Training an MLP and Optimization

BADEREDDINE Haitham

March 7, 2025

1 Introduction

Artificial neural networks have emerged as a fundamental component in machine learning, providing robust solutions to intricate problems. Despite their effectiveness, training these networks presents significant challenges, particularly in mitigating overfitting. To tackle this issue, several strategies have been introduced, including the application of diverse optimizers and techniques to prevent overfitting, such as dropout layers, batch normalization, and L2 regularization.

In this study, we investigate the influence of various optimizers and overfitting prevention methods on the training and optimization processes of neural networks.

2 Part One: Training an MLP with Keras

2.1 Discussing the Training Time of the Models

	model	duration
0	Batch Gradient Descent	15.033991
1	Mini Batch Gradient Descent	69.924013
2	Stochastic Gradient Descent	3918.229174
3	Mini Batch GD With Learning rate decay	70.980195
4	Mini Batch GD With Learning rate decay and mom...	73.890594
5	Adam	75.754597
6	RMSProp	73.812522

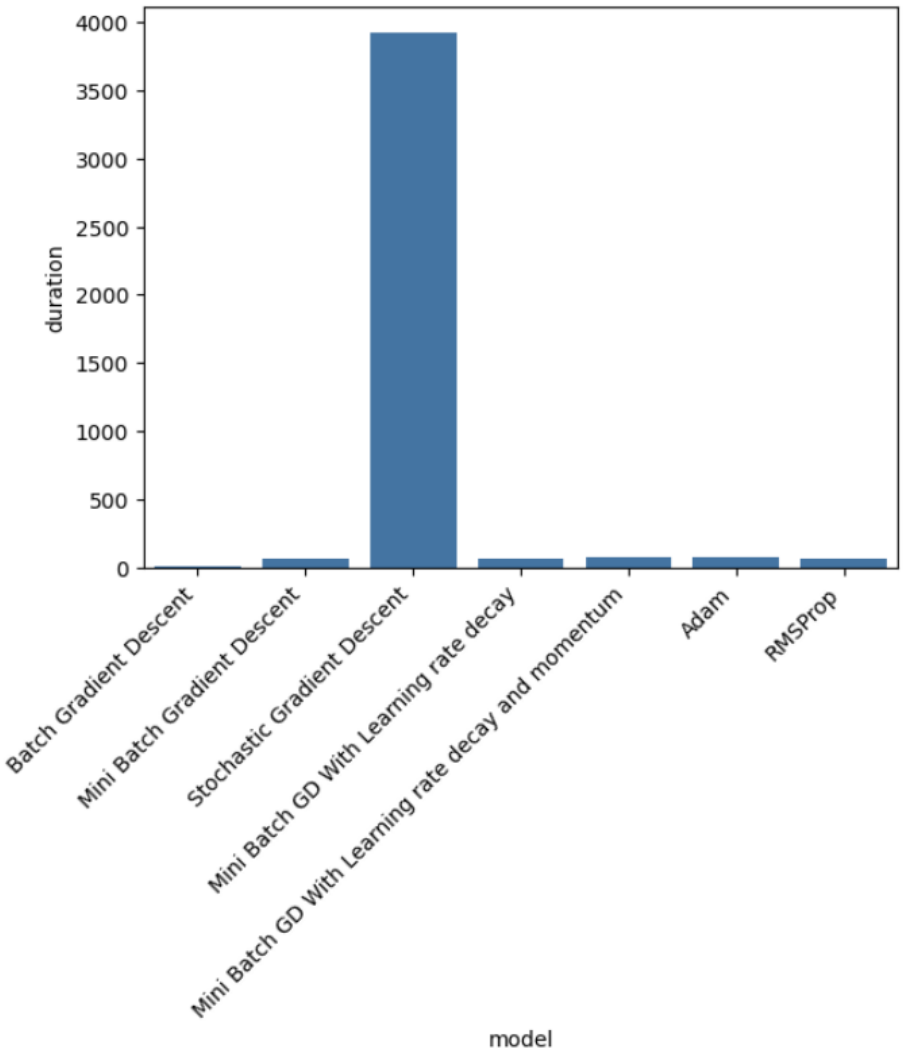

Figure 1: The histogram of the training time for each model

Neural networks typically leverage a type of machine instruction known as SIMD (Single Instruction, Multiple Data), which is available on GPUs and modern CPUs. SIMD instructions enable the same operation to be applied to multiple data points simultaneously, significantly accelerating tasks such as array addition or matrix multiplication. This parallel processing capability is why data is often processed in batches during training. Larger batch sizes allow for greater utilization of SIMD parallelism, improving computational efficiency.

For instance, Stochastic Gradient Descent (SGD) processes one data point at a time, failing to exploit the parallel nature of SIMD instructions, which results in longer training times. In contrast, Batch Gradient Descent processes the entire training set at once, fully leveraging SIMD parallelism and thus achieving faster training times.

2.2 Discussing the Performances of the Different Models

model	loss	accuracy
Batch Gradient Descent	2.076424	0.357833
Mini Batch Gradient Descent	0.106913	0.968750
Stochastic Gradient Descent	0.208053	0.975083
Mini Batch GD With Learning rate decay	0.113862	0.964750
Mini Batch GD With Learning rate decay and momentum	0.111840	0.977083
Adam	0.171247	0.976583
RMSProp	0.177938	0.979083

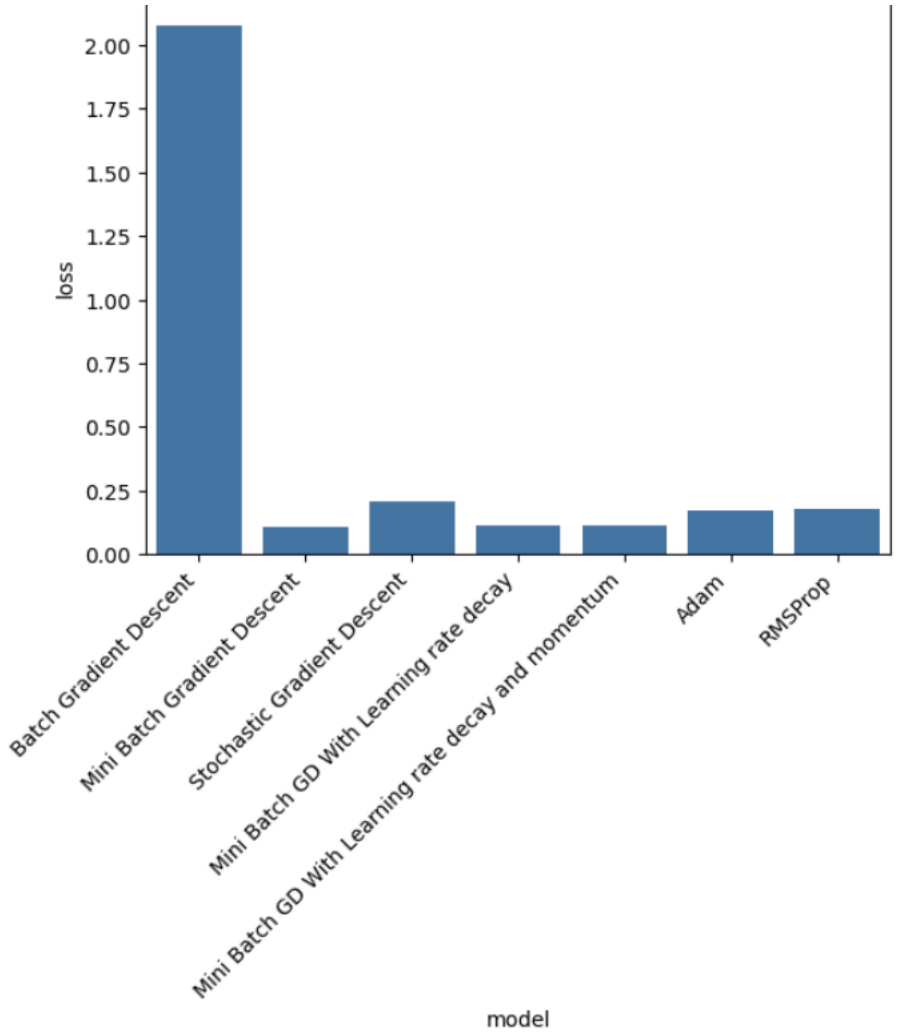

Figure 2: The loss histogram for different models on the validation set

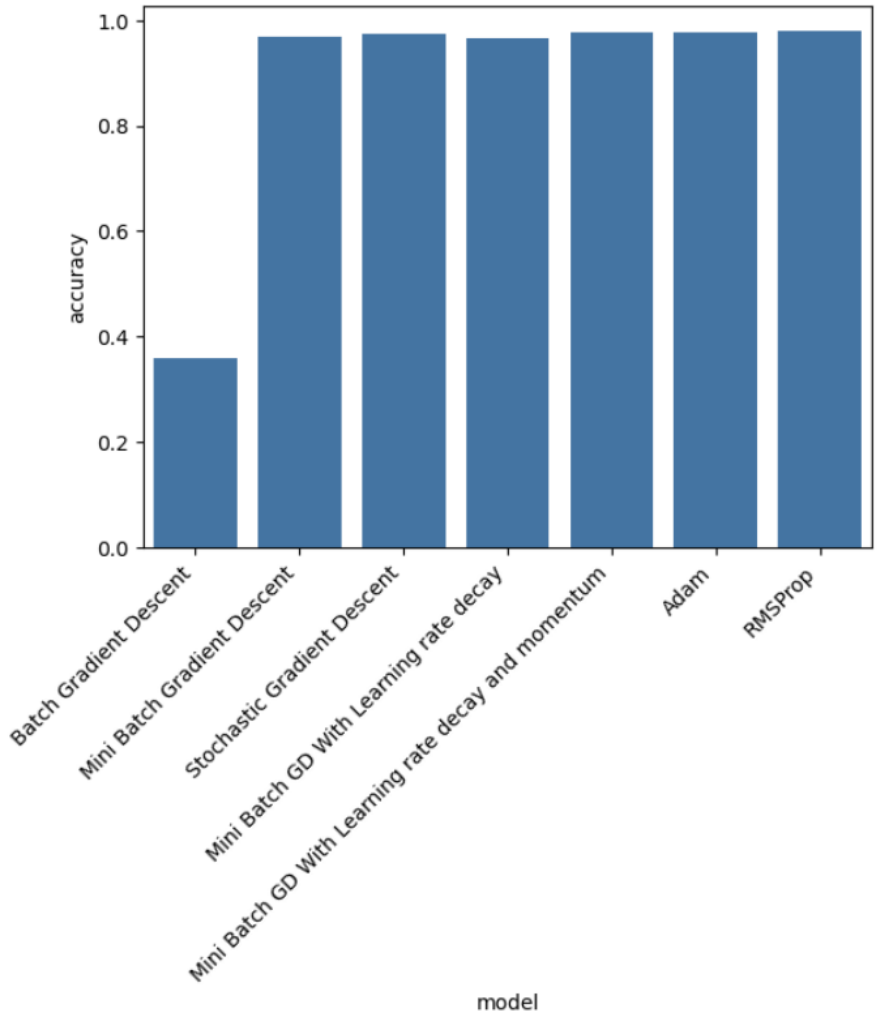

Figure 3: The accuracy histogram for different models on the validation set

Overall, we observe that the Batch Gradient Descent model tends to perform the poorest due to its limited number of iterations. Since Batch Gradient Descent processes the entire training set in a single update, the number of iterations equals the number of epochs (i.e., num_iterations=num_epochs). In contrast, Mini-Batch Gradient Descent with a batch size of 32, for example, and a training set containing 48,000 data points, performs significantly more iterations. Specifically, the number of iterations is calculated as

$$\text{num\_iterations} = \text{num\_epochs} \times \left(\frac{\text{training set size}}{\text{batch size}} \right).$$

For instance, with 50 epochs, this results in $50 \times (48000/32) = 75,000$ iterations.

However, this does not imply that smaller batches are always preferable. While smaller batches increase the number of iterations, they also slow down the training process and introduce more noise into the gradient updates. This explains why Stochastic Gradient Descent (SGD), which uses a batch size of 1, often exhibits relatively higher loss compared to Mini-Batch Gradient Descent (ignoring the poor performance of Batch Gradient Descent).

2.3 Comparing the Learning Graphs for the Different Models

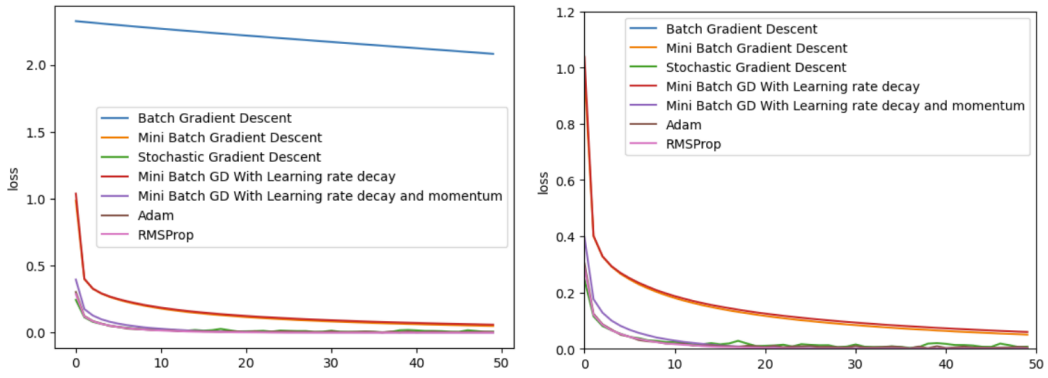

Figure 4: The training loss for the different models

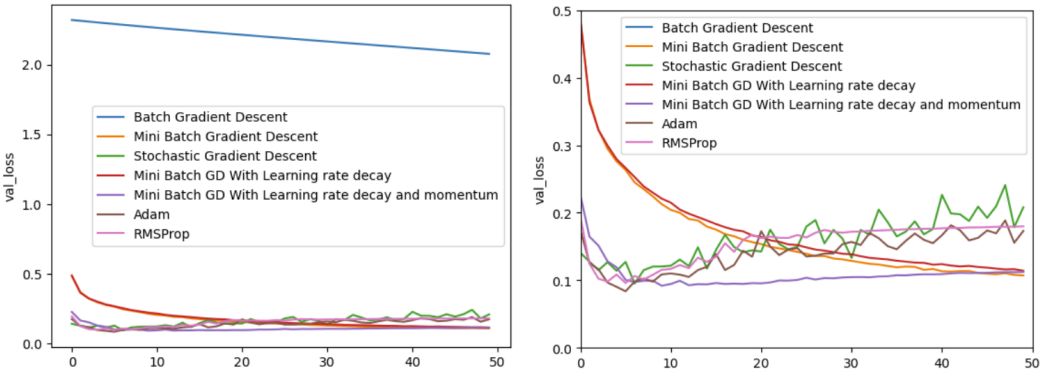

Figure 5: The validation loss for the different models

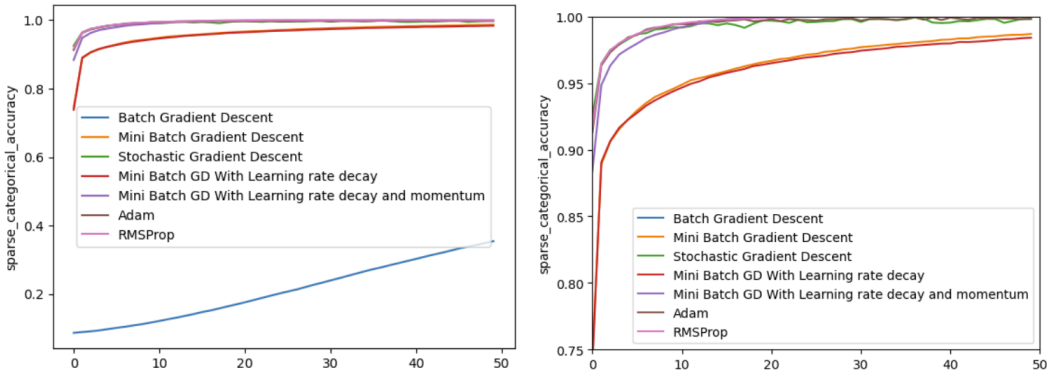

Figure 6: The training accuracy for the different models

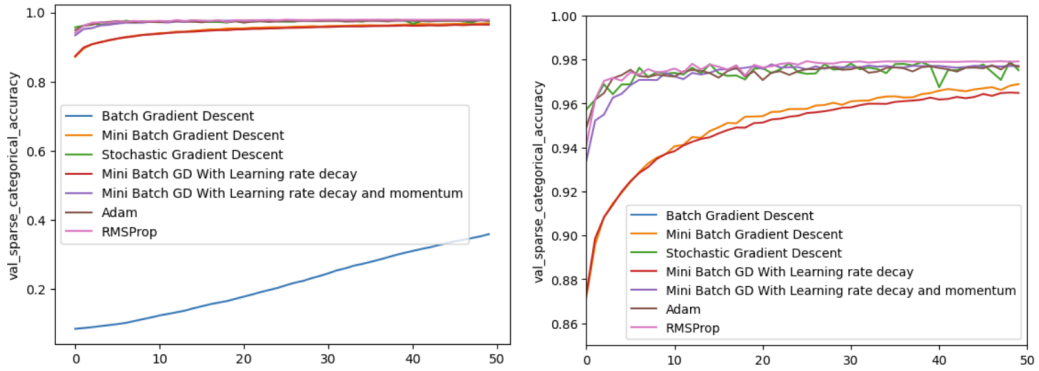

Figure 7: The validation accuracy for the different models

We observe that optimization algorithms such as Adam, RMSProp, and Mini-Batch Gradient Descent with momentum converge faster than standard Mini-Batch Gradient Descent. This is because these methods incorporate information from both the current gradient and past gradients, which helps to mitigate oscillations and noisy updates. By doing so, they avoid taking large steps in suboptimal directions, leading to more stable and efficient convergence.

However, towards the final epochs, we notice that the loss curves for Adam and RMSProp become less stable. This instability arises as the gradients diminish in magnitude, which can amplify fluctuations in the updates. A potential solution to this issue is to reduce the learning rate during the later stages of training, which can help stabilize the optimization process and improve convergence.

2.4 The Convergence Graphs for Each Model

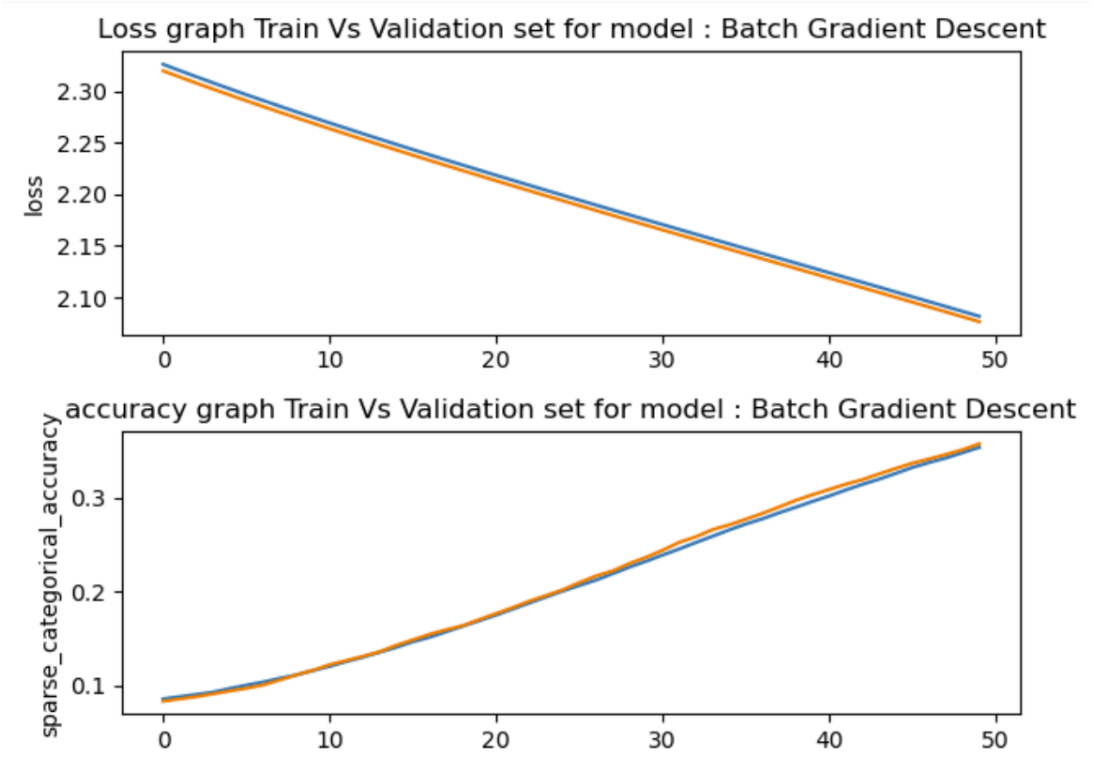

Figure 8: The convergence graph for the model : Batch Gradient Descent

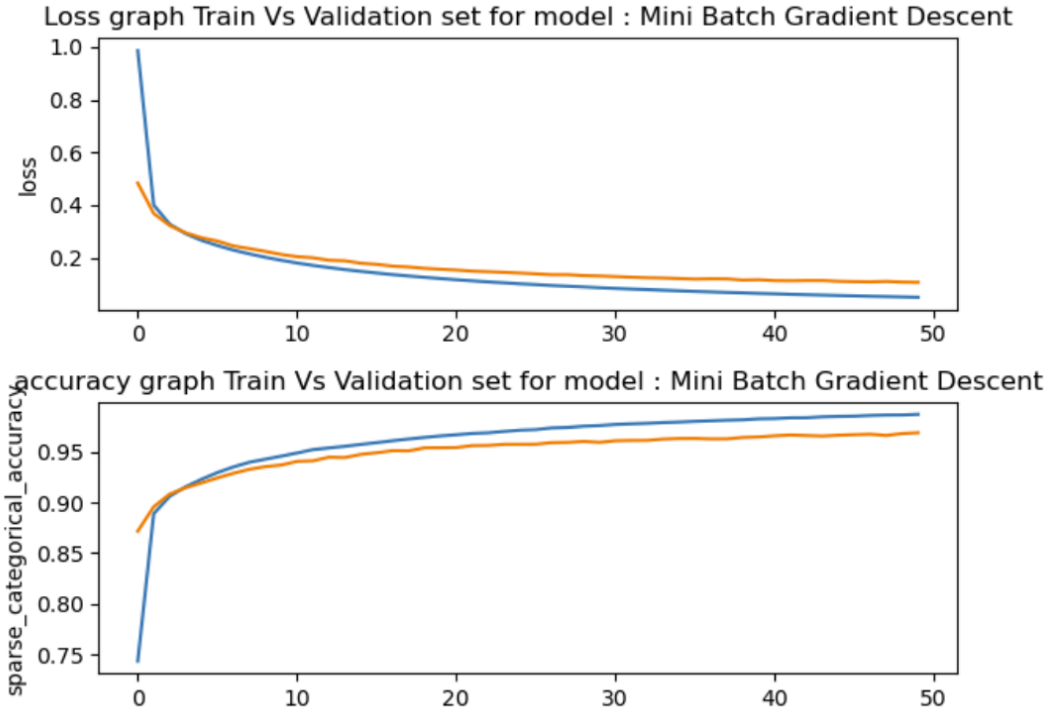

Figure 9: The convergence graph for the model : Mini Batch Gradient Descent

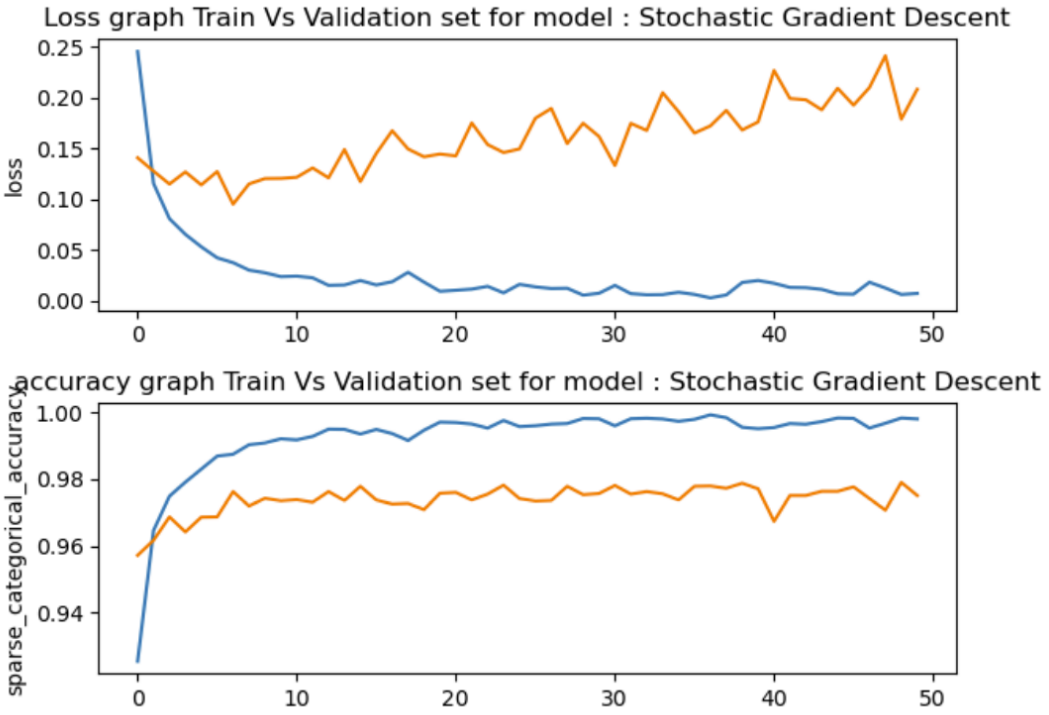

Figure 10: The convergence graph for the model : Stochastic Gradient Descent

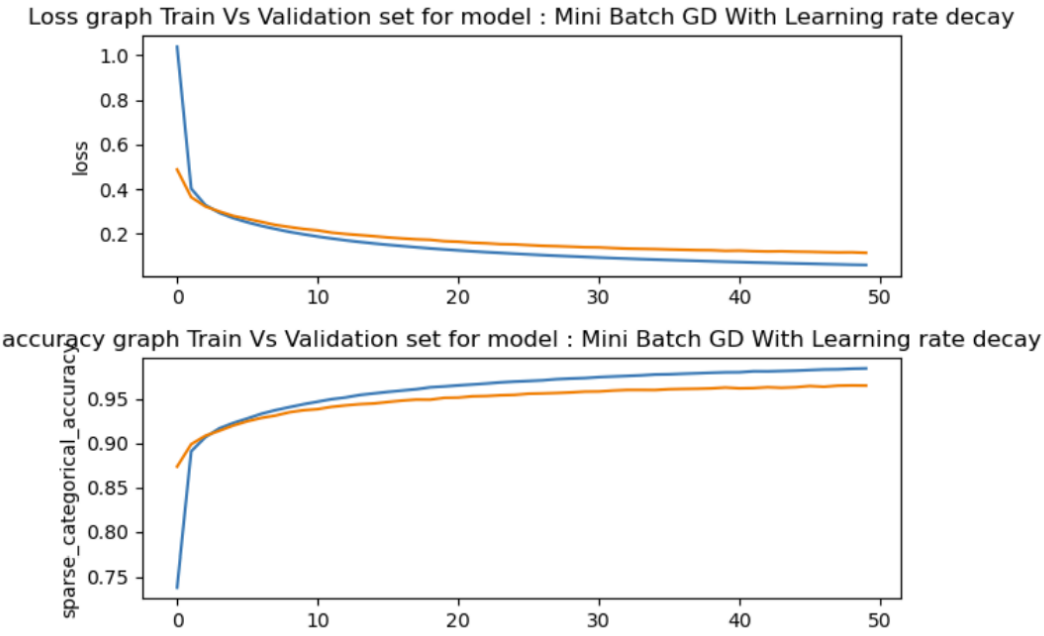

Figure 11: The convergence graph for the model : Mini Batch Gradient Descent With Learning rate decay

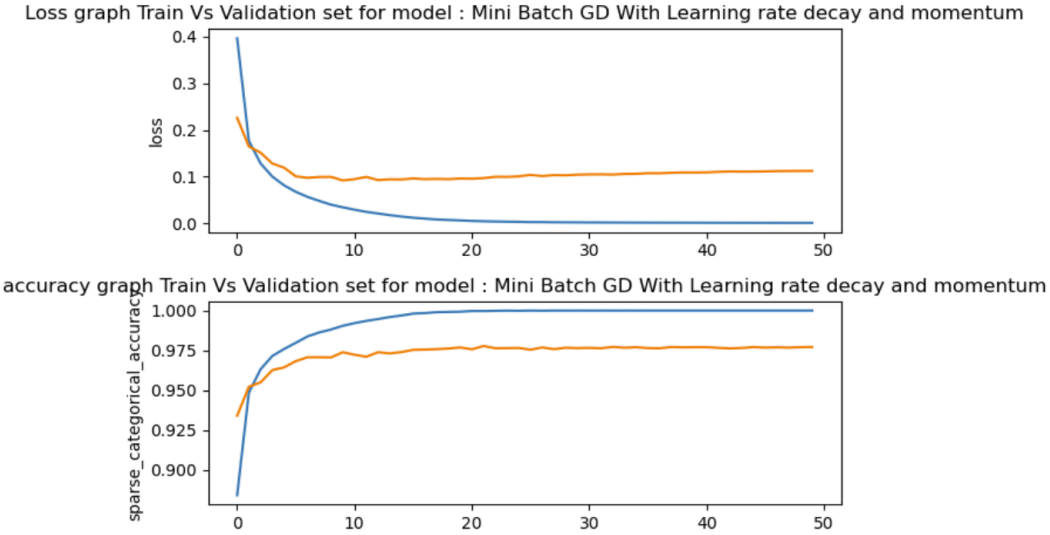

Figure 12: The convergence graph for the model : Mini Batch Gradient Descent With Learning rate decay and momentum

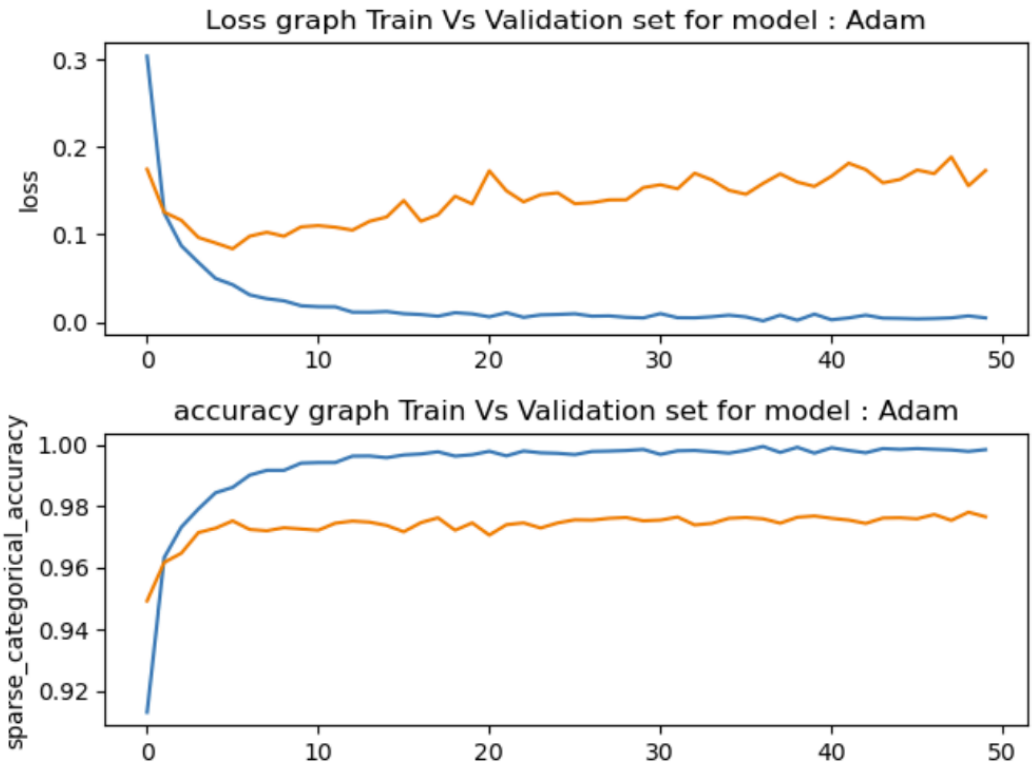

Figure 13: The convergence graph for the model : Adam

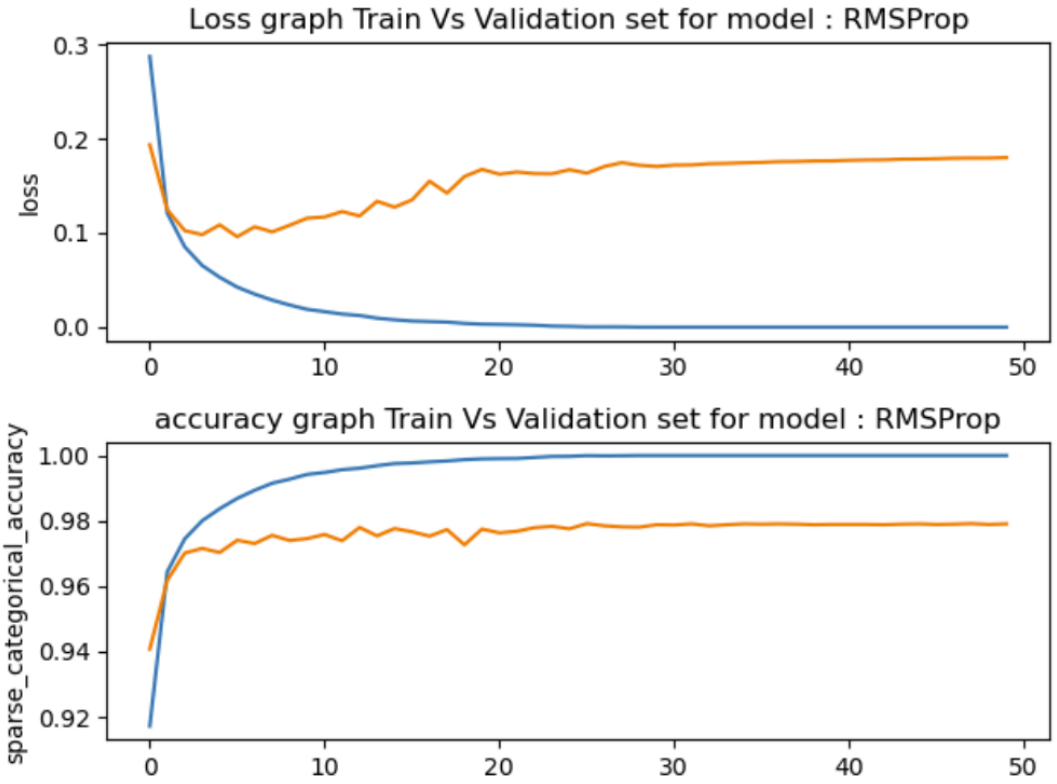

Figure 14: The convergence graph for the model : RMSProp

Even though Adam and RMSProp converges faster they can lead to overfitting if trained for longer than needed with a large value of the learning rate, the stochastic gradient descent however performs worst on the validation set due to the noise introduced during training (the bigger the batch size the noisier the training).

3 Part Two: Optimizing Hyperparameters

3.1 Discussing the Training Time of the Models

	model	duration
0	Mini-Batch SGD	102.652152
1	Mini-Batch SGD + Early Stopping	57.742547
2	Mini-Batch SGD With L2 Nomralization	100.922874
3	Mini-Batch SGD With Dropout Rate = 0.2	105.407015
4	Mini-Batch SGD With Dropout Rate = 0.3	105.240382
5	Mini-Batch SGD With Dropout Rate = 0.5	106.039060
6	Mini-Batch SGD With Batch Normalization	107.224656
7	Random Search	187.643153

Figure 15: The histogram of the training time for each model

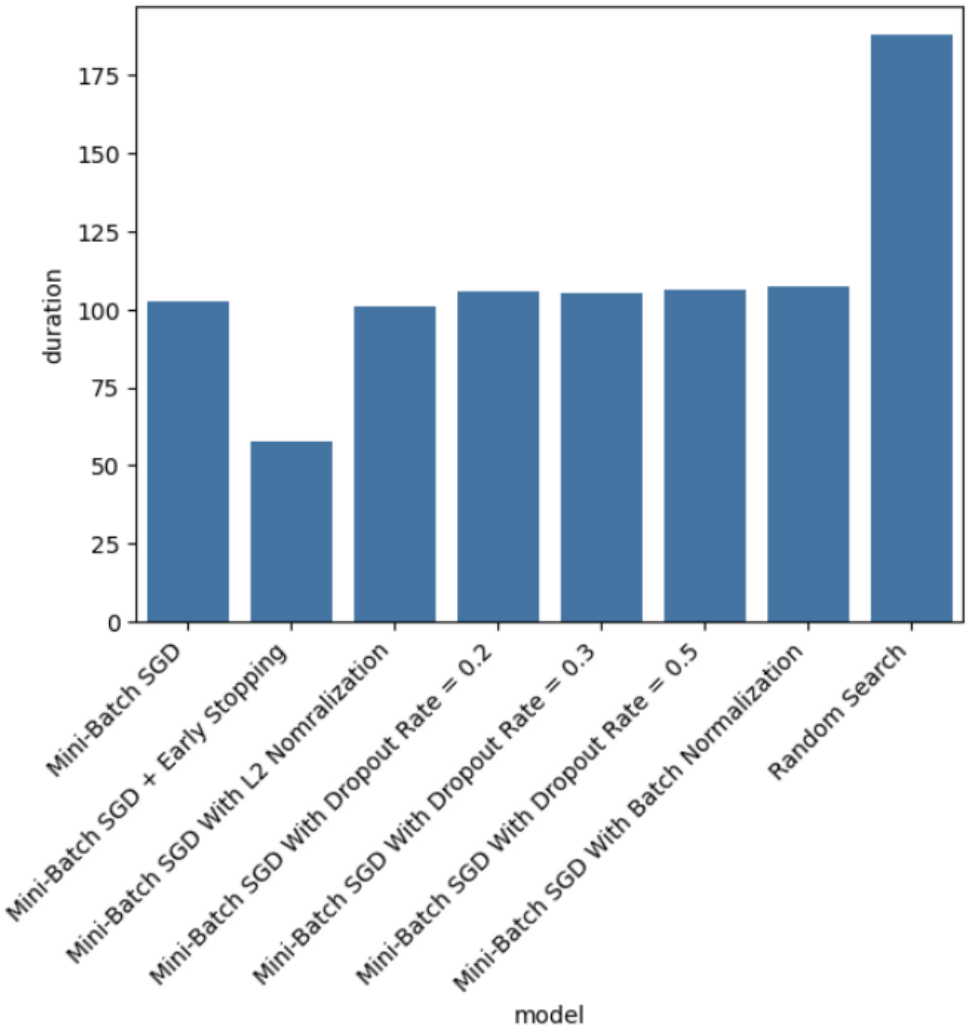

Figure 16: The histogram of the training time for each model

The training time for most models is roughly consistent, with the exception of the model utilizing early stopping, which demonstrates a significantly shorter training duration. This reduction occurs because training halts at epoch 28 instead of continuing until epoch 50. In contrast, the random search model exhibits the longest training time among all models. This is due to the optimal batch size identified through random search being 16, which is smaller than the batch sizes used for training the other models. As smaller batch sizes inherently increase the number of iterations and computational overhead, they result in longer training times.

3.2 Discussing the Performances of the Different Models

model	loss	accuracy
Mini-Batch SGD	1.506944	0.4957
Mini-Batch SGD + Early Stopping	1.466845	0.4882
Mini-Batch SGD With L2 Nomralization	1.528138	0.4837
Mini-Batch SGD With Dropout Rate = 0.2	1.495598	0.4828
Mini-Batch SGD With Dropout Rate = 0.3	1.455339	0.4884
Mini-Batch SGD With Dropout Rate = 0.5	1.491055	0.4762
Mini-Batch SGD With Batch Normalization	1.740329	0.4395
Random Search	1.460039	0.4928

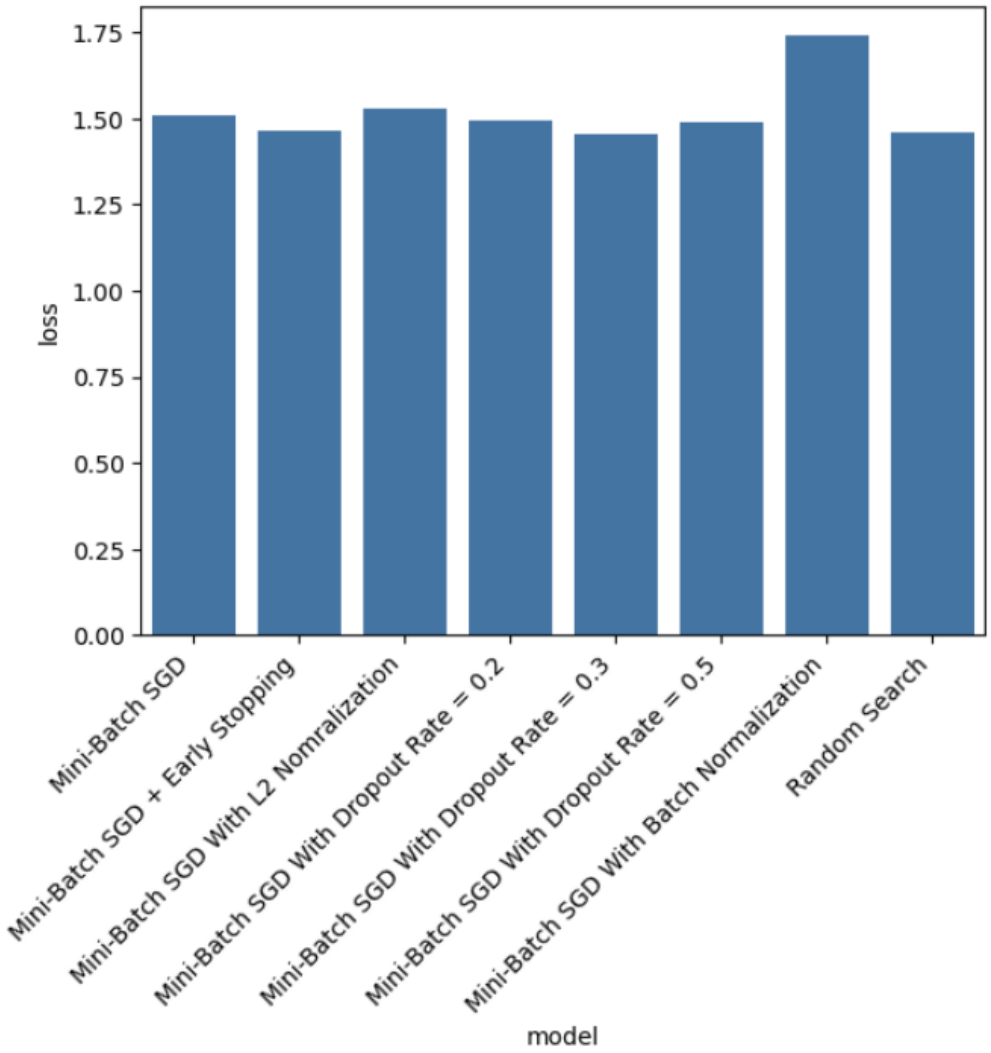

Figure 17: The loss histogram for different models on the validation set

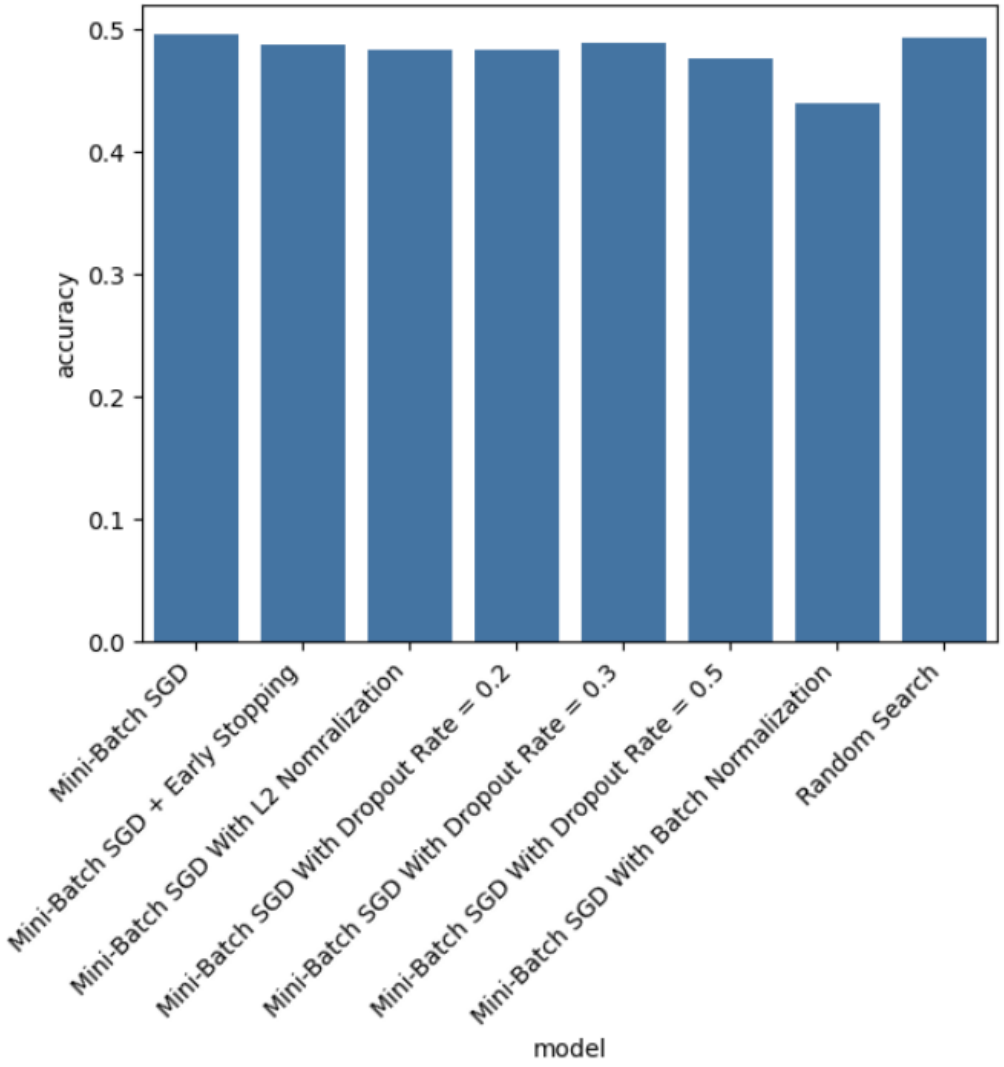

Figure 18: The accuracy histogram for different models on the validation set

The performance across the models is quite comparable, with the models trained using Mini-Batch Gradient Descent (Mini-Batch GD) and Mini-Batch GD combined with a dropout rate of 0.3 showing similar results. Unsurprisingly, the model trained with the best hyperparameters identified through random search performs slightly better than the others.

On the other hand, the model incorporating batch normalization in the second hidden layer exhibits the lowest accuracy and the highest loss. This underperformance may be attributed to the small batch size used during training, which can lead to unreliable estimates of the statistical parameters (mean μ and variance σ) as well as the learned parameters (α and β). Inaccurate estimations of these parameters can negatively impact the effectiveness of batch normalization, ultimately degrading the model's performance.

3.3 Comparing the Learning Graphs for the Different Models

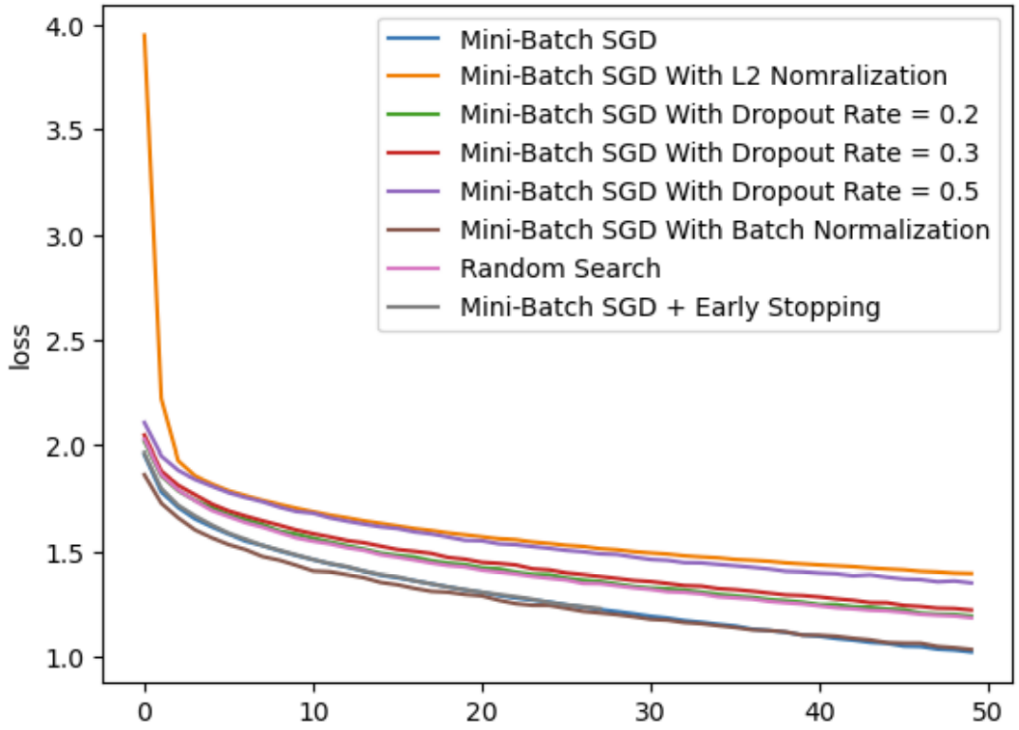

Figure 19: Learning Graphs

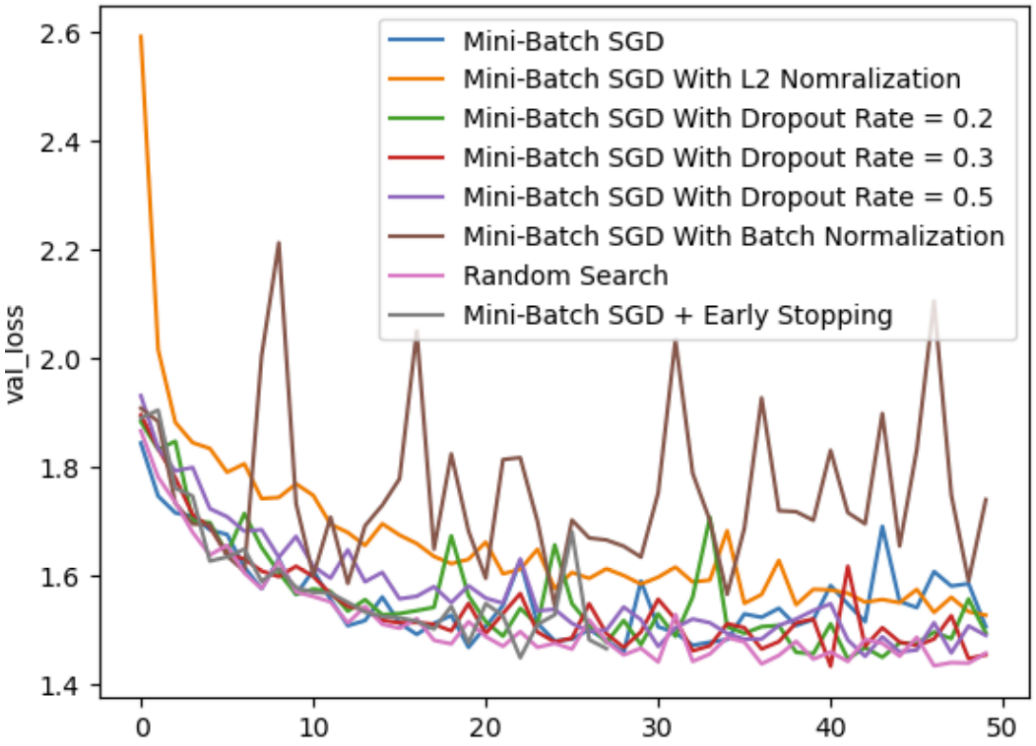

Figure 20: Learning Graphs

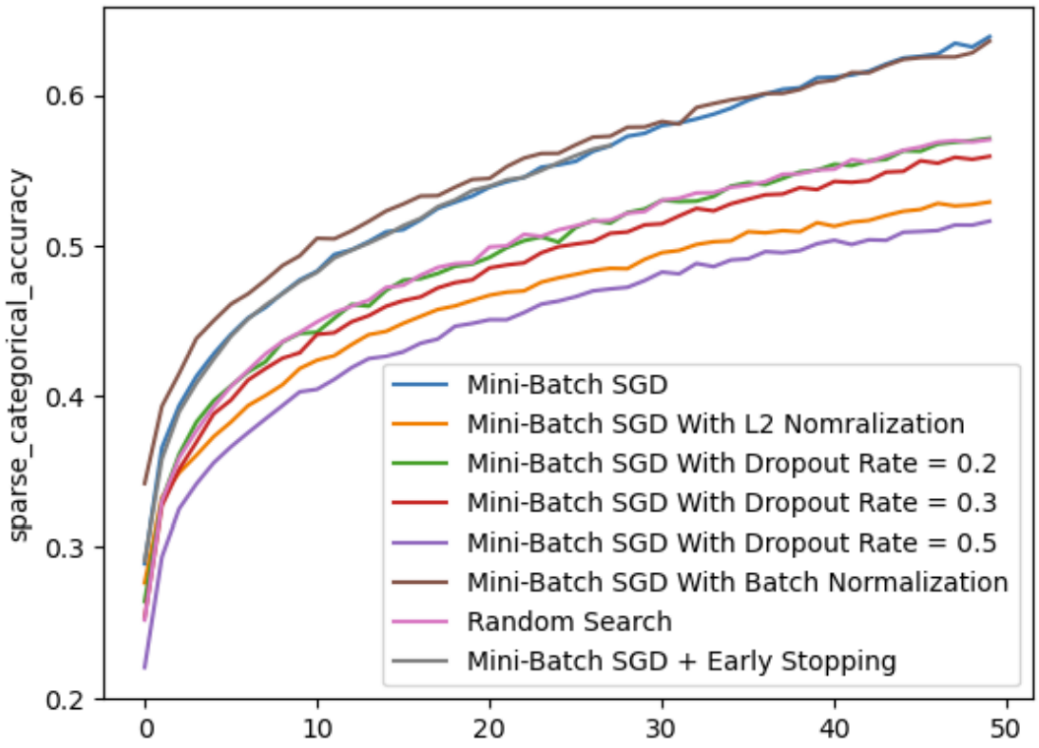

Figure 21: Learning Graphs

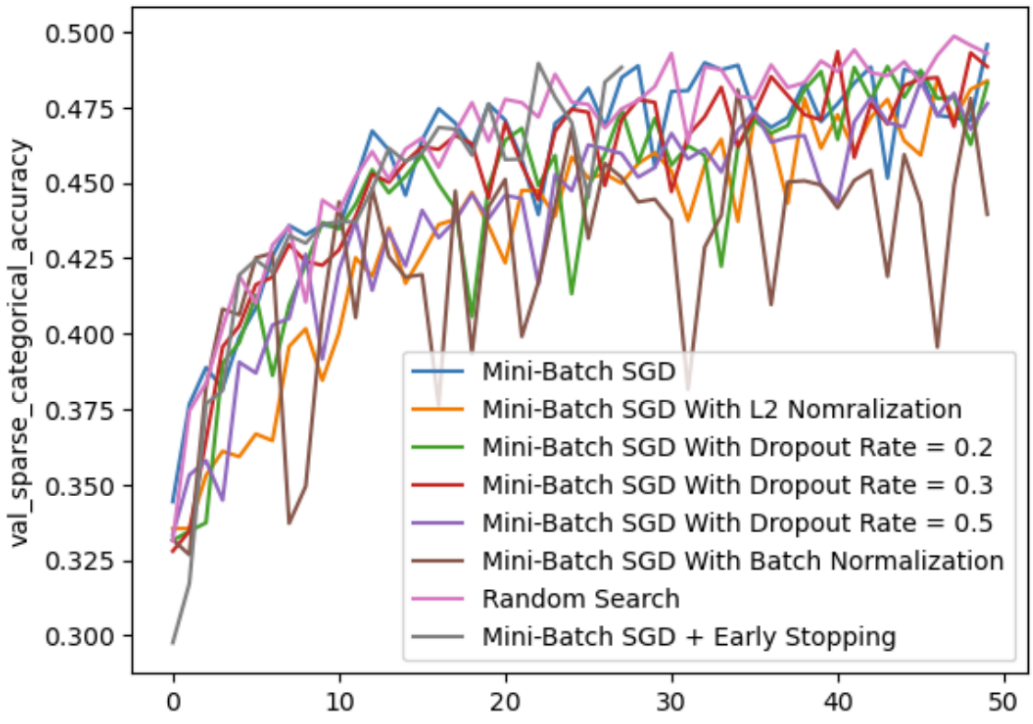

Figure 22: Learning Graphs

We notice that training graphs don't show much oscillations and the loss is decreasing steadily, with the model trained with L2 regularization taking a few more epochs for its loss to be in comparable range with the other model, it's also the model with highest loss which maybe due to a large regularization parameter for this particular problem (which may cause the model to even underfit), the model trained with batch normalization have the lowest training loss.

But the validation graphs don't tell the same story, there is so much oscillations specially in the case of the model trained with batch normalization which maybe due to the batch being small and the network not being very deep as it is recommended to use batch normalization in the case of very deep neural networks.

3.4 The Convergence Graphs for Each Model

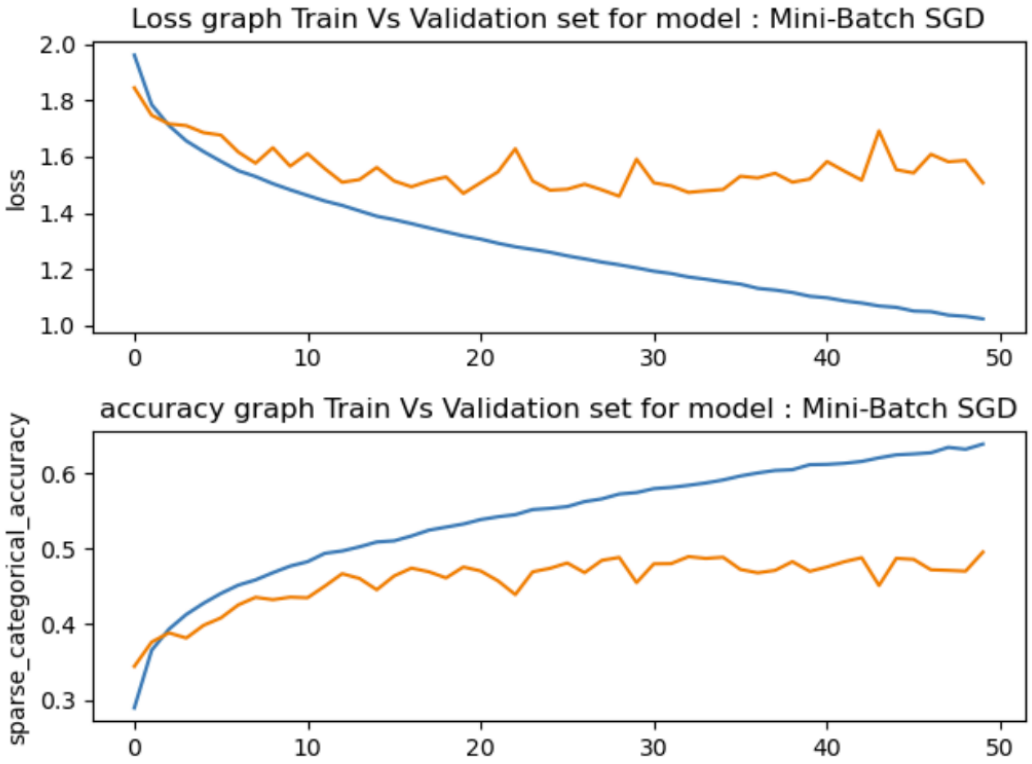

Figure 23: The convergence graph for the model : Mini-Batch GD

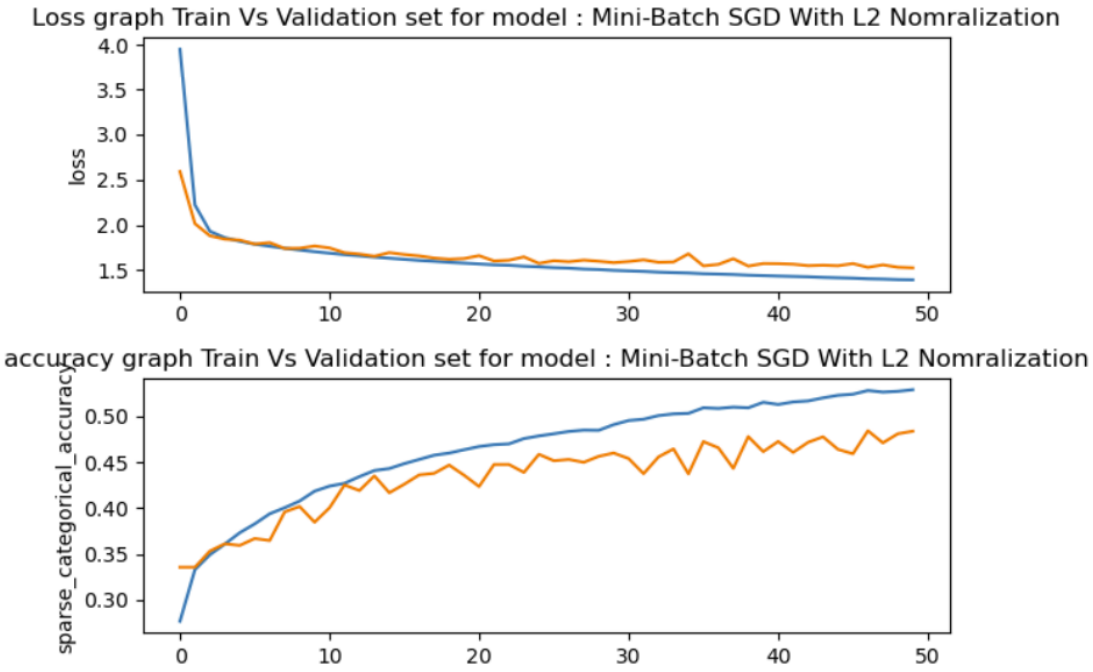

Figure 24: The convergence graph for the model : Mini Batch GD + L2 regularization

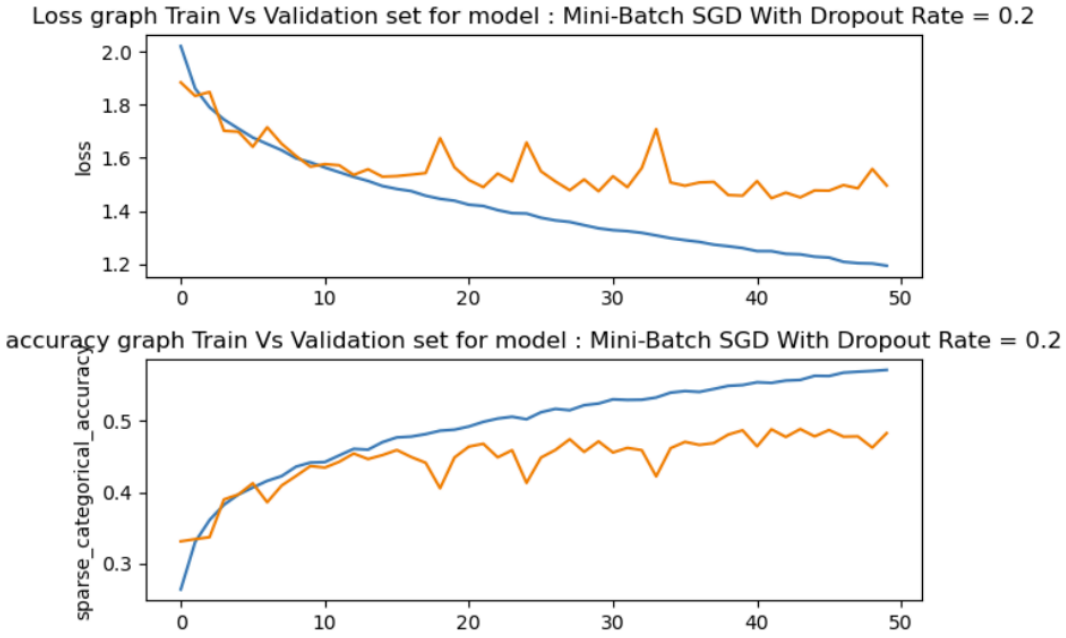

Figure 25: : The convergence graph for the model : Mini-Batch GD with dropout rate = 0.2

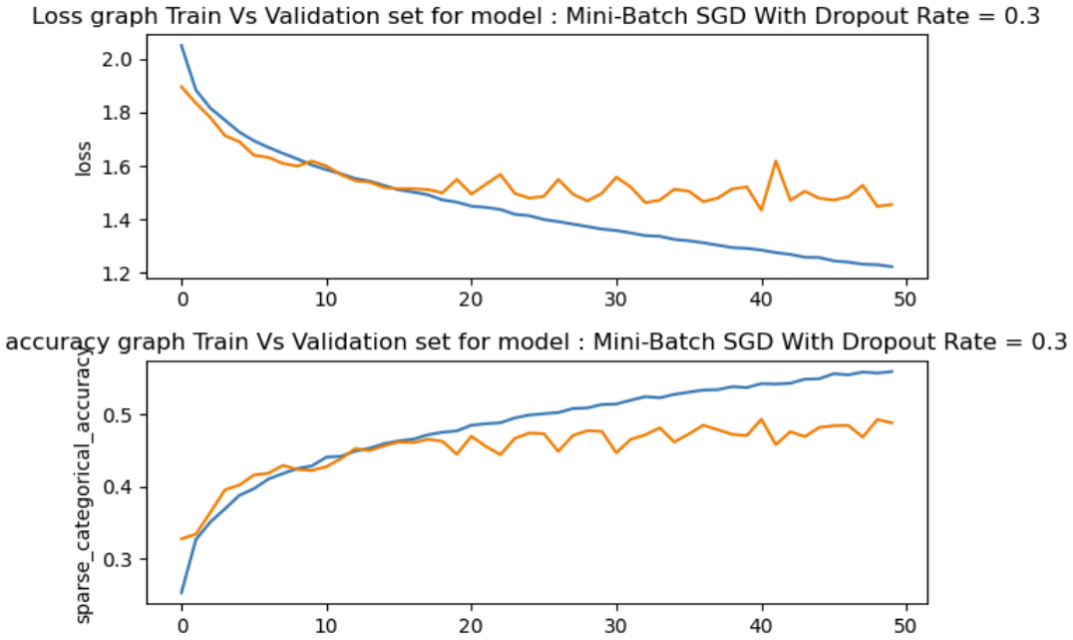

Figure 26: The convergence graph for the model : Mini-Batch GD with dropout rate = 0.3

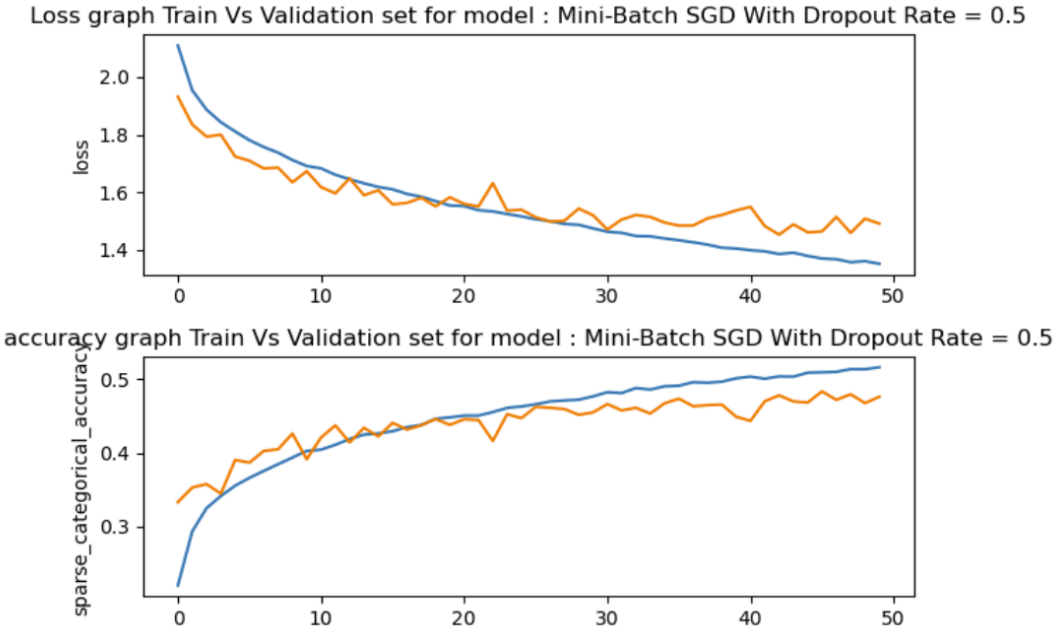

Figure 27: The convergence graph for the model : Mini-Batch GD with dropout rate = 0.5

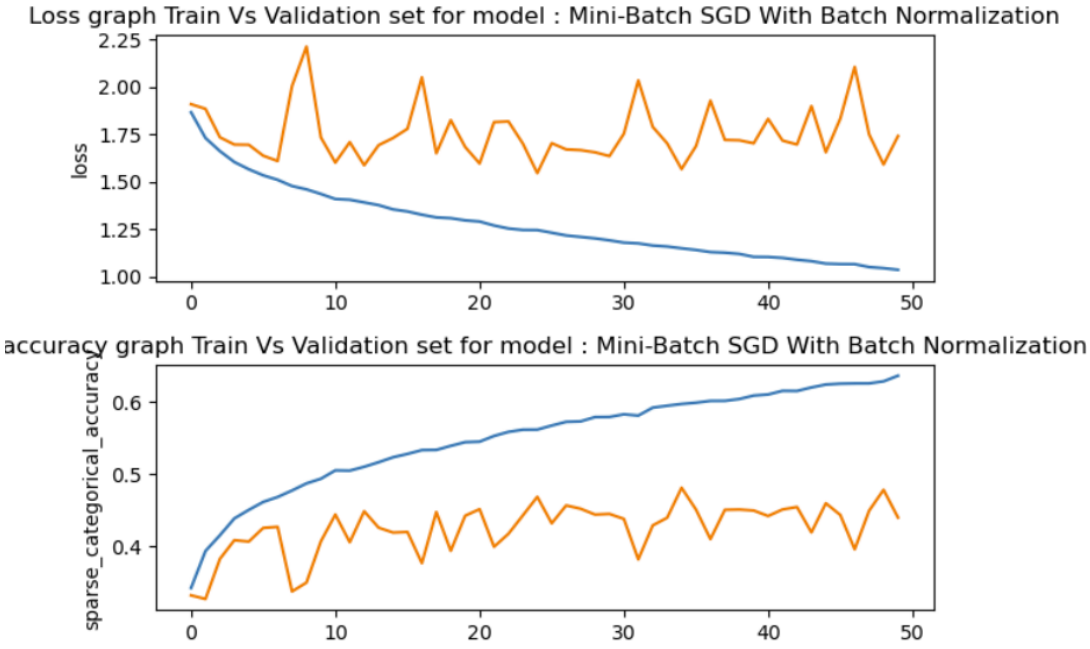

Figure 28: : The convergence graph for the model : Adam

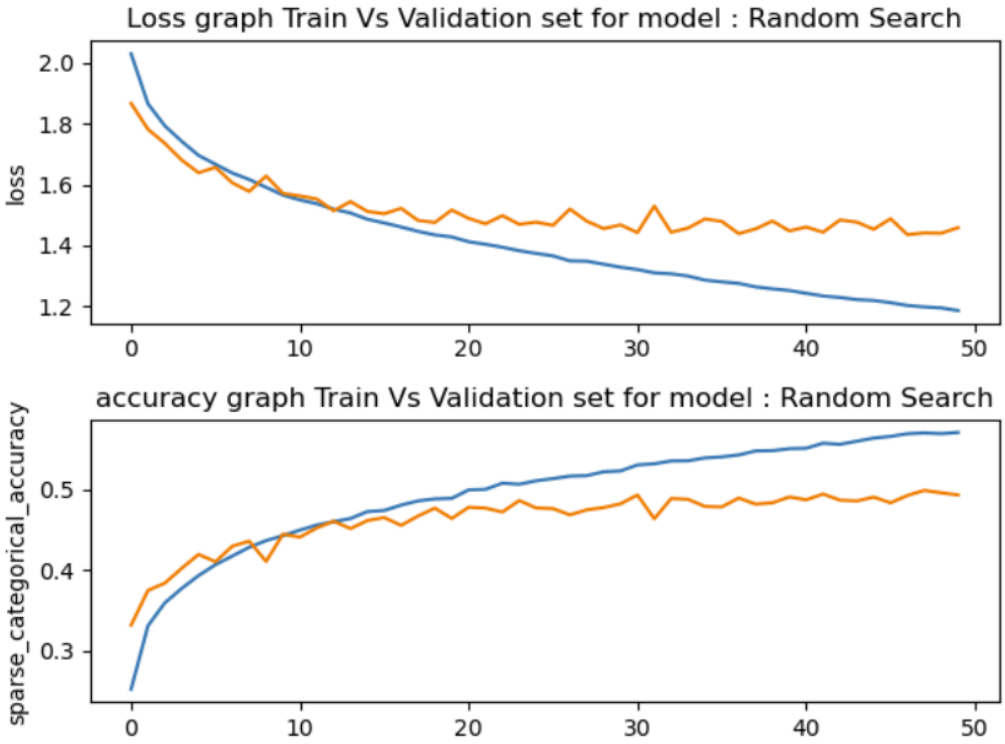

Figure 29: The convergence graph for the model : RMSProp

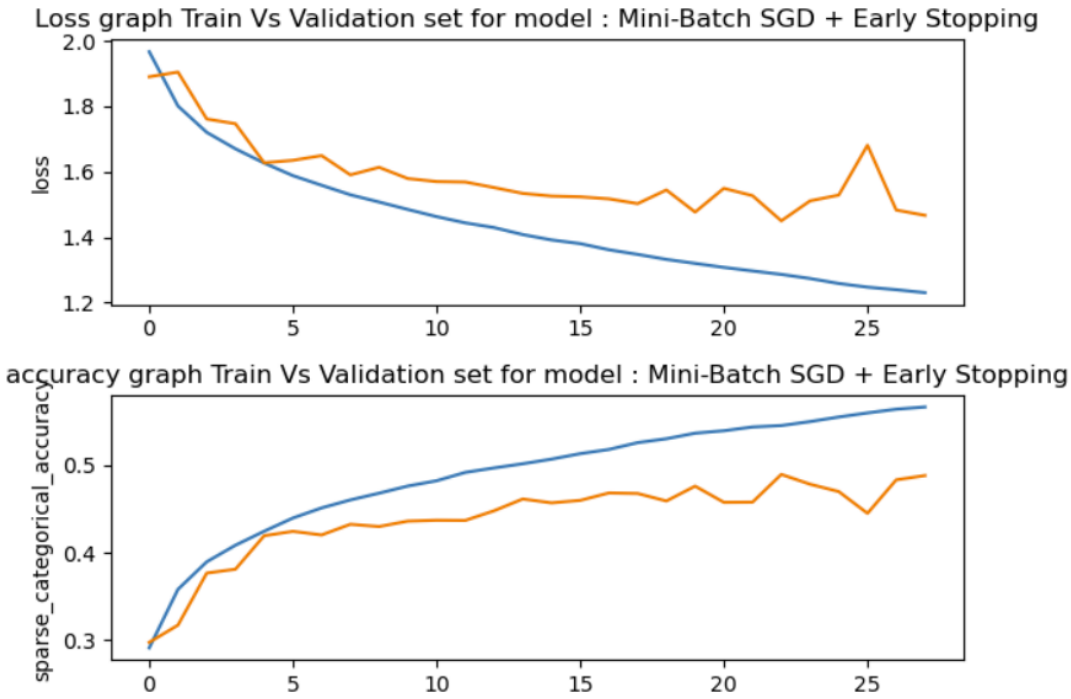

Figure 30: The convergence graph for the model : Mini-Batch GD + Early stopping

From the convergence graphs we can clearly visualize how can dropout and L2 regularization significantly reduce overfitting but also how a mis-configured batch normalization can lead to even worst results.

4 Conclusion

In conclusion, our experiments have yielded valuable insights into the training and optimization of neural networks using various optimizers and overfitting prevention techniques. We observed that the choice of optimizer plays a critical role in model performance, with Adam generally surpassing SGD and RMSprop in terms of both convergence speed

and final accuracy. Furthermore, we demonstrated that overfitting avoidance strategies, such as dropout layers, batch normalization, and L2 regularization, can significantly enhance the model's generalization capabilities by reducing overfitting and improving performance on unseen data.

Overall, our findings underscore the importance of carefully selecting and fine-tuning these components during the training of neural networks. By understanding the impact of different optimizers and overfitting prevention techniques, we can optimize the training process, ultimately leading to better model performance and improved generalization.