The goal of this assignment is to get hands-on experience with deep learning. Starting from a baseline architecture we provided, you will design an improved deep net architecture to classify (small) images of 100 categories. You will evaluate the performance of your architecture by uploading your predictions to this Kaggle competition and submit your code and report describing your implementation choices to Compass2g. This assignment must be done in your project groups and the due date is Wednesday, May 2, 11:59:59PM.

Dataset

The `CIFAR100` dataset consists of 60000 32x32 color images from 100 classes, with 600 images per class. There are 50000 training images and 10000 test images. The images in `CIFAR100` are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size.
We have modified the standard dataset to create the `CIFAR100_CS543` dataset which consists of 45000 training images (450 of each class), 5000 validation images (50 of each class), and 10000 test images (100 of each class). The train and val datasets have labels while all the labels in the test set are set to 0. You can evaluate your model on validation set as labels are provided. When you have an improved model, you could obtain your performance on the test set by uploading a CSV file to this Kaggle competition. Note that individual teams are limited to 5 submissions a day, so try to tune your model before uploading CSV files. Also, you must make at least one submission for your final system for full credit. The best performance will be considered.

Deep Learning Framework - PyTorch

We encourage the use of PyTorch for this assignment, as it is very easy to pick up. It has a lot of tutorials and an active community answering questions on its discussion forums. This assignment has been adapted from a PyTorch tutorial on the CIFAR-10 dataset.

You will be using the following neural network layers:

- Convolutional, i.e. `nn.Conv2d`
- Pooling, e.g. `nn.MaxPool2d`
- Fully-connected (linear), i.e. `nn.Linear`
- Non-linear activations, e.g. `nn.ReLU`
- Normalization, e.g. `nn.batchnorm2d`

We created a BaseNet that you can run and get a baseline accuracy. The architecture of the network is defined below:

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Output Channels (for conv layers)
1	conv2d	5	32 | 28	3 | 6
2	relu	-	28 | 28	-
3	maxpool2d	2	28 | 14	-
4	conv2d	5	14 | 10	6 | 16
5	relu	-	10 | 10	-
6	maxpool2d	2	5 | 5	-
7	linear	-	400 | 50	-
8	relu	-	50 | 50	-
9	linear	-	50 | 100	-

It consists of two convolutional modules (`conv-relu-maxpool`) and two linear layers. Try understanding how each entry of this table is related to the provided code. Your goal is to edit this class or make new classes for devising better deep net architectures. This tutorial by PyTorch is helpful in gearing up on using deep nets. Also this lecture on CNN by Andrej Karpathy is a good resource for anyone starting with deep nets. It talks about architectural choices, output dimension of conv layers based on layer parameters, and regularization methods. All of these can help improve your network design in this assignment. You would have to report a table similar to the one linked above to illustrate your improved network.

Steps to run the code

Refer to the BW guide for bluewaters related resources.

1. SSH into bluewaters
2. Mount your BW repo on your local machine using `sshfs`
3. Copy both `.py` files in Data section of Kaggle competition to your BW repo

Every time you want to run your code:

1. Run an interactive session
2. Move from login node of bluewaters to a compute node using ccm
3. Activate torchvision virtual environment we have set for you
4. Run code using `python cifar100_cs543.py` or using an interactive python environment like `ipython`or `jupyter-notebook`

Before you design your own architecture, you should start by getting familiar with the BaseNet architecture already provided, the meaning of hyper-parameters and the function of each layer. Please go through the above CNN lecture and read the comments in the code provided. This lecture for learning rates and preventing overfitting is a good additional read.

Improve model

The BaseNet architecture provided gives a classification accuracy of around ~23% on the test set. Your goal is to create an improved deep net by making judicious architecture and implementation choices, as outlined below. A good combination of choices can get your accuracy close to 50%. A fair submission with more than 40% accuracy will be given full credit.

Todo #1> Data Normalization Normalizing input data makes training easier and more robust. Similar to normalized epipolar geometry estimation, data in this case too could be made zero mean and fixed standard deviation (sigma=1 is the to-go choice). Use `transforms.Normalize()` with the right parameters to make the data well conditioned (zero mean, std dev=1) for improved training. After your edits, make sure that test_transform has the same data normalization parameters as train_transform.

Todo #2> Data augmentation Try using `transforms.RandomCrop()` and/or `transforms.RandomHorizontalFlip()` to augment training data. You shouldn't have any data augmentation in test_transform (val or test data is never augmented). If you need a better understanding, try reading through PyTorch tutorial for transforms.

Todo #3> Deeper network Following the guidelines laid out by this lecture on CNN, experiment by adding more convolutional and fully connected layers. Add more conv layers with increasing output channels and also add more linear (fc) layers. Do not have a maxpool layer after every conv layer in your deeper network as it leads to too much loss of information.

Todo #4> Normalization layers Normalization layers help reduce overfitting of the model to your data. Pytorch's normalization layers are an easy way of incorporating them in your model. Add normalization layers after conv layers (`nn.BatchNorm2d`). Add normalization layers after linear and experiment inserting them before or after ReLU layer (`nn.BatchNorm1d`).

Todo #5> train-val-test set based early stopping. After how many epochs to stop training? This answeron stackexhange is a good summary of using train-val-test splits to reduce overfitting. This blog is a good reference for early stopping. Always remember, you should never use the test-set in anything but the final evaluation. Seeing the train loss and validation accuracy plot, decide for how many epochs to train your model. Not too many (as that leads to overfitting) and not too few (else your model hasn't learnt enough).

Finally, there are a lot of approaches to improve a model beyond what we stated above. Feel free to try out your own ideas, or interesting ML/CV approaches you read about. Since there are limited computational resources for each student, we encourage to rationally limit training time and model size.

Submission checklist

One member of your team must submit the following before the deadline:

1> CSV file of your predicted test labels on Kaggle. Your performance will not directly affect your grade unless it is below our threshold of 40% or unless you are the winning team (the winner will get extra credit).

2> The edited `cifar100_cs543.py` on compass 2g. This should run and create a CSV file which is approximately similar to the one above (stochastic nature of gradient descent could lead to difference in accuracy, this is fine).

3> PDF report on compass2g. The report should include the following:

- Team name (should match what you list on kaggle)
- Best accuracy (should match your accuracy on kaggle)
- Factors which helped improve your model performance. Explain each factor in 2-3 lines.

- Table defining your final architecture. An example table has been inserted for reference in the report template.
- Final architecture's plot for training loss and validation accuracy. This would have been auto-generated by `cifar100_cs543.py`.
- Ablation study to validate the above choices, i.e., a comparison of performance for two variants of a model, one with and one without a certain feature or implementation choice.
- [New relaxed EC opportunitites] If you attempt extra credit (EC), see @377 and @390 on piazza. If your main report model is different from the model used for EC, you should include the code for this model too. Details are on the piazza post and also the report template.
  For more details, please refer to the following template.

Tips for this assignment

- Do not lift existing code or torchvision models.
- All edits to BaseNet which lead to a significant accuracy improvement must be listed in the report. Try also to support this explanation by adding an entry in the ablation study. We understand running experiments for each change might get cumbersome if you are trying a lot of things. So use your judgment to decide what are the most important factors to getting improved performance and documenting how you evolved your model.
- Do not run experiments for very high numbers of epochs. Test for a single epoch to know everything runs fine. ~15 epochs was enough for the model which obtained 50% accuracy

Acknowledgements

Developed by Unnat Jain. We thank the BW support staff for their help in organizing this assignment.