PyTorch Seminar

Day2. CIFAR10 dataset training (2)

Duhyeon Kim / May 2025

PyTorch

PyTorch Seminar

Day2. CIFAR10 dataset training

Duhyeon Kim / May 2025

PyTorch

Contents

Things needed to Train CIFAR10

A. Neural Network Composition

B. CIFAR10 dataset

C. Custom Dataloader (role of transform function)

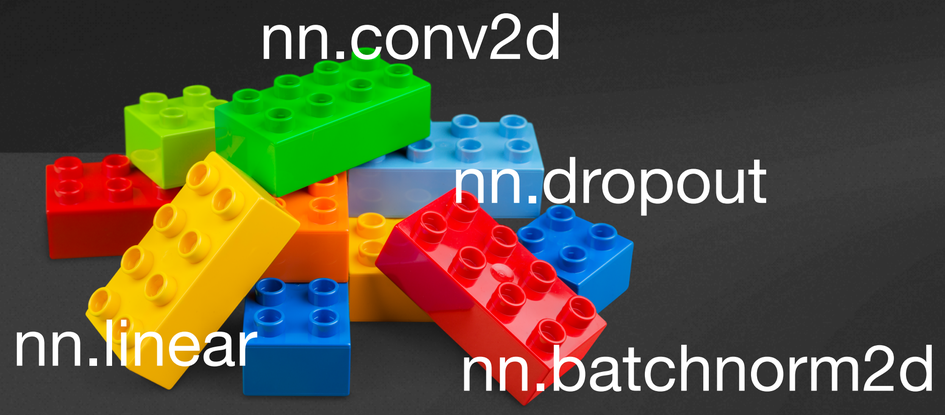
D. Custom Model (what is import ~ ?)

PyTorch

PyTorch nn composition

nn.Module

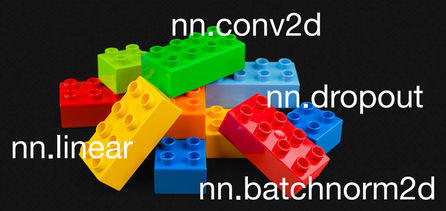
```
import torch.nn as nn
```



```
class CustomModel(nn.Module):
```

PyTorch nn composition

nn.Module

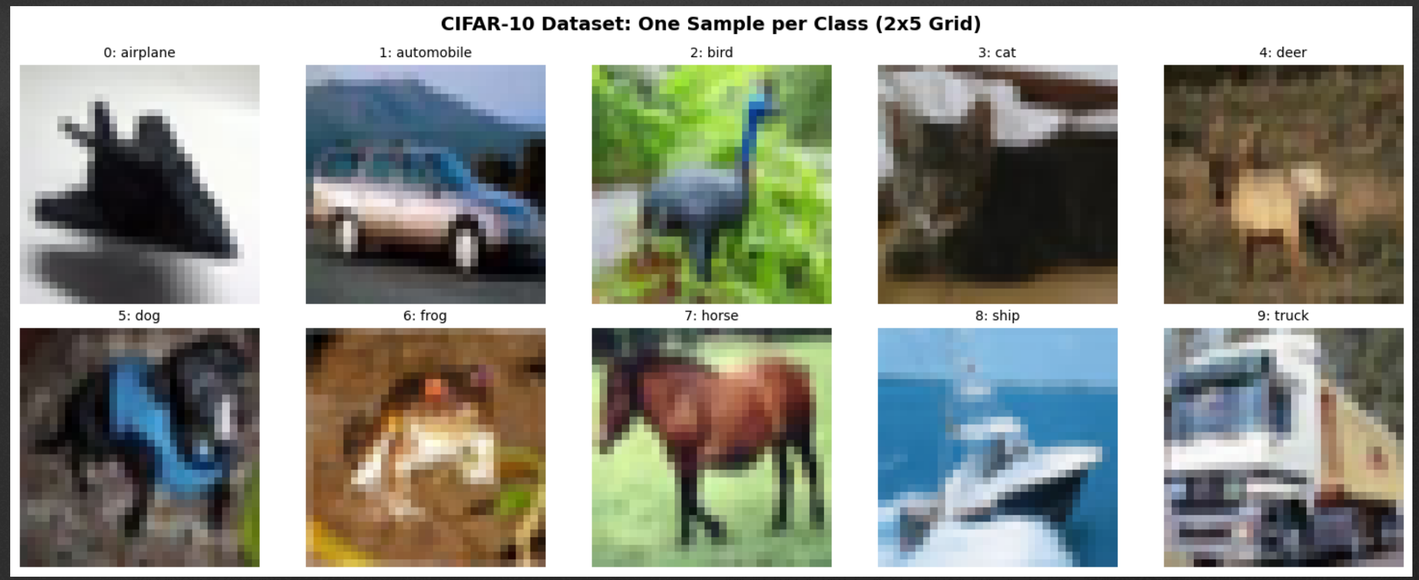


https://colab.research.google.com/drive/16tkCZEnTU0bhtTvB0vBppk6TrO7IC47k?usp=sharing

CIFAR-10 dataset

Overview

- 10 classes
  {'airplane': 0,’automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4, 'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}
- 32x32 RGB images
- Train 50000 + Test 10000

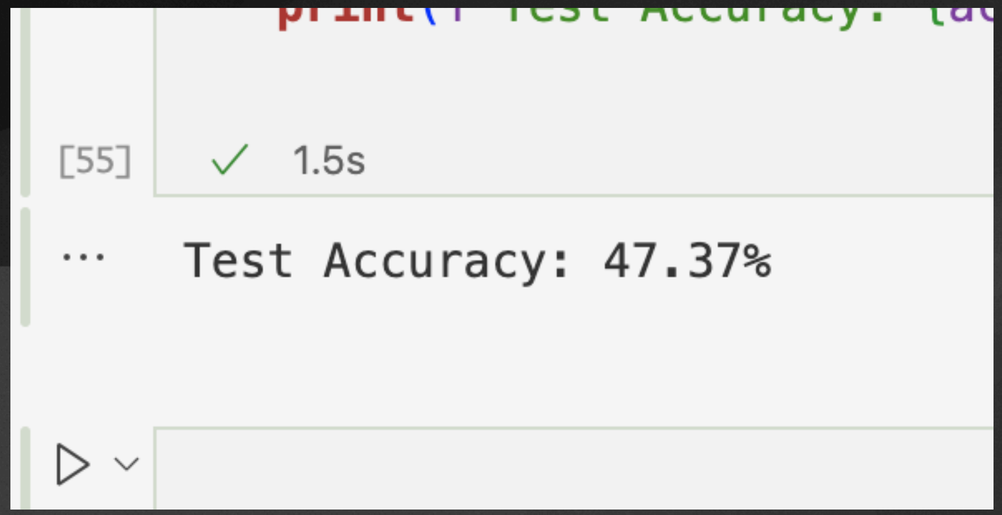


https://www.cs.toronto.edu/~kriz/cifar.html

Simplest CIFAR-10 Training

Simplest CIFAR-10 Training

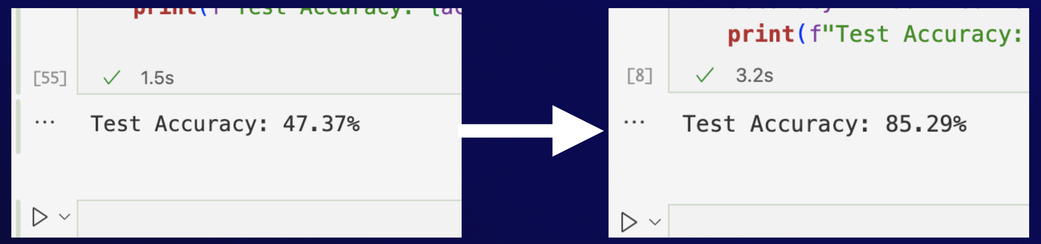
Evaluation Accuracy



47% > 10% (random sampling)

How to further **improve** Model Performance?

Model Improvement Techniques



Model Improvement Techniques

Roads to 100% accur.

A. Deeper Model

B. Data Augmentation (torchvision.transform)

C. Optimizer

D. When it comes to LLM (Deepseek)

Deeper Model

AlexNet (Krizhevsky et al., 2012)

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

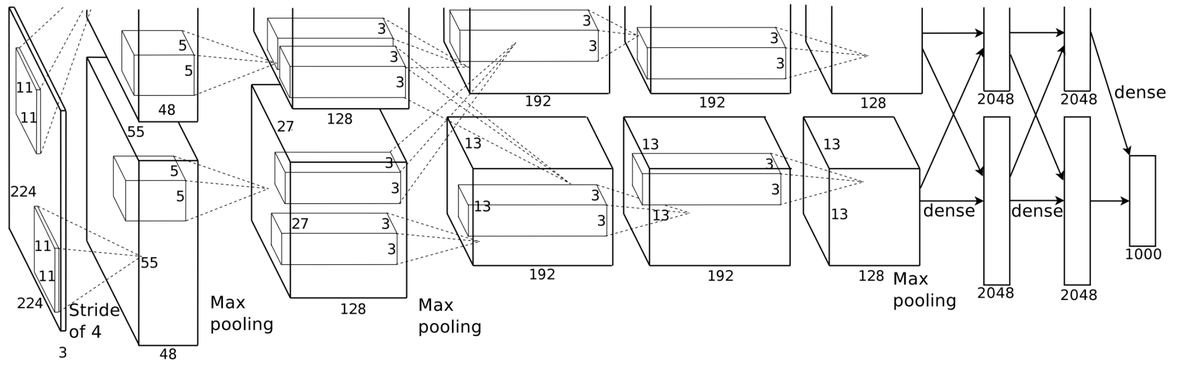
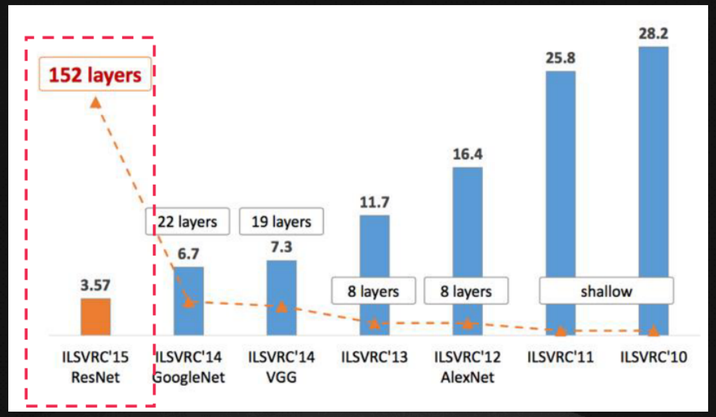


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Deeper Model

AlexNet (Krizhevsky et al., 2012)



Key Milestones

- **DEEP** Convolutional Network
  - 5 (conv) + 3 (FC)
- ReLU activation
  - Training Speed up (6x)
  - Gradient Vanishing solved
- GPU in practice
  - DEEP -> GPU

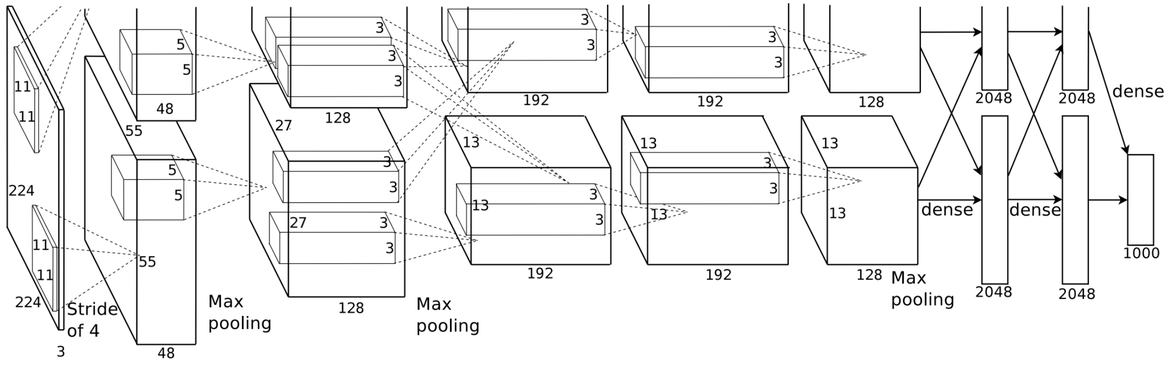


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Data Augmentation

torchvision.transforms

Meaning
& Effect of
Augmenting Data

- increase dataset size and diversity
- Overfitting Prevention
  - sees diff view of the same image
- Data Efficiency
  - one datum -> several data

Performance Boost

https://docs.pytorch.org/vision/0.9/transforms.html

Data Augmentation

torchvision.transforms

TORCHVISION.TRANSFORMS

Transforms are common image transformations. They can be chained together using `Compose`. Additionally, there is the `torchvision.transforms.functional` module. Functional transforms give fine-grained control over the transformations. This is useful if you have to build a more complex transformation pipeline (e.g. in the case of segmentation tasks).

All transformations accept PIL Image, Tensor Image or batch of Tensor Images as input. Tensor Image is a tensor with `(C, H, W)` shape, where `C` is a number of channels, `H` and `W` are image height and width. Batch of Tensor Images is a tensor of `(B, C, H, W)` shape, where `B` is a number of images in the batch. Deterministic or random transformations applied on the batch of Tensor Images identically transform all the images of the batch.

```python
# MODIFIED: Enhanced data augmentation techniques and added normalization
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(32, padding=4),  # MODIFIED: Added random crop
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])
```

Transforms on torch.*Tensor only

PIL image

transforms.Compose

RandomHorizontalFlip

RandomCrop

ToTensor

Normalize

Tensor image

Transform is used in dataset __getitem__

https://docs.pytorch.org/vision/0.9/transforms.html

Optimizer

SGD to Adam

Stochastic Gradient Descent

$$\theta_t + 1 = \theta_t - \eta \nabla L(\theta t; x, y)$$

or

$$v_{t+1} = \beta v_t + \eta \nabla L(\theta_t; x, y)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

Adaptive Moment Estimation

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_{t-1})$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(\theta_{t-1}))^2$$

Momentum
Velocity

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_t = \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Carefully and meaningfully update Model Parameter

Custom Model

Computer Vision

- Residual Network https://arxiv.org/abs/1512.03385
- 3x3 kernel https://arxiv.org/abs/1409.1556
- Generator and Discriminator https://arxiv.org/abs/1406.2661
- VAE https://arxiv.org/abs/1312.6114
- Feature Fushion https://openaccess.thecvf.com/content/WACV2021/papers/Dai_Attentional_Feature_Fusion_WACV_2021_paper.pdf

Custom Model

LLM (transformers)

- Multi-Head Attention
- Multi-Head latent Attention
- Multi-Query Attention
- Group-Query Attention

Memory + Inference Speed

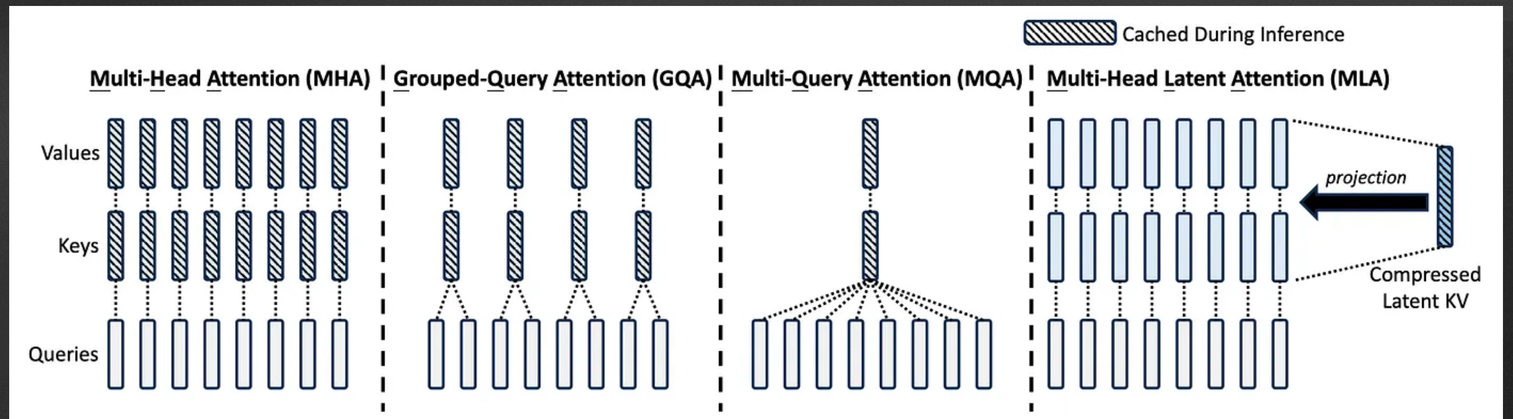


Figure 3 | Simplified illustration of Multi-Head Attention (MHA), Grouped-Query Attention (GQA), Multi-Query Attention (MQA), and Multi-head Latent Attention (MLA). Through jointly compressing the keys and values into a latent vector, MLA significantly reduces the KV cache during inference.

https://arxiv.org/pdf/2405.04434

Model Improvement Techniques in Code

Assignment #1

Your own dataset training

Dataset Search

https://datasetsearch.research.google.com/

https://www.kaggle.com/datasets

https://paperswithcode.com/datasets

kaggle

Assignment #1

Your own dataset training

Submit a Jupyter Notebook(.ipynb) that demonstrates:

✅ **Step 1: Custom Model Declaration**
Define your custom model class with proper architecture initialization and forward pass implementation.

📦 Step 2: Checkpoint Loading
Load the saved model weights from your checkpoint file into the custom model instance.

📊 Step 3: Dataset & Evaluation
Load the specified dataset and perform model evaluation to demonstrate the trained model's performance.

Thank you

Summary

Overall Code Structure

CNN

MLP

1 2 3 4 5 6 7 8 9 0

Predicted

```
{
    'cab': 0.08239,
    'dog': 0.04366,
    'cat': 0.09582,
    'car': 0.22994,
    'bird': 0.03967,
    'plane': 0.03967,
    'boat': 0.24323,
    'tree': 0.10801,
    'horse': 0.03135,
    'bike': 0.08626
}
```

Answer

```
{
    'cab': 1,
    'dog': 0,
    'cat': 0,
    'car': 0,
    'bird': 0,
    'plane': 0,
    'boat': 0,
    'tree': 0,
    'horse': 0,
    'bike': 0
}
```

Dataloader

CustomDataset

batches

Cross Entropy Loss

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

$$y_j = 1, \quad y_{i \neq j} = 0$$

$$L = -\log(\hat{y}_j)$$

SGD optimizer

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta \cdot lr$$