Analysis and Comparison of The Interaction Between Optimizers, Learning Rates and Batch Sizes

Săvianu Bogdan

May 24, 2025

Contents

1 Introduction

This report provides an analysis of the training and validation performance obtained from various runs. The focus is on comparing different optimizers, the impact of learning rates, the effect of changing batch sizes, and consistency across different K-Folds.

I will examine results from the following main configurations:

- Learning Rate 0.0001 with Cross Entropy Loss with batch sizes 16 and 64
- Learning Rate 0.001 with Cross Entropy Loss with batch sizes 16 and 64
- Learning Rate 0.001 with KLDivLoss (only on Adam optimizer to cut down runtime)

2 General Observations on Batch Size Change

Increasing batch size has led to:

- Smoother training loss curves due to more stable gradient estimates.
- Potentially faster training per epoch in terms of wall-clock time. However, this did not come without exceptions.

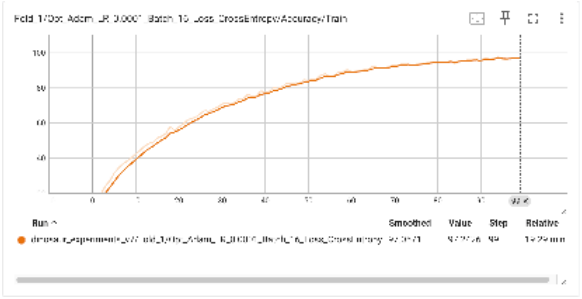
3 Analysis by Configuration

3.1 LR 0.0001, Cross Entropy Loss

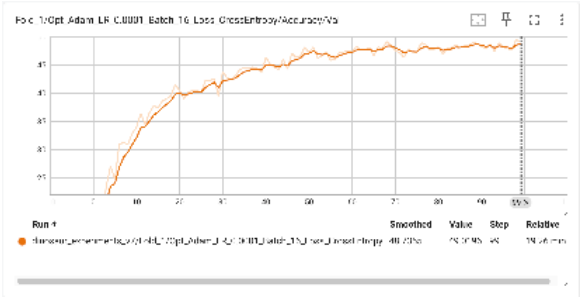
In this section I have analyzed optimizers (Adam, RMSProp, SGD) with a learning rate of 0.0001 and Cross Entropy loss, comparing batch sizes 16 and 64 for Fold 1.

3.1.1 Adam (LR 0.0001, CrossEntropy, Fold 1)

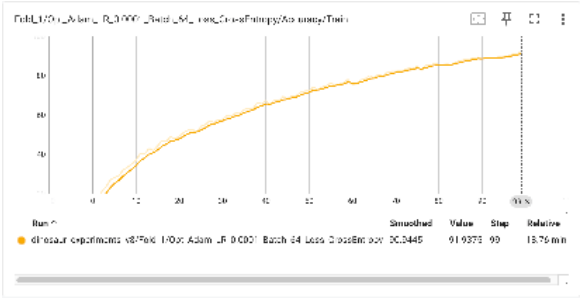
- **Performance (b64 vs b16, Fold 1):** Adam with b16 showed a slightly quicker convergence, and also slightly higher overfitting. Training seemed to be faster with b64, but marginally. The higher batch size did not seem to impact my GPU performance, which, by analyzing with the asitop command line tool, was never used to its full power. On the other hand, the CPU often hit 99% pressure when preparing the data for training.



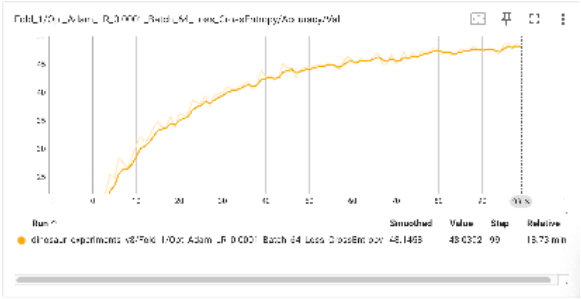
(a) Adam Train (LR 0.0001, CE, B16, F1)



(b) Adam Val (LR 0.0001, CE, B16, F1)



(c) Adam Train (LR 0.0001, CE, B64, F1)

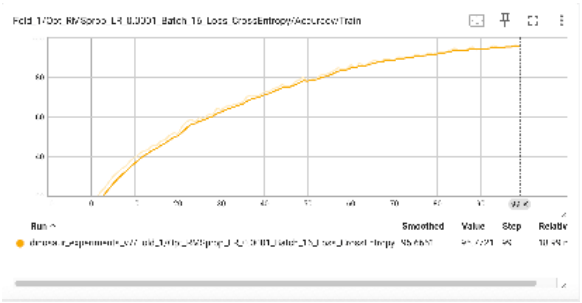


(d) Adam Val (LR 0.0001, CE, B64, F1)

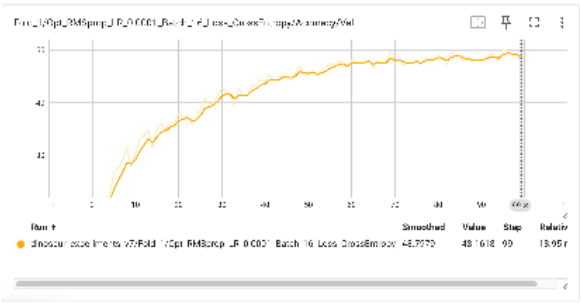
Figure 1: Adam Performance (LR 0.0001, CrossEntropy, Fold 1)

3.1.2 RMSProp (LR 0.0001, CrossEntropy, Fold 1)

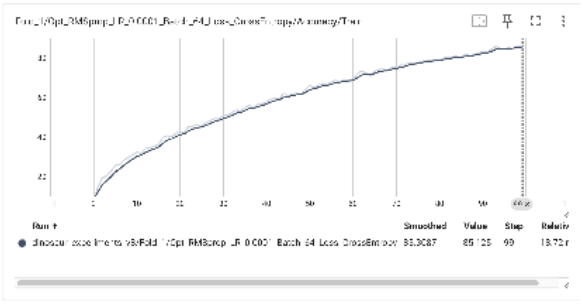
- **Performance (b64 vs b16, Fold 1):** Similar to Adam, overfitting is enormous especially in the case of b16 which reached 95% accuracy by step 99. validation also converged a little more quickly in the case of b16, but the end result was fairly close.



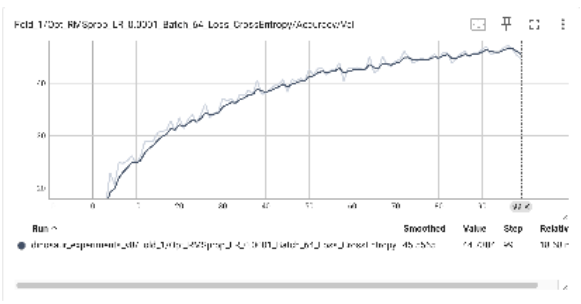
(a) RMSProp Train (LR 0.0001, CE, B16, F1)



(b) RMSProp Val (LR 0.0001, CE, B16, F1)



(c) RMSProp Train (LR 0.0001, CE, B64, F1)

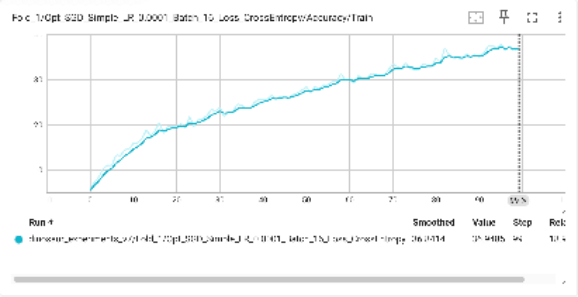


(d) RMSProp Val (LR 0.0001, CE, B64, F1)

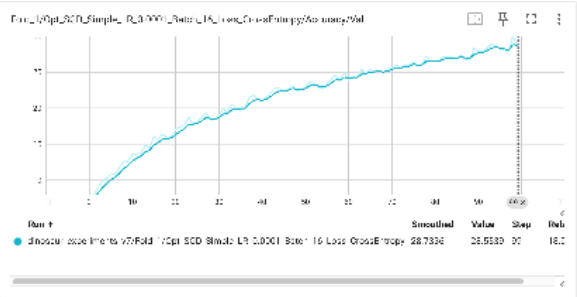
Figure 2: RMSProp Performance (LR 0.0001, CrossEntropy, Fold 1)

3.1.3 SGD (LR 0.0001, CrossEntropy, Fold 1)

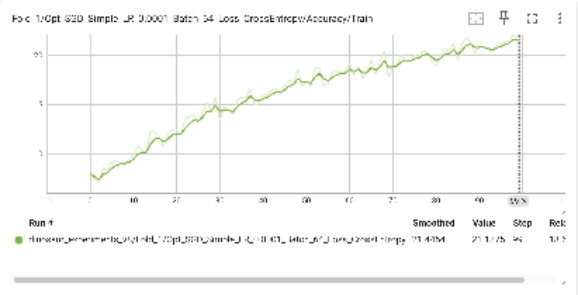
- **Performance (b64 vs b16, Fold 1):** In the case of SGD, we notice that overfitting is considerably higher when using b16 and the validation result was also twice as hight with b16, though the value was still rising in both cases. It can also be noted that validation with b64 was more erratic which makes it less predictable.



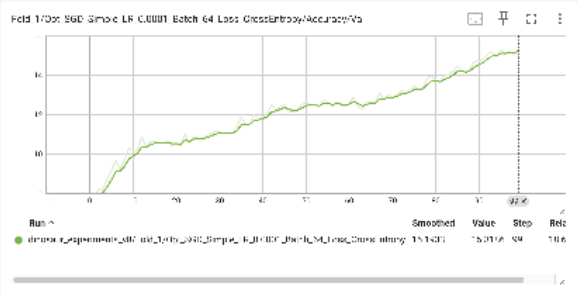
(a) SGD Train (LR 0.0001, CE, B16, F1)



(b) SGD Val (LR 0.0001, CE, B16, F1)



(c) SGD Train (LR 0.0001, CE, B64, F1)



(d) SGD Val (LR 0.0001, CE, B64, F1)

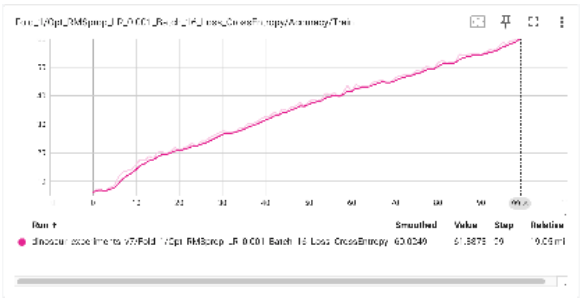
Figure 3: SGD Performance (LR 0.0001, CrossEntropy, Fold 1)

3.2 LR 0.001, Cross Entropy Loss - Comparison with LR 0.0001

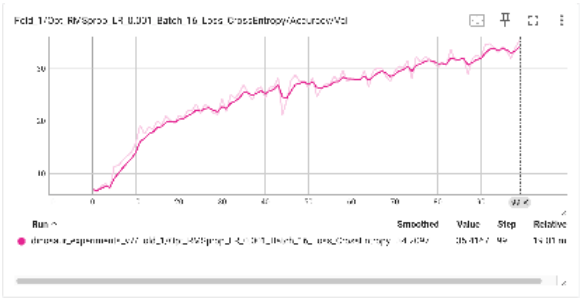
This section analyzes optimizers (Adam, RMSProp, SGD) with a learning rate of 0.001 and Cross Entropy loss, comparing their performance against the LR 0.0001 results from the previous section for the same optimizer and batch size.

3.2.1 RMSProp (LR 0.001 vs LR 0.0001, CrossEntropy)

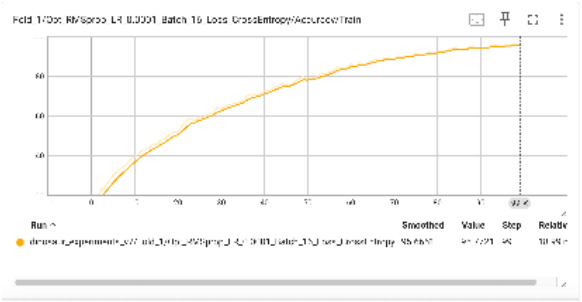
- **LR Impact (Batch Size 16, Fold 1):** It can be noticed that the smaller learning rate lead to much faster learning in this case along with higher overfitting. The graph also shows a smoother curve for validation when using LR 0.0001. LR 0.0001 managed to obtain 48% validation which seems quite high given the task of classifying dinosaurs.
- **LR Impact (Batch Size 64, Fold 1):** The aforementioned trends seem to apply here as well: quicker and faster learning with the smaller LR.



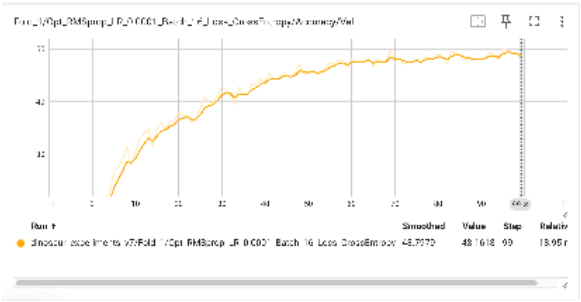
(a) RMSProp Train (LR 0.001, CE, B16, F1)



(b) RMSProp Val (LR 0.001, CE, B16, F1)

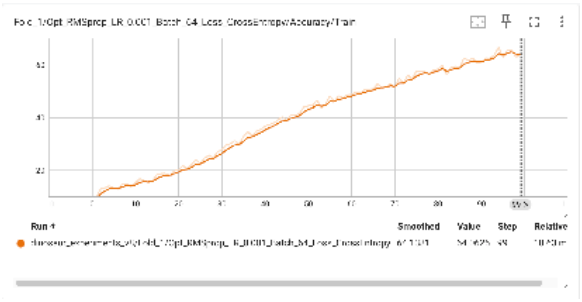


(c) RMSProp Train (LR 0.0001, CE, B16, F1)

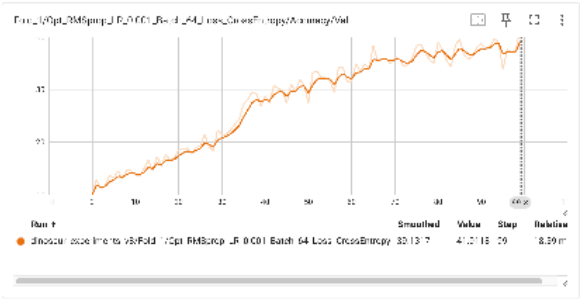


(d) RMSProp Val (LR 0.0001, CE, B16, F1)

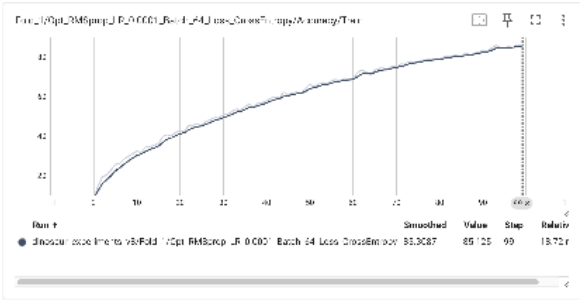
Figure 4: RMSProp LR Impact (CE, Batch 16, Fold 1, LR 0.001 vs LR 0.0001)



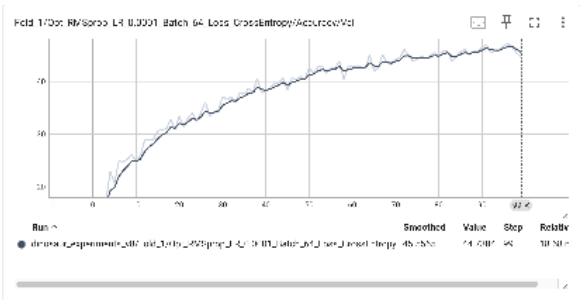
(a) RMSProp Train (LR 0.001, CE, B64, F1)



(b) RMSProp Val (LR 0.001, CE, B64, F1)



(c) RMSProp Train (LR 0.0001, CE, B64, F1)

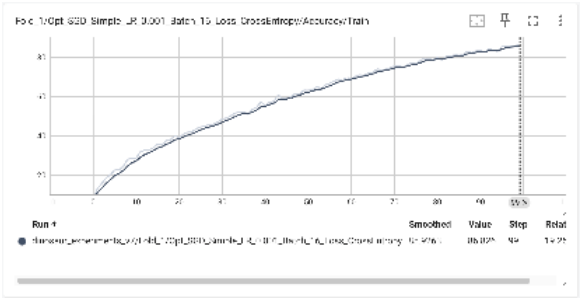


(d) RMSProp Val (LR 0.0001, CE, B64, F1)

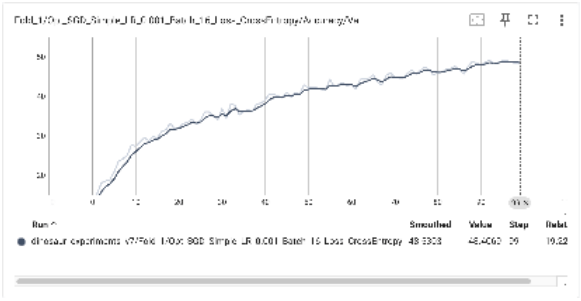
Figure 5: RMSProp LR Impact (CE, Batch 64, Fold 1, LR 0.001 vs LR 0.0001)

3.2.2 SGD (LR 0.001 vs LR 0.0001, CrossEntropy)

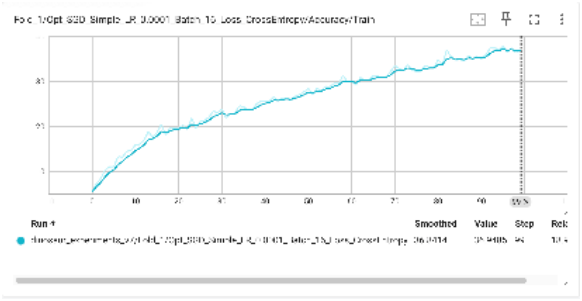
- **LR Impact (Batch Size 16, Fold 1):** Contrary to what has been noted for RMSProp, the smaller learning rate in the case of SGD actually made it learn more slowly, but overfitting was reduced proportionally too.
- **LR Impact (Batch Size 64, Fold 1):** The same trend as in SGD B16



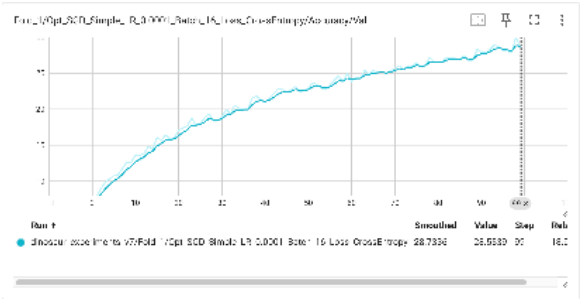
(a) SGD Train (LR 0.001, CE, B16, F1)



(b) SGD Val (LR 0.001, CE, B16, F1)

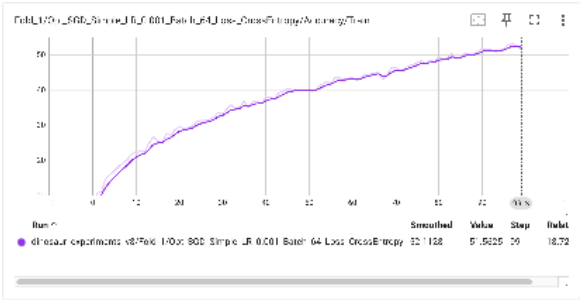


(c) SGD Train (LR 0.0001, CE, B16, F1)

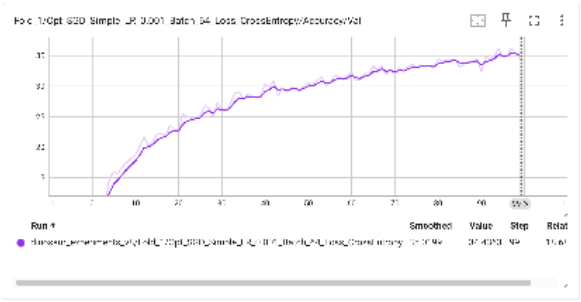


(d) SGD Val (LR 0.0001, CE, B16, F1)

Figure 6: SGD LR Impact (CE, Batch 16, Fold 1, LR 0.001 vs LR 0.0001)

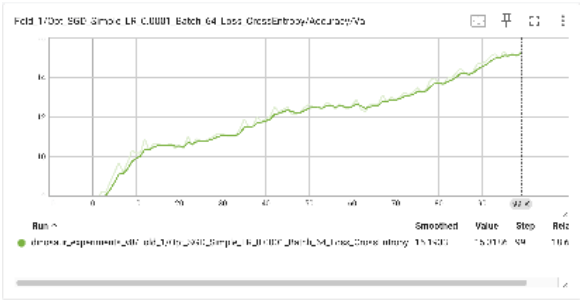


(a) SGD Train (LR 0.001, CE, B64, F1)



(b) SGD Val (LR 0.001, CE, B64, F1)

(c) SGD Train (LR 0.0001, CE, B64, F1)

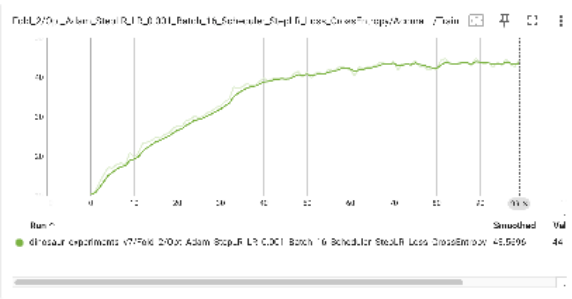


(d) SGD Val (LR 0.0001, CE, B64, F1)

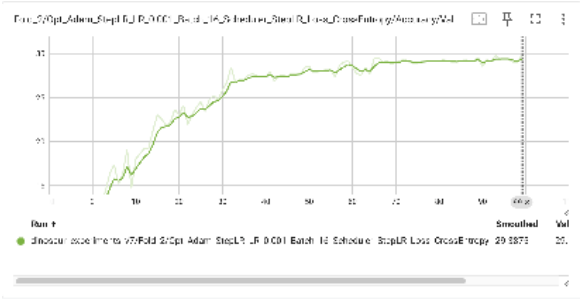
Figure 7: SGD LR Impact (CE, Batch 64, Fold 1, LR 0.001 vs LR 0.0001)

3.2.3 Adam with StepLR (LR 0.001, CrossEntropy) - Scheduler Effect Analysis

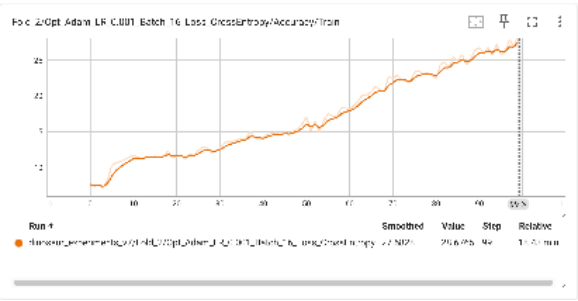
- **Effect of StepLR (Fold 2, Batch Size 16):** The scheduler made the results converge much sooner. As for the final validation result, not much concrete can be said since the model without the scheduler had not converged yet after 100 steps.
- **Effect of StepLR (Fold 2, Batch Size 64):** The final validation result was not too different, about 4% in favor of the scheduler, but overfitting was reduced by 30%.



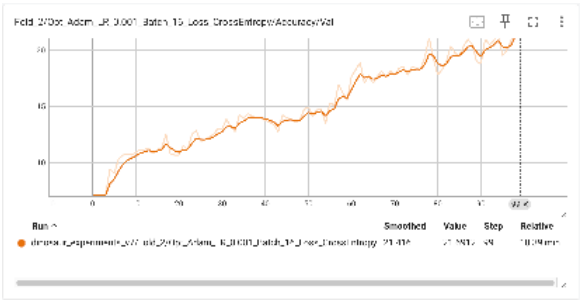
(a) Adam w/ StepLR Train (B16, F2)



(b) Adam w/ StepLR Val (B16, F2)

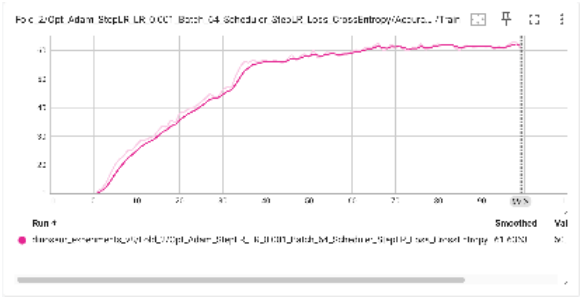


(c) Adam w/o StepLR Train (B16, F2)

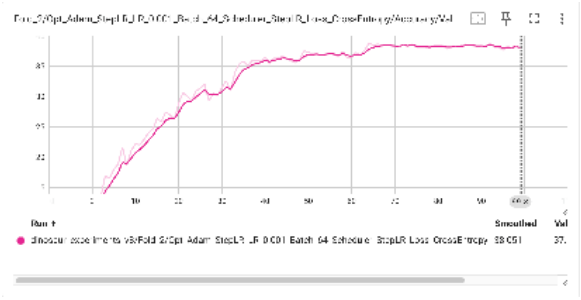


(d) Adam w/o StepLR Val (B16, F2)

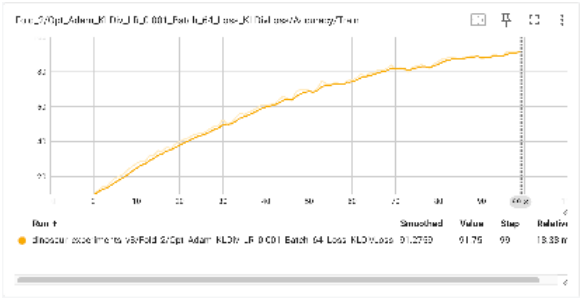
Figure 8: Adam StepLR vs NoLR (LR 0.001, CrossEntropy, Fold 2, Batch 16)



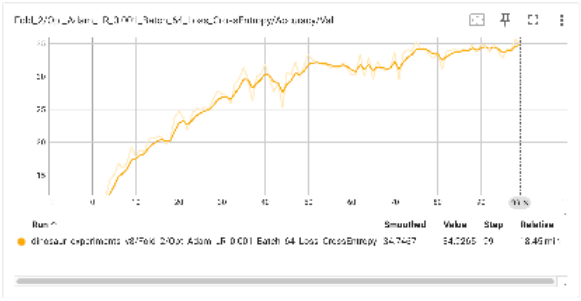
(a) Adam w/ StepLR Train (B64, F2)



(b) Adam w/ StepLR Val (B64, F2)



(c) Adam w/o StepLR Train (B64, F2)



(d) Adam w/o StepLR Val (B64, F2)

Figure 9: Adam StepLR vs NoLR (LR 0.001, CrossEntropy, Fold 2, Batch 64)

3.3 LR 0.001, KLDivLoss

This section focuses on the Adam optimizer with KLDivLoss at LR 0.001.

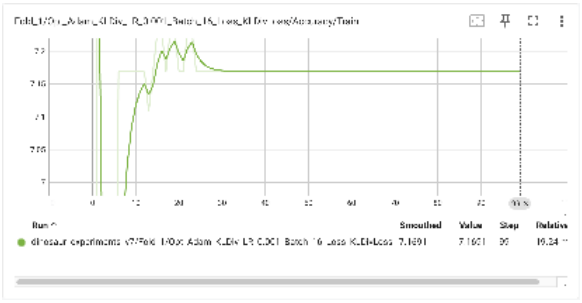
3.3.1 Adam (LR 0.001, KLDivLoss)

- **Fold 1 (b64 vs b16):** The most interesting observation of the entire project happened right here. Using a combination batch size 16 and KLDiv loss function resulted in a terrible outcome where the model hit a plateau at 7.2% both in training and validation. No amount of training helped it get out of that spot.

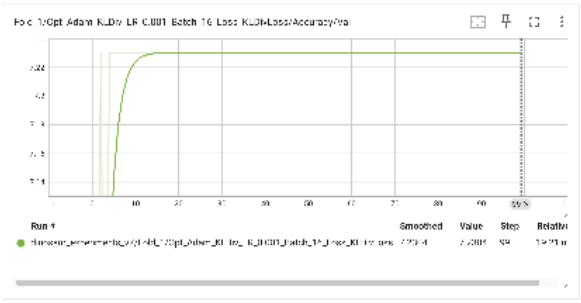
  Surprisingly, a larger batch size solved the previous issue. The validation result was the best among all the other models at 49%, but the training value was the highest as well at 98%.

- **Fold 2 (b64 vs b16):** Even more surprising, by splitting the dataset in 3 sets using the K-Fold method, the model did not suffer the same fate when using batch size 16.

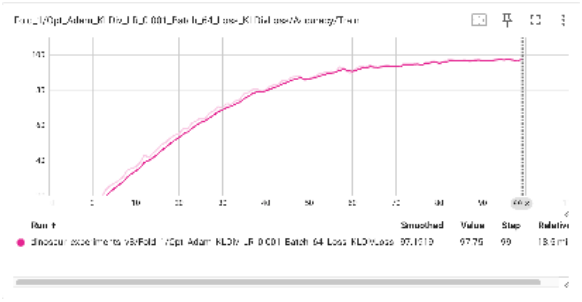
  Also, in fold 2 batch size 64 presented higher overfitting but the validation result stayed the same.



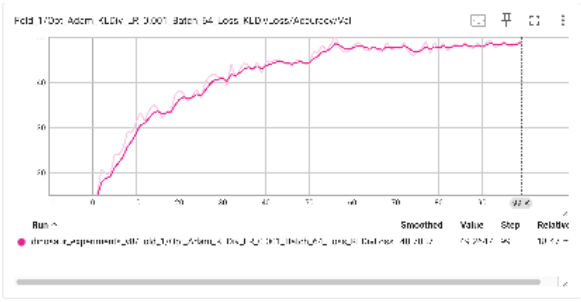
(a) Adam Train (LR 0.001, KLDiv, B16, F1)



(b) Adam Val (LR 0.001, KLDiv, B16, F1)

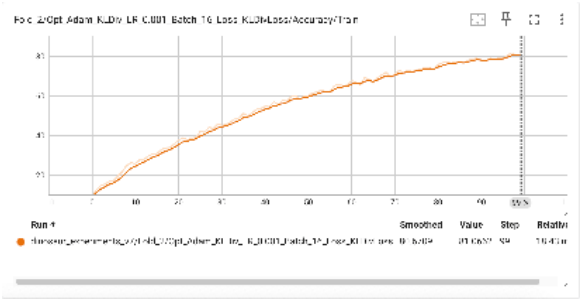


(c) Adam Train (LR 0.001, KLDiv, B64, F1)

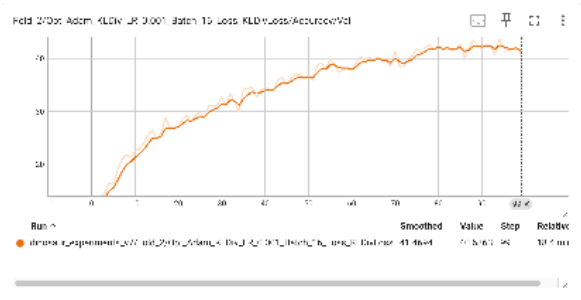


(d) Adam Val (LR 0.001, KLDiv, B64, F1)

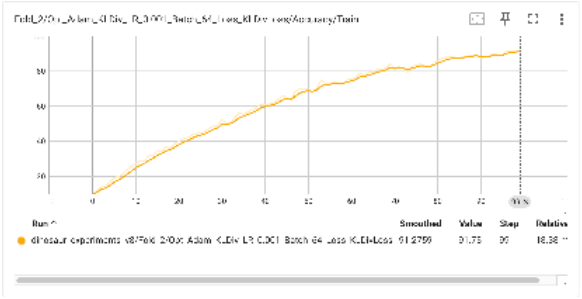
Figure 10: Adam Performance (LR 0.001, KLDivLoss, Fold 1)



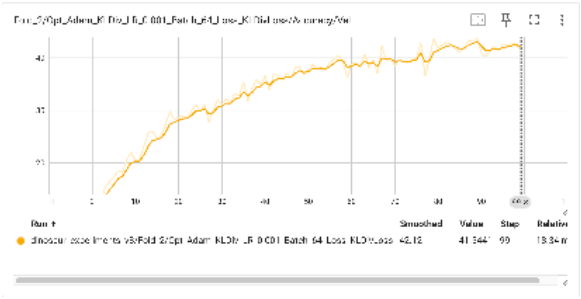
(a) Adam Train (LR 0.001, KLDiv, B16, F2)



(b) Adam Val (LR 0.001, KLDiv, B16, F2)



(c) Adam Train (LR 0.001, KLDiv, B64, F2)



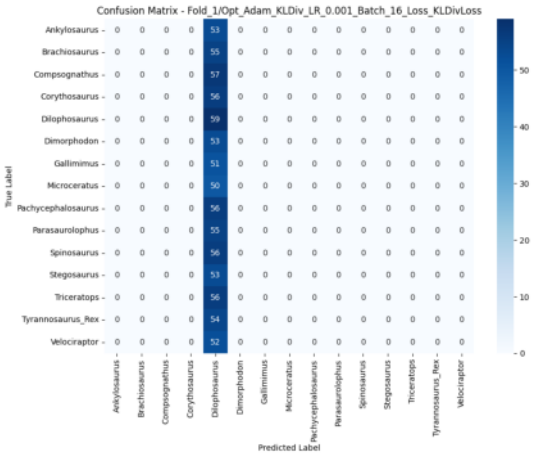
(d) Adam Val (LR 0.001, KLDiv, B64, F2)

Figure 11: Adam Performance (LR 0.001, KLDivLoss, Fold 2)

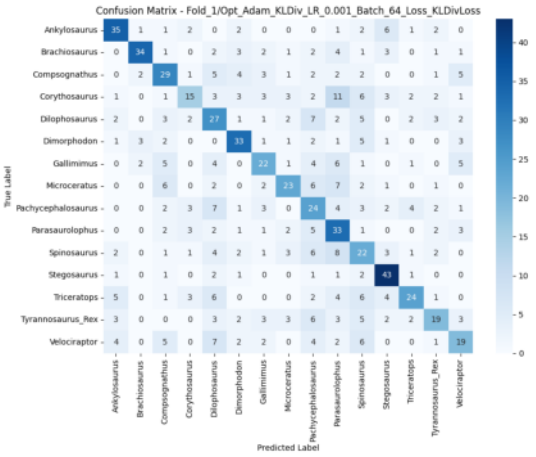
4 Confusion Matrices

By viewing the confusion matrices, the weird behavior of Adam with KLDiv Loss and a learning rate of 0.001 can be visualized more clearly. It seems like it erroneously predicted that absolutely all of the images were part of the Dilophosaurus class and the only correct answers were, in fact, only for the actual Dilophosaurus images.

Side by side we have an image of the most successful model where the 49% is visible by the pattern where the main diagonal is quite pronounced, further confirming the decent accuracy of the model (again, keeping in mind it was performing a complex task).



(a) CM: Adam, LR 0.001, KLDiv, B16, F1 (Plateaued at 7.2% Acc)



(b) CM: Adam, LR 0.001, KLDiv, B64, F1 (49% Val Acc)

Figure 12: Comparison of Confusion Matrices: KLDivLoss with Adam (Fold 1)

5 Comparative Analysis Across Configurations

5.1 Optimizer Comparison

Across the Cross Entropy Loss experiments, particularly at LR 0.0001, Adam and RMSProp generally showed faster convergence and achieved higher training accuracies compared to SGD, but this often came with significant overfitting, especially with batch size 16. For instance, RMSProp (LR 0.0001, Cross Entropy, b16) reached 95% training accuracy quickly. SGD, while slower to train, sometimes showed less severe overfitting or, in the case of LR 0.0001 b64, more erratic validation performance. The Adam optimizer, when combined with KLDivLoss (LR 0.001, b64), yielded the highest validation accuracy (49%), though it also had the highest training accuracy (98%), indicating substantial overfitting. The StepLR scheduler with Adam (LR 0.001, Cross Entropy, b64) notably reduced overfitting by 30% while achieving good validation results.

5.2 Learning Rate Impact (0.001 vs 0.0001 with CrossEntropyLoss)

The impact of increasing the learning rate from 0.0001 to 0.001 for Cross Entropy Loss varied by optimizer. For RMSProp (both b16 and b64), the smaller learning rate (0.0001) paradoxically led to much faster learning, higher overfitting, and smoother validation curves, achieving 48% validation accuracy in one instance. Conversely, for SGD (both b16 and b64, Fold 1), the smaller learning rate (0.0001) resulted in slower learning but also proportionally reduced overfitting. This suggests that the optimal learning rate is highly dependent on the chosen optimizer.

5.3 Batch Size Impact (16 vs 64)

Increasing batch size from 16 to 64 generally led to smoother training curves, as noted in the general observations. However, its impact on performance was mixed and optimizer-dependent. For Adam and RMSProp with LR 0.0001 and Cross Entropy, batch size 16 led to slightly quicker convergence and higher overfitting. For SGD (LR 0.0001, Cross Entropy), batch size 16 showed considerably higher overfitting and a higher validation result, while batch size 64 had more erratic validation.

The most dramatic impact was observed with Adam using KLDivLoss (LR 0.001, Fold 1!!!), where batch size 16 resulted in the model's performance plateauing at a mere 7.2% accuracy for both training and validation. Increasing the batch size to 64 solved this issue, leading to the highest observed validation accuracy of 49%. This highlights a critical interaction between batch size, loss function, and optimizer that can lead to wildly different behavior. The use of a StepLR scheduler with Adam (LR 0.001, Cross Entropy) showed benefits for both batch sizes, but particularly for b64 where it reduced overfitting significantly.

5.4 Cross-Fold Consistency

The most striking example of cross-fold inconsistency was observed with Adam using KLDivLoss (LR 0.001) and batch size 16. In Fold 1, this configuration led to a performance plateau at 7.2% accuracy. However, in Fold 2, the same configuration (Adam, LR

0.001, KLDiv, b16) did not suffer this fate and trained successfully, indicating a strong sensitivity to the specific data split in Fold 1 for this particular setup. For Adam with KLDivLoss and batch size 64, Fold 1 yielded a 49% validation accuracy, while in Fold 2, the validation result was similar, though overfitting was higher in Fold 2. This suggests that while b64 was more robust than b16 for Adam with KLDivLoss, some variance across folds, particularly in overfitting, can still be expected.

6 Conclusion

Experiments comparing batch sizes 16 and 64, alongside different optimizers, learning rates (0.001, 0.0001), and loss functions (CrossEntropy, KLDivLoss), revealed several key trends. The increase to batch size 64 generally resulted in smoother training curves, but its effect on validation accuracy and overfitting was varied and depended significantly on the optimizer and learning rate.

No single optimizer consistently outperformed others across all metrics and configurations. Adam with KLDivLoss (LR 0.001, b64, Fold 1) achieved the highest validation accuracy (49%), but also demonstrated high overfitting. The Adam optimizer with a StepLR scheduler (LR 0.001, Cross Entropy, b64) showed promise in mitigating overfitting while maintaining good performance. The choice of learning rate showed different effects depending on the optimizer: a smaller LR (0.0001) was beneficial for RMSProp but detrimental to SGD's learning speed when using CrossEntropyLoss.

The Adam KLDIVLoss configuration with LR 0.001 and batch size 16 in Fold 1 highlighted potential training pathologies, with performance catastrophically plateauing at 7.2%; this issue was resolved by increasing the batch size to 64 or by changing the data fold. This underscores the critical interplay of hyperparameters and data characteristics. Significant overfitting was a common issue across many configurations.