Introduction to Deep Learning with PyTorch

Training State-of-the-Art Models Using Modern Training Frameworks

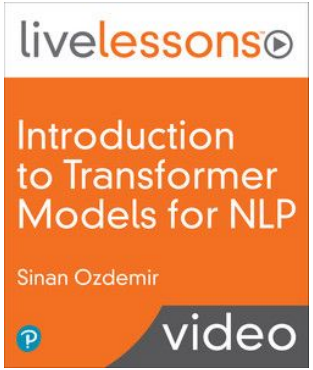
Sinan Ozdemir

Data Scientist, Entrepreneur,
Author, Lecturer

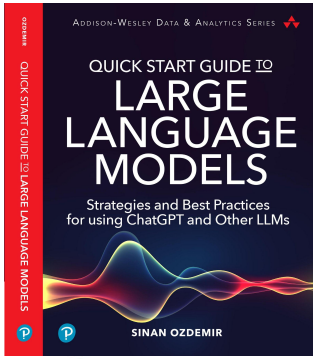
Pearson

Welcome!

My name is **Sinan Ozdemir** (in/sinan-ozdemir + @prof_oz)



- Current **founder** of Loop Genius (using GPT3 to help entrepreneurs get their first 100 customers)
- Current **lecturer** for O'Reilly and Pearson
- Founder of Kylie.ai (Funded by OpenAI Founder + Acquired)
- **Masters** in Theoretical Math from **Johns Hopkins**
- Former lecturer of Data Science at Johns Hopkins



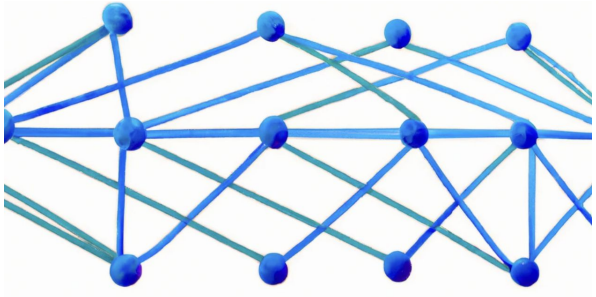
Author of ML textbooks and online series, including

- The Principles of Data Science
- Introduction to Transformer Models for NLP
- Quick Start Guide to LLMs (Fall 2023)

Pearson

Introduction to deep learning and neural networks

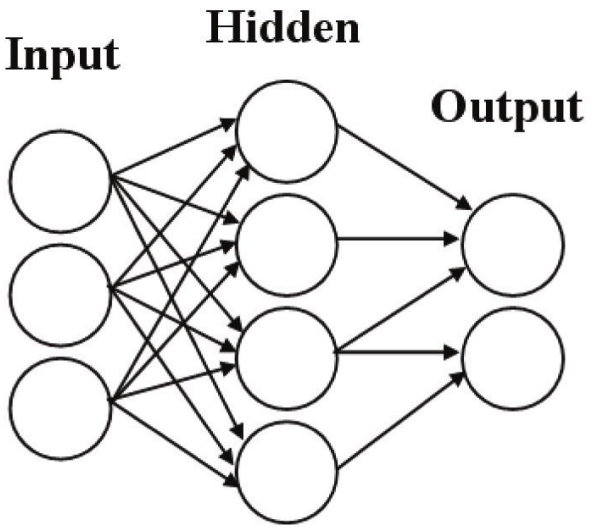
- Deep learning: a subfield of machine learning focused on neural networks with many layers
- Neural networks: interconnected layers of artificial neurons that process input data



- Neural networks can learn complex patterns and representations from large datasets
- Applications: image recognition, natural language processing, speech recognition, and more

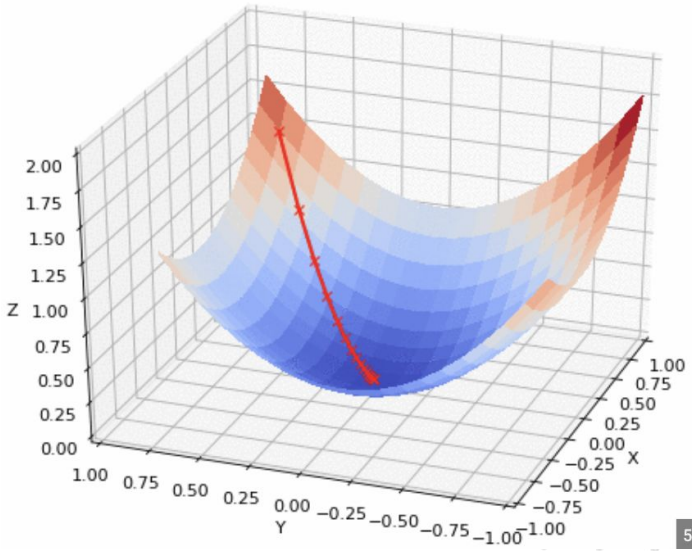
Basic components of a neural network

- Neurons: basic processing units of a neural network
- Layers: groups of neurons arranged in a sequence
  - Input layer: receives input data
  - Hidden layers: intermediate layers responsible for processing and learning features
  - Output layer: generates predictions or decisions based on learned features
- Weights and biases: parameters that the network learns during training
- Activation functions: non-linear functions applied to neuron outputs (e.g., ReLU, sigmoid, softmax)



Gradient descent and backpropagation

- Gradient descent: iterative optimization algorithm used to minimize the loss function
- Loss function: quantifies the difference between predictions and ground truth
- Backpropagation: algorithm for computing gradients of the loss function with respect to network parameters
- Learning rate: hyperparameter controlling the step size of gradient descent updates



Pearson

Introduction to PyTorch

- PyTorch: open-source deep learning framework developed by Facebook's AI Research lab
- Eager execution: dynamic computation graph for easy debugging and flexibility
- Autograd: automatic differentiation system for computing gradients
- GPU acceleration: efficient computation on GPUs using CUDA
- Pre-built modules: commonly used neural network layers and functions

PyTorch autograd & dynamic computation graph

- Autograd: PyTorch's automatic differentiation engine
- Computes gradients of loss function with respect to model parameters
- Dynamic computation graph: built on-the-fly as you execute operations
- Allows for flexible and efficient model construction and modification

Eg: A simple neural network in PyTorch

- Define neural network architecture (**`nn.Module`**)
- Initialize layers (e.g., **`nn.Linear`**) & activation functions (e.g., **`nn.ReLU`**)

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

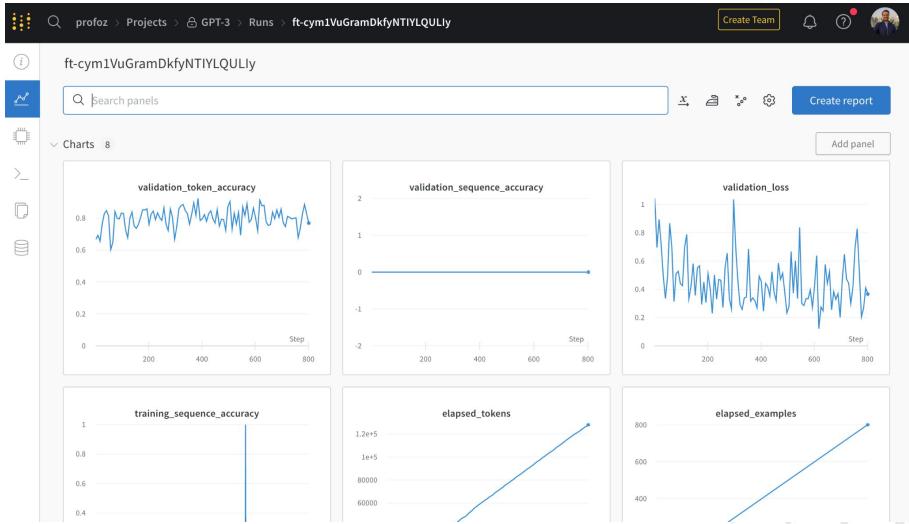
- Implement forward pass in forward method

Eg. Train a neural network w. gradient descent

- Load and preprocess the MNIST dataset
- Define a neural network architecture for image classification
- Train the model using gradient descent
- Evaluate model performance on test dataset

Experiment tracking and reproducibility

- Experiment tracking: logging model parameters, hyperparameters, performance metrics, and artifacts
- Reproducibility: ability to re-run experiments and obtain the same results
- Ensures reliable and consistent model development process
- Enables collaboration and sharing of results with others



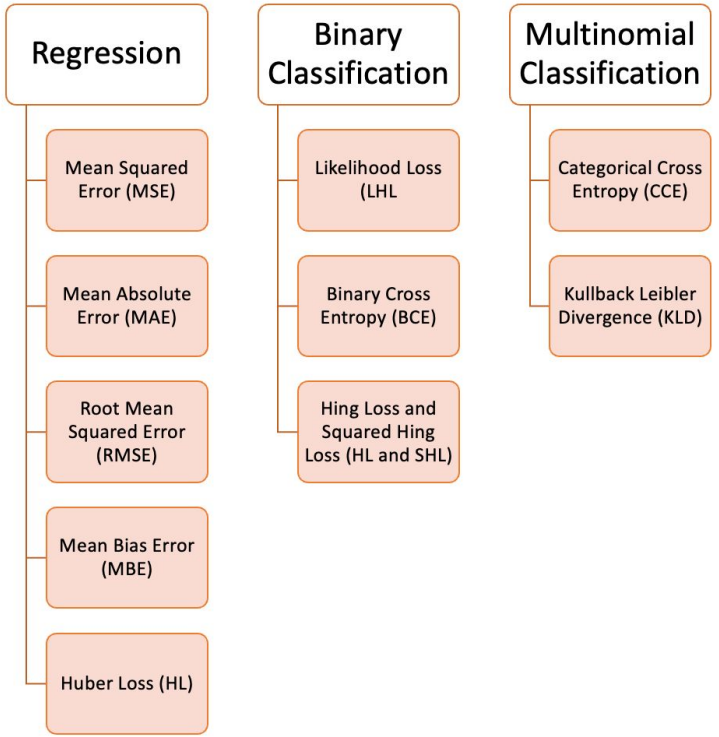
Pearson

Experiment tracking tools

- Weights & Biases: platform-agnostic tool for experiment tracking and collaboration
- Track model parameters, hyperparameters, metrics, and artifacts
- Compare experiments and visualize performance

Loss functions in deep learning

- Quantify the difference between model predictions and ground truth
- Common loss functions:
  - Mean squared error (MSE)
  - Cross-entropy loss
  - Binary cross-entropy loss
  - Hinge loss
- Chosen based on the type of problem (e.g., regression, classification)



Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fpub.towardsai.net%2F10-commonly-used-loss-functions-explained-with-python-code-59967e1f3c8d

Quantify deep learning performance

- Metrics measure model performance on test data
- Common metrics:
  - Accuracy
  - Precision, Recall, F1-score
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Area Under the Curve (AUC-ROC)
- Chosen based on the type of problem and evaluation criteria

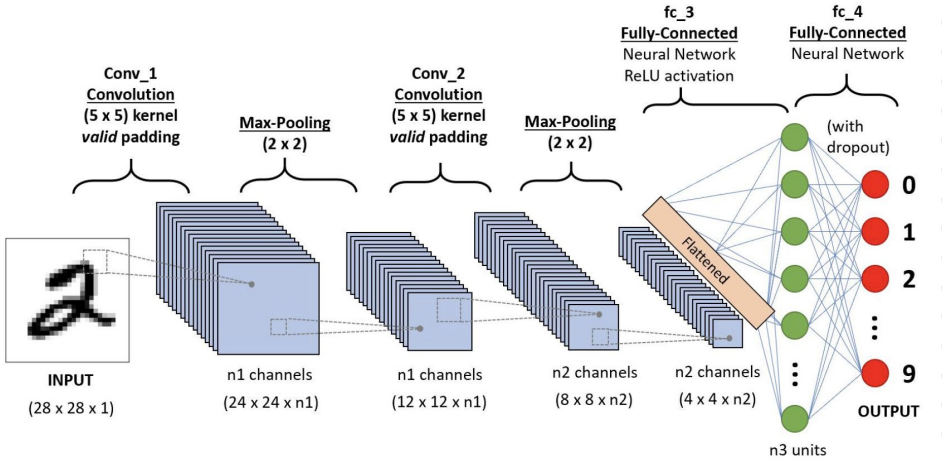
Eg: Train an MNIST classifier using various losses

- Train an MNIST classifier using different loss functions
- Evaluate model performance using appropriate metrics
- Compare the performance of models trained with different loss functions

```
# Define the loss functions to compare
loss_functions = {'CrossEntropyLoss': nn.CrossEntropyLoss(),
                  'NLLLoss': nn.NLLLoss(),
                  'KLDivLoss': nn.KLDivLoss()}
```

Convolutional Neural Networks (CNNs)

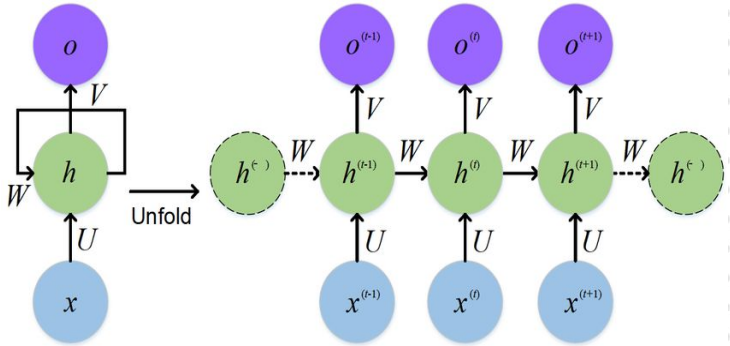
- Specialized neural networks for processing grid-like data (e.g., images)
- Convolutional layers: learn local features in input data
- Pooling layers: reduce spatial dimensions and enhance feature representations
- Fully connected layers: perform classification based on learned features



Source:
https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture

Recurrent Neural Networks (RNNs)

- Neural networks designed for processing sequences of data
- Maintain internal hidden states to capture temporal dependencies



- Variants of RNNs:
  a. Long Short-Term Memory (LSTM)
  b. Gated Recurrent Unit (GRU)

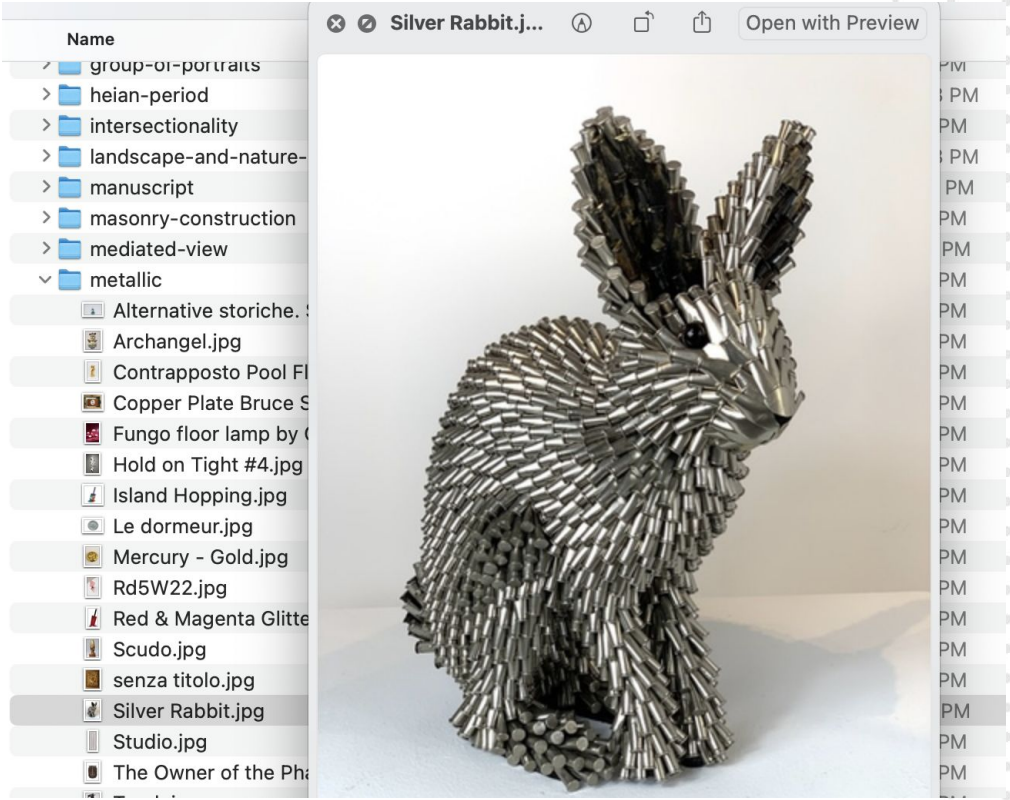
- Applications: text classification, language modeling, sequence-to-sequence learning

Pearson

Source:
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FThe-standard-RNN-and-unfolded-RNN_fig1_318332317

Eg: Train a CNN for image classification

- Load and preprocess art style dataset
- Define a CNN architecture for image classification
- Train the model using gradient descent
- Evaluate model performance on test dataset



Eg: Train an LSTM for Tweet classification

- Load and preprocess the Twitter disaster dataset
- Define an RNN architecture for classification (e.g., LSTM)
- Train the model using gradient descent
- Evaluate model performance on test dataset

Transfer learning and pre-trained models

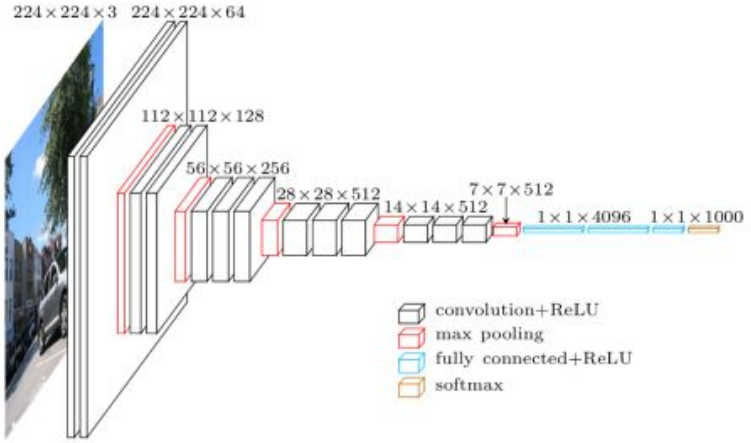
- Transfer learning: leveraging knowledge from pre-trained models to improve performance on new tasks
- Pre-trained models: models trained on large datasets for specific tasks (e.g., VGG-11, BERT)
- Fine-tuning: training a pre-trained model on a new task with a smaller dataset
- Faster convergence and improved performance compared to training from scratch

Eg: Fine-tune a pre-trained VGG-11 model

- Load and preprocess the custom image dataset
- Load pre-trained VGG-11 model
- Replace the final classification layer to

  match the number of

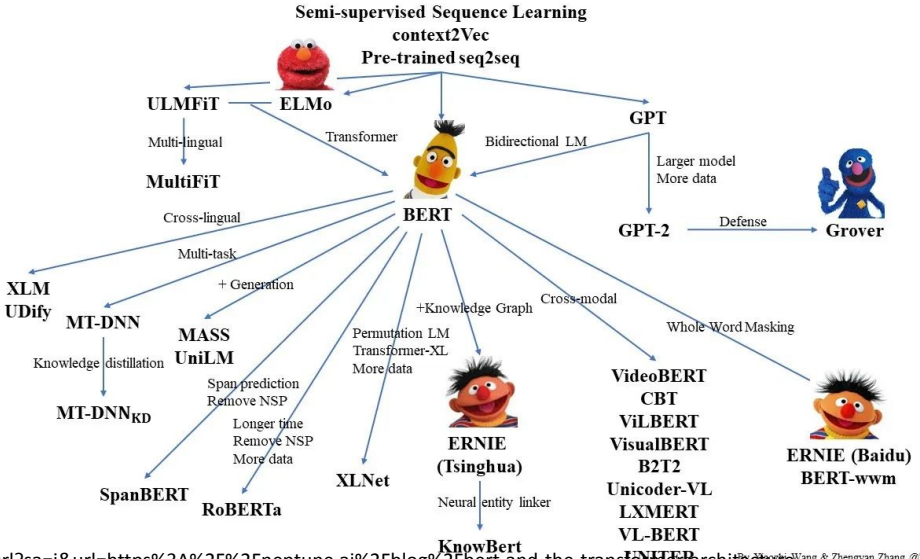
  classes in the custom dataset
- Fine-tune the model using gradient descent
- Evaluate model performance on test dataset



Pearson

Eg: Fine-tuning a pre-trained BERT model

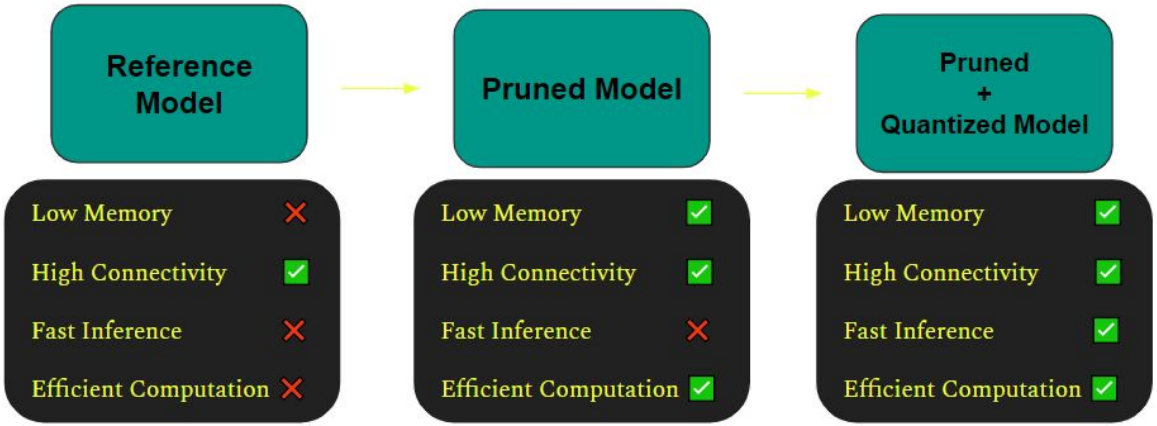
- Load and preprocess the custom text dataset
- Load the pre-trained BERT model using Transformers library
- Set up a training configuration with the Trainer class
- Train the model on the custom text dataset
- Evaluate model performance on test dataset



Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fneptune.ai%2Fblog%2Fbert-and-the-transformer-architecture

Model pruning, quantization, and deployment

- Pruning: removing redundant or less important connections in the network to reduce model size
- Quantization: reducing the precision of model parameters to lower memory and computational requirements



Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fgithub.com%2FArvindSubramaniam%2FExplicit-Connection-Sensitivity-Pruning-and-Quantization

Eg: Prune and quantize a VGG-11 model

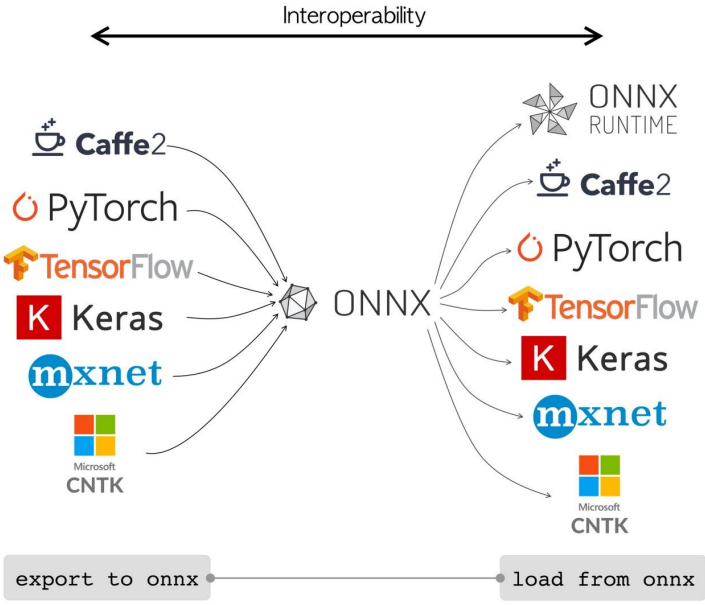
- Prune the VGG-11 model using PyTorch's pruning utilities
- Quantize the BERT model using PyTorch's quantization utilities

Simple Flask API for serving the VGG-11 model

- Flask: lightweight web framework for serving Python applications
- Set up a Flask application to serve the VGG-11 model for image classification
- Implement a RESTful API for receiving image input and returning classification results

ONNX conversion for model interoperability

- ONNX: Open Neural Network Exchange format for model interoperability
- Facilitates the deployment of models across deep learning frameworks and platforms
- Convert PyTorch models to ONNX format for greater flexibility in deployment
- Image: Diagram showing the conversion of a PyTorch model to ONNX format



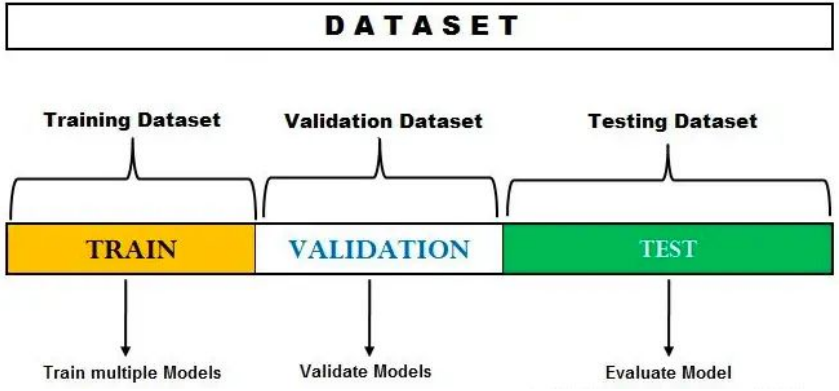
Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fcms-ml.github.io%2Fdocumentation%2Finference%2Fonnx.html

Eg: Convert the VGG-11 model to ONNX

- Convert the VGG-11 model to ONNX format using PyTorch's torch.onnx.export() function

The importance of using a holdout set

- Holdout set: a separate dataset not used during training or validation
- Ensures a fair evaluation of the model's performance on unseen data
- Reduces the risk of overfitting and biased model evaluation
- When to use a holdout set: small datasets, model selection, and fine-tuning pre-trained models



GPU/CUDA integration in deep learning

- GPUs: specialized hardware for parallel processing, ideal for deep learning tasks
- CUDA: NVIDIA's parallel computing platform and programming model
- PyTorch has aseamless integration with CUDA
- Use to() and cuda() methods to move models and data to GPU
- Enable GPU acceleration during training by specifying the device (e.g., 'cuda:0')

```
# Check if GPU is available and use it, otherwise use CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

device
```

```
device(type='cpu')
```

Pearson

Course wrap-up and next steps

- Recap of main topics covered in the course
- Further resources for deep learning and PyTorch:
  - Books: "Applied Deep Learning with TensorFlow 2" by Umberto Michelucci, "Deep Learning with Python" by Francois Chollet
  - Videos: "Machine Learning with scikit-learn LiveLessons" by David Mertz, "Real-World Machine Learning" by Henrik Brink, Joseph W. Richards, and Mark Fetherolf
- Encourage students to continue learning and applying deep learning techniques in their projects