CSIT6000M Assignment-1

> Name: Zhang Hexiao
>
> Student Id: 20932780
>
> Email: hzhangea@connect.ust.hk

In this task, I built the neural network with `Pytorch` and visualized the results using `wandb` and `matplotlib`. In addition, I used `sklearn` and `skimage` to calculate the test metrics.

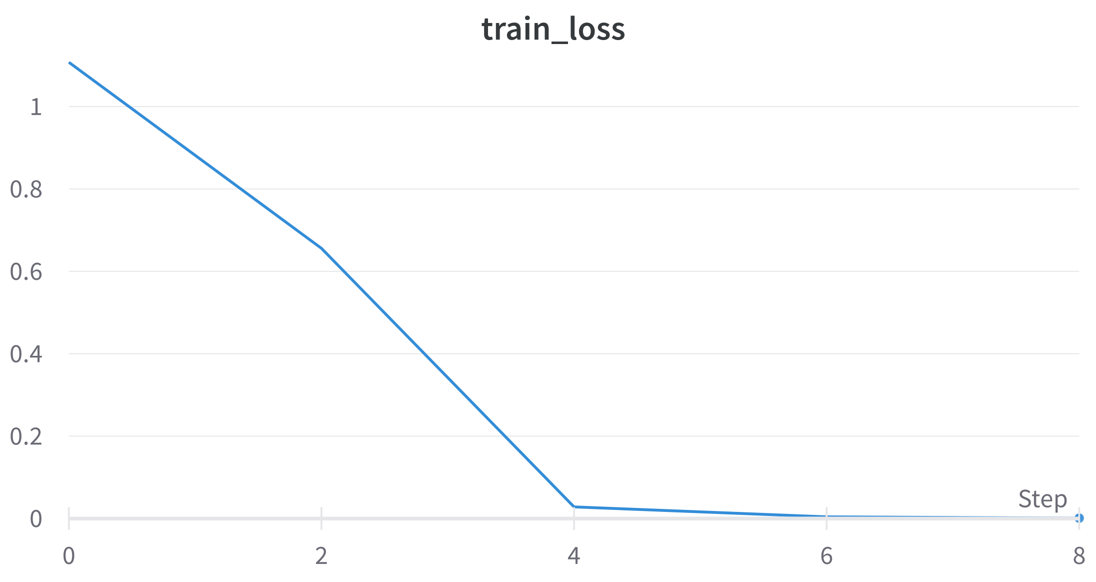
You can reproduce my results by reading the guidelines in `README.md` under the `src` folder.

Task 1

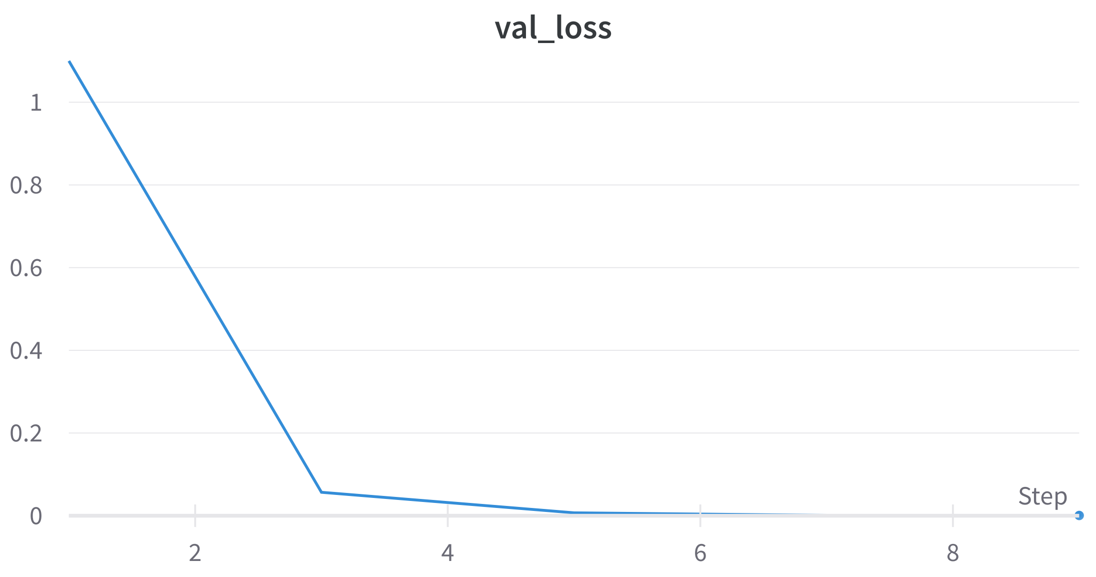
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
├─Sequential: 1-1                        [-1, 3]                   --
|    └─Conv2d: 2-1                       [-1, 16, 32, 32]          160
|    └─ReLU: 2-2                         [-1, 16, 32, 32]          --
|    └─Conv2d: 2-3                       [-1, 32, 32, 32]          4,640
|    └─ReLU: 2-4                         [-1, 32, 32, 32]          --
|    └─MaxPool2d: 2-5                    [-1, 32, 16, 16]          --
|    └─Conv2d: 2-6                       [-1, 16, 16, 16]          4,624
|    └─ReLU: 2-7                         [-1, 16, 16, 16]          --
|    └─Conv2d: 2-8                       [-1, 16, 16, 16]          2,320
|    └─ReLU: 2-9                         [-1, 16, 16, 16]          --
|    └─MaxPool2d: 2-10                   [-1, 16, 8, 8]            --
|    └─Flatten: 2-11                     [-1, 1024]                --
|    └─Linear: 2-12                      [-1, 64]                  65,600
|    └─ReLU: 2-13                        [-1, 64]                  --
|    └─Linear: 2-14                      [-1, 3]                   195
================================================================================
==========
```

Task 2

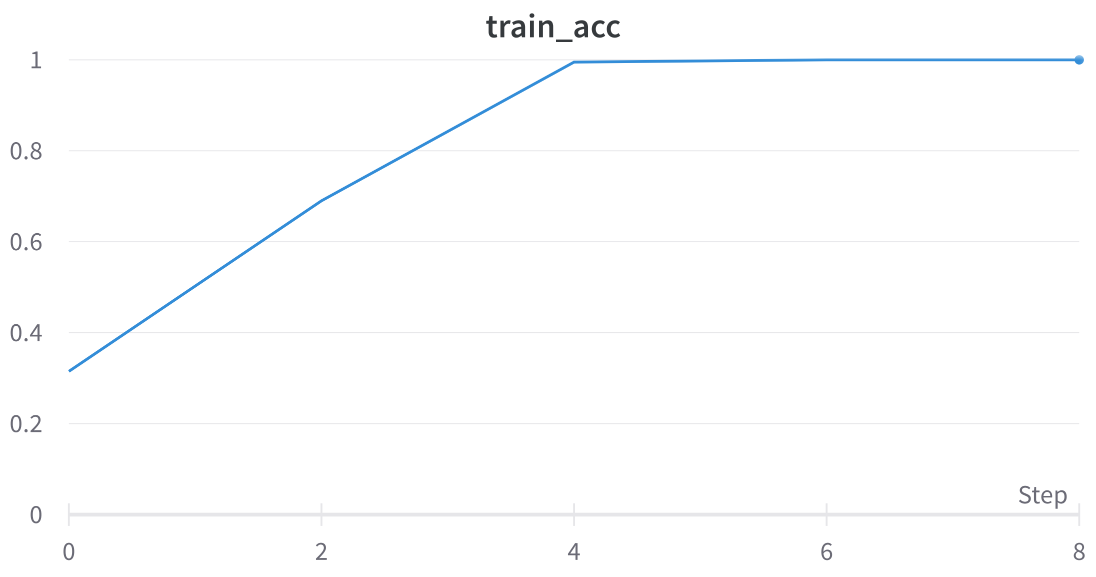
Dataset	**Loss**	**Acc**
Training	0.00088	1.0
Validation	0.00058	1.0



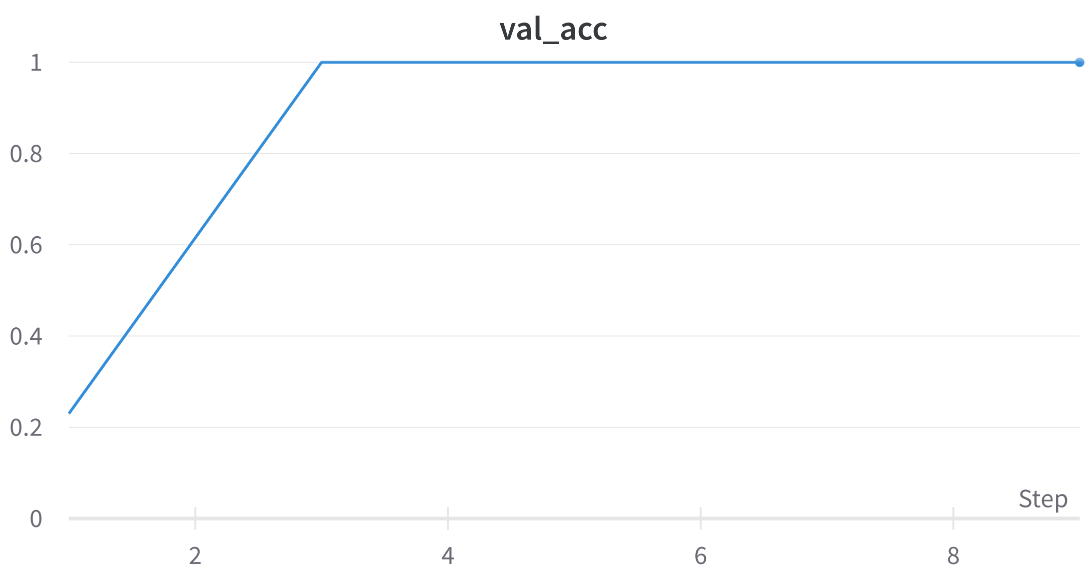
train_loss
1
0.8
0.6
0.4
0.2
0
Step
0
2
4
6
8



val_loss
1
0.8
0.6
0.4
0.2
0
Step
2
4
6
8



train_acc
1
0.8
0.6
0.4
0.2
0
Step
0
2
4
6
8



Task 3

Dataset	Loss	Acc
Test	0.40593	0.83

```
              precision    recall  f1-score   support

           1       0.96      1.00      0.98        65
           2       0.68      0.96      0.79        68
           3       1.00      0.54      0.70        67

    accuracy                           0.83       200
   macro avg       0.88      0.83      0.82       200
weighted avg       0.88      0.83      0.82       200
```

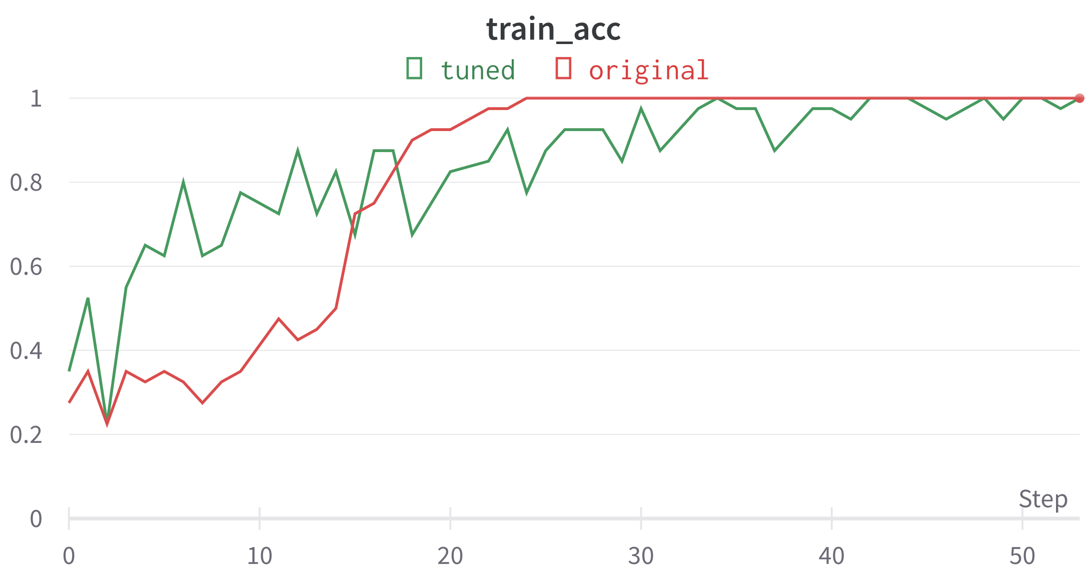
From the test results, I found that the model was accurate when there was only one ball in the figure. **When there are two or three balls, the model is prone to errors.**

In addition, the model is very accurate in training and validation, but does not perform well on the test set. Apparently, it is **overfitted**.

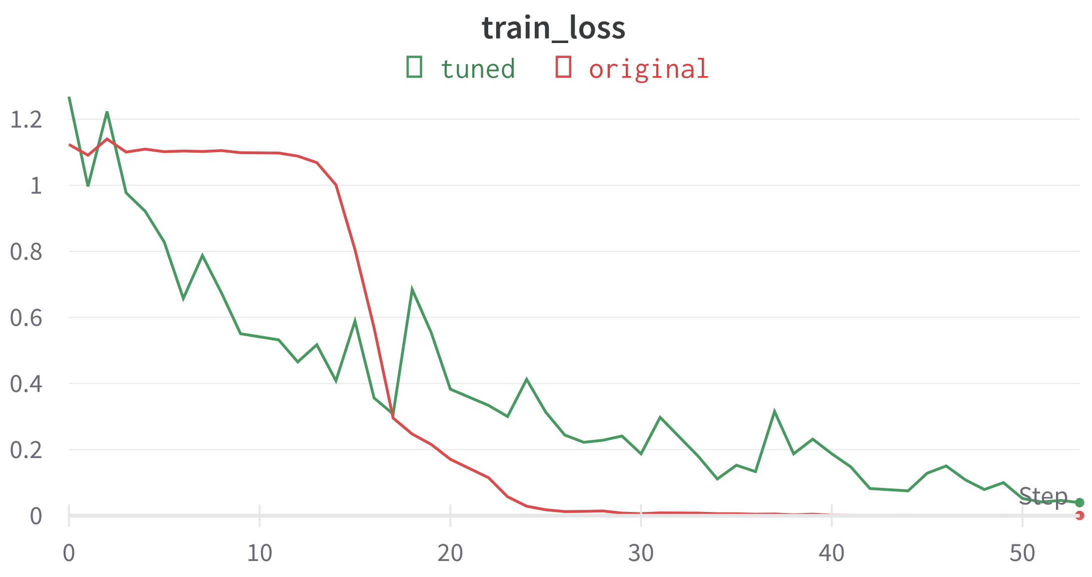
Task 4

To mitigate the overfitting problem, I added **L2 loss** with `weight_decay=0.001` and added **Batch normalization** operations behind each convolutional layer.

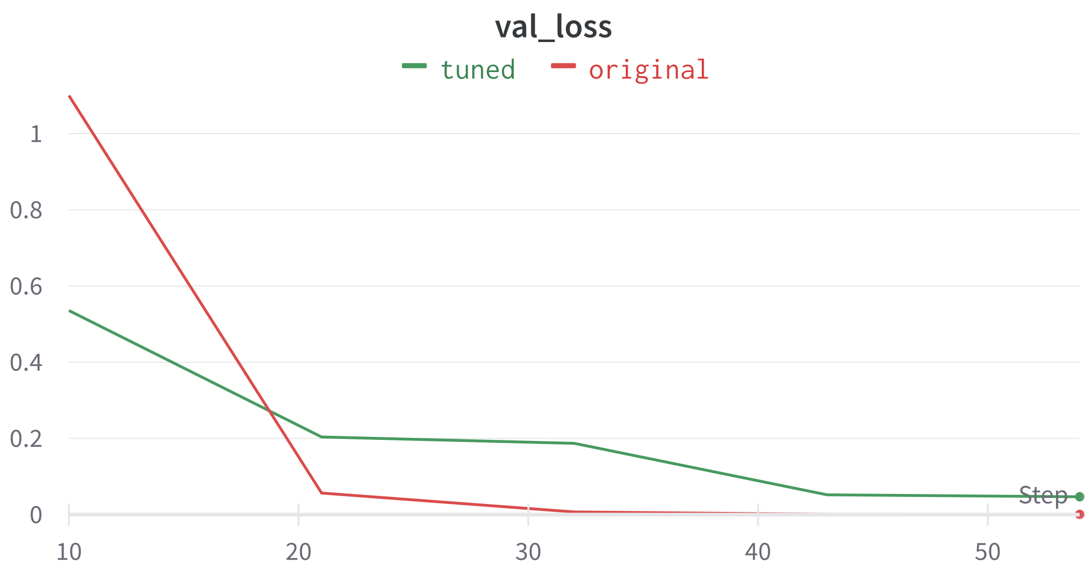
Batch normalization also helps to smooth out the loss surface of the neural network, which can make it easier to optimize the network during training. This can prevent the network from getting stuck in local minima and help it to converge to a better solution. By reducing the sensitivity of the network to the specific values of the input data, we can help prevent overfitting and improve the generalization performance of the network.



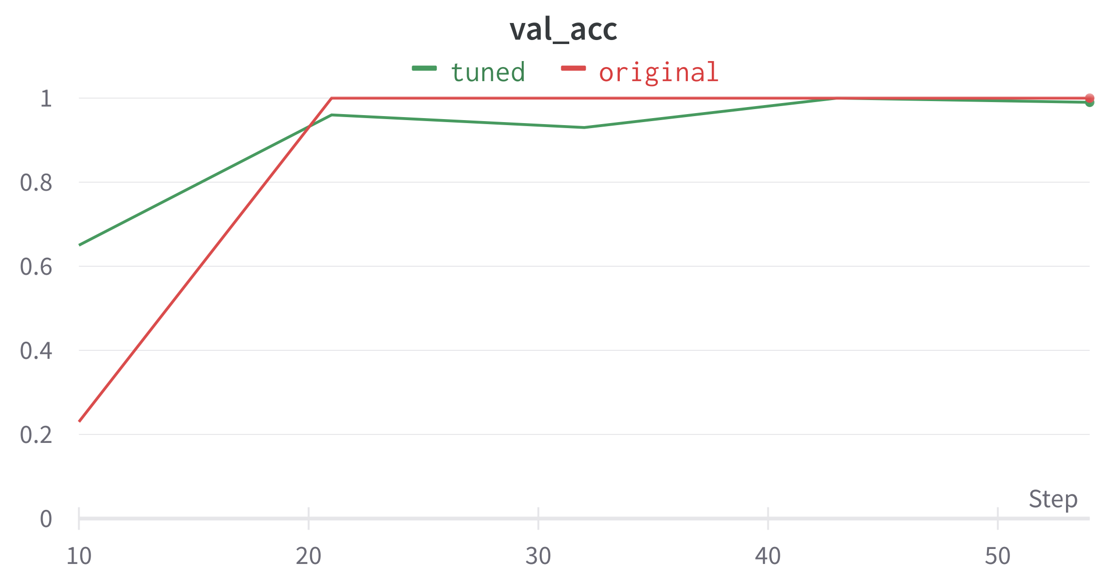
train_acc
tuned
original
1
0.8
0.6
0.4
0.2
0
Step
0
10
20
30
40
50



train_loss
tuned
original
1.2
1
0.8
0.6
0.4
0.2
0
Step
0
10
20
30
40
50



val_loss
tuned
original
1
0.8
0.6
0.4
0.2
0
Step
10
20
30
40
50



Model	Test Loss	Test Acc
Original	0.40593	0.83
Tuned	0.1101	0.97

```
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        65
           2       0.97      0.94      0.96        68
           3       1.00      0.97      0.98        67

    accuracy                           0.97       200
   macro avg       0.97      0.97      0.97       200
weighted avg       0.97      0.97      0.97       200
```

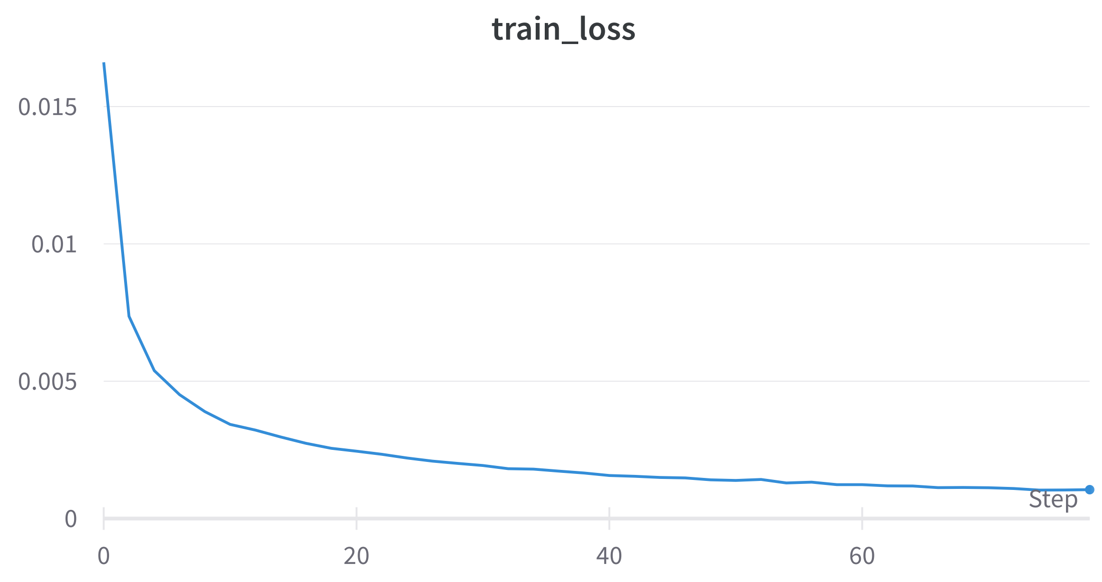
I also tried to find the optimal `weight_decay` using the hyperparameter search library `optuna`. When `weight_decay=0.000167...`, the test error is zero. However, because the data set is too small, such operations can only lead to overfitting on the test set.

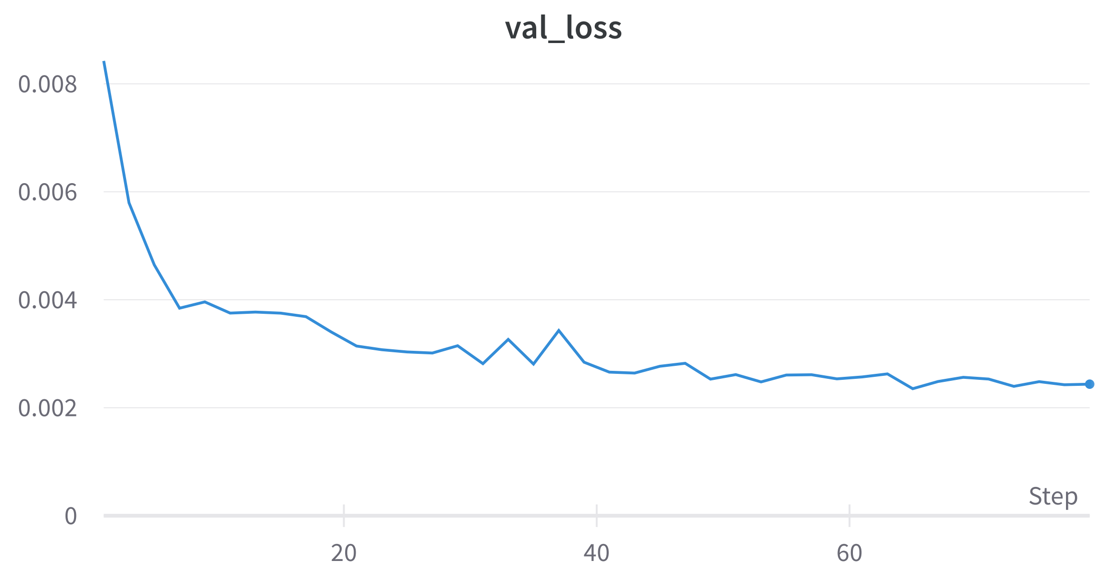
Task 5

Like ResNet, skip connections allow the gradients to flow more easily through the network during backpropagation. This can help **improve the convergence of the network and prevent vanishing gradients**.

Also, skip connections allow the network to **reuse features** learned at different levels of the hierarchy. This means that the network can use both low-level and high-level features to make its predictions.

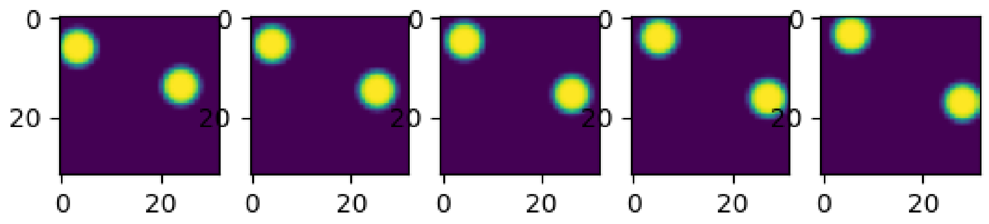
Task 6

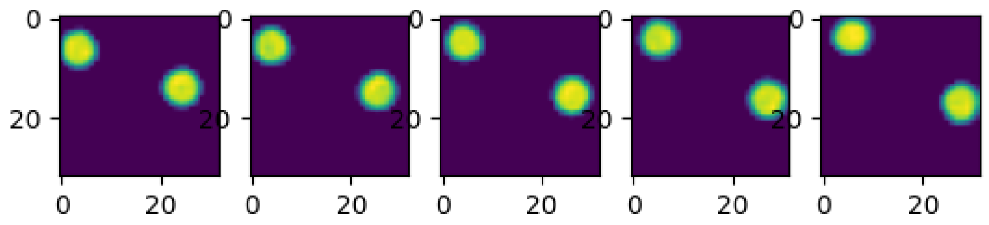




PSNR on Validation set	SSIM on Validation set
30.04602	0.97005

In the figure below, the top one is the ground truth and the bottom one is the output of the model. The model output is basically the same as the actual situation in terms of the position of balls, with a slight difference in color.

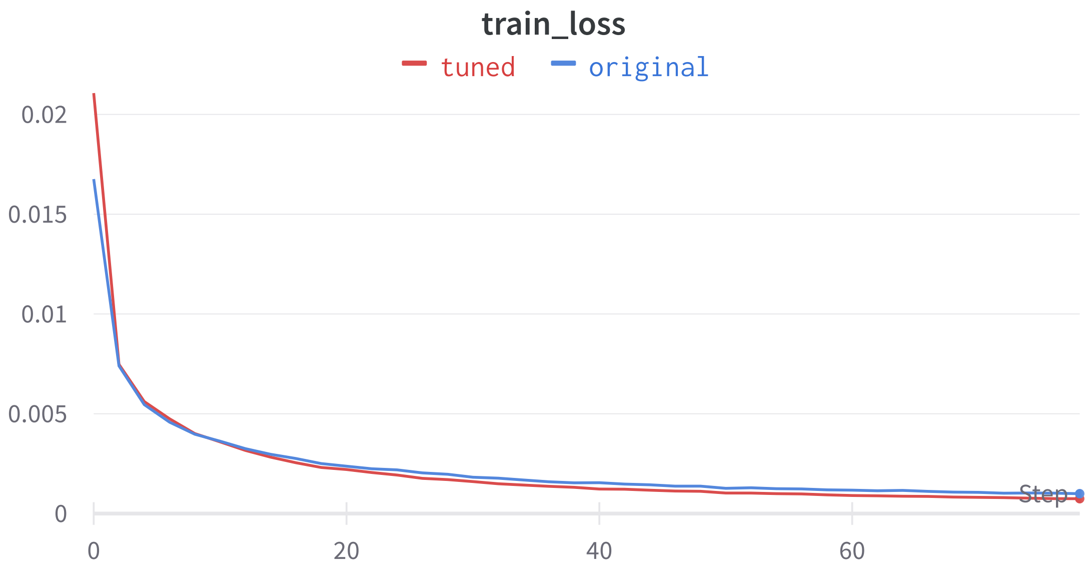


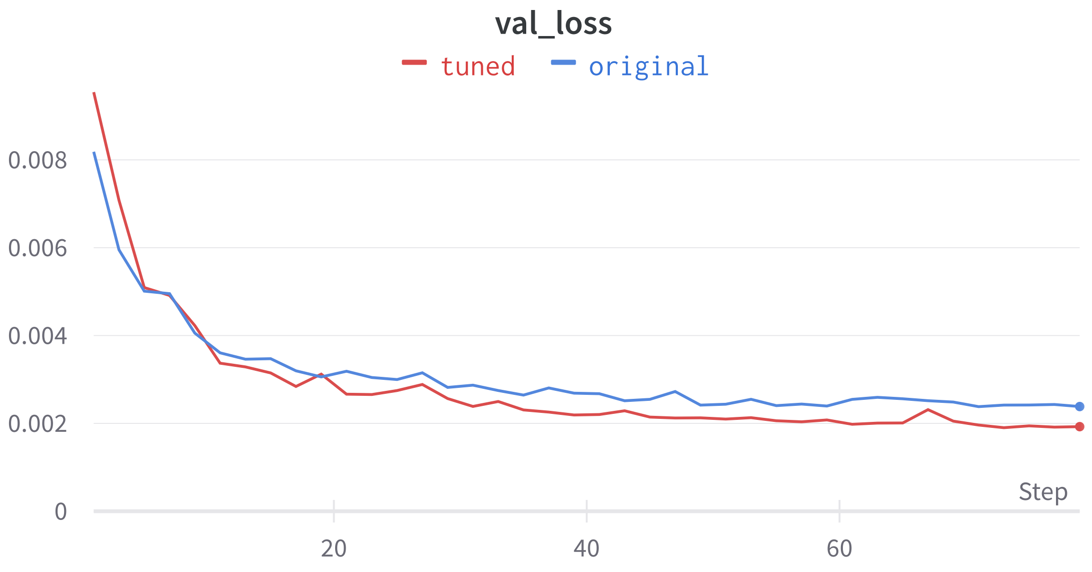


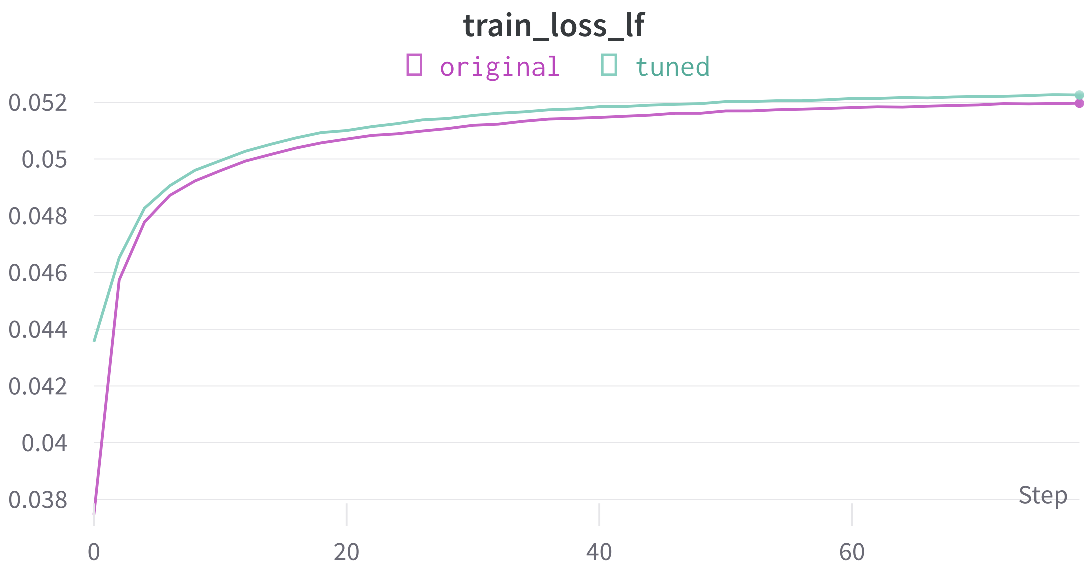
Fine-tuning the hyperparameters can slightly improve the performance of the model, and batch normalization is often effective as well. After a dozen manual grid searches, the hyperparameters I finally picked are:

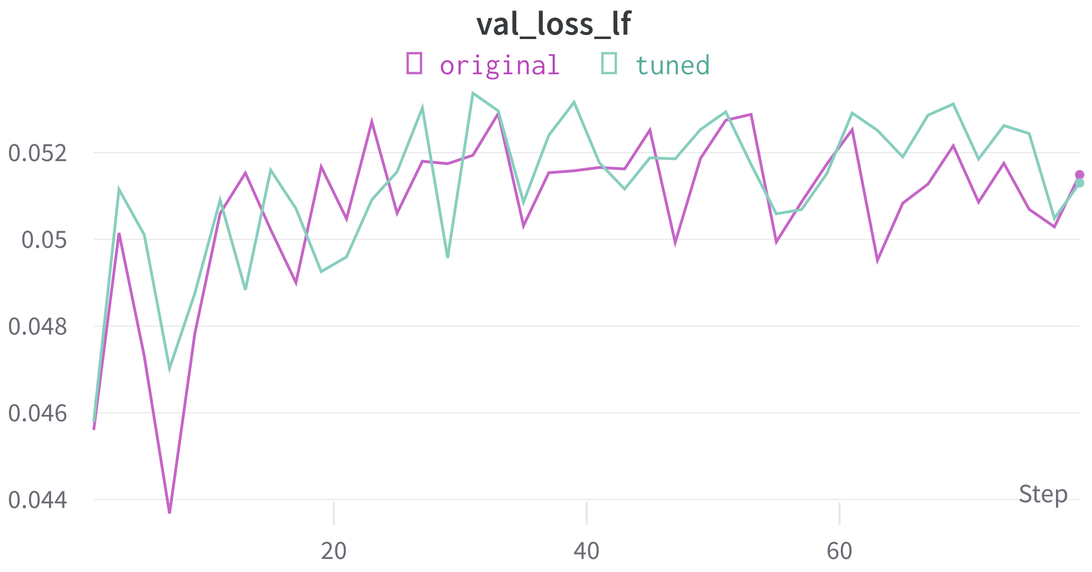
- learning rate = 0.0005
- L2 weight decay = 0

On the validation set metrics of both tasks, the new model achieved some improvements.


train_loss
tuned
original
0.02
0.015
0.01
0.005
0
Step
0
20
40
60


val_loss
tuned
original
0.008
0.006
0.004
0.002
0
Step
20
40
60


train_loss_lf
original
tuned
0.052
0.05
0.048
0.046
0.044
0.042
0.04
0.038
Step
0
20
40
60



Model	PSNR	PSNR with Last Frame	SSIM	SSIM with Last Frame
Original	30.04	13.90	0.970	0.668
Tuned	31.13	13.89	0.975	0.665

Reflection

I have always found finding optimal hyperparameters for specific tasks to be very laborious and painful, especially when the dataset is large training process cannot be done in a shorter time. There are often multiple hyperparameters to consider, and the influence between them is very confusing. Hyperparametric search tools can sometimes help us, but once the problem gets a little bit complex, they seem out of reach.