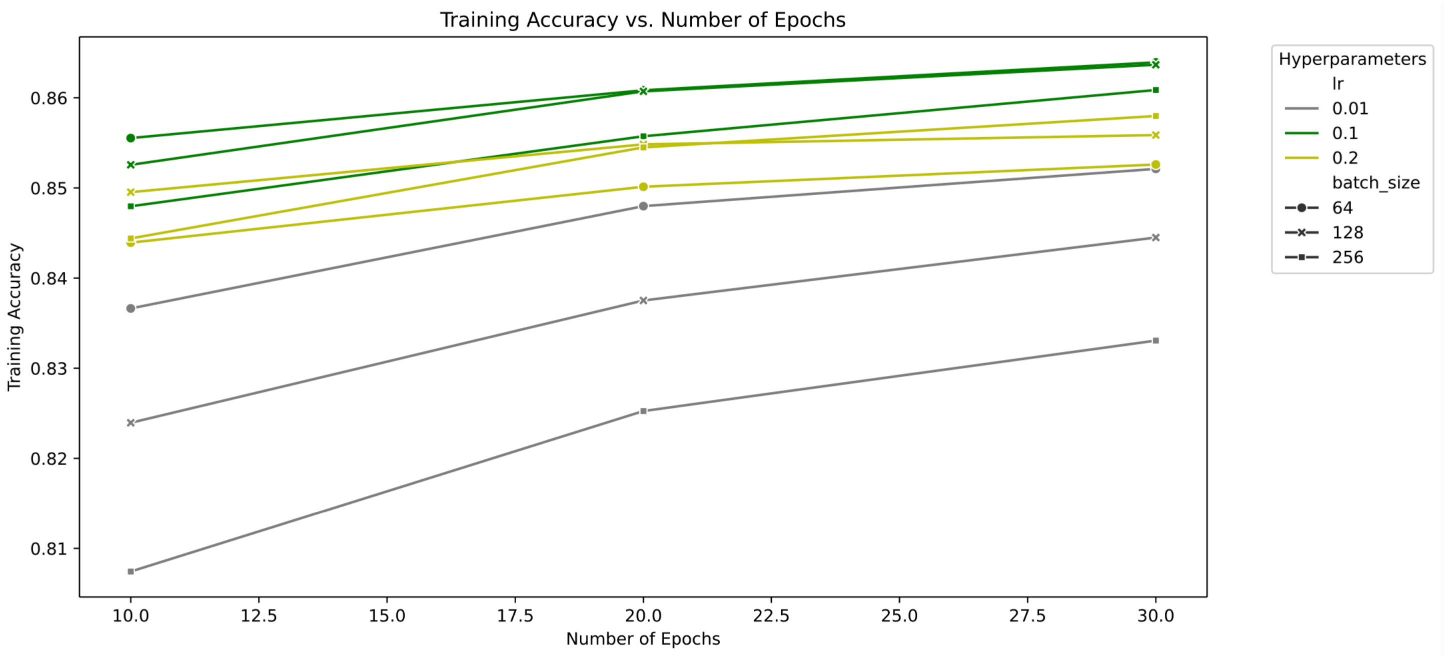
Assignment 5.2

batch_size	num_epochs	lr	train_loss	train_acc	test_acc
256	10	0.10	0.447237	0.847967	0.8307
256	10	0.01	0.606075	0.807433	0.7948
256	10	0.20	0.450400	0.844400	0.8369
256	20	0.10	0.419751	0.855733	0.8400
256	20	0.01	0.537254	0.825250	0.8127
256	20	0.20	0.420886	0.854500	0.8087
256	30	0.10	0.407390	0.860867	0.8414
256	30	0.01	0.505789	0.833067	0.8207
256	30	0.20	0.410185	0.857983	0.8283
128	10	0.10	0.427356	0.852567	0.8343
128	10	0.01	0.539896	0.823950	0.8090
128	10	0.20	0.445005	0.849533	0.8104
128	20	0.10	0.406431	0.860717	0.8432
128	20	0.01	0.488414	0.837517	0.8261
128	20	0.20	0.421837	0.854833	0.8385
128	30	0.10	0.395988	0.863667	0.8424
128	30	0.01	0.464547	0.844500	0.8305
128	30	0.20	0.419843	0.855867	0.8340
64	10	0.10	0.417927	0.855533	0.8365
64	10	0.01	0.490335	0.836633	0.8244
64	10	0.20	0.466685	0.843933	0.8102
64	20	0.10	0.401180	0.860833	0.8383
64	20	0.01	0.451474	0.847983	0.8324
64	20	0.20	0.448947	0.850133	0.8073
64	30	0.10	0.393610	0.863933	0.8332
64	30	0.01	0.434257	0.852117	0.8368

64	30	0.20	0.439175	0.852600	0.8318



Learning Behavior Observed
Learning rate lr plays a significant role in model performance. Too small a learning rate (0.01) leads to slower convergence, reflected by higher training loss. Too high a learning rate (0.2), on the other hand, can destabilize the learning, as evidenced by the fluctuating test accuracy. Smaller batch sizes (like 64) tend to have slightly better test accuracy. It's probably because smaller batches provide a regularization effect and lower generalization error. However, it's also computationally more expensive. More epochs generally result in better training accuracy but don't always translate to better test accuracy due to the risk of overfitting.

Learning Rate
0.1: Seems to be just right. Both the training and test accuracies are high, and the model converges well. The 86% accuracy mark is reached in some configurations. Across all batch sizes and epoch numbers, this learning rate generally provides the highest test accuracy. For example, with a batch size of 128 and 20 epochs, the test accuracy peaks at 84.32%. It allows the model to converge efficiently, demonstrated by the consistently lower training losses when compared to other learning rates.

0.01: Is slow and steady but it doesn't win here. The model is underfitting. It's learning, but it doesn't have enough kick to get to the optimal state. This rate is too cautious. Yes, it's moving, but not fast enough. You can clearly see it in the higher training losses, such as 0.606 for a batch size of 256 and 10 epochs. The model hasn't adapted well enough to the data. This is evident from the lower training and test accuracies, like 80.74% test accuracy for 256 batch size and 10 epochs.

0.2: Learns fast, but it could miss the global minimum. t's going too fast to notice the details. Sometimes it performs well, but other times it misses the mark. This is particularly noticeable in the test accuracies, which don't show a consistent trend.

Batch Size
256: Is efficient but not as effective. The model generalizes well but doesn't capture the nuances, based on the test accuracy. While it's computationally efficient, it tends to generalize too much, missing out on the finer details in the data. The highest test accuracy achieved is 84.14%, with 30 epochs and a learning rate of 0.1.

128: Is a balanced choice, it's computationally less expensive than 64 and performs almost as well. This size is the middle ground in terms of computational cost and performance. It achieves the highest test accuracy of 84.32% with 20 epochs and a learning rate of 0.1.

64: Performs slightly better on the test set, which is the real indicator of performance. But it's computationally demanding. This batch size allows the model to learn the nuances in the data better. It's reflected in the higher test accuracies, but it's computationally expensive. It could be improved by implementing early stopping to avoid unnecessary calculations.

Epochs
10: Is a bit rushed. The model hasn't seen the data enough to make strong predictions. It gets the main points but not the details.

20: Is reasonable. Most of the configurations with 20 epochs have solid performance metrics. This seems to be the sweet spot where the model has seen the data enough times to generalize well without overfitting.

30: Might be overfitting. The model starts to memorize the training data, evident from the high training accuracies but not-so-impressive test accuracies. Implementing dropout layers could be an improvement here.

Final Thoughts
If you want a fast, decent model, go with a higher batch size and learning rate of 0.1. If you're going for that extra 1% in accuracy, consider lowering the batch size and perhaps using early stopping with more epochs. If you're planning to run for more epochs, consider implementing some regularization techniques like dropout. If you need a quick but sound model, a batch size of 128 with a learning rate of 0.1 seems to be the most efficient choice. If you're aiming for the best possible model and computational cost is not a concern, go for a smaller batch size like 64 with more epochs. But watch out for overfitting!