

# Deep Learning Report

## **Lab Assignment - 8**

---

Ayush Abrol

B20AI052

## Question 01

### Aim:

Train a ResNet18 model for classification on even classes of CIFAR-10 (B20AI052 - even roll no) for 30 epochs. Analyze the training of models using the wandb library and “weights & biases” tool covered in the class.

- Show comparison of the performance of the hyperparameter tuning using plots using the “weights & biases” tool. Hyperparameters: choice of activation function, optimizer
- Write the observations about GPU (like GPU utilization, temperature etc).
- Show all the logs about accuracy, loss using plots using “weights & biases” tool
- Generate the report of the entire experiment using the “weights & biases” tool. Attach this report to your assignment report.
- Add a visual result analysis on the “weights & biases” tool taught by Dr Anush, where he shows the prediction for different images. Include this in the assignment video in addition to the other demonstrations.

**Note:** The comparison should be done at least between 3-4 models (resulting from the choice of hyperparameters)

### Wandb server where the project is deployed:


[https://wandb.ai/ayushabrol13/B20AI052\\_Lab\\_Assignment\\_8](https://wandb.ai/ayushabrol13/B20AI052_Lab_Assignment_8)

### Procedure:

- Imported all the necessary libraries and modules such as torch (nn, optim, utils), torchvision(transforms, datasets, Dataloader, Subset, models), numpy, pandas, tqdm, random, math and wandb
- Set up the device to GPU (cuda).
- Logged in to wandb using the API key for my “Weights and Biases” website credentials.
- Created a hyperparameters dictionary as:

```
hyperparams = {
    "num_epochs": 30,
    "batch_size": 128,
    "learning_rate": 3e-4,
    "activation": ["Relu", "Leaky_relu", "Sigmoid", "Tanh"],
    "optimizer": ["Adam", "SGD", "RMSprop", "Adagrad"],
}
```

- Created a transform for CIFAR10 dataset where I augmented the data using RandomHorizontalFlip, RandomCrop and then converted the image to a tensor and then normalized it using mean = (0.5, 0.5, 0.5) and std = (0.5, 0.5, 0.5).
- Selected the even classes and created an even subclasses Subset of the training and testing data loader.
- Then, I created a function named log\_image\_table() that logs a wandb (Weights and Biases) Table with image predictions, labels, and scores.
  - It takes four input arguments: images, predicted, labels, and probs.
    - images is a tensor of images
    - predicted is a tensor of predicted classes for each image
    - labels is a tensor of actual classes for each image
    - probs is a tensor of predicted probabilities for each class
  - The function creates a wandb Table with columns named "image", "pred", "target", and "score\_i" (where i is an even number between 0 and 9). It then iterates through the input tensors, converts them to CPU, and adds each image, prediction, label, and score to the wandb Table using the add\_data() method.
  - Finally, the function logs the wandb Table to the "predictions\_table" key using the wandb.log() method. The commit=False argument tells WandB not to commit the data to the server immediately.
- Then I implemented the ResNet18 architecture as a subclass of the nn.Module class. The ResNet18 model is pre-trained on ImageNet and modified for a specific classification task by replacing the fully connected layer with a new linear layer.
  - The class takes one argument activation, which specifies the type of activation function to be used.
  - The \_\_init\_\_ method initializes the ResNet18 model and replaces the last fully connected layer with a new linear layer that maps the output to 10

- 
- classes. It also sets the activation function based on the input argument and moves the model to the specified device.
- The forward method defines how input data is processed through the model. The input tensor `x` is passed through the ResNet18 model, and the output is passed through an activation function specified by the activation argument. The output tensor is returned.
  - The available activation functions are `relu`, `leaky_relu`, `sigmoid`, and `tanh`.
- 
- Then, I created an evaluate function which computes performance of the model on the validation dataset and log a wandb.Table for the same.
- 
- Then, finally I created a loop for a ResNet18 model. It trains the model with different hyperparameters and logs the results using the Weights & Biases (wandb) platform.
  - In each iteration, it randomly selects an activation function and optimizer from a predefined set of hyperparameters. Then, it initializes a wandb experiment with the selected hyperparameters, creates a ResNet18 model with the selected activation function, and initializes a criterion (cross-entropy loss) and optimizer (Adam, SGD, RMSprop, or Adagrad) based on the selected hyperparameters.
  - The model is trained for a specified number of epochs. In each epoch, it trains on the training dataset, calculates the loss and accuracy, and logs the results to wandb. At the end of each epoch, it evaluates the model on the validation dataset, calculates the loss and accuracy, and logs the results to wandb.
  - Finally, it logs the training loss and finishes the wandb experiment.
  - Experiment is repeated 6 times for different optimizers and activation functions and all the results were logged in the wandb server.



## Results:

- All the results obtained from the “Weights and Biases” tool are attached in the report below which is generated by the tool itself.

Link:

[https://drive.google.com/file/d/17Gij6lad188BlsI8Jzr9K6m5orhH8m0F/view?usp=share\\_link](https://drive.google.com/file/d/17Gij6lad188BlsI8Jzr9K6m5orhH8m0F/view?usp=share_link)

This link contains:

- Visual result analysis on the “weights & biases” which shows the prediction for different images
- Comparison of the performance of the hyperparameter tuning using plots using the “weights & biases” tool for all the 6 different models.
  - Training
  - Testing
- Observations about GPU (like GPU utilization, temperature etc).

- Other than that, all the logs about accuracy, loss using plots using “weights & biases” tool are as follows:

```

ayushabrol13 > Projects > B20AI052_Lab_Assignment_8 > Runs >
Epoch: 1, Train Loss: 1.6725033868332297, Train Accuracy: 0.3586
Epoch: 1, Val Loss: 1.6193197965621948, Val Accuracy: 0.3874
Epoch: 2, Train Loss: 1.589922431172157, Train Accuracy: 0.40564
Epoch: 2, Val Loss: 1.5945470333099365, Val Accuracy: 0.403
Epoch: 3, Train Loss: 1.573796925496082, Train Accuracy: 0.41534
Epoch: 3, Val Loss: 1.5839979648590088, Val Accuracy: 0.4092
Epoch: 4, Train Loss: 1.565506776984857, Train Accuracy: 0.4227
Epoch: 4, Val Loss: 1.5639013051986694, Val Accuracy: 0.4223
Epoch: 5, Train Loss: 1.5626475476488775, Train Accuracy: 0.42294
Epoch: 5, Val Loss: 1.567138671875, Val Accuracy: 0.4192
Epoch: 6, Train Loss: 1.5568611901633593, Train Accuracy: 0.42924
Epoch: 6, Val Loss: 1.5613850355148315, Val Accuracy: 0.4237
Epoch: 7, Train Loss: 1.5526475346818263, Train Accuracy: 0.43272
Epoch: 7, Val Loss: 1.5727458000183105, Val Accuracy: 0.4208
Epoch: 8, Train Loss: 1.5484499529916413, Train Accuracy: 0.43498
Epoch: 8, Val Loss: 1.561082363128662, Val Accuracy: 0.427
Epoch: 9, Train Loss: 1.5468670519030825, Train Accuracy: 0.43602
Epoch: 9, Val Loss: 1.5525298118591309, Val Accuracy: 0.4326
Epoch: 10, Train Loss: 1.5444661956660601, Train Accuracy: 0.43672
Epoch: 10, Val Loss: 1.5553332567214966, Val Accuracy: 0.4312
Epoch: 11, Train Loss: 1.545638375136317, Train Accuracy: 0.4366
Epoch: 11, Val Loss: 1.5537842512130737, Val Accuracy: 0.4261
Epoch: 12, Train Loss: 1.540599759744138, Train Accuracy: 0.44036
Epoch: 12, Val Loss: 1.549453616142273, Val Accuracy: 0.4338
Epoch: 13, Train Loss: 1.5385694047626184, Train Accuracy: 0.44306
Epoch: 13, Val Loss: 1.5486810207366943, Val Accuracy: 0.4325
Epoch: 14, Train Loss: 1.5361365104208187, Train Accuracy: 0.44358
Epoch: 14, Val Loss: 1.545793890953064, Val Accuracy: 0.4362
Epoch: 15, Train Loss: 1.5357848290277987, Train Accuracy: 0.4432
Epoch: 15, Val Loss: 1.545941948890686, Val Accuracy: 0.4368
Epoch: 16, Train Loss: 1.5336924517641262, Train Accuracy: 0.4456
Epoch: 16, Val Loss: 1.5513206720352173, Val Accuracy: 0.4358
Epoch: 17, Train Loss: 1.5330767284850686, Train Accuracy: 0.44644
Epoch: 17, Val Loss: 1.546811819076538, Val Accuracy: 0.4363
Epoch: 18, Train Loss: 1.5321478843688965, Train Accuracy: 0.44588
Epoch: 18, Val Loss: 1.542970061302185, Val Accuracy: 0.4406
Epoch: 19, Train Loss: 1.5316213059182069, Train Accuracy: 0.44706
Epoch: 19, Val Loss: 1.551324486732483, Val Accuracy: 0.4371
Epoch: 20, Train Loss: 1.5310830473899841, Train Accuracy: 0.44738
Epoch: 20, Val Loss: 1.5407224893569946, Val Accuracy: 0.4387
Epoch: 21, Train Loss: 1.530449248698293, Train Accuracy: 0.44812
Epoch: 21, Val Loss: 1.552518367767334, Val Accuracy: 0.4294
Epoch: 22, Train Loss: 1.5261838874038385, Train Accuracy: 0.44966
Epoch: 22, Val Loss: 1.546096682548523, Val Accuracy: 0.4373
Epoch: 23, Train Loss: 1.5305154828392729, Train Accuracy: 0.44772
Epoch: 23, Val Loss: 1.5408629179000854, Val Accuracy: 0.4408
Epoch: 24, Train Loss: 1.5244083185585178, Train Accuracy: 0.45134
Epoch: 24, Val Loss: 1.5399190187454224, Val Accuracy: 0.438
Epoch: 25, Train Loss: 1.525113537603495, Train Accuracy: 0.45156
Epoch: 25, Val Loss: 1.5413883924484253, Val Accuracy: 0.4449
Epoch: 26, Train Loss: 1.5246072892023592, Train Accuracy: 0.45104
Epoch: 26, Val Loss: 1.5464837551116943, Val Accuracy: 0.4387
Epoch: 27, Train Loss: 1.5234519985257362, Train Accuracy: 0.45426
Epoch: 27, Val Loss: 1.5454425811767578, Val Accuracy: 0.4381

```

These are the logs for one out of the 6 models, all the similar such logs can be found on the server link attached at the top of this report.