Question-1

a. **Prepare CIFAR-10 with proper normalization and data augmentation. Specify the transforms used. (5)**

- The training data is pre-processed using several data augmentation transformations to improve model generalization on the CIFAR-10 dataset. These transformations help simulate diverse visual conditions and prevent overfitting. The augmentations applied in the GetCifar10() function are:
  i. RandomCrop(32, padding=4)
     1. Randomly crops a 32×32 region from the image after padding it by 4 pixels on each side.
  ii. RandomHorizontalFlip()
     1. Randomly flips the image horizontally with a probability of 0.5.
  iii. CIFAR10Policy()
     1. Applies one randomly selected sub-policy from 25 predefined ones.
     2. Each sub-policy consists of two image transformations chosen from operations like:
        a. Rotate, ShearX/Y, TranslateX/Y, Solarize, Posterize, Color, Contrast, Brightness, Sharpness, Invert, Equalize, Autocontrast, etc.
  iv. Normalize(mean, std)
     1. Normalizes each channel using the dataset mean and standard deviation:
        a. Mean = (0.4914, 0.4822, 0.4465)
        b. Std = (0.2023, 0.1994, 0.2010)
  v. Cutout(n_holes=1, length=16)
     1. Randomly masks out one square region (of size 16×16) from the image.

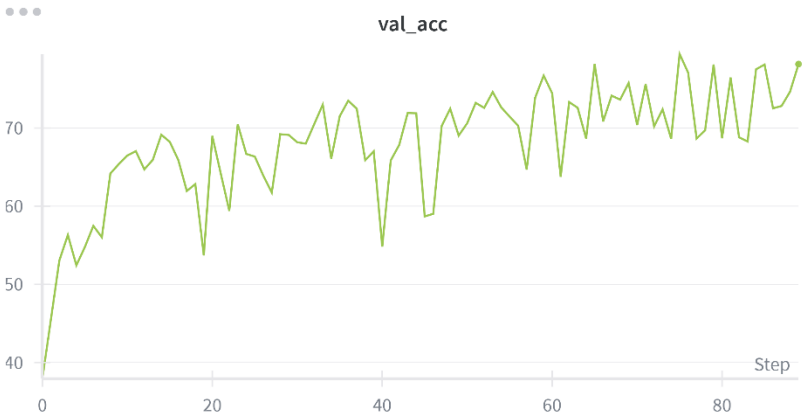
b. **Train the model with one chosen configuration (preferably the one that gives the best test accuracy). (3)**

- The configuration which gives the best validation accuracy as per the training results is with below hyperparameters:
  i. activation=selu
  ii. batch_norm=True
  iii. batch_size=64
  iv. epochs=90
  v. learning_rate=0.00351147544245589
  vi. momentum=0.8247395191150262
  vii. optimizer=nadam
  viii. weight_decay=2.0531225329636637e-05
- The best validation accuracy achieved is 78.18%.
- The best network architecture is:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): SELU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): SELU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): SELU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): SELU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

c. Report the final test top-1 accuracy and include loss/accuracy curves. (2)

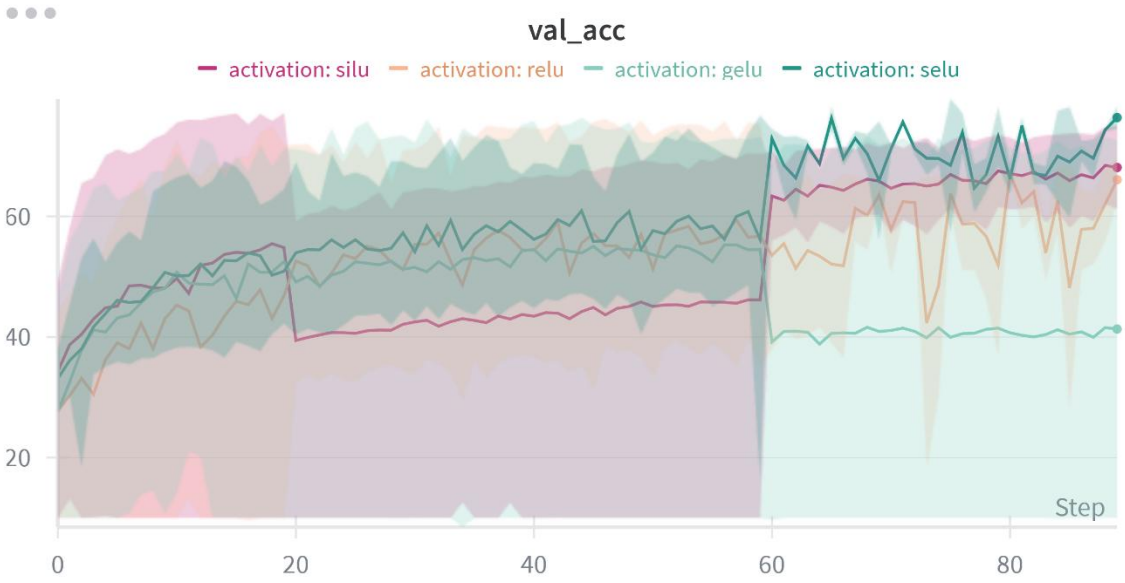
- Final test top-1 accuracy is 78.18%
- Below is the test accuracy plot.



Question 2. Model Performance on Different Configurations (60 points)

d. Vary the activation function. Use different activations such as ReLU, Sigmoid, Tanh, SiLU, GELU, etc. Describe how model performance changes when the activation function is varied. (20)

- The activation function was varied among ReLU, SiLU (Swish), GELU, and SELU.
- The figure below shows the validation accuracy curves for each activation across epochs:



- Observations From the Plot:
  i. SiLU
     1. Achieved the highest and most stable validation accuracy (~65–70%).
     2. Showed smoother convergence and smaller fluctuations across epochs.

3. The smooth, non-monotonic activation allows better gradient flow compared to ReLU.

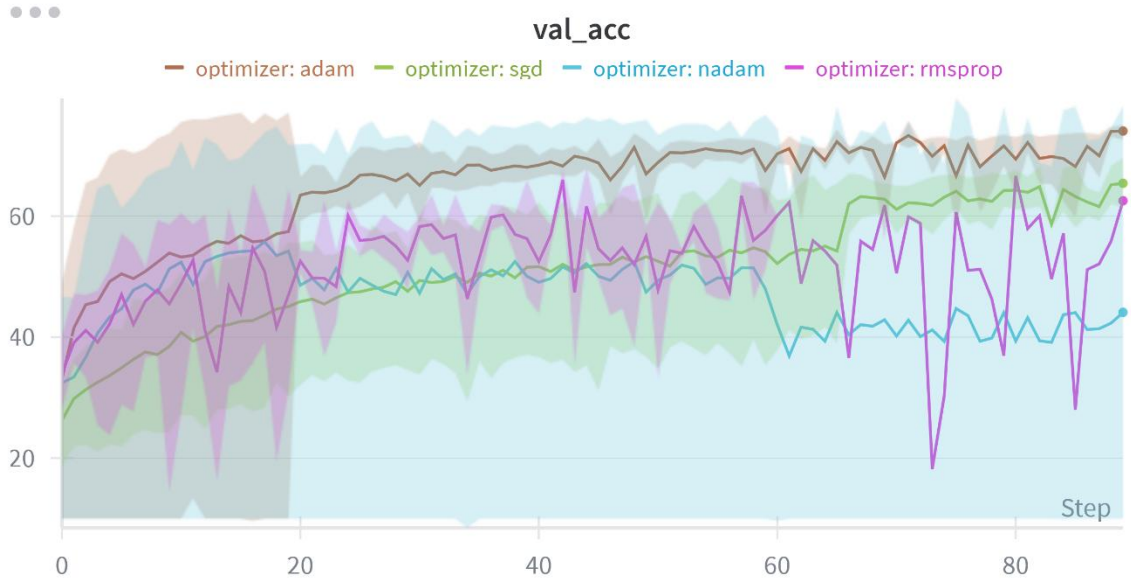
ii. ReLU
1. Provided decent performance (around 60–65% validation accuracy).
2. Converged quickly but exhibited slightly higher variance across epochs.
3. Serves as a strong and efficient baseline but suffers from "dead neurons" in some layers.

iii. SELU
1. Showed moderate accuracy (~60–68%), close to SiLU in later epochs.
2. Convergence was slower initially but became stable later.
3. **Works best in self-normalizing networks without BatchNorm. Hence, when Batch normalization was enabled, this achieved best validation accuracy.**

iv. GELU
1. Training was less stable, possibly due to small model depth and dataset scale, where GELU's probabilistic behavior provides less advantage.

e. **Vary the optimizer. Use different optimizers such as SGD, Nesterov-SGD, Adam, Adagrad, RMSprop, Nadam, etc. Explain how each optimizer affects convergence and how they differ from one another. (30)**

- The optimizer was varied among SGD, Adam, RMSprop, Nadam.
- The figure below presents the validation accuracy (val_acc) trends across epochs for each optimizer:

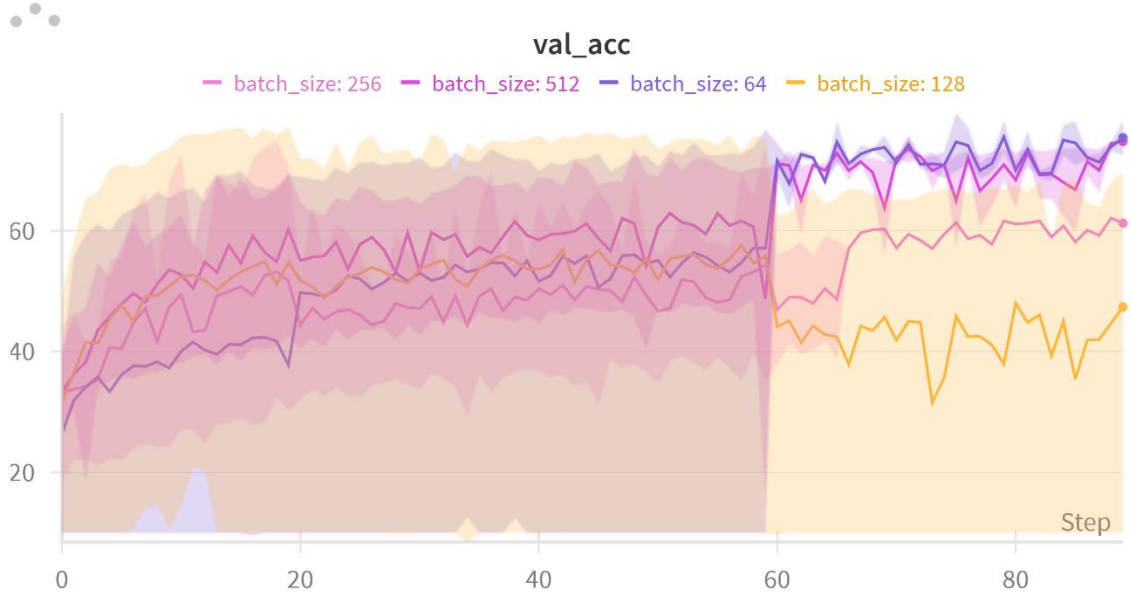


-

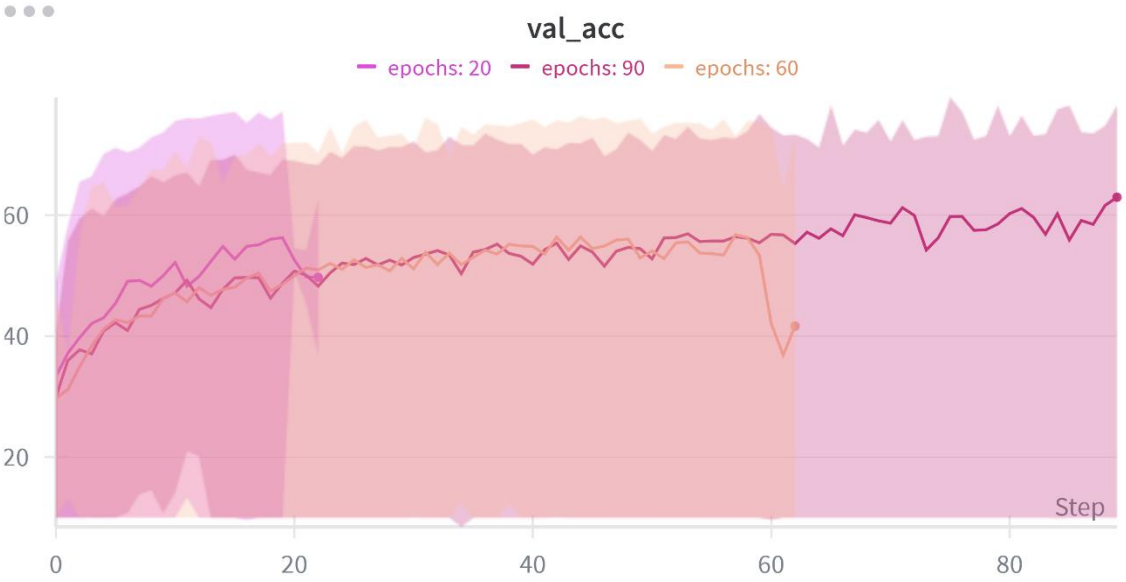
- Observations from the Plot:
    i. Adam
        1. Achieved smooth, stable convergence. (70-80%).
        2. The adaptive learning rate mechanism helps balance speed and stability, allowing faster optimization compared to vanilla SGD.
        3. Combines the ideas of momentum and RMSprop. Keeps track of both the average gradient (momentum) and the average of squared gradients.
        4. Automatically adjusts learning rates for each parameter.
        5. Fast convergence, less sensitive to hyperparameter tuning.
    ii. SGD
        1. Reached moderate final accuracy (~60–63%) with slower but steady improvement.
        2. Training curve is smooth and stable but converges gradually.
        3. As expected, requires careful tuning of learning rate and momentum for best results.
        4. Updates weights using the gradient of the loss function with respect to parameters on a small random batch (mini-batch) of data.
        5. Uses a fixed learning rate.
        6. Simple and reliable but may converge slowly and get stuck in local minima.
    iii. RMSprop
        1. Showed good early convergence but suffered from high oscillations and instability across epochs.
        2. Final accuracy (~60–65%) fluctuates significantly, indicating sensitivity to learning rate and batch normalization.
        3. Uses an exponentially decaying average of past squared gradients to adjust the learning rate for each parameter.
        4. Adaptive learning rate per parameter.
        5. Learns faster than SGD; more responsive but less stable.
    iv. Nadam
        1. **Achieved the highest validation accuracy (~80%).**
        2. Nadam combines Adam's adaptivity with Nesterov momentum.
        3. Extends Adam by incorporating Nesterov momentum, which looks ahead before updating the parameters, giving a more accurate adjustment direction.
        4. Often provides slightly faster convergence than Adam.

f. Vary the batch size, number of epochs, and learning rate. Explain how the convergence speed and performance vary with these changes. (10)

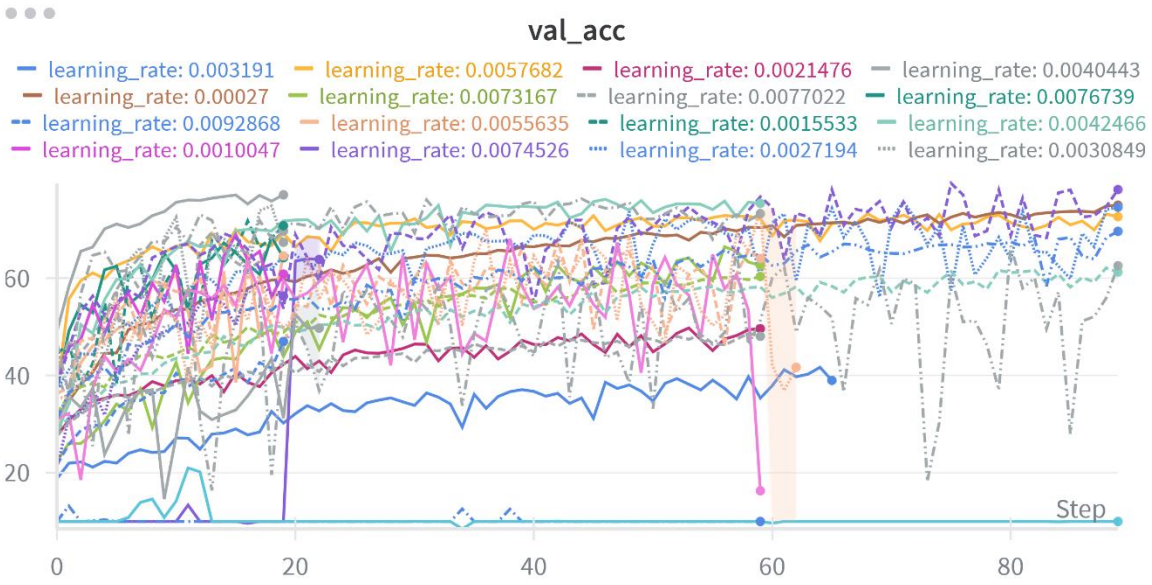
- **Varying Batch Size:**



-
- **Varying Epochs:**



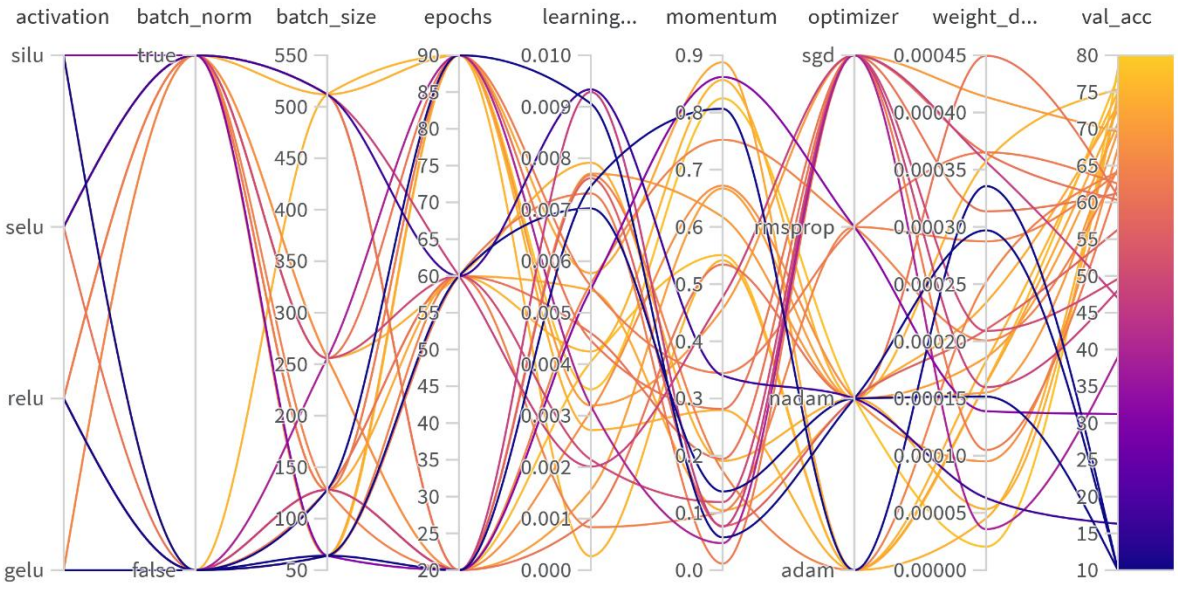
- **Varying Learning Rate:**



-

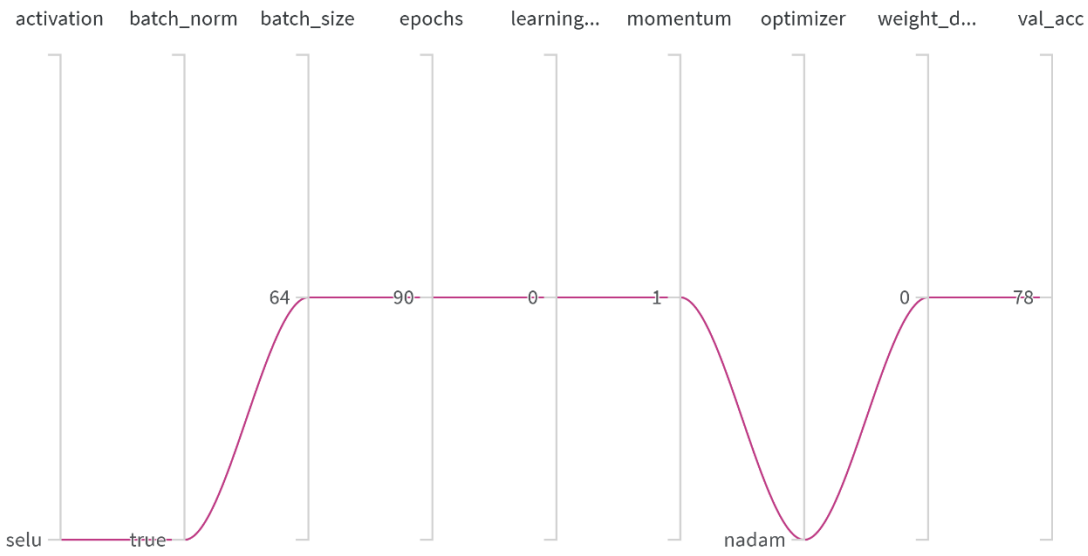
Question 3. Plots (10 points)

a. **Provide the W&B parallel-coordinate plot that shows which configuration achieves what accuracy (a sample plot is given below).**



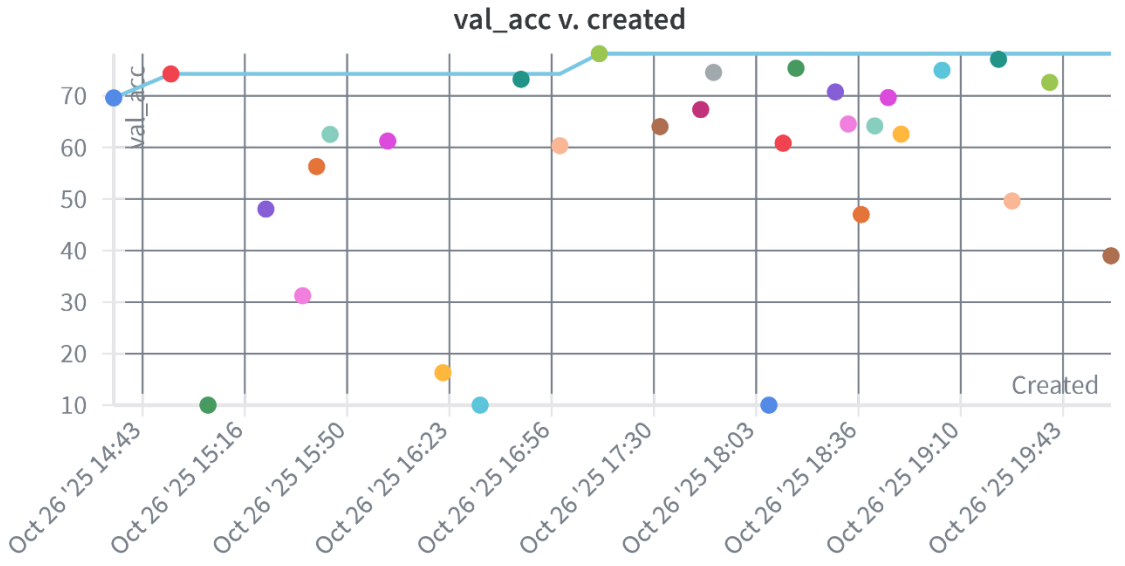
b.

c. W&B plot for the best configuration:



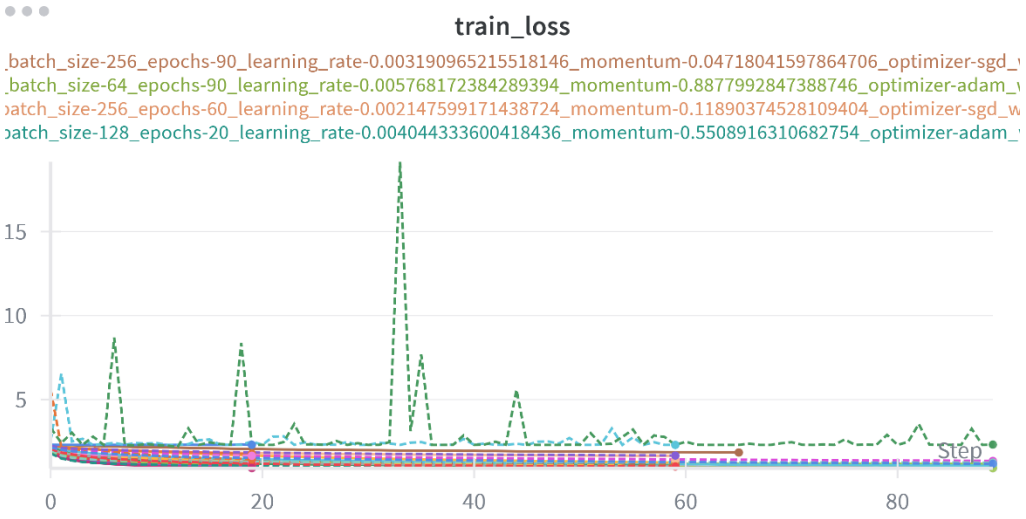
d.

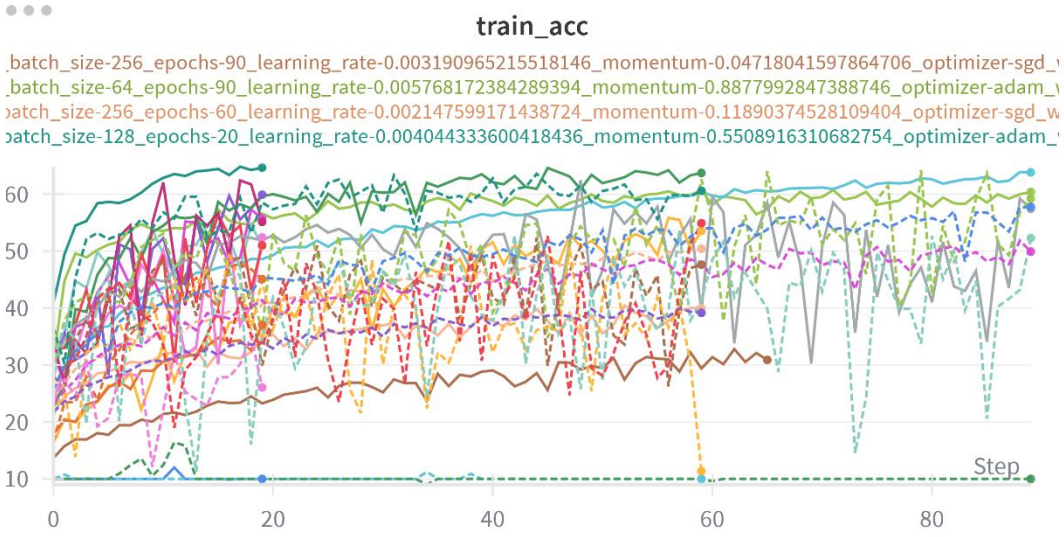
e. Provide the validation accuracy vs. step (scatter) plot.

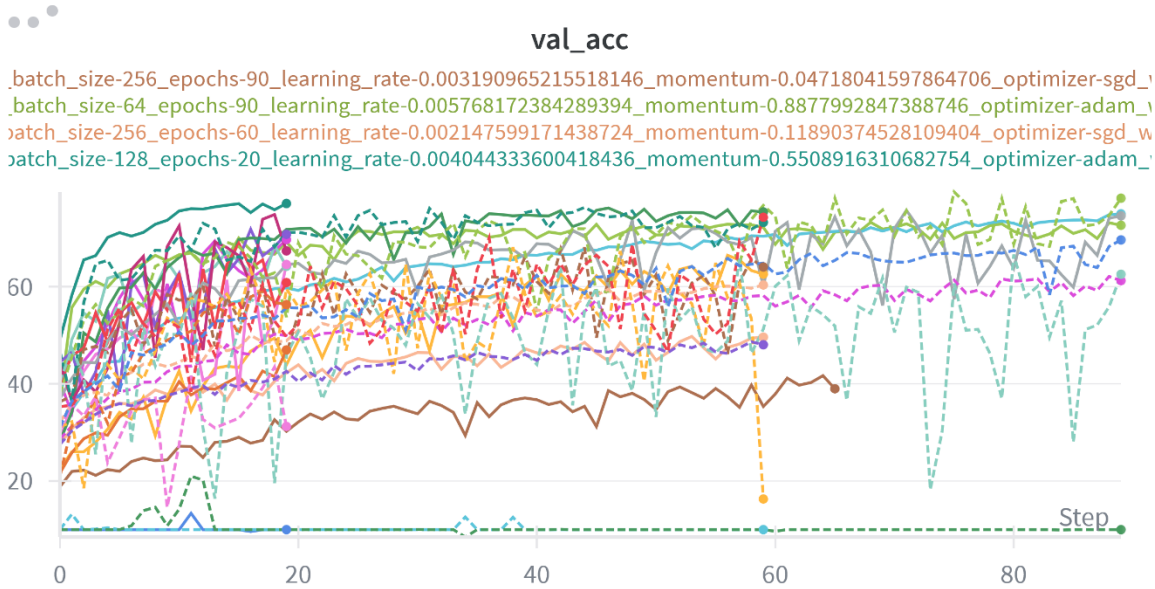


f.

g. Provide the plots for training loss, training accuracy, validation loss, and validation accuracy respectively. (Sample plots are shown below. Make sure you generate these automatically using W&B.)


train_loss
_batch_size-256_epochs-90_learning_rate-0.003190965215518146_momentum-0.04718041597864706_optimizer-sgd_
_batch_size-64_epochs-90_learning_rate-0.005768172384289394_momentum-0.8877992847388746_optimizer-adam_
batch_size-256_epochs-60_learning_rate-0.002147599171438724_momentum-0.11890374528109404_optimizer-sgd_w
batch_size-128_epochs-20_learning_rate-0.004044333600418436_momentum-0.5508916310682754_optimizer-adam_
15
10
5
Step
0
20
40
60
80


train_acc
_batch_size-256_epochs-90_learning_rate-0.003190965215518146_momentum-0.04718041597864706_optimizer-sgd_
_batch_size-64_epochs-90_learning_rate-0.005768172384289394_momentum-0.8877992847388746_optimizer-adam_
batch_size-256_epochs-60_learning_rate-0.002147599171438724_momentum-0.11890374528109404_optimizer-sgd_w
batch_size-128_epochs-20_learning_rate-0.004044333600418436_momentum-0.5508916310682754_optimizer-adam_
60
50
40
30
20
10
Step
0
20
40
60
80


val_acc
_batch_size-256_epochs-90_learning_rate-0.003190965215518146_momentum-0.04718041597864706_optimizer-sgd_
_batch_size-64_epochs-90_learning_rate-0.005768172384289394_momentum-0.8877992847388746_optimizer-adam_
batch_size-256_epochs-60_learning_rate-0.002147599171438724_momentum-0.11890374528109404_optimizer-sgd_w
batch_size-128_epochs-20_learning_rate-0.004044333600418436_momentum-0.5508916310682754_optimizer-adam_
60
40
20
Step
0
20
40
60
80

Question 4. Final Model Performance (10 points)

Based on the W&B parallel plot, provide the configuration that achieved the best validation accuracy. Do not list multiple configurations. The configuration mentioned here will be verified by re-running your model. Ensure that this configuration indeed reproduces the reported best accuracy.

- The configuration which gives the best validation accuracy as per the training results is with below hyperparameters:
    - activation=selu
    - batch_norm=True
    - batch_size=64
    - epochs=90
    - learning_rate=0.00351147544245589
    - momentum=0.8247395191150262
    - optimizer=nadam
    - weight_decay=2.0531225329636637e-05
- The best validation accuracy achieved is 78.18%.
- The best network architecture is:

VGG(

(features): Sequential(

(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(2): SELU(inplace=True)

(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(5): SELU(inplace=True)

(6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(9): SELU(inplace=True)

(10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(12): SELU(inplace=True)

(13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

)

(classifier): Sequential(

(0): Linear(in_features=128, out_features=10, bias=True)

)

)

Question 5. Reproducibility and Repository (10 points)

Github repo link: **https://github.com/neeraj1397/cs6886w_assignment1_ee23m522.git**