

Task 1

1.1. Data visualization

The analysis of the dataset is in the file “data_visualization.ipynb”.

- Cifar10 dataset includes 10 classes: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
- In the training set 50,000 samples in the test set – 10,000 samples. Each class contains the same number of samples. The data is balanced.
- The images are 32 x 32 RGB.
- Examples of the images of training set:

airplane



automobile



bird



cat



deer



dog



frog



horse



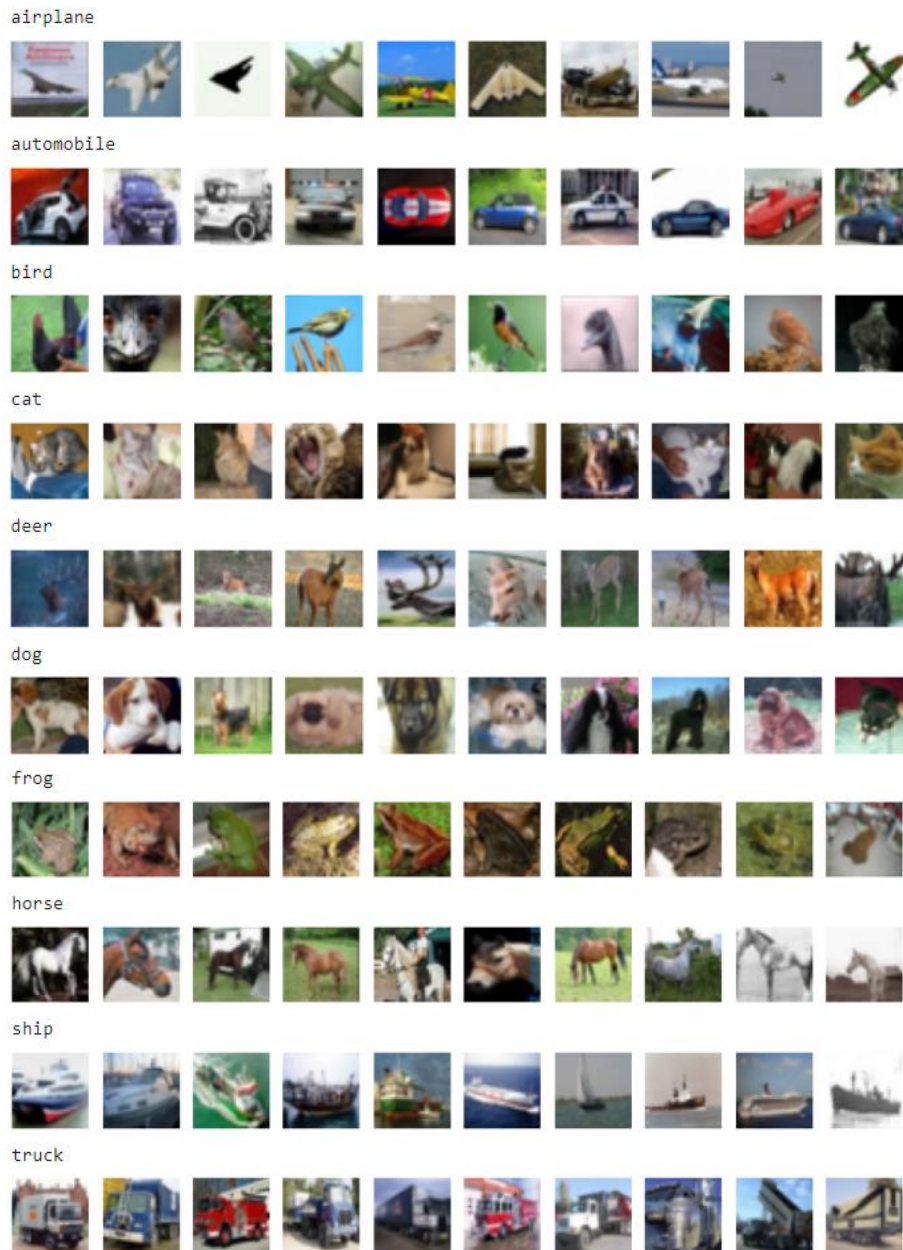
ship



truck

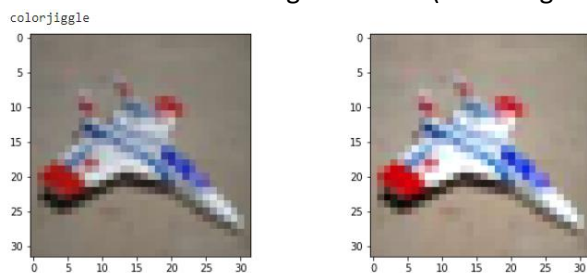


- Examples of the images of the test set:

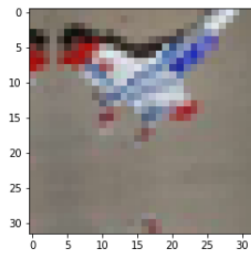
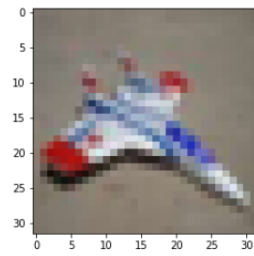


- There are different libraries for augmentations:
 - 1) torchvision => <https://pytorch.org/vision/stable/transforms.html> => used for training
 - 2) albumentations => https://albumentations.ai/docs/getting_started/image_augmentation/
 - 3) Kornia => <https://kornia.readthedocs.io/augmentation.html> => used for visualization in "data_visualization.ipynb".

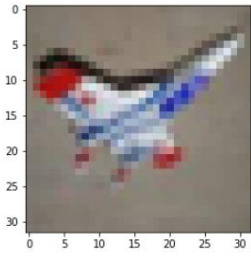
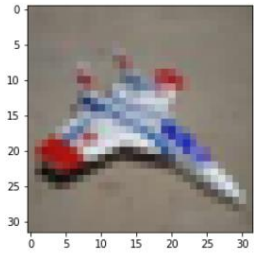
Some visualizations of Kornia augmentation (left – original image, right – image after augmentation):



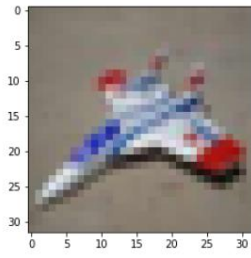
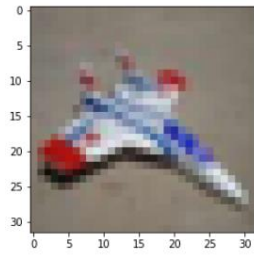
randomaffine



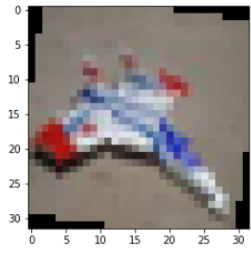
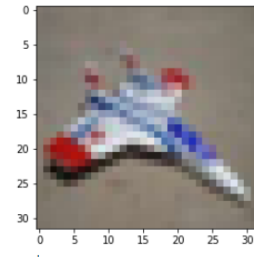
randomverticalflip



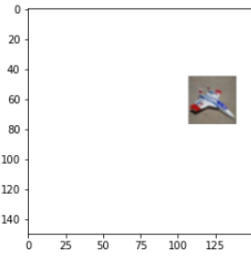
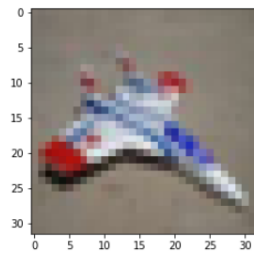
randomhorizontalflip



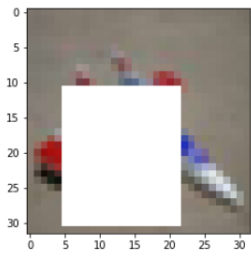
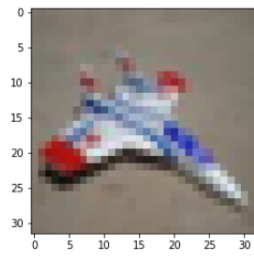
randomrotation



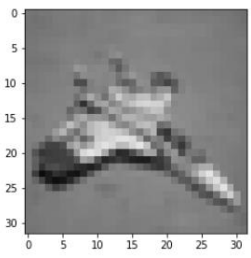
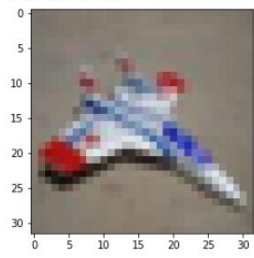
randomcrop

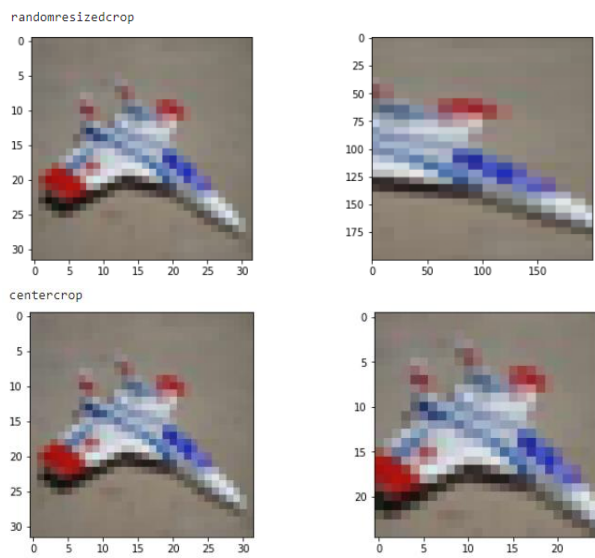


randomerasing



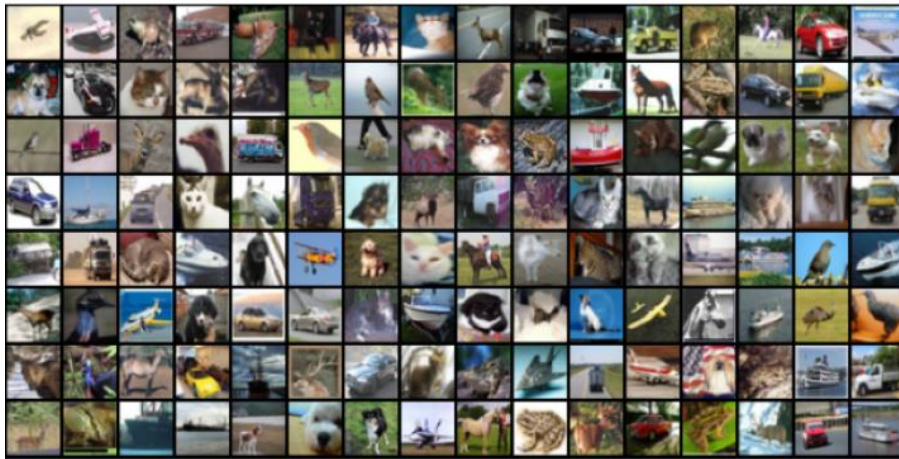
randomgrayscale



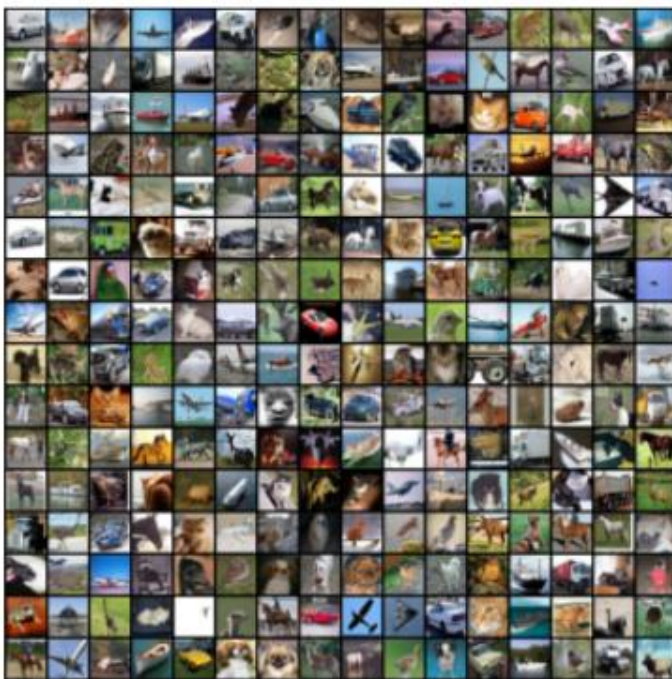


- Visualization of training/validation 10% shuffle split. Training batch = 128, validation batch = test batch = 256.

Training batch



Validation batch



Test batch



1.2. Model Training

1.2.1. Reproducibility (<https://pytorch.org/docs/stable/notes/randomness.html>)

Experiment description.

Run id	Description	Results' path
1	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 2 --optimization "Adam" --experiment_name "CNN_No_run_1" --model "CNN"	LSF_out/CNN_No_run_1_out_423841
2		LSF_out/CNN_No_run_2_out_423842
3		LSF_out/CNN_No_run_3_out_423854
4		LSF_out/CNN_No_run_4_out_423858
5		LSF_out/CNN_No_run_5_out_423861

Experiment results.

Run id	Run details per epoch
1	Epoch [0], train_loss: 1.9014, train_acc: 0.2822, val_loss: 1.5055, val_acc: 0.4414, val_precision: 0.4655, val_recall: 0.4414, val_f1: 0.4229 Epoch [1], train_loss: 1.3065, train_acc: 0.5222, val_loss: 1.1419, val_acc: 0.5794, val_precision: 0.5871, val_recall: 0.5794, val_f1: 0.5726
2	Epoch [0], train_loss: 1.9014, train_acc: 0.2822, val_loss: 1.5055, val_acc: 0.4414, val_precision: 0.4655, val_recall: 0.4414, val_f1: 0.4229 Epoch [1], train_loss: 1.3065, train_acc: 0.5222, val_loss: 1.1419, val_acc: 0.5794, val_precision: 0.5871, val_recall: 0.5794, val_f1: 0.5726
3	Epoch [0], train_loss: 1.9014, train_acc: 0.2822, val_loss: 1.5055, val_acc: 0.4414, val_precision: 0.4655, val_recall: 0.4414, val_f1: 0.4229 Epoch [1], train_loss: 1.3065, train_acc: 0.5222, val_loss: 1.1419, val_acc: 0.5794, val_precision: 0.5871, val_recall: 0.5794, val_f1: 0.5726
4	Epoch [0], train_loss: 1.9014, train_acc: 0.2822, val_loss: 1.5055, val_acc: 0.4414, val_precision: 0.4655, val_recall: 0.4414, val_f1: 0.4229 Epoch [1], train_loss: 1.3065, train_acc: 0.5222, val_loss: 1.1419, val_acc: 0.5794, val_precision: 0.5871, val_recall: 0.5794, val_f1: 0.5726
5	Epoch [0], train_loss: 1.9014, train_acc: 0.2822, val_loss: 1.5055, val_acc: 0.4414, val_precision: 0.4655, val_recall: 0.4414, val_f1: 0.4229 Epoch [1], train_loss: 1.3065, train_acc: 0.5222, val_loss: 1.1419, val_acc: 0.5794, val_precision: 0.5871, val_recall: 0.5794, val_f1: 0.5726

Conclusion:

- There is no randomness in training => 100% reproducibility.

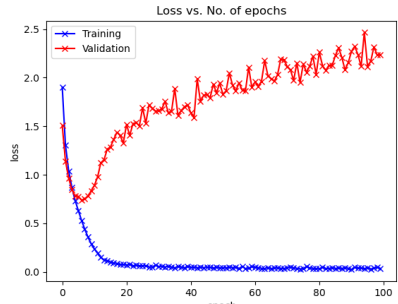
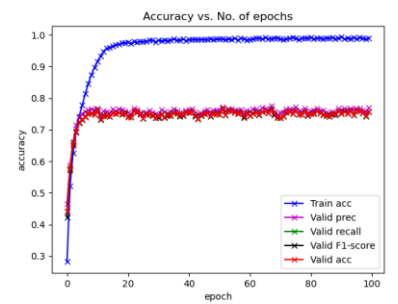
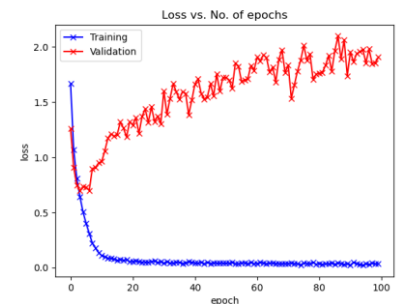
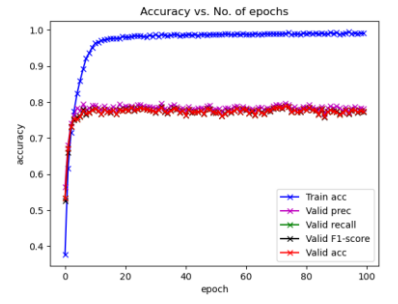
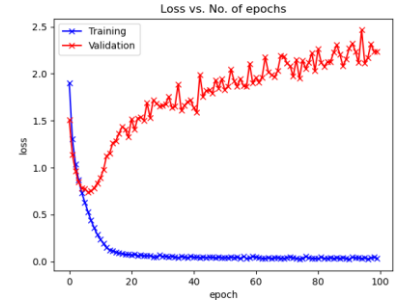
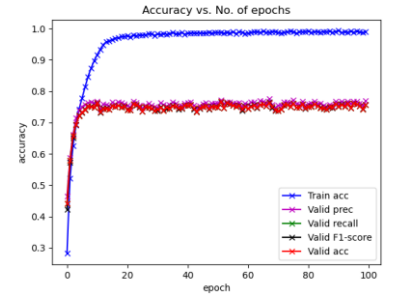
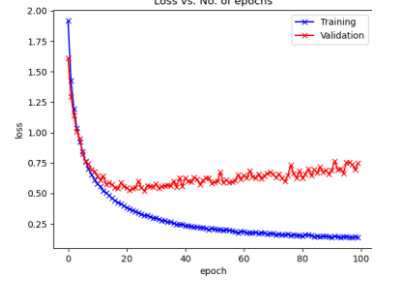
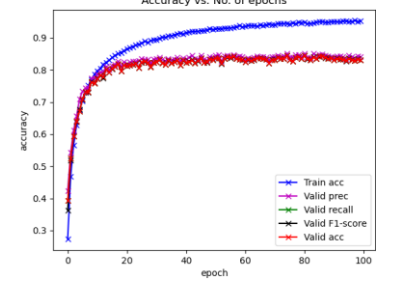
1.2.2. Preprocessing => normalization/augmentation

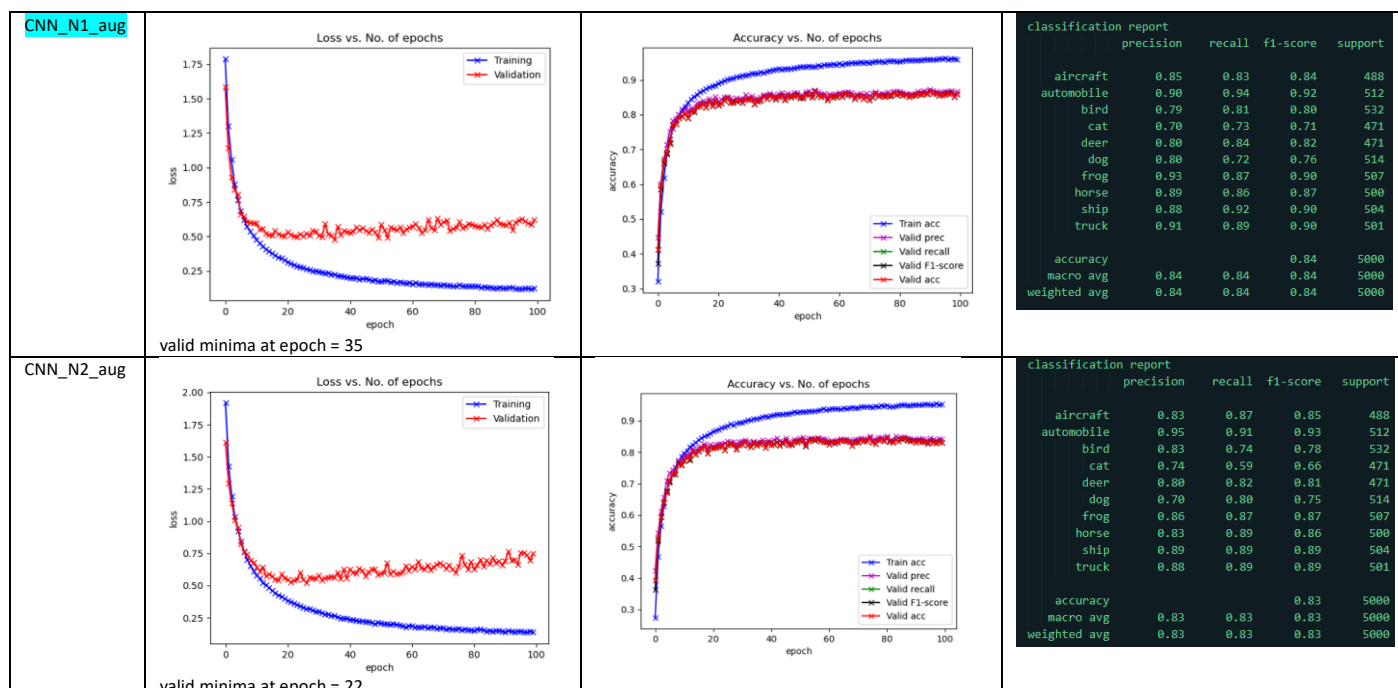
Experiment description.

Test name	Normalization	Augmentation
CNN_No	No	No
CNN_N1	mean & stdev of the training dataset	No
CNN_N2	mean = 0, stdev = 1	No
CNN_N1_aug	mean & stdev of the training dataset	RandomHorizontalFlip (p=0.5); RandomCrop(32, padding=4)
CNN_N2_aug	mean = 0, stdev = 1	RandomHorizontalFlip (p=0.5); RandomCrop(32, padding=4)
CNN_aug	No	RandomHorizontalFlip (p=0.5); RandomCrop(32, padding=4)

Test name	Description	Results' path
CNN_No	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_No" --model "CNN"	results/CNN_No_2024-02-28_08-21-21 LSF/CNN_No_out_423862
CNN_N1	python cifar_10_train_rev_2.py --normalization "N1" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_N1" --model "CNN"	results/CNN_N1_2024-02-28_08-21-30 LSF/CNN_N1_out_423872
CNN_N2	python cifar_10_train_rev_2.py --normalization "N2" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_N2" --model "CNN"	results/CNN_N2_2024-02-28_08-21-57 LSF/CNN_N2_out_423873
CNN_N1_aug	python cifar_10_train_rev_2.py --normalization "N1_aug" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_N1_aug" --model "CNN"	results/CNN_N1_aug_2024-02-28_08-22-25 LSF/CNN_N1_aug_out_423875
CNN_N2_aug	python cifar_10_train_rev_2.py --normalization "N2_aug" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_N2_aug" --model "CNN"	results/CNN_N2_aug_2024-02-28_08-22-28 LSF/CNN_N2_aug_out_423886
CNN_aug	python cifar_10_train_rev_2.py --normalization "aug" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_aug" --model "CNN"	results/CNN_aug_2024-02-28_11-26-46 LSF/CNN_aug_out_442178

Experiment results.

Test name	Loss/accuracy trend chart		Validation set (best model*)
CNN_No	 <p>valid minima at epoch = 6</p>		<pre> classification report precision recall f1-score support aircraft 0.73 0.83 0.78 488 automobile 0.88 0.89 0.88 512 bird 0.62 0.72 0.66 532 cat 0.52 0.54 0.53 471 deer 0.61 0.72 0.66 471 dog 0.68 0.66 0.67 514 frog 0.87 0.61 0.72 507 horse 0.83 0.78 0.80 500 ship 0.89 0.83 0.86 504 truck 0.86 0.82 0.84 501 accuracy 0.74 5000 macro avg 0.75 0.74 0.74 5000 weighted avg 0.75 0.74 0.74 5000 </pre>
CNN_N1	 <p>valid minima at epoch = 6</p>		<pre> classification report precision recall f1-score support aircraft 0.80 0.80 0.80 488 automobile 0.87 0.90 0.89 512 bird 0.78 0.53 0.63 532 cat 0.61 0.47 0.53 471 deer 0.61 0.81 0.69 471 dog 0.65 0.70 0.67 514 frog 0.79 0.75 0.77 507 horse 0.78 0.82 0.80 500 ship 0.82 0.89 0.86 504 truck 0.82 0.86 0.84 501 accuracy 0.75 5000 macro avg 0.75 0.75 0.75 5000 weighted avg 0.76 0.75 0.75 5000 </pre>
CNN_N2	 <p>valid minima at epoch = 6</p>		<pre> classification report precision recall f1-score support aircraft 0.73 0.83 0.78 488 automobile 0.88 0.89 0.88 512 bird 0.62 0.72 0.66 532 cat 0.52 0.54 0.53 471 deer 0.61 0.72 0.66 471 dog 0.68 0.66 0.67 514 frog 0.87 0.61 0.72 507 horse 0.83 0.78 0.80 500 ship 0.89 0.83 0.86 504 truck 0.86 0.82 0.84 501 accuracy 0.74 5000 macro avg 0.75 0.74 0.74 5000 weighted avg 0.75 0.74 0.74 5000 </pre>
CNN_aug	 <p>valid minima at epoch = 26</p>		<pre> classification report precision recall f1-score support aircraft 0.83 0.87 0.85 488 automobile 0.95 0.91 0.93 512 bird 0.83 0.74 0.78 532 cat 0.74 0.59 0.66 471 deer 0.80 0.82 0.81 471 dog 0.70 0.80 0.75 514 frog 0.86 0.87 0.87 507 horse 0.83 0.89 0.86 500 ship 0.89 0.89 0.89 504 truck 0.88 0.89 0.89 501 accuracy 0.83 5000 macro avg 0.83 0.83 0.83 5000 weighted avg 0.83 0.83 0.83 5000 </pre>



*Best model – is the model which saved at minimal validation loss.
Last model – is the model which was saved at the end of the training.

Conclusion:

- Using 100 epochs without any preprocessing leading to overfitting (validation loss decreases and after increases, showed minima).
- We are observing stable validation accuracy, recall, precision, and F1-score values even after the validation loss starts to increase (overfitting region), it could indicate that the model has reached a stable point in terms of its generalization ability. At this point the model was saved as “best model”. Best from a generalization point of view.

Results' path	LSF/CNN_No_out_423862	
	Test set	
	Best model	Last model
loss	0.7475	2.3915
accuracy	0.7432	0.7471
precision	0.7603	0.7596
recall	0.7432	0.7471
F1	0.7450	0.7470

- N1 normalization & N2 normalization don't solve the problem of overfitting. Theoretically, normalization can be efficient in speed of convergence (scale the input features to a similar range). But for this experiment I didn't see improvement in convergence speed: without augmentation the minima is reached within 6 epoch (for both with and without normalization), using augmentation – only N2 normalization increased speed of convergence from 26 to 22 epochs. For N1 normalization, the speed decreased from 22 epoch to 35 epochs.
- The simple augmentation, like here – random horizontal flip and random crop, almost removes the overfitting.
- Looking at precision/recall/F1/accuracy values for CNN_N1_aug and CNN_N2_aug => we can conclude that N1 normalization is a little bit better, but the difference is insignificant and N2 can be used to save the resources for calculation of mean and stdev for specific dataset.
- The optimal => N1 + augmentation

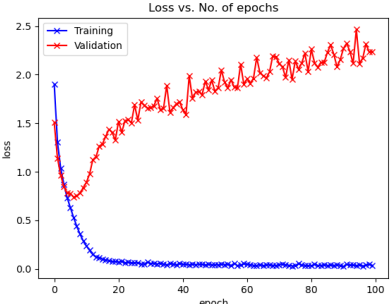
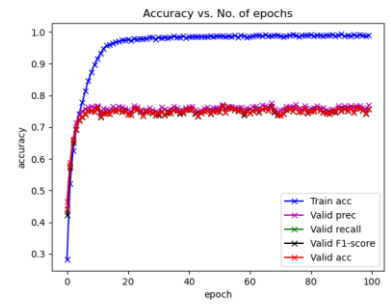
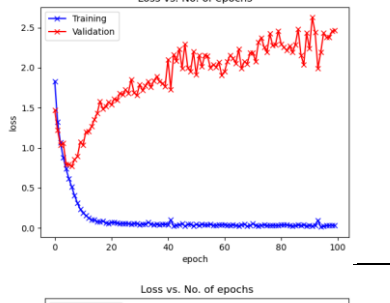
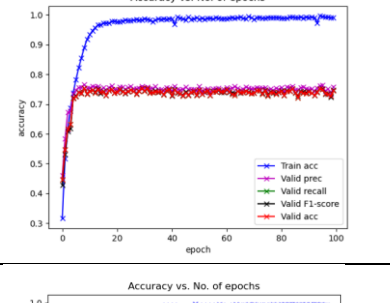
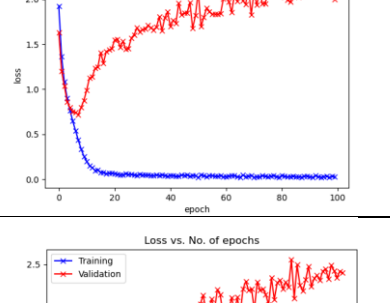
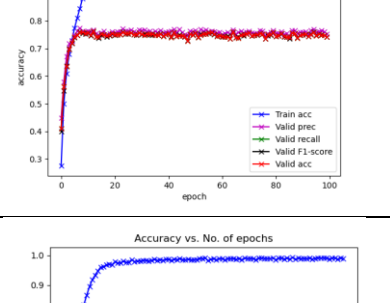
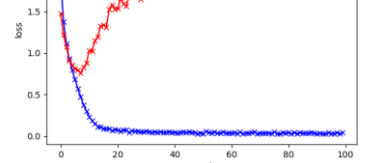
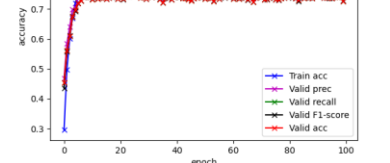
1.2.3. Validation size

Experiment description.

Validation set size *, %	Description	Results' path
10	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_No" --model "CNN"	results/CNN_No_2024-02-28_08-21-21 LSF/CNN_No_out_423862
15	python cifar_10_train_rev_2.py --normalization "No" --val_size 15 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_val_15" --model "CNN"	results/CNN_val_15_2024-02-28_08-23-02 LSF/CNN_val_15_out_423887
20	python cifar_10_train_rev_2.py --normalization "No" --val_size 20 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_val_20" --model "CNN"	results/CNN_val_20_2024-02-28_08-23-08 LSF/CNN_val_20_out_423888
25	python cifar_10_train_rev_2.py --normalization "No" --val_size 25 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_val_25" --model "CNN"	results/CNN_val_25_2024-02-28_08-23-35 LSF/CNN_val_25_out_423889

* Percent of the training dataset

Experiment results.

Validation set size, %	Loss/accuracy trend chart		Validation set (best model*)
10			<pre> classification report precision recall f1-score support aircraft 0.73 0.83 0.78 488 automobile 0.88 0.89 0.88 512 bird 0.62 0.72 0.66 532 cat 0.52 0.54 0.53 471 deer 0.61 0.72 0.66 471 dog 0.68 0.66 0.67 514 frog 0.87 0.61 0.72 587 horse 0.83 0.78 0.80 580 ship 0.89 0.83 0.86 584 truck 0.86 0.82 0.84 581 accuracy 0.74 5000 macro avg 0.75 5000 weighted avg 0.75 5000 </pre>
15			<pre> classification report precision recall f1-score support aircraft 0.80 0.76 0.78 732 automobile 0.91 0.85 0.88 761 bird 0.66 0.61 0.64 793 cat 0.53 0.56 0.54 697 deer 0.58 0.81 0.68 730 dog 0.74 0.54 0.62 759 frog 0.78 0.79 0.79 749 horse 0.76 0.80 0.78 779 ship 0.90 0.84 0.87 752 truck 0.82 0.85 0.84 748 accuracy 0.74 7500 macro avg 0.75 7500 weighted avg 0.75 7500 </pre>
20			<pre> classification report precision recall f1-score support aircraft 0.84 0.76 0.79 973 automobile 0.93 0.82 0.87 1010 bird 0.72 0.63 0.67 1006 cat 0.51 0.64 0.57 967 deer 0.65 0.75 0.70 963 dog 0.70 0.61 0.66 1024 frog 0.85 0.78 0.81 1025 horse 0.83 0.79 0.81 1035 ship 0.83 0.90 0.86 998 truck 0.81 0.88 0.84 999 accuracy 0.76 10000 macro avg 0.77 10000 weighted avg 0.77 10000 </pre>
25			<pre> classification report precision recall f1-score support aircraft 0.72 0.79 0.75 1287 automobile 0.94 0.79 0.86 1266 bird 0.68 0.64 0.66 1289 cat 0.55 0.58 0.56 1215 deer 0.67 0.71 0.69 1198 dog 0.67 0.58 0.62 1253 frog 0.81 0.81 0.81 1272 horse 0.76 0.80 0.78 1291 ship 0.89 0.80 0.84 1251 truck 0.76 0.91 0.83 1258 accuracy 0.74 12500 macro avg 0.74 12500 weighted avg 0.75 12500 </pre>

Conclusion:

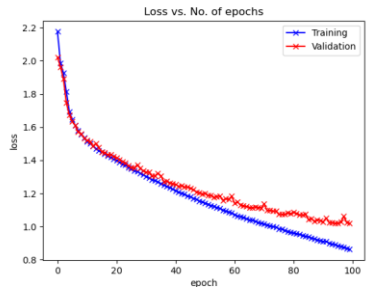
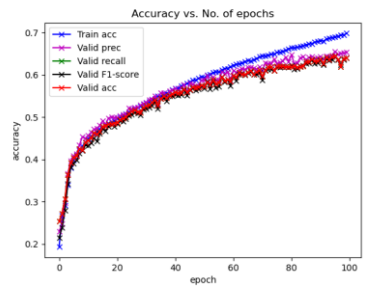
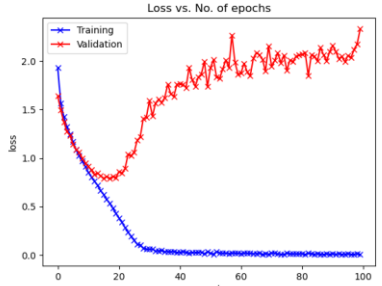
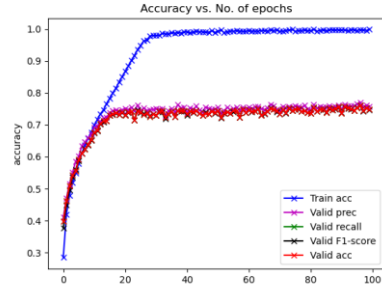
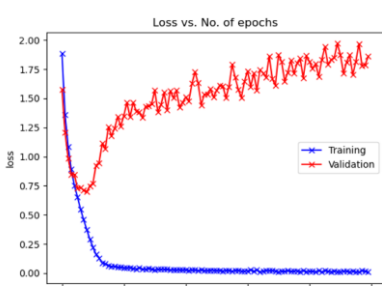
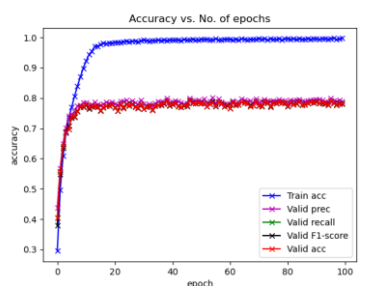
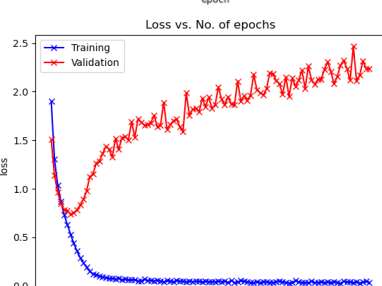
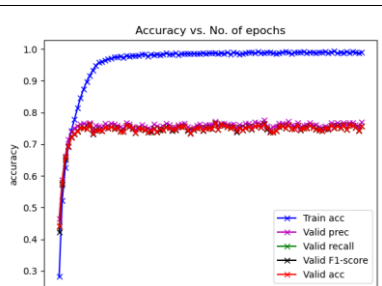
- Validation set is not used in the model training. But the high validation set can decrease the number of sample points in the training set and the training can be poor. For this experiment validation set size doesn't impact the training because we have 50,000 samples (relatively large training).
- No impact => validation size = 10% (of the total training set) can be taken for optimal model.

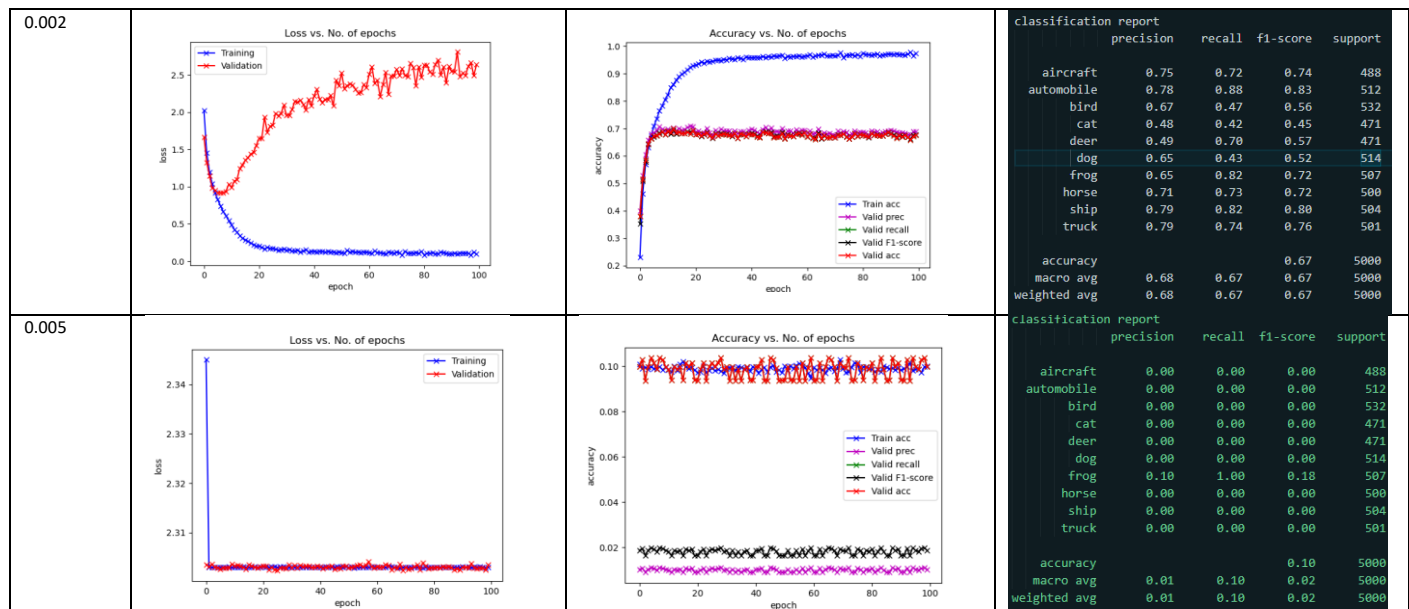
1.2.4. Learning rate

Experiment description.

Learning rate	Description	Results' path
0.00001	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.00001 --epochs 100 --optimization "Adam" --experiment_name "CNN_LR_00001" --model "CNN"	results/CNN_LR_00001_2024-02-28_10-14-34 LSF/CNN_LR_00001_out_432580
0.0001	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.0001 --epochs 100 --optimization "Adam" --experiment_name "CNN_LR_0001" --model "CNN"	results/CNN_LR_0001_2024-02-28_08-26-47 LSF/CNN_LR_0001_out_423913
0.0005	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.0005 --epochs 100 --optimization "Adam" --experiment_name "CNN_LR_0005" --model "CNN"	results/CNN_LR_0005_2024-02-28_08-26-22 LSF/CNN_LR_0005_out_423912
0.001	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_No" --model "CNN"	results/CNN_No_2024-02-28_08-21-21 LSF/CNN_No_out_423862
0.002	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.002 --epochs 100 --optimization "Adam" --experiment_name "CNN_LR_0.002" --model "CNN"	results/CNN_LR_0.002_2024-02-28_10-15-52 LSF/CNN_LR_0.002_out_432596
0.005	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.005 --epochs 100 --optimization "Adam" --experiment_name "CNN_LR_005" --model "CNN"	results/CNN_LR_005_2024-02-28_08-23-54 LSF/CNN_LR_005_out_423900

Experiment results.

Learning rate	Loss/accuracy trend chart		Validation set (best model*)
0.00001			<pre>classification report precision recall f1-score support aircraft 0.70 0.69 0.70 488 automobile 0.74 0.84 0.79 512 bird 0.51 0.62 0.56 532 cat 0.41 0.45 0.43 471 deer 0.61 0.46 0.52 471 dog 0.57 0.55 0.56 514 frog 0.78 0.63 0.70 507 horse 0.68 0.71 0.69 500 ship 0.78 0.73 0.75 504 truck 0.67 0.71 0.69 501 accuracy 0.64 5000 macro avg 0.65 0.64 0.64 5000 weighted avg 0.65 0.64 0.64 5000</pre>
0.0001			<pre>classification report precision recall f1-score support aircraft 0.73 0.82 0.77 488 automobile 0.89 0.83 0.86 512 bird 0.69 0.64 0.66 532 cat 0.56 0.53 0.54 471 deer 0.60 0.75 0.67 471 dog 0.73 0.55 0.63 514 frog 0.81 0.76 0.79 507 horse 0.74 0.79 0.76 500 ship 0.80 0.86 0.83 504 truck 0.83 0.82 0.83 501 accuracy 0.74 5000 macro avg 0.74 0.74 0.73 5000 weighted avg 0.74 0.74 0.74 5000</pre>
0.0005			<pre>classification report precision recall f1-score support aircraft 0.79 0.84 0.81 488 automobile 0.87 0.92 0.89 512 bird 0.69 0.70 0.69 532 cat 0.56 0.58 0.57 471 deer 0.71 0.67 0.69 471 dog 0.70 0.61 0.65 514 frog 0.81 0.82 0.81 507 horse 0.83 0.81 0.82 500 ship 0.89 0.87 0.88 504 truck 0.83 0.88 0.85 501 accuracy 0.77 5000 macro avg 0.77 0.77 0.77 5000 weighted avg 0.77 0.77 0.77 5000</pre>
0.001			<pre>classification report precision recall f1-score support aircraft 0.73 0.83 0.78 488 automobile 0.88 0.89 0.88 512 bird 0.62 0.72 0.66 532 cat 0.52 0.54 0.53 471 deer 0.61 0.72 0.66 471 dog 0.68 0.66 0.67 514 frog 0.87 0.61 0.72 507 horse 0.83 0.78 0.80 500 ship 0.89 0.83 0.86 504 truck 0.86 0.82 0.84 501 accuracy 0.74 5000 macro avg 0.75 0.74 0.74 5000 weighted avg 0.75 0.74 0.74 5000</pre>



Conclusion:

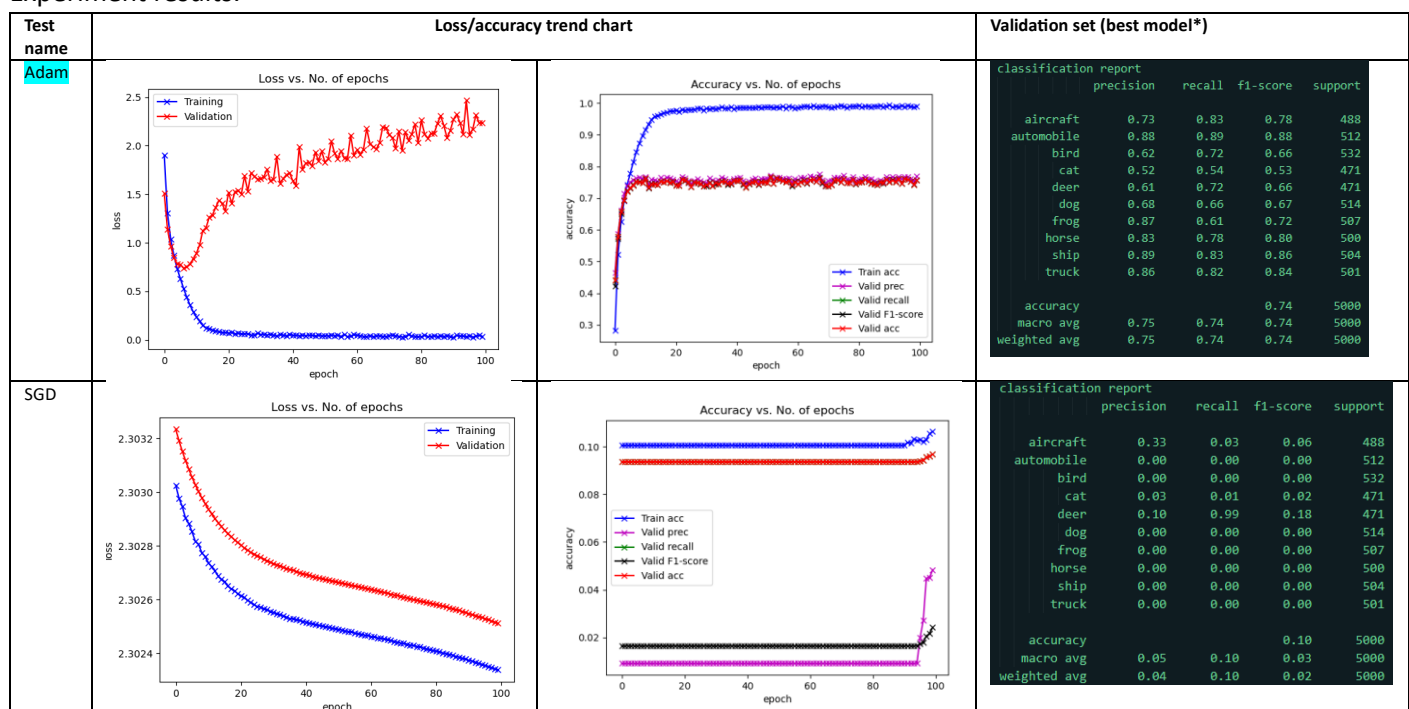
- Low LR => slow training.
- High LR => divergence (divergence occurs when the parameters of the model oscillate or diverge to infinity rather than converging to optimal values.)
- The optimal => LR = 0.001

1.2.5. Optimizer

Experiment description.

Optimizer	Description	Results' path
Adam	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_No" --model "CNN"	results/CNN_No_2024-02-28_08-21-21 LSF/CNN_No_out_423862
SGD	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "SGD" --experiment_name "CNN_SGD" --model "CNN"	results/CNN_SGD_2024-02-28_08-27-01 LSF/CNN_SGD_out_423914

Experiment results.



Conclusion:

- SGD showed poor performance than Adam.
- The optimal => Adam.

1.2.6. Model

Experiment description.

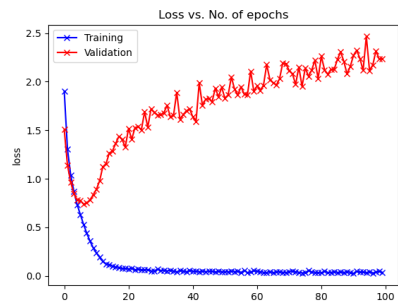
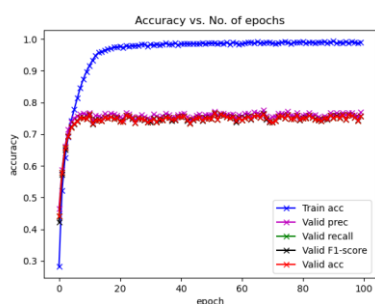
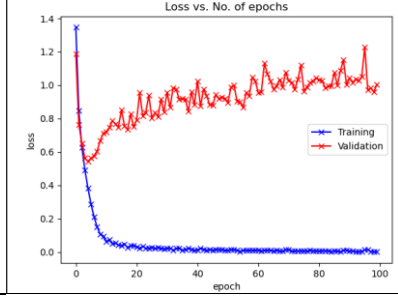
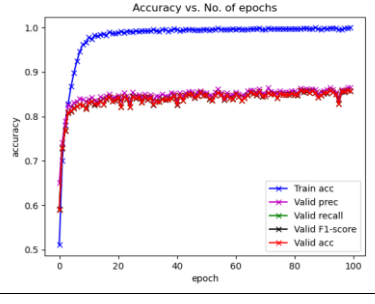
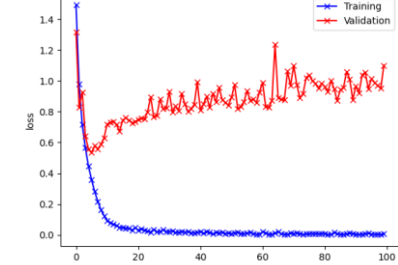
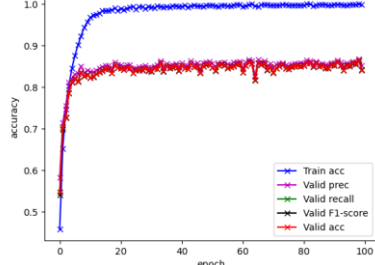
Test name	Description	Results' path
CNN ⁽¹⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_No" --model "CNN"	results/CNN_No_2024-02-28_08-21-21 LSF/CNN_No_out_423862
ResNet_18 ⁽²⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "ResNet_18" --model "ResNet_18"	results/ResNet_18_2024-02-28_08-27-56 LSF/ResNet_18_out_423926
ResNet_34 ⁽²⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "ResNet_34" --model "ResNet_34"	results/ResNet_34_2024-02-28_08-28-44 LSF/ResNet_34_out_423927
ViT ⁽³⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "ViT" --model "ViT"	results/ViT_2024-02-28_08-28-40 LSF/ViT_out_423930
ViT_small ⁽³⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "ViT_small" --model "ViT_small"	results/ViT_small_2024-02-28_08-28-54 LSF/ViT_small_out_423935
ViT_tiny ⁽³⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "ViT_tiny" --model "ViT_tiny"	results/ViT_tiny_2024-02-28_08-28-57 LSF/ViT_tiny_out_423939
ViT_simple ⁽³⁾	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "ViT_simple_correct" --model "ViT_simple"	results/ViT_simple_correct_2024-02-28_09-38-05 LSF/ViT_simple_correct_out_429077

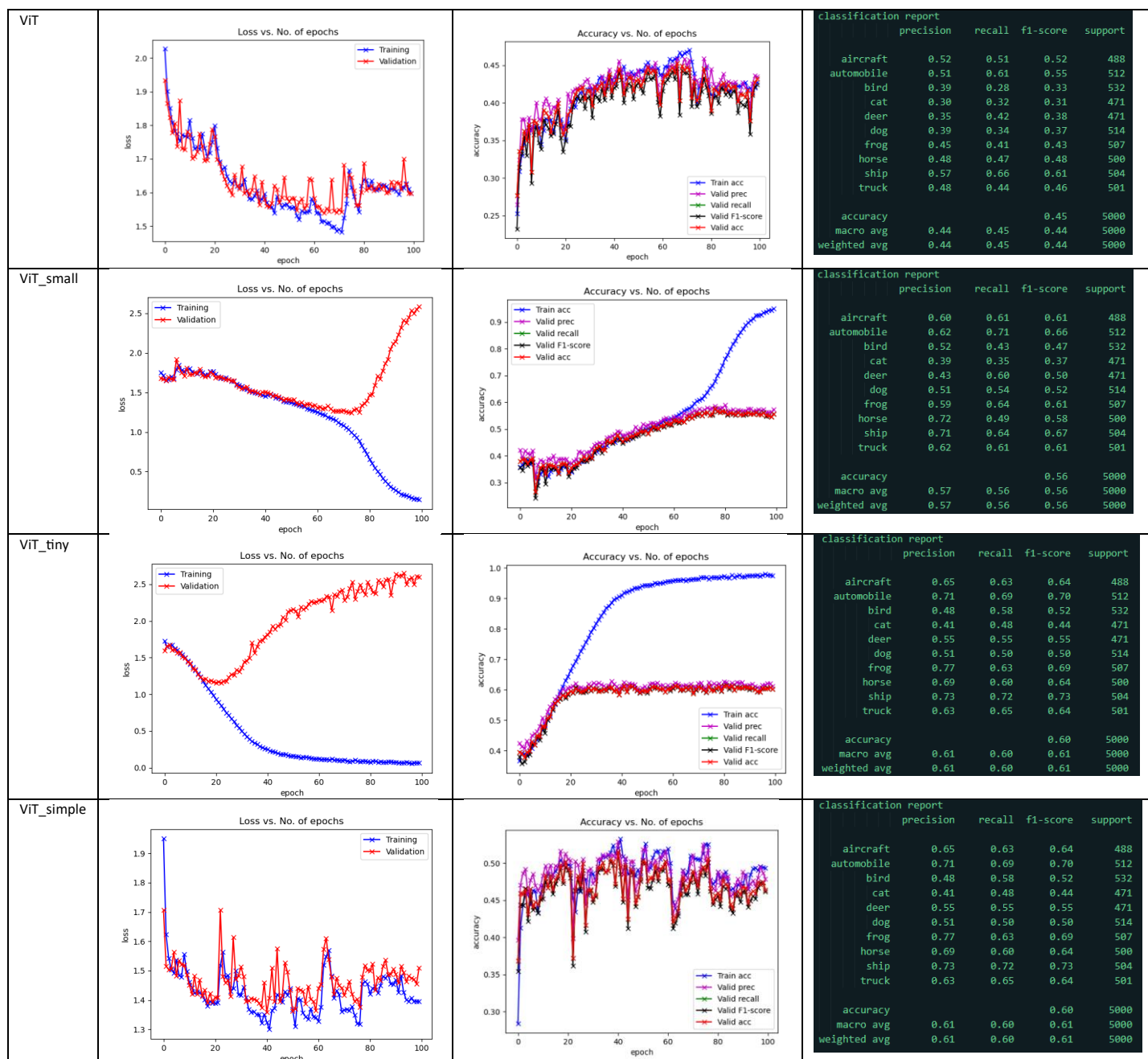
⁽¹⁾ <https://www.kaggle.com/code/shadabhussain/cifar-10-cnn-using-pytorch>

⁽²⁾ <https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py>

⁽³⁾ <https://github.com/kentaroy47/vision-transformers-cifar10/tree/main/models>

Experiment results.

Test name	Loss/accuracy trend chart		Validation set (best model*)																																																																											
CNN			<table><tr><th colspan="5">classification report</th></tr><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>aircraft</td><td>0.73</td><td>0.83</td><td>0.78</td><td>488</td></tr><tr><td>automobile</td><td>0.88</td><td>0.89</td><td>0.88</td><td>512</td></tr><tr><td>bird</td><td>0.62</td><td>0.72</td><td>0.66</td><td>532</td></tr><tr><td>cat</td><td>0.52</td><td>0.54</td><td>0.53</td><td>471</td></tr><tr><td>deer</td><td>0.61</td><td>0.72</td><td>0.66</td><td>471</td></tr><tr><td>dog</td><td>0.68</td><td>0.66</td><td>0.67</td><td>514</td></tr><tr><td>frog</td><td>0.87</td><td>0.61</td><td>0.72</td><td>507</td></tr><tr><td>horse</td><td>0.83</td><td>0.78</td><td>0.80</td><td>500</td></tr><tr><td>ship</td><td>0.89</td><td>0.83</td><td>0.86</td><td>504</td></tr><tr><td>truck</td><td>0.86</td><td>0.82</td><td>0.84</td><td>501</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.74</td><td>5000</td></tr><tr><td>macro avg</td><td>0.75</td><td>0.74</td><td>0.74</td><td>5000</td></tr><tr><td>weighted avg</td><td>0.75</td><td>0.74</td><td>0.74</td><td>5000</td></tr></table>	classification report						precision	recall	f1-score	support	aircraft	0.73	0.83	0.78	488	automobile	0.88	0.89	0.88	512	bird	0.62	0.72	0.66	532	cat	0.52	0.54	0.53	471	deer	0.61	0.72	0.66	471	dog	0.68	0.66	0.67	514	frog	0.87	0.61	0.72	507	horse	0.83	0.78	0.80	500	ship	0.89	0.83	0.86	504	truck	0.86	0.82	0.84	501	accuracy			0.74	5000	macro avg	0.75	0.74	0.74	5000	weighted avg	0.75	0.74	0.74	5000
classification report																																																																														
	precision	recall	f1-score	support																																																																										
aircraft	0.73	0.83	0.78	488																																																																										
automobile	0.88	0.89	0.88	512																																																																										
bird	0.62	0.72	0.66	532																																																																										
cat	0.52	0.54	0.53	471																																																																										
deer	0.61	0.72	0.66	471																																																																										
dog	0.68	0.66	0.67	514																																																																										
frog	0.87	0.61	0.72	507																																																																										
horse	0.83	0.78	0.80	500																																																																										
ship	0.89	0.83	0.86	504																																																																										
truck	0.86	0.82	0.84	501																																																																										
accuracy			0.74	5000																																																																										
macro avg	0.75	0.74	0.74	5000																																																																										
weighted avg	0.75	0.74	0.74	5000																																																																										
ResNet_18			<table><tr><th colspan="5">classification report</th></tr><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>aircraft</td><td>0.80</td><td>0.84</td><td>0.82</td><td>488</td></tr><tr><td>automobile</td><td>0.78</td><td>0.98</td><td>0.87</td><td>512</td></tr><tr><td>bird</td><td>0.77</td><td>0.71</td><td>0.74</td><td>532</td></tr><tr><td>cat</td><td>0.74</td><td>0.58</td><td>0.65</td><td>471</td></tr><tr><td>deer</td><td>0.82</td><td>0.80</td><td>0.81</td><td>471</td></tr><tr><td>dog</td><td>0.75</td><td>0.72</td><td>0.74</td><td>514</td></tr><tr><td>frog</td><td>0.75</td><td>0.93</td><td>0.83</td><td>507</td></tr><tr><td>horse</td><td>0.91</td><td>0.83</td><td>0.87</td><td>500</td></tr><tr><td>ship</td><td>0.93</td><td>0.87</td><td>0.90</td><td>504</td></tr><tr><td>truck</td><td>0.90</td><td>0.86</td><td>0.88</td><td>501</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.81</td><td>5000</td></tr><tr><td>macro avg</td><td>0.82</td><td>0.81</td><td>0.81</td><td>5000</td></tr><tr><td>weighted avg</td><td>0.82</td><td>0.81</td><td>0.81</td><td>5000</td></tr></table>	classification report						precision	recall	f1-score	support	aircraft	0.80	0.84	0.82	488	automobile	0.78	0.98	0.87	512	bird	0.77	0.71	0.74	532	cat	0.74	0.58	0.65	471	deer	0.82	0.80	0.81	471	dog	0.75	0.72	0.74	514	frog	0.75	0.93	0.83	507	horse	0.91	0.83	0.87	500	ship	0.93	0.87	0.90	504	truck	0.90	0.86	0.88	501	accuracy			0.81	5000	macro avg	0.82	0.81	0.81	5000	weighted avg	0.82	0.81	0.81	5000
classification report																																																																														
	precision	recall	f1-score	support																																																																										
aircraft	0.80	0.84	0.82	488																																																																										
automobile	0.78	0.98	0.87	512																																																																										
bird	0.77	0.71	0.74	532																																																																										
cat	0.74	0.58	0.65	471																																																																										
deer	0.82	0.80	0.81	471																																																																										
dog	0.75	0.72	0.74	514																																																																										
frog	0.75	0.93	0.83	507																																																																										
horse	0.91	0.83	0.87	500																																																																										
ship	0.93	0.87	0.90	504																																																																										
truck	0.90	0.86	0.88	501																																																																										
accuracy			0.81	5000																																																																										
macro avg	0.82	0.81	0.81	5000																																																																										
weighted avg	0.82	0.81	0.81	5000																																																																										
ResNet_34			<table><tr><th colspan="5">classification report</th></tr><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>aircraft</td><td>0.84</td><td>0.86</td><td>0.85</td><td>488</td></tr><tr><td>automobile</td><td>0.89</td><td>0.94</td><td>0.92</td><td>512</td></tr><tr><td>bird</td><td>0.91</td><td>0.58</td><td>0.71</td><td>532</td></tr><tr><td>cat</td><td>0.74</td><td>0.63</td><td>0.68</td><td>471</td></tr><tr><td>deer</td><td>0.76</td><td>0.82</td><td>0.79</td><td>471</td></tr><tr><td>dog</td><td>0.68</td><td>0.81</td><td>0.74</td><td>514</td></tr><tr><td>frog</td><td>0.84</td><td>0.86</td><td>0.85</td><td>507</td></tr><tr><td>horse</td><td>0.80</td><td>0.88</td><td>0.84</td><td>500</td></tr><tr><td>ship</td><td>0.88</td><td>0.92</td><td>0.90</td><td>504</td></tr><tr><td>truck</td><td>0.90</td><td>0.90</td><td>0.90</td><td>501</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>5000</td></tr><tr><td>macro avg</td><td>0.82</td><td>0.82</td><td>0.82</td><td>5000</td></tr><tr><td>weighted avg</td><td>0.83</td><td>0.82</td><td>0.82</td><td>5000</td></tr></table>	classification report						precision	recall	f1-score	support	aircraft	0.84	0.86	0.85	488	automobile	0.89	0.94	0.92	512	bird	0.91	0.58	0.71	532	cat	0.74	0.63	0.68	471	deer	0.76	0.82	0.79	471	dog	0.68	0.81	0.74	514	frog	0.84	0.86	0.85	507	horse	0.80	0.88	0.84	500	ship	0.88	0.92	0.90	504	truck	0.90	0.90	0.90	501	accuracy			0.82	5000	macro avg	0.82	0.82	0.82	5000	weighted avg	0.83	0.82	0.82	5000
classification report																																																																														
	precision	recall	f1-score	support																																																																										
aircraft	0.84	0.86	0.85	488																																																																										
automobile	0.89	0.94	0.92	512																																																																										
bird	0.91	0.58	0.71	532																																																																										
cat	0.74	0.63	0.68	471																																																																										
deer	0.76	0.82	0.79	471																																																																										
dog	0.68	0.81	0.74	514																																																																										
frog	0.84	0.86	0.85	507																																																																										
horse	0.80	0.88	0.84	500																																																																										
ship	0.88	0.92	0.90	504																																																																										
truck	0.90	0.90	0.90	501																																																																										
accuracy			0.82	5000																																																																										
macro avg	0.82	0.82	0.82	5000																																																																										
weighted avg	0.83	0.82	0.82	5000																																																																										



Conclusion:

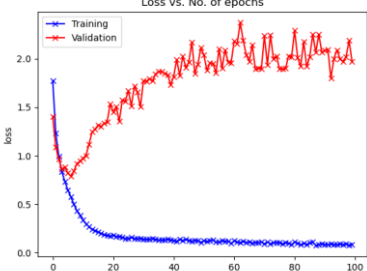
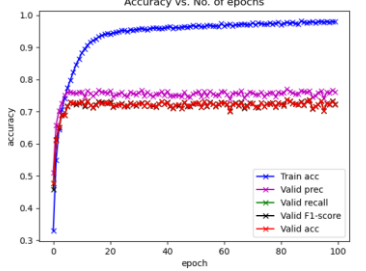
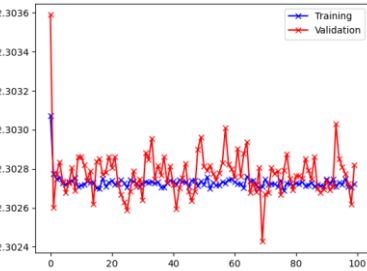
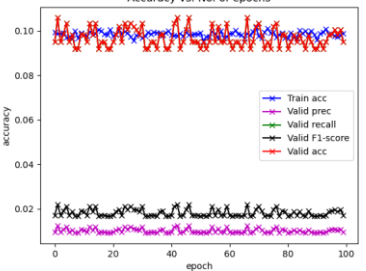
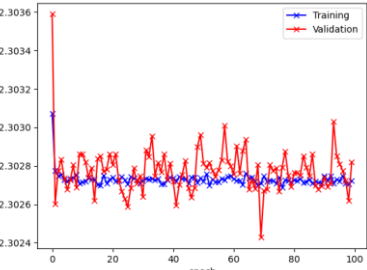
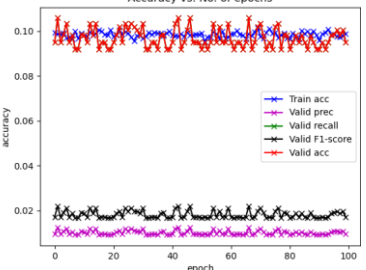
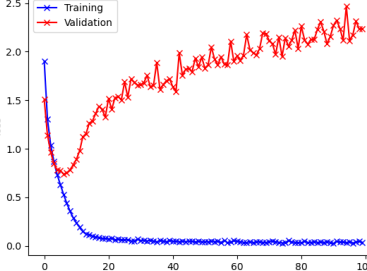
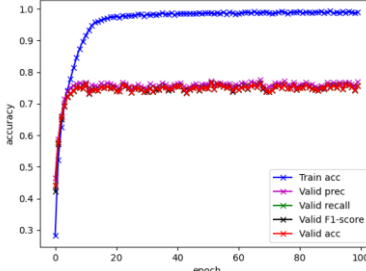
- The task was object detection and classification. A good option for the model could be YOLO. But due to time limitation I didn't quickly find the YOLO structure to check on cifar10. There for only classification task was analyzed here.
- The best predictions were using ResNet34 model.
- The worst was Vision Transformer (ViT) models. Generally, for the last 3 years, ViTs are the best in the class for image classification tasks. But for this task this model failed. The reason could be CIFAR-10 is a relatively small dataset with 60,000 32x32 color images in 10 classes. ViTs were originally designed for large-scale datasets with high-resolution images, such as ImageNet. ViTs require a large amount of data to perform well due to the massive number of parameters. With a smaller dataset like CIFAR-10, ViTs may not have had enough data to effectively learn meaningful representations. I didn't have time for fine-tuning of ViT model hyperparameters => it can be the reason of poor training as well.
- The optimal => Resnet34.

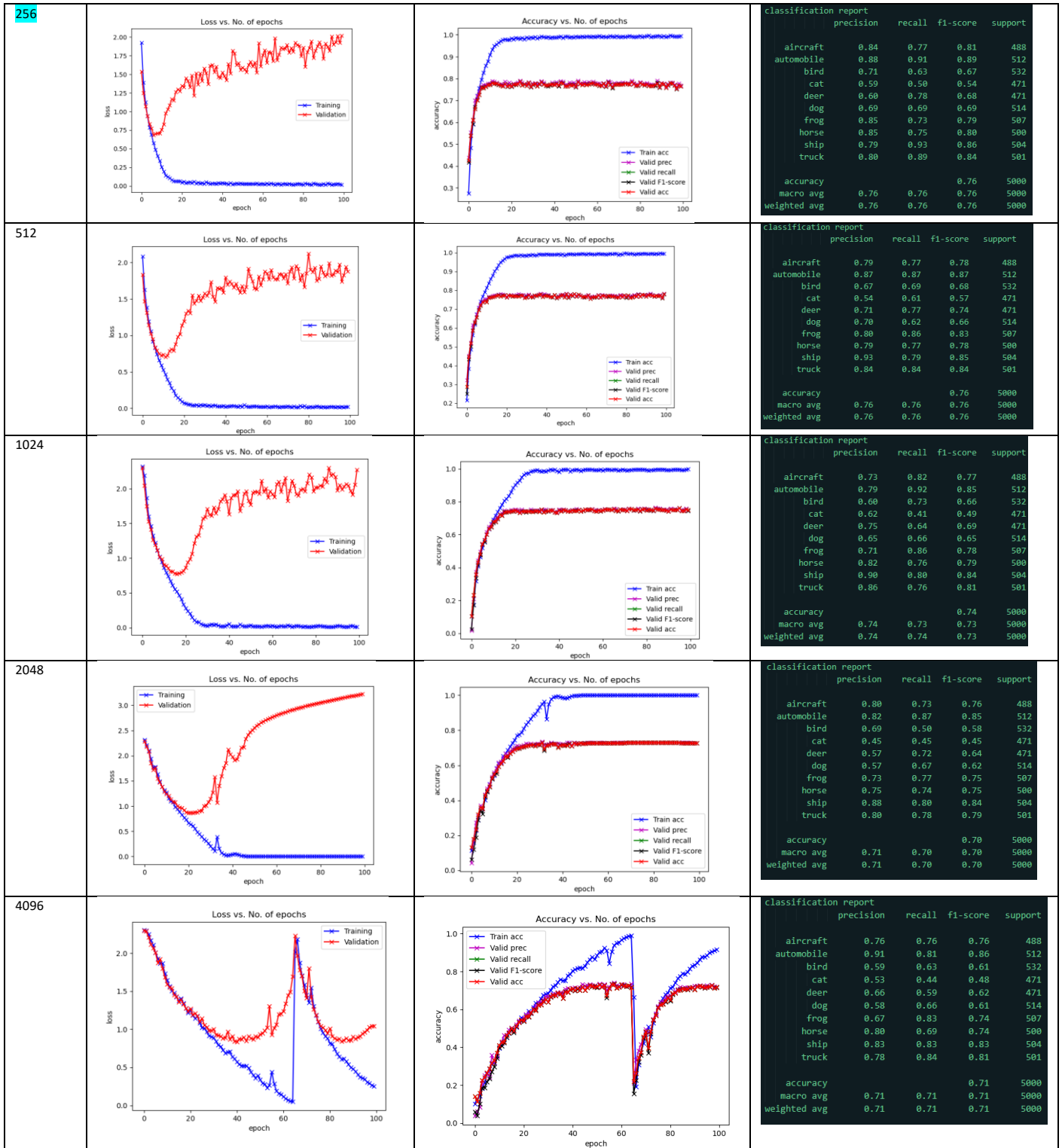
1.2.7. Batch size

Experiment description.

Batch size	Description	Results' path
32	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 32 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_32" --model "CNN"	results/CNN_batch_32_2024-02-28_08-29-43 LSF/CNN_batch_32_out_423953
64	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 64 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_64" --model "CNN"	results/CNN_batch_64_2024-02-28_08-29-56 LSF/CNN_batch_64_out_423954
	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 64 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_64_rev_2" --model "CNN"	results/CNN_batch_64_rev_2_2024-02-28_10-43-49 LSF/CNN_batch_64_rev_2_out_437228
128	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 128 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_No" --model "CNN"	results/CNN_No_2024-02-28_08-21-21 LSF/CNN_No_out_423862
256	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 256 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_256" --model "CNN"	results/CNN_batch_256_2024-02-28_08-29-54 LSF/CNN_batch_256_out_423967
512	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 512 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_512" --model "CNN"	results/CNN_batch_512_2024-02-28_08-29-54 LSF/CNN_batch_512_out_423968
1024	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 1024 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_1024" --model "CNN"	results/CNN_batch_1024_2024-02-28_08-30-01 LSF/CNN_batch_1024_out_423979
2048	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 2048 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_2048" --model "CNN"	results/CNN_batch_2048_2024-02-28_10-46-20 LSF/CNN_batch_2048_out_438261
4096	python cifar_10_train_rev_2.py --normalization "No" --val_size 10 --batch_size 4096 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "CNN_batch_4096" --model "CNN"	results/CNN_batch_4096_2024-02-28_10-58-09 LSF/CNN_batch_4096_out_439814

Experiment results.

Batch size	Loss/accuracy trend chart		Validation set (best model*)
32			<pre> classification report precision recall f1-score support aircraft 0.75 0.74 0.74 488 automobile 0.87 0.88 0.88 512 bird 0.62 0.66 0.64 532 cat 0.51 0.52 0.52 471 deer 0.63 0.73 0.68 471 dog 0.64 0.59 0.61 514 frog 0.83 0.73 0.78 507 horse 0.81 0.75 0.78 500 ship 0.82 0.83 0.82 504 truck 0.82 0.83 0.82 501 accuracy 0.73 5000 macro avg 0.73 0.73 0.73 5000 weighted avg 0.73 0.73 0.73 5000 </pre>
64			<pre> classification report precision recall f1-score support aircraft 0.00 0.00 0.00 488 automobile 0.00 0.00 0.00 512 bird 0.00 0.00 0.00 532 cat 0.00 0.00 0.00 471 deer 0.00 0.00 0.00 471 dog 0.00 0.00 0.00 514 frog 0.10 1.00 0.18 507 horse 0.00 0.00 0.00 500 ship 0.00 0.00 0.00 504 truck 0.00 0.00 0.00 501 accuracy 0.10 5000 macro avg 0.01 0.10 0.02 5000 weighted avg 0.01 0.10 0.02 5000 </pre>
64 second run			<pre> classification report precision recall f1-score support aircraft 0.00 0.00 0.00 488 automobile 0.00 0.00 0.00 512 bird 0.00 0.00 0.00 532 cat 0.00 0.00 0.00 471 deer 0.00 0.00 0.00 471 dog 0.00 0.00 0.00 514 frog 0.10 1.00 0.18 507 horse 0.00 0.00 0.00 500 ship 0.00 0.00 0.00 504 truck 0.00 0.00 0.00 501 accuracy 0.10 5000 macro avg 0.01 0.10 0.02 5000 weighted avg 0.01 0.10 0.02 5000 </pre>
128			<pre> classification report precision recall f1-score support aircraft 0.73 0.83 0.78 488 automobile 0.88 0.89 0.88 512 bird 0.62 0.72 0.66 532 cat 0.52 0.54 0.53 471 deer 0.61 0.72 0.66 471 dog 0.68 0.66 0.67 514 frog 0.87 0.61 0.72 507 horse 0.83 0.78 0.80 500 ship 0.89 0.83 0.86 504 truck 0.86 0.82 0.84 501 accuracy 0.74 5000 macro avg 0.75 0.74 0.74 5000 weighted avg 0.75 0.74 0.74 5000 </pre>



Conclusion:

- Low batch size => noisy gradients, slow convergence
- High batch size => poor optimization, poor generalization, as the model may not learn the subtle patterns present in the data, divergence.
- Batch size = 64 => the worst in the class. I don't have explanation for this outlier => need additional examination for the issue.
- The optimal => batch size = 256 (highest accuracy/recall/precision/F1 for the best model).

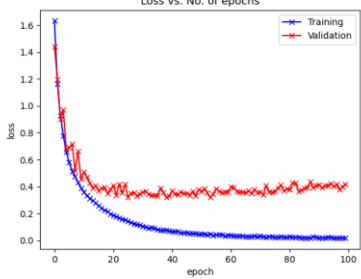
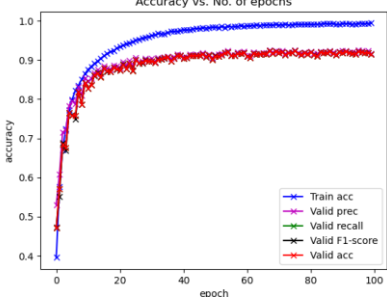
1.2.8. Optimal model

Experiment description.

Parameter	Optimal
preprocess	N1_aug
Validation set size, %	10
LR	0.001
Optimizer	Adam
Model	ResNet34
batch size	256

Test name	Description	Results' path
optimal model	python cifar_10_train_rev_2.py --normalization "N1_aug" --val_size 10 --batch_size 256 --num_workers 4 --lr 0.001 --epochs 100 --optimization "Adam" --experiment_name "optimal_model" --model "ResNet_34"	results/optimal_model_2024-02-28_12-23-42 LSF/optimal_model_out_443340

Experiment results.

Test name	Loss/accuracy trend chart		Validation set (best model*)				
optimal model			classification report				
			precision	recall	f1-score	support	
			aircraft	0.95	0.92	0.93	488
			automobile	0.97	0.97	0.97	512
			bird	0.90	0.89	0.90	532
			cat	0.82	0.87	0.84	471
			deer	0.92	0.91	0.91	471
			dog	0.88	0.85	0.86	514
			frog	0.91	0.95	0.93	507
			horse	0.95	0.92	0.94	500
			ship	0.95	0.96	0.96	504
			truck	0.95	0.97	0.96	501
			accuracy			0.92	5000
			macro avg	0.92	0.92	0.92	5000
			weighted avg	0.92	0.92	0.92	5000

Conclusion:

- The optimal model is stable, without overfitting.
- The minimal validation loss = 0.3202 reached on epoch = 53.

	Best model epoch = 53	Last model epoch = 100
loss	0.3202	0.4189
accuracy	0.9178	0.9140
precision	0.9203	0.9165
recall	0.9178	0.9140
F1	0.9179	0.9139

1.3. Inference

The summary of model performance using the test set is in the file “inference.ipynb”:

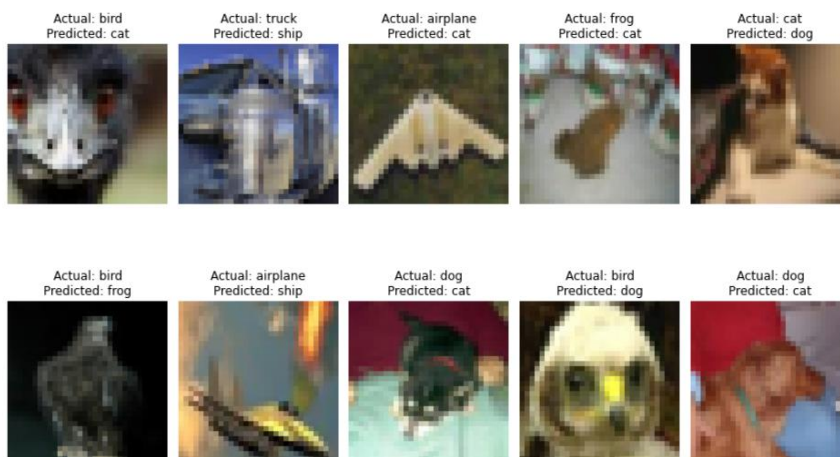
```
** accuracy: 0.9170
--
confusion matrix
[[916  4 18 11  7  0  7  4 28  5]
 [ 1970  0  1  1  0  2  0  7 18]
 [ 18  0 886 19 25 14 29  4  4  1]
 [ 7  1 35 826 16 67 31  9  4  4]
 [ 2  0 15 15 918 15 20 14  1  0]
 [ 3  0 14 79 10 869  8 14  1  2]
 [ 5  1 21 15  2  3 951  1  0  1]
 [ 5  0 10 13 11 13  2 943  1  2]
 [18  8  3  5  2  0  2  0 950 12]
 [ 8 34  0  5  0  1  1  1  9 941]]
--
classification report
      precision    recall  f1-score   support

   aircraft      0.93      0.92      0.92      1000
  automobile      0.95      0.97      0.96      1000
         bird      0.88      0.89      0.89      1000
          cat      0.84      0.83      0.83      1000
         deer      0.93      0.92      0.92      1000
          dog      0.88      0.87      0.88      1000
          frog      0.90      0.95      0.93      1000
         horse      0.95      0.94      0.95      1000
          ship      0.95      0.95      0.95      1000
          truck      0.95      0.94      0.95      1000

   accuracy              0.92      10000
  macro avg              0.92      10000
 weighted avg              0.92      10000
```

The model accuracy is above 90%, which is not bad. According to precision and recall, the prediction of cats, dogs and birds are the hardest. The predictions of auto, horse, ship, and truck are the best.

Visualization of the first 10 wrong predictions:



Visualization of the first 10 correct predictions:



Task 2

2.1.

1) If dataset will have large scale/high resolution RGB images there can be either bottle neck within OOM of GPU or the training can be very slow. For such tasks I will use small batches and use DDP multi-GPU processing using torch lightning => <https://lightning.ai/docs/pytorch/stable/>.

In this situation the small batches are running on separate GPUs and the gradient of NN is updated per average of all separate batches per step. The effective batch size is the batch size * # GPU in multiprocessing.

For effective batch size < 8K there is no degradation on model precision => <https://arxiv.org/pdf/1706.02677.pdf>

I did some experiments with torch lightning using MNIST database => see Appendix A for details.

2) If there are a lot of images (no labels needed here), we can do image embedding using autoencoder, when the inner layer of autoencoder will be image embedding vector. After use these embedded images for model input.

3) Reduce image resolution => but can lead to accuracy degradation, so proposal

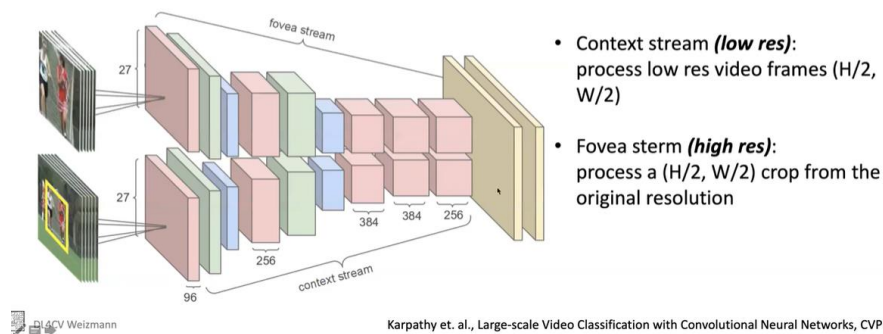
(https://www.youtube.com/watch?v=DJlpYiUemiw&list=PL_Z_U9MIJdNgFM7-f2fZ9ZxiVRP_jhJv&index=17) use model with combination of low resolution H/2, W/2 frames in combination with high resolution central cropped H2/W2 frames:

Models for Videos: Multi-scale

How can we reduce computational cost while maintaining accuracy?

Reduce video resolution → lower performance

Reduce network's capacity → lower performance



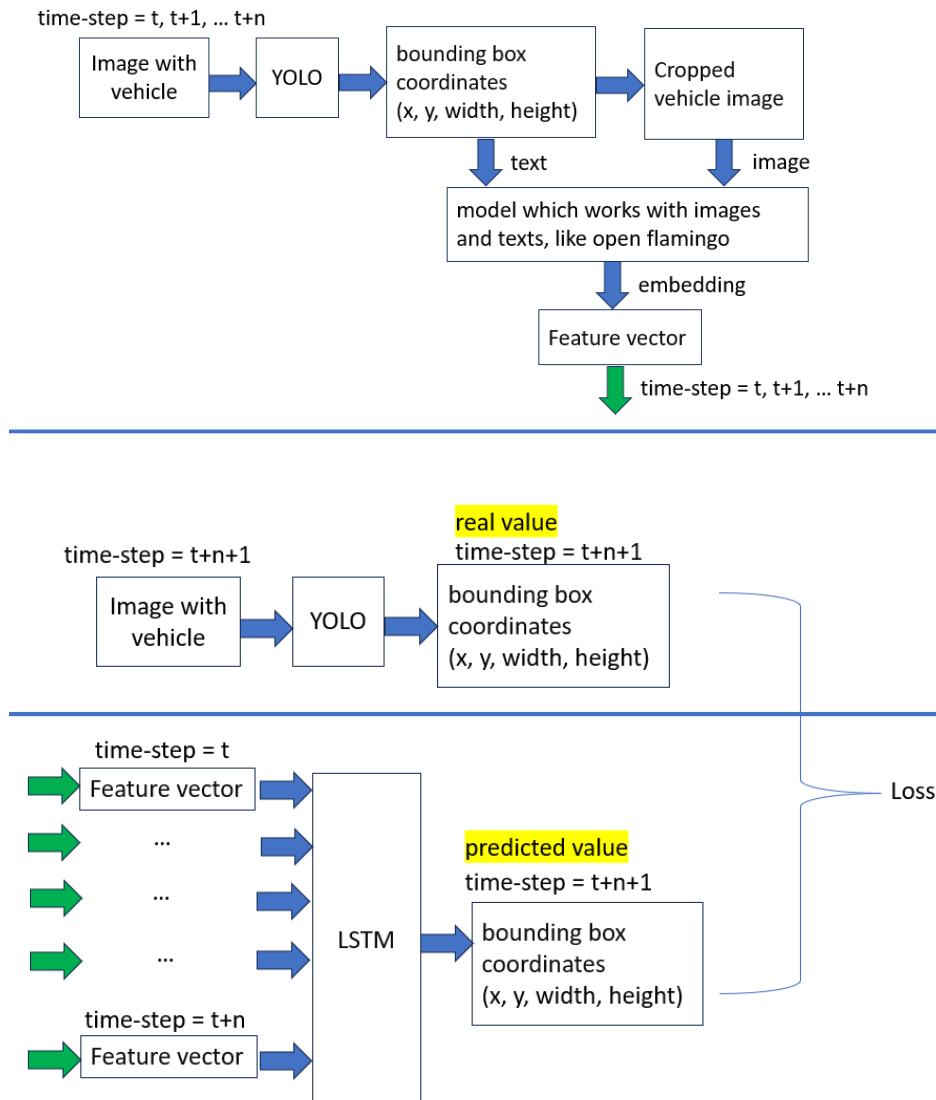
- Context stream (**low res**): process low res video frames (H/2, W/2)
- Fovea stream (**high res**): process a (H/2, W/2) crop from the original resolution

2.2.

1) Use convLSTM which used for the next frame prediction method in video =>

<https://sladewinter.medium.com/video-frame-prediction-using-convlstm-network-in-pytorch-b5210a6ce582>

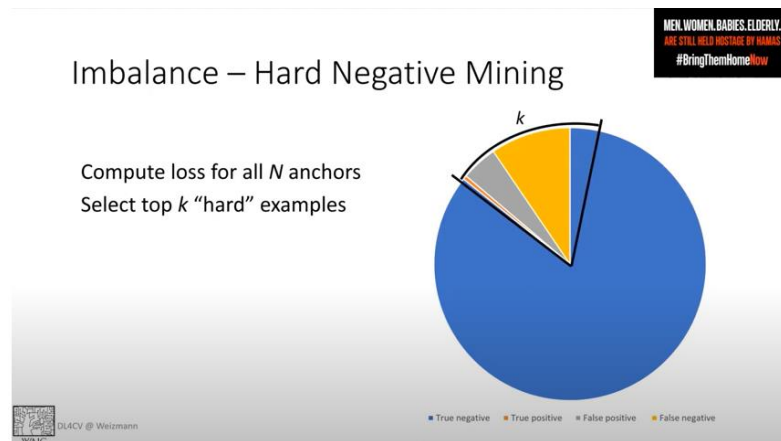
2) Use some hybrid structure like:



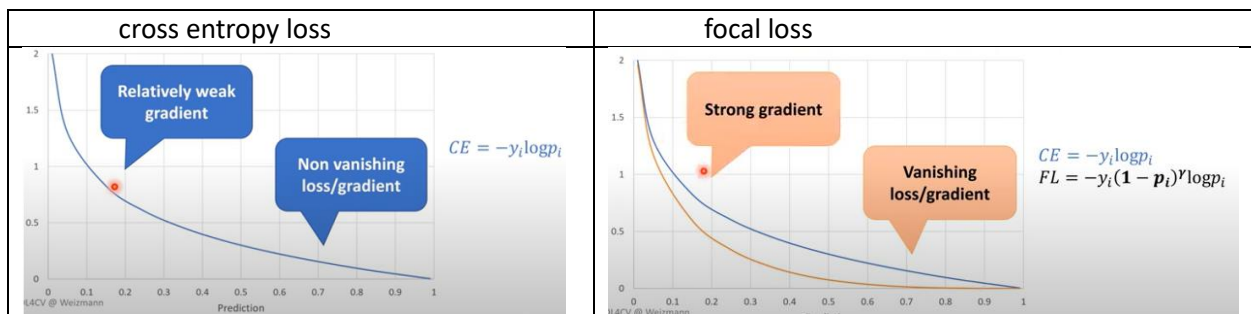
2.3.

If there is class imbalance issue, we can do:

- 1) Generate synthetic samples for minor class (GAN can be used).
- 2) Do data augmentation to generate more samples for minor class.
- 3) Apply higher weight on minor class during loss calculation (algorithms like `class_weight` parameter in scikit-learn or `weight` parameter in PyTorch's loss functions can be used).
- 4) During loss calculation take the limited number of samples of major class to be match the # of samples in minor class (<https://www.youtube.com/watch?v=kUzkEEUygEA>):



- 5) Use Focal loss functions instead of cross entropy (<https://www.youtube.com/watch?v=kUzkEEUygEA>):



Additional references:

- Johnson, J.M., Khoshgoftaar, T.M. Survey on deep learning with class imbalance. *J Big Data* 6, 27 (2019).
<https://doi.org/10.1186/s40537-019-0192-5>
- Focal loss:
Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980-2988) <https://doi.org/10.1109/ICCV.2017.324>
- Online hard example mining:
Shrivastava, A., Gupta, A., & Girshick, R. (2016). Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 761-769).
<https://doi.org/10.1109/CVPR.2016.89>

Appendix A

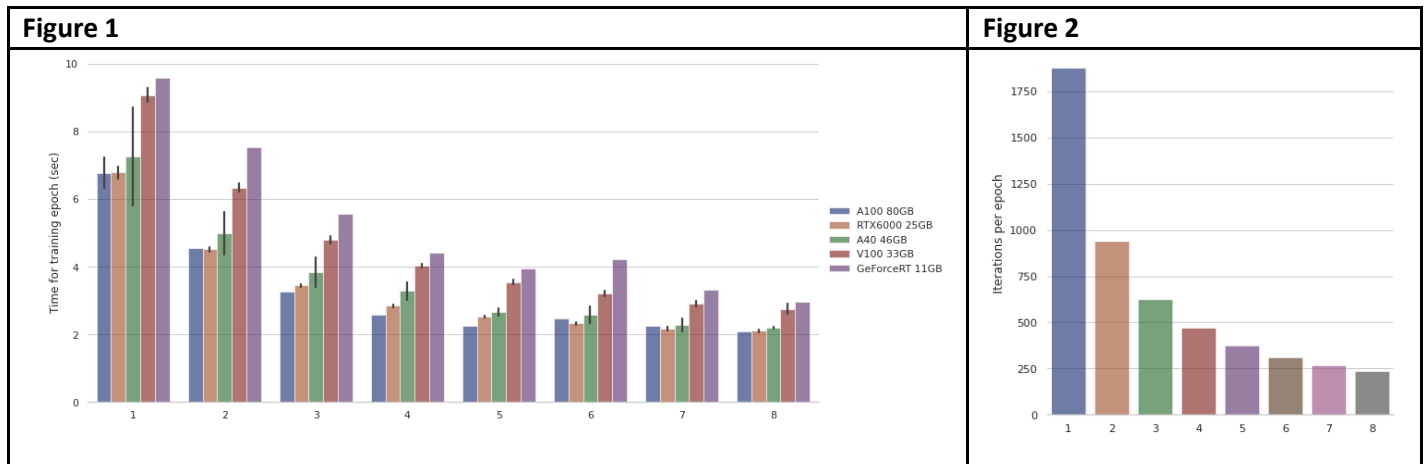
Running details: MNIST data classification, batch size=32, 2 epochs run (~5 min to have minimal impact of waic resources), DDP accelerator used for multi node execution.

Analysis:

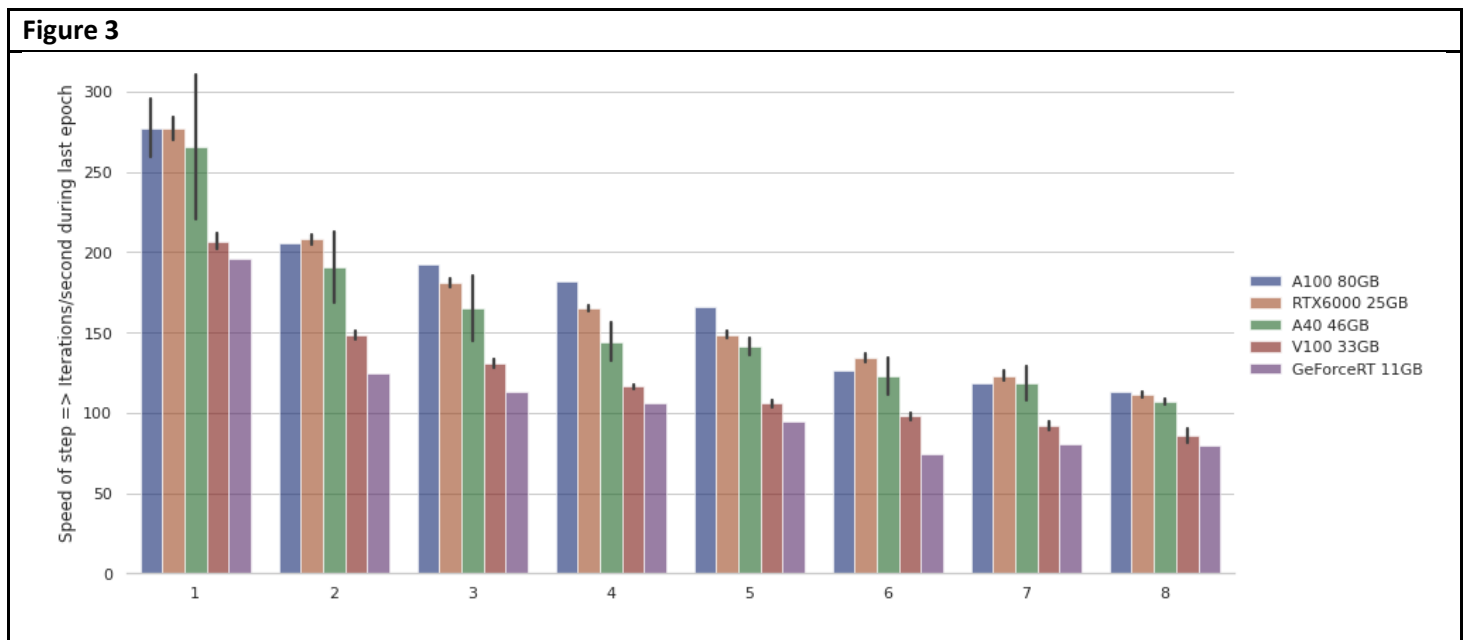
1. Local run (when server was empty from jobs) on 17 different servers with #GPU change from 1 to 8 (Figure 1).

The best server is A100 (blue), the worst – V100 (red) & GeForceRT (purple). RTX6000 is a little bit better than A40 for this specific problem (but A40 have 46GB versus 25GB GPU memory advantage).

When #GPU ↑, the training time ↓ (Figure 1). Logical, because the parallel processes executed and # iterations per epoch ↓ when #GPU ↑ (Figure 2).



When #GPU ↑, the speed of iteration step ↓ (Figure 3) => can be related to impact of synchronization process (more GPU involved => more complicated synchronization process => impact on speed of iteration).



2. I ran 8 GPU training on the single node with 8 GPU per node (local run => Table 1) versus 8 GPU training using 8 nodes with 1 GPU per node (LSF run => Table 2)

Table 1

run ID	server		total # GPU	# nodes	# GPU/node	iterations/epoch	iteration/sec	training time, sec/epoch	constrain on GPU type?	comment
1	hgn46/44/41/42/47/54/50/49	A40 46GB	8	8	1	235	113.22	2.08	waic_2023_gpu	LSF run
2	hgn44/41/47/43/54/45/50/42	A40 46GB	8	8	1	235	88.78	2.65	waic_2023_gpu	LSF run
3	hgn41/54/44/47/45/50/42/43	A40 46GB	8	8	1	235	124.63	1.89	waic_2023_gpu	LSF run
4	hgn41/54/44/47/45/50/42/43	A40 46GB	8	8	1	235	105.12	2.24	waic_2023_gpu	LSF run
5	hgn44/45/50/42/43/52/55/51	A40 46GB	8	8	1	235	113.8	2.07	waic_2023_gpu	LSF run
6	hgn44/45/50/52/55/42/51/53	A40 46GB	8	8	1	235	124.55	1.89	waic_2023_gpu	LSF run
7	hgn44/45/50/52/55/42/51/53	A40 46GB	8	8	1	235	124.81	1.88	waic_2023_gpu	LSF run
8	hgn44/45/50/52/55/42/51/53	A40 46GB	8	8	1	235	119.06	1.97	waic_2023_gpu	LSF run
9	hgn44/41/54/52/55/45/47/51	A40 46GB	8	8	1	235	99.76	2.36	waic_2023_gpu	LSF run
10	hgn44/41/54/45/47/50/42/43	A40 46GB	8	8	1	235	100.35	2.34	waic_2023_gpu	LSF run
median =							113.51	2.07		
stdev =							12.49	0.26		

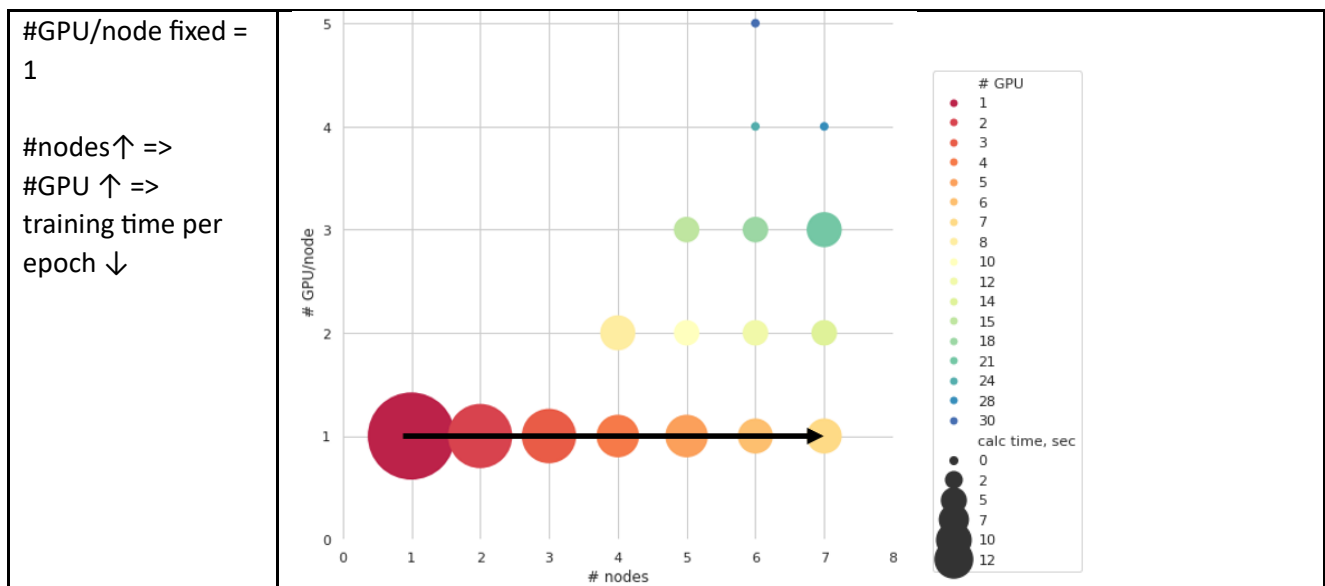
Table 2

run ID	server		total # GPU	# nodes	GPU/node	iterations/epoch	iteration/sec	training time, sec/epoch	constrain on GPU type?	comment
1	hgn56	A40 46GB	8	1	8	235	105.35	2.23	n/a	local run
2	hgn41	A40 46GB	8	1	8	235	110.06	2.14	n/a	local run
3	hgn45	A40 46GB	8	1	8	235	110.82	2.12	n/a	local run
4	hgn47	A40 46GB	8	1	8	235	113.1	2.08	n/a	local run
5	hgn49	A40 46GB	8	1	8	235	109.2	2.15	n/a	local run
6	hgn53	A40 46GB	8	1	8	235	110.54	2.13	n/a	local run
median =							110.30	2.13		
stdev =							2.56	0.05		

The expectation was that for the same # of GPU the multi node execution will be worst.
But for MNIST problem => multi-node DDP showed higher speed (113.51 versus 110.30) and lower training time (2.07 versus .13).
On the other side, multi-node DDP showed the higher run-to-run variation in results when single node execution was more stable.

3. Multi-node LSF execution analysis.

In this experiment #nodes changed from 1 to 7 and #GPU per node changed from 1 to 5.
Up to 30 GPUs per run were checked.
All the nodes were of the same type => A40 46GB.



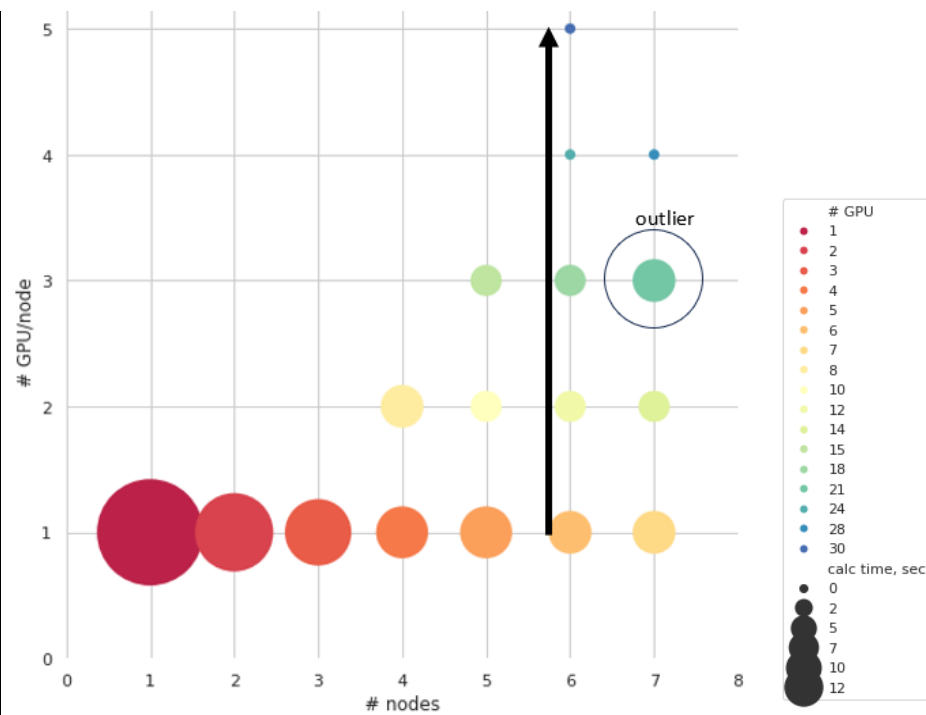
#nodes fixed = 6

#GPU/node fixed ↑

=>

#GPU ↑ =>

training time per epoch ↓

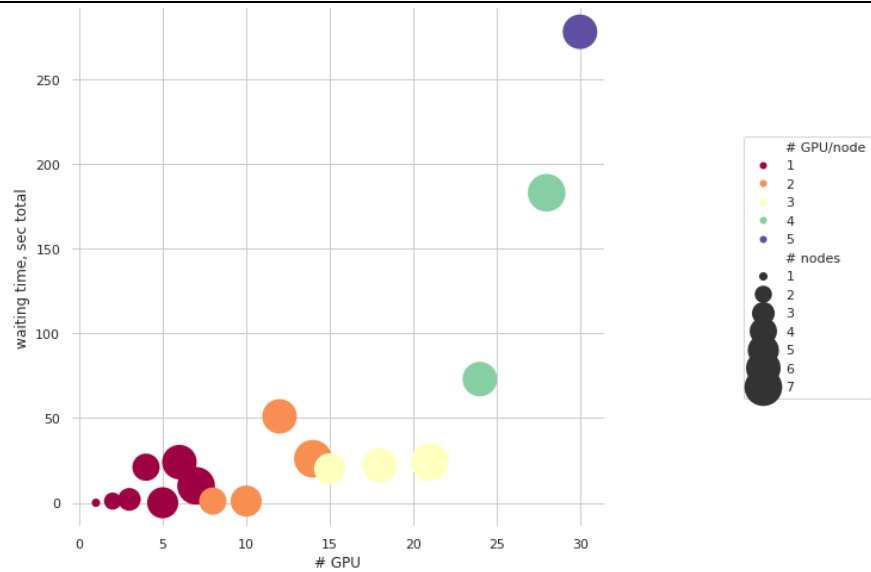


requested GPU ↑

=>

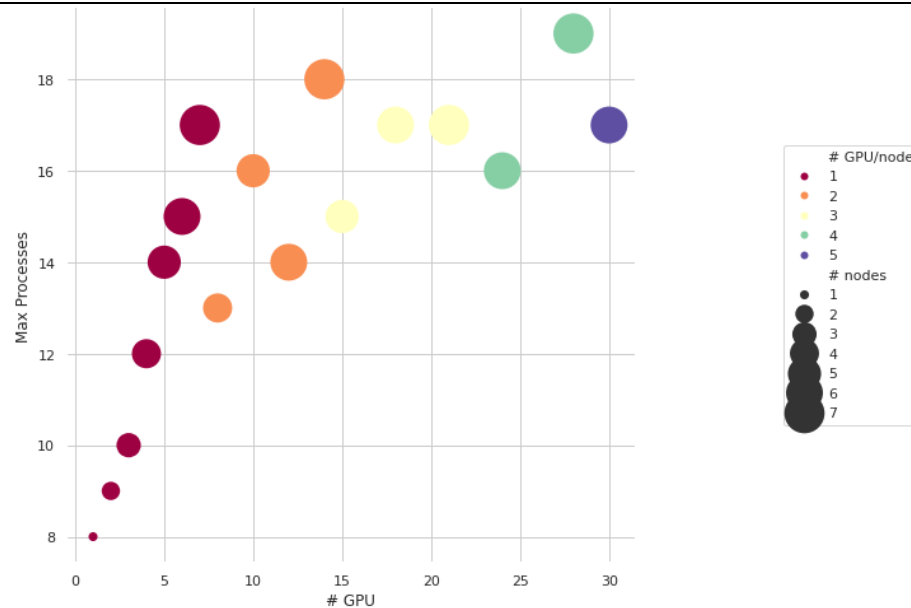
waiting time for job ↑

↑

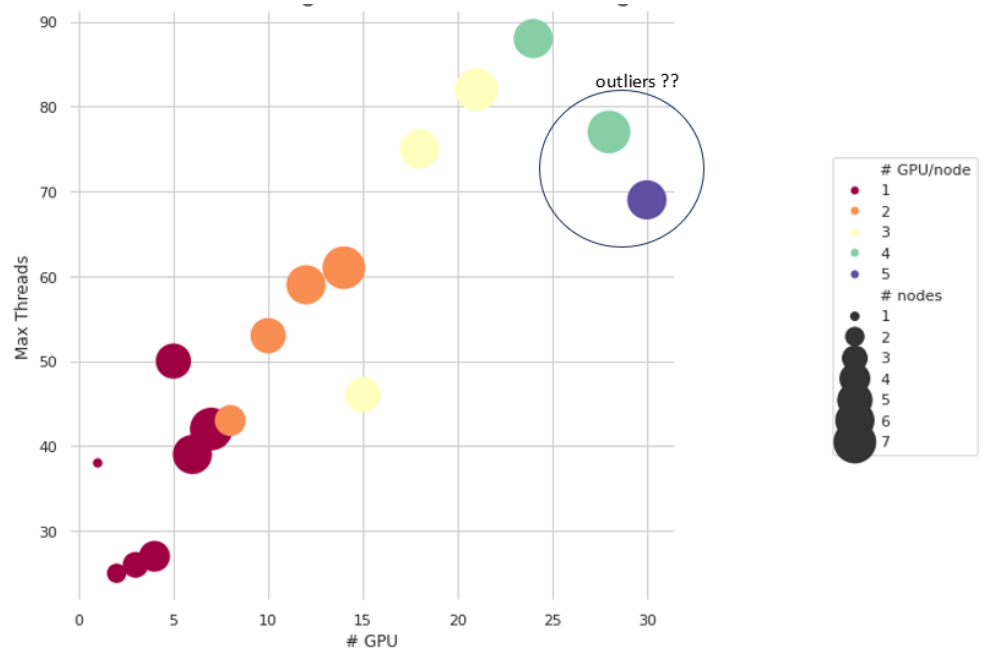


GPU ↑ =>
Max Processes ↑

But I don't understand why for 30 GPU => max processes is 17. Per DDP is need be 30 as well



GPU ↑ =>
Max Threads ↑

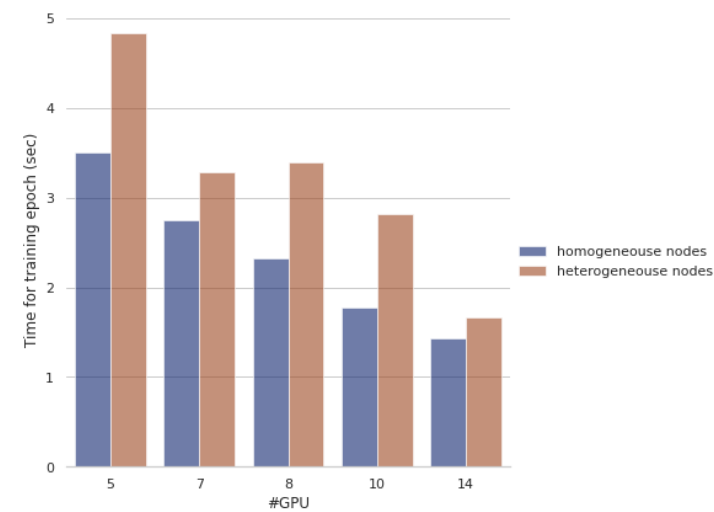


#GPU	steps/epoch	effective batch size	constrain on GPU type?	comment	it/s	calc time, sec	node	# nodes	# GPU/node	# GPU	job submitting	job starting	waiting time	Run time, sec	Turnaround time, sec	CPU time, sec	Max memory, MB	Average Memory, MB	Max Processes	Max Threads
1	1875	32	waic_2023_gpu	LSF run	136.82	13.70	hgn47	1	1	1	11:16:59	11:16:59	0:00:00	78	76	47	3277	1552	8	38
2	938	64	waic_2023_gpu	LSF run	128.85	7.28	hgn47/53	2	1	2	11:20:41	11:20:42	0:00:01	62	64	82	2283	1278	9	25
3	625	96	waic_2023_gpu	LSF run	118.09	5.29	hgn47/53/56	3	1	3	11:23:24	11:23:26	0:00:02	87	89	120	2287	762	10	26
4	469	128	waic_2023_gpu	LSF run	125.33	3.74	hgn47/53/51/56	4	1	4	11:29:05	11:29:26	0:00:21	83	100	138	3430	1118	12	27
5	375	160	waic_2023_gpu	LSF run	107.02	3.50	hgn47/43/53/51/56	5	1	5	11:31:48	11:31:48	0:00:00	104	100	342	10446	2012	14	50
6	313	192	waic_2023_gpu	LSF run	113.42	2.76	hgn47/43/45/53/51/56	6	1	6	11:35:24	11:35:48	0:00:24	126	143	521	13027	1630	15	39
7	268	224	waic_2023_gpu	LSF run	97.34	2.75	hgn44/47/43/45/53/51/56	7	1	7	11:38:38	11:38:48	0:00:10	93	102	423	15769	1132	17	42
8	235	256	waic_2023_gpu	LSF run	101.28	2.32	hgn49/43/45/51	4	2	8	11:49:27	11:49:28	0:00:01	75	76	315	8233	2137	13	43
10	188	320	waic_2023_gpu	LSF run	106.02	1.77	hgn49/43/45/51/56	5	2	10	11:51:39	11:51:40	0:00:01	141	143	897	15147	3408	16	53
12	156	384	waic_2023_gpu	LSF run	92.66	1.68	hgn49/43/45/51/53/56	6	2	12	11:55:13	11:56:24	0:00:51	132	181	1118	18872	2550	14	59
14	134	448	waic_2023_gpu	LSF run	93.57	1.43	hgn49/52/43/45/51/53/56	7	2	14	11:59:19	11:59:45	0:00:26	116	142	1011	30014	3811	18	61
15	125	480	waic_2023_gpu	LSF run	96.83	1.29	hgn44/45/51/52/53	5	3	15	12:03:56	12:04:16	0:00:20	46	67	253	6654	3293	15	46
18	105	576	waic_2023_gpu	LSF run	89.18	1.18	hgn44/45/52/53/42/51	6	3	18	12:05:54	12:06:16	0:00:22	76	98	368	40487	2162	17	75
21	90	672	waic_2023_gpu	LSF run	38.87	2.32	hgn44/45/49/51/52/53/42	7	3	21	12:08:20	12:08:44	0:00:24	86	108	618	31211	2074	17	82
24	79	768	waic_2023_gpu	LSF run	93.31	0.85	hgn44/49/52/42/53/51/56	6	4	24	12:11:29	12:12:42	0:01:13	151	221	1796	35549	3879	16	88
28	67	896	waic_2023_gpu	LSF run	87.18	0.77	hgn44/49/52/42/53/51/56	7	4	28	12:12:23	12:15:26	0:03:03	137	319	989	15622	2005	19	77
30	63	960	waic_2023_gpu	LSF run	92.45	0.68	hgn44/42/51/43/52/56	6	5	30	12:13:22	12:18:00	0:04:38	93	369	498	14339	2506	17	69

4. The last analysis is answering the question how the homogeneity of nodes influences the iteration speed.

During LSF run I removed the constraint to run only on A40 machines (waic_2023_gpu) and left the condition to LSF use only machines with GPU mem lower limit = 30G.

From the figure below we can see that it is better to use homogeneous nodes.



#GPU	steps/epoch	steps/epoch calc	effective batch size	constrain on GPU type?	comment	it/s	calc time, sec	node	# nodes	# GPU/node	# GPU	group
5	375	375	160	waic_2023_gpu	LSF run	107.02	3.50	hgn47/43/53/51/56	5	1	5	homogeneous nodes
5	375	375	160	gmem=30G	LSF run	77.58	4.83	ibdgx008/007/004/003, hgn44	5	1	5	heterogeneous nodes
7	268	268	224	waic_2023_gpu	LSF run	97.34	2.75	hgn44/47/43/45/53/51/56	7	1	7	homogeneous nodes
7	268	268	224	gmem=30G	LSF run	81.71	3.28	ibdgx007/004/003, hgn43/52/48/56	7	1	7	heterogeneous nodes
8	235	234	256	waic_2023_gpu	LSF run	101.28	2.32	hgn49/43/45/51	4	2	8	homogeneous nodes
8	235	234	256	gmem=30G	LSF run	69.37	3.39	ibdgx007/004/003, hgn43	4	2	8	heterogeneous nodes
10	188	188	320	waic_2023_gpu	LSF run	106.02	1.77	hgn49/43/45/51/56	5	2	10	homogeneous nodes
10	188	188	320	gmem=30G	LSF run	66.74	2.82	ibdgx004/003, hgn43/52/48	5	2	10	heterogeneous nodes
14	134	134	448	waic_2023_gpu	LSF run	93.57	1.43	hgn49/52/43/45/51/53/56	7	2	14	homogeneous nodes
14	134	134	448	gmem=30G	LSF run	80.46	1.67	ibdgx004/003, hgn43/52/48/56, ibdgxa01	7	2	14	heterogeneous nodes