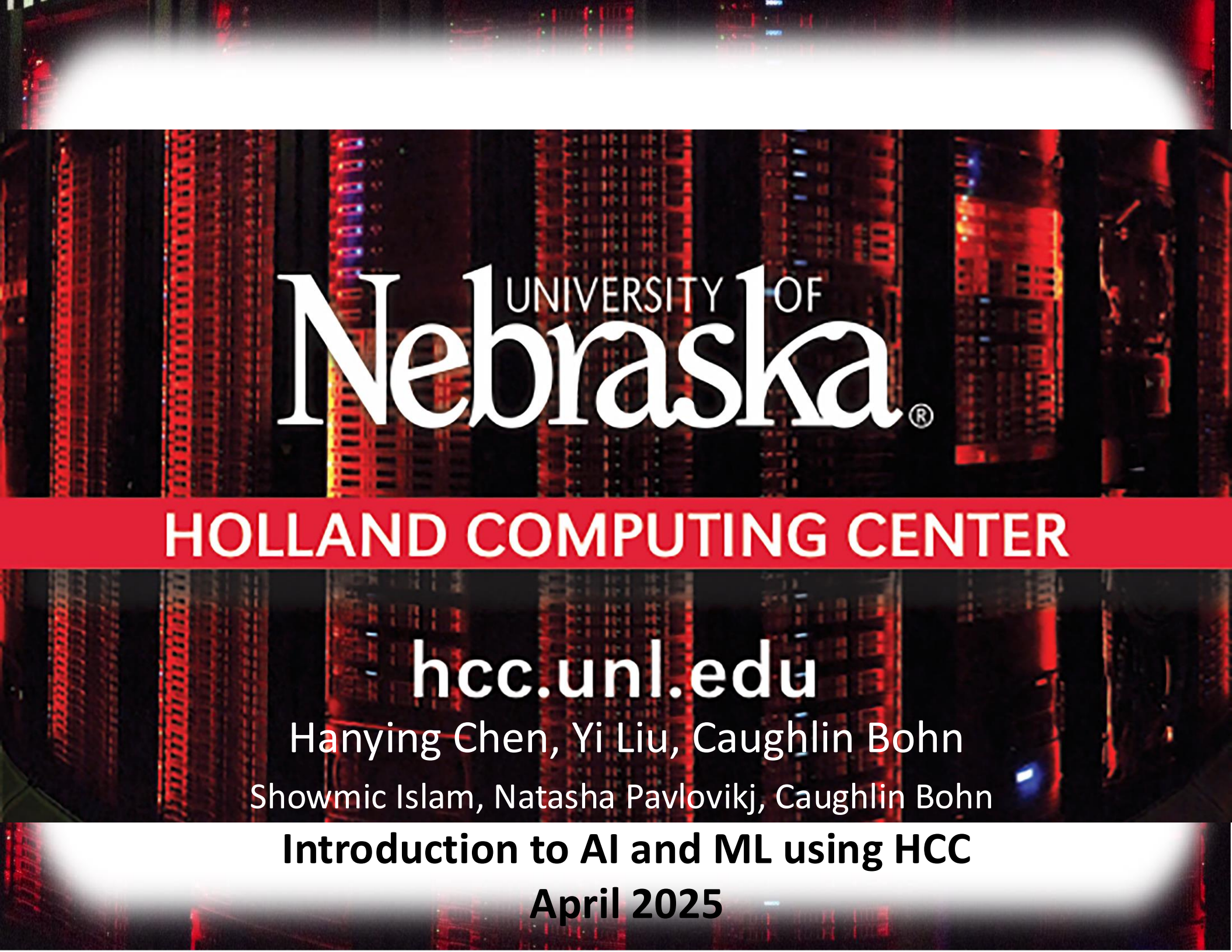


Hanying Chen, Yi Liu, Caughlin Bohn

Showmic Islam, Natasha Pavlovikj, Caughlin Bohn

Introduction to AI and ML using HCC

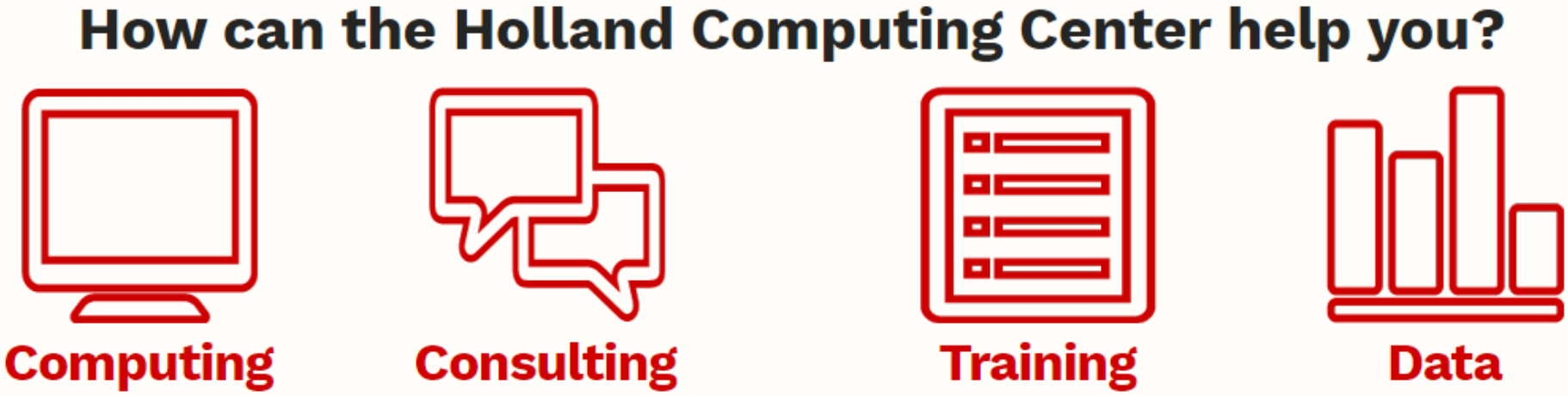
April 2025

Schedule

timestamp	Session
12:00 ~ 12:15pm	Setup and Support
12:15 ~ 1:00pm	**Introduction to HCC resources for ML and AI (45 Min)**
1:00 ~ 1:15pm	Short Break (15 Min)
1:15 ~ 2:00pm	**ML and AI workflows (45 Min)**
2:00 ~ 2:10pm	Short Break (10 Min)
2:10 ~ 2:50pm	**Introduction to PyTorch (40 Min)**
2:50 ~ 3:00pm	Break (10 Min)
3:00 ~ 3:50pm	**Introduction to PyTorch cont. (50 Min)**
3:50 ~ 4:00pm	Break (10 Min)
4:00 ~ 4:15pm	**Introduction to National Research Platform (NRP)**
4:15 ~ 4:30pm	Open Questions

Welcome!

- Hands-on AI practice on HCC
  - Computer Vision – image classification using CNN
  - Natural Language Processing (NLP) – text classification using BERT

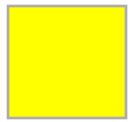


Setup

In this session, we will use Jupyter Lab via Swan's Open OnDemand portal for hands-on practice.

- At the end of the last session, we finished setting up the environment
- Now, please verify whether the data has been successfully downloaded in the "**00_prepare_datasets.ipynb**" notebook
  - You should see output confirming that the datasets are downloaded

If you **are done**, please put up your yellow sticky note.



If you **need help**, please put up your red sticky note.

Takeaways

In this session, you will:

- Practice a **computer vision task**: Classify images using a **Convolutional Neural Network (CNN)**
- Explore the **AI/ML development workflow** from design, data preparation to model training and evaluation
- Learn how to **build and train a CNN using PyTorch** step-by-step
- Hands-on practice **case study on CIFAR-10 image classification**

Recap:
Command Line vs. Open OnDemand

Command line:

- Allow extended running time using SLURM (7 days maximum)
- Better for running experiments

Jupyter Lab in Open OnDemand:

- GUI interface
- Limited running time (8 hours maximum)
- Better for developing code

In this session, we will use **Jupyter Lab**

- Interactive environment ideal for exploring PyTorch models step-by-step
- We can write, run, and visualize the PyTorch code

Development Workflow

Depending on the task, purpose, and data, the workflow has five steps in general:

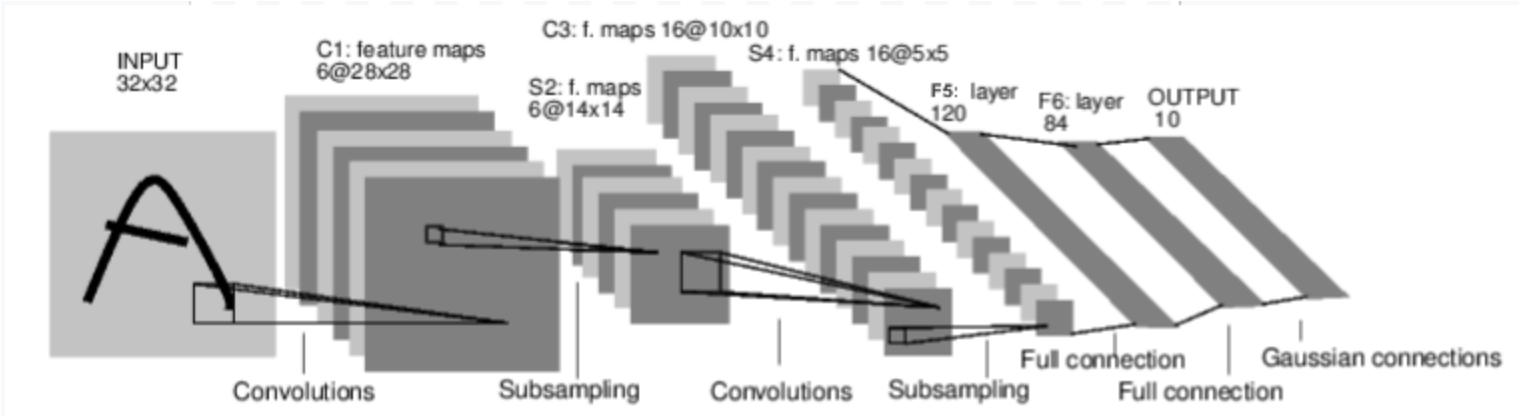
1. Design:
   define objectives of the task
2. Data preparation:
   collect, label, and pre-process data
3. Model selection:
   select AI/ML algorithms based on the task
4. Training
   explore effective hyper-parameters
   train the model on the data
5. Evaluation
   *test the model on **unseen** data to evaluate the model's ability on generalization*

Hands-On Practice 1

Overview:

- Task: Categorize natural scene images
- Data: CIFAR10*

- We will build a convolutional neural network (CNN) in PyTorch to classify images in the CIFAR10 dataset into 10 different classes



https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

* Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

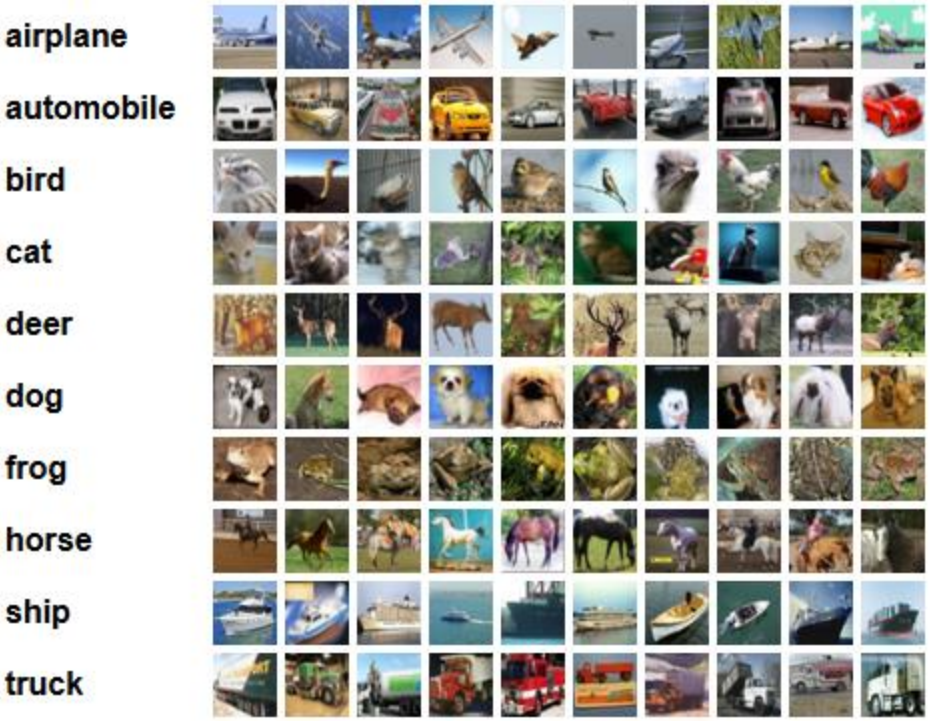
Design

Ways to identify design based on the purpose and task:

- We are looking at an image categorization task
- Models have different advantages at specific task types:
  - Categorization – Convolutional Neural Network (CNN)
  - Segmentation – U-Net
  - Generation – Diffusion model
- Or we can look for the state-of-the-art design in the similar tasks
  - CIFAR10 is commonly used to benchmark CNNs

CIFAR10

Let’s take a look into CIFAR10 in Jupyter Lab



If you **are done**, please put up your yellow sticky note.

If you **need help**, please put up your red sticky note.

Data Preparation

Data preparation can involve different parts, for example:

- **Pre-processing**: Clean data – removing noise, eliminating duplicates, and ensuring proper labeling
- **Data Balancing**: Even class distribution to prevent model bias
- **Data Normalization**: Rescale values
  (e.g., mapping pixel values from 0-255 to a range of -1 to 1)
- **Data Splitting**: Divide the dataset into training and validation sets for model training, and a separate test set for evaluation
- **Data Augmentation**: Enrich dataset variety

Data Preparation

Data preparation can involve different parts, for example:

- **Pre-processing**: Clean data – removing noise, eliminating duplicates, and ensuring proper labeling
- **Data Balancing**: Even class distribution to prevent model bias
- **Data Normalization**: Rescale values
  (e.g., mapping pixel values from 0-255 to a range of -1 to 1)
- **Data Splitting**: Divide the dataset into training and validation sets for model training, and a separate test set for evaluation
- **Data Augmentation**: Enrich dataset variety

Data Preparation

Let’s prepare data and its data loader in Jupyter Lab

```
transform = transforms.Compose([
    transforms.ToTensor(),
    # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

If you **are done**, please put up your yellow sticky note.

If you **need help**, please put up your red sticky note.

Model Selection

General ideas:

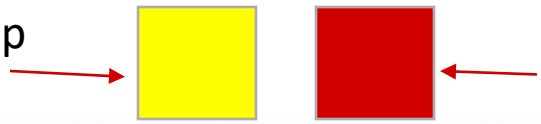
- **Current state-of-the-art**: e.g., “paper with code” https://paperswithcode.com/sota
- **Hardware constraints**: check available GPUs on HCC https://hcc.unl.edu/docs/submitting_jobs/submitting_gpu_jobs/
- **Data sample size**: more complex model more training data needed
- **Existing similar projects**: use models known effective to similar tasks
- **Hybrid or custom approach**: develop a new model by combining the above

Model Selection

For practice purposes, let's explore building our own models using PyTorch

```
class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.layer1 = self.ConvModule(in_features=3, out_features=64)      #16,16
        self.layer2 = self.ConvModule(in_features=64, out_features=128)    #8,8
        self.layer3 = self.ConvModule(in_features=128, out_features=256)   #4,4
        self.layer4 = self.ConvModule(in_features=256, out_features=512)   #2,2
        self.classifier = nn.Sequential(nn.Flatten(),
                                        nn.Linear(2*2*512, 1024),
                                        nn.ReLU(),
                                        nn.Linear(1024, 512),
                                        nn.ReLU(),
                                        nn.Linear(512,10),
                                        nn.Softmax())
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.classifier(x)
        return x
    def ConvModule(self, in_features, out_features):
        return nn.Sequential(nn.Conv2d(in_channels=in_features,
                                       out_channels=out_features,
                                       kernel_size=3, padding=1),
                             nn.BatchNorm2d(out_features),
                             nn.ReLU(),
                             nn.MaxPool2d(2,2))
```

If you **are done**, please put up your yellow sticky note.



If you **need help**, please put up your red sticky note.

Training

Difference among training and testing sets

Set	Purpose	Usage	Typical Size
Training Set	Used for model learning; model parameters are adjusted based on this data.	Directly used during the training phase.	Largest portion (~60-80%)
Testing Set	Used to evaluate the final model performance objectively.	Only used after training is fully completed.	Moderate portion (~10-20%)

Training

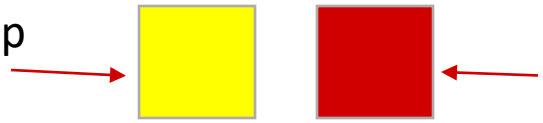
In general:

Models learn from the **training set**. We can tune the hyper-parameters of the model to help models learn effectively.

Let's see

- how different hyperparameters affect the model's performance
- how to use GPU acceleration

If you **are done**, please put up your yellow sticky note.



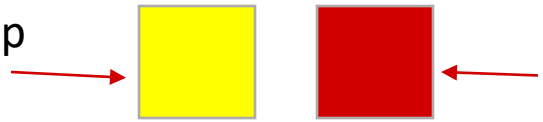
If you **need help**, please put up your red sticky note.

Evaluation

We evaluate the effectiveness of the model using **the testing set**. It's crucial that the testing set **remains completely unseen** throughout the training and validation phases to avoid data contamination for accurate evaluation.

- Let's evaluate the model we just trained

If you **are done**, please put up your yellow sticky note.



If you **need help**, please put up your red sticky note.

More Tips

- Testing performance are often lower than the training performance (e.g., accuracy), why?
- Overfitting can reduce the model's ability to generalize effectively to new, unseen data.
- Tuning or performing a grid search for optimal model selection and hyperparameter configurations is often necessary to achieve the best model performance.

Summary

In this session:

- we practiced image classification using a custom CNN model built with PyTorch.
- we used GPU resources provided by HCC to accelerate the training process.

In our next session, we'll explore text classification using a pre-trained BERT model.

Schedule

timestamp	Session
12:00 ~ 12:15pm	Setup and Support
12:15 ~ 1:00pm	**Introduction to HCC resources for ML and AI (45 Min)**
1:00 ~ 1:15pm	Short Break (15 Min)
1:15 ~ 2:00pm	**ML and AI workflows (45 Min)**
2:00 ~ 2:10pm	Short Break (10 Min)
2:10 ~ 2:50pm	**Introduction to PyTorch (40 Min)**
2:50 ~ 3:00pm	Break (10 Min)
3:00 ~ 3:50pm	**Introduction to PyTorch cont. (50 Min)**
3:50 ~ 4:00pm	Break (10 Min)
4:00 ~ 4:15pm	**Introduction to National Research Platform (NRP)**
4:15 ~ 4:30pm	Open Questions

Schedule

timestamp	Session
12:00 ~ 12:15pm	Setup and Support
12:15 ~ 1:00pm	**Introduction to HCC resources for ML and AI (45 Min)**
1:00 ~ 1:15pm	Short Break (15 Min)
1:15 ~ 2:00pm	**ML and AI workflows (45 Min)**
2:00 ~ 2:10pm	Short Break (10 Min)
2:10 ~ 2:50pm	**Introduction to PyTorch (40 Min)**
2:50 ~ 3:00pm	Break (10 Min)
3:00 ~ 3:50pm	**Introduction to PyTorch cont. (50 Min)**
3:50 ~ 4:00pm	Break (10 Min)
4:00 ~ 4:15pm	**Introduction to National Research Platform (NRP)**
4:15 ~ 4:30pm	Open Questions

Takeaways

In this session, you will:

- Practice a **Natural Language Processing (NLP)** task: Classify texts using a pre-trained **Bidirectional Encoder Representations from Transformers (BERT)** model

- Hands-on **AI/ML development workflow** from design, data preparation to model training and evaluation

- Learn how to transfer pre-trained model and fine tuning **using PyTorch** step-by-step

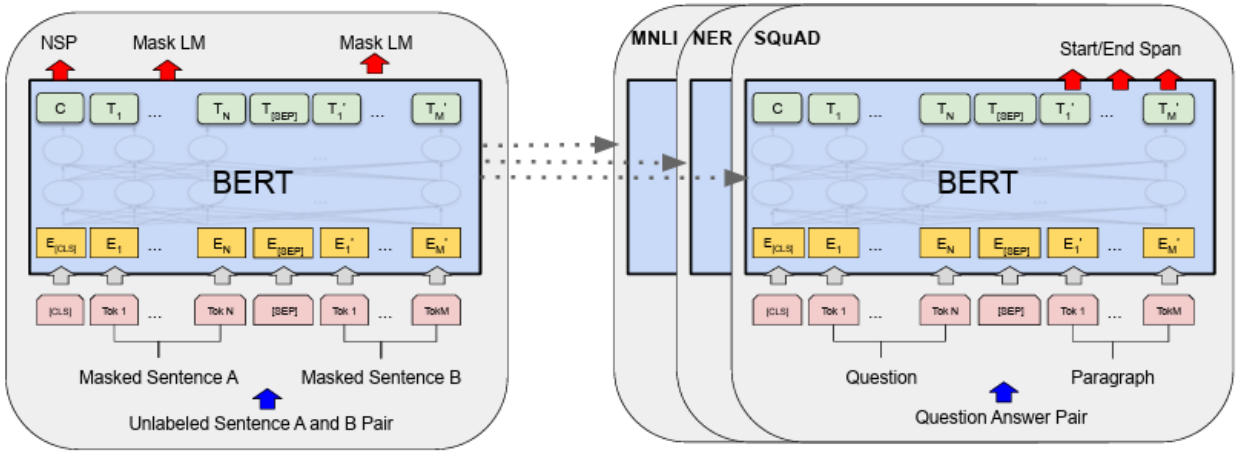
- Hands-on practice **case study on text classification**

Hands-On Practice 2

Overview:

- Task: Classify review comments as positive or negative
- Data: IMDb reviews*

- We will transfer a pre-trained BERT model in PyTorch to classify review comments of movies into positive or negative comments



Pre-training Fine-Tuning

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171-4186. 2019.

* https://www.kaggle.com/datasets/atulanandjha/imdb-50k-movie-reviews-test-your-bert

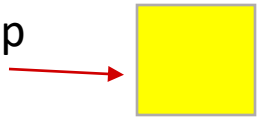
Design

- We are looking at a text categorization task
- Models have different advantages at specific task types:
  - Categorization – BERT
  - Generation – Generative Pre-trained Transformer (GPT)
- Or we can look for the state-of-the-art design in the similar tasks
  - IMDb is commonly used to benchmark BERT models

IMDb

Let’s take a look into IMDb in Jupyter Lab

If you **are done**, please put up your yellow sticky note.



If you **need help**, please put up your red sticky note.

Data Preparation

For text classification, we need to prepare tokenization:

- **Tokenization**: raw texts → understandable by the model
  - Splitting sentences into words/subword-chunks
  - Encoding the chunk into numerical representation

- Typical steps:
  - Using a tokenizer provided with the pre-trained model (e.g., BERT tokenizer)
  - Generating token IDs, attention masks, and padding sequences to a uniform length
  - Making sure input data matches the model's expected input format

Model Selection

BERT model:

- Bidirectional Encoder Representations from Transformers
- Captures complex language semantics thanks to its design, bidirectional context
- Excellent performance on sentiment analysis
- Reduced need for extensive labeled data with transfer learning

Training and Validation

Difference among training, validation and testing sets

Set	Purpose	Usage	Typical Size
Training Set	Used for model learning; model parameters are adjusted based on this data.	Directly used during the training phase.	Largest portion (~60-80%)
Validation Set	Used to tune hyperparameters and prevent overfitting during training.	Evaluated periodically to guide model adjustments.	Moderate portion (~10-20%)
Testing Set	Used to evaluate the final model performance objectively.	Only used after training is fully completed.	Moderate portion (~10-20%)

Training and Validation

Similar to previous hands-on practice:

Models learn from the **training set**. Then, we use **validation set** to monitor performance in real time. Based on this performance, we tune the hyper-parameters of the model to help models learn effectively.

- This session, we will also practice on monitoring the model using TensorBoard

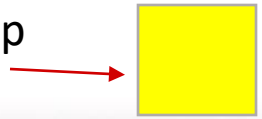
TensorBoard

1. In Swan's Open OnDemand portal: https://swan-ood.unl.edu/
2. Select "Interactive Apps" → "TensorBoard: Swan"

Open OnDemand Settings

Parameter	Value
Tensorflow version	tensorflow-gpu/py311/2.17
Tensorboard logdir	/work/groupname/username/hcc-ai-ml-workshop-2025/log
Number of cores	4
Running time in hours	1
Requested RAM in GBs	16
QoS type	short
Partition selection	---Leave BLANK---
Reservation	---Leave BLANK---
GRES	---Leave BLANK---
Job Constraints	---Leave BLANK---

If you **are done**, please put up your yellow sticky note.



If you **need help**, please put up your red sticky note.

Training and Validation

- Let's practice in Jupyter Lab
- TensorBoard should be updating as the training progresses.

If you **are done**, please put up your yellow sticky note.

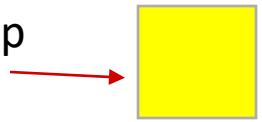
If you **need help**, please put up your red sticky note.

Evaluation

Same as the previous practice:
We evaluate the effectiveness of the model using **the testing set**. It's crucial that the testing set **remains completely unseen** throughout the training and validation phases to avoid data contamination for accurate evaluation.

- Let's evaluate the model we just trained

If you **are done**, please put up your yellow sticky note.



If you **need help**, please put up your red sticky note.

More Tips

- BERT has many variants now, choose the one that benefits your project
- BERT often has input length limit, truncates the long inputs into splits
- Tuning to set optimal hyper-parameters using a validation set is important
- Dataset balance is crucial to reduce over-sampling or under-sampling during training

Summary

In this session:

- we practiced text classification using a pre-trained BERT model with PyTorch.
- we used GPU resources provided by HCC to accelerate the training process.

Schedule

timestamp	Session
12:00 ~ 12:15pm	Setup and Support
12:15 ~ 1:00pm	**Introduction to HCC resources for ML and AI (45 Min)**
1:00 ~ 1:15pm	Short Break (15 Min)
1:15 ~ 2:00pm	**ML and AI workflows (45 Min)**
2:00 ~ 2:10pm	Short Break (10 Min)
2:10 ~ 2:50pm	**Introduction to PyTorch (40 Min)**
2:50 ~ 3:00pm	Break (10 Min)
3:00 ~ 3:50pm	**Introduction to PyTorch cont. (50 Min)**
3:50 ~ 4:00pm	Break (10 Min)
4:00 ~ 4:15pm	**Introduction to National Research Platform (NRP)**
4:15 ~ 4:30pm	Open Questions