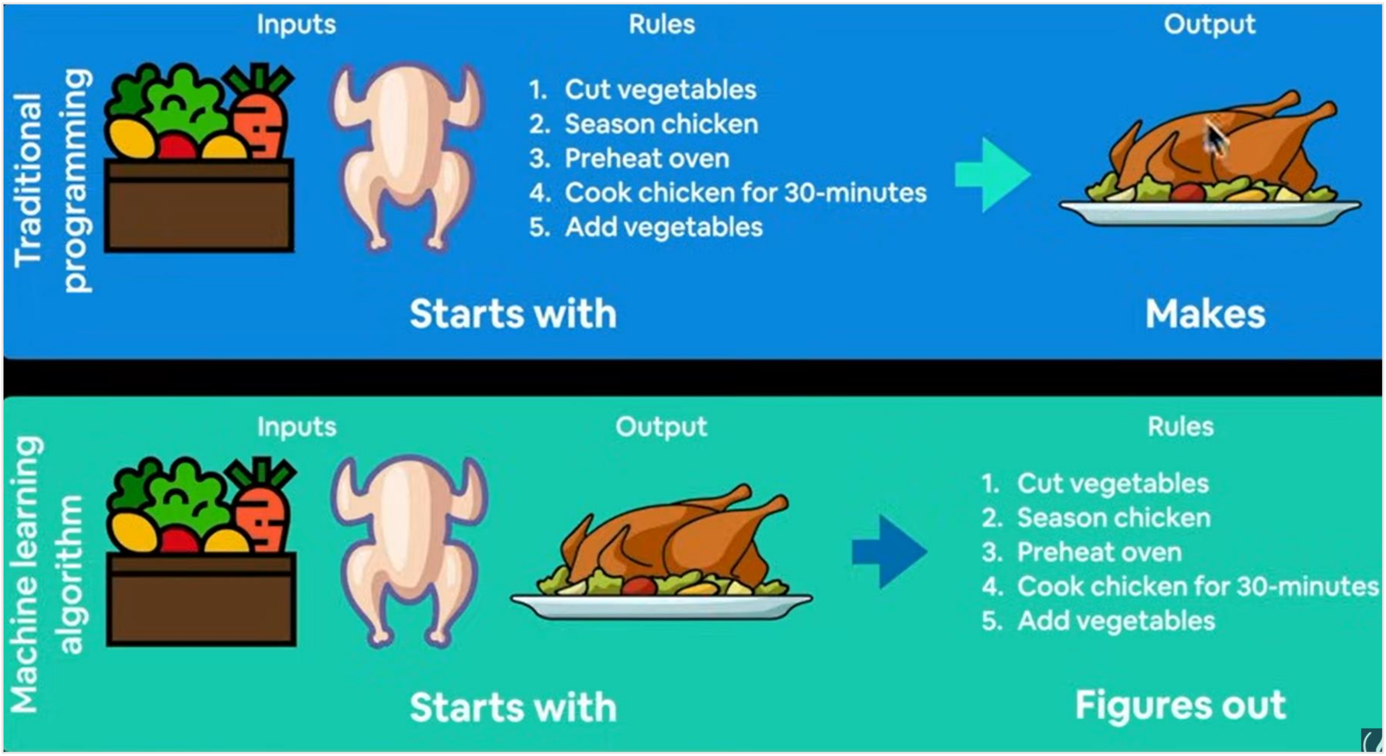
Imran Nawar | DIP Lab | Islamia College, Peshawar
LinkedIn | Aug 5, 2024

PyTorch for Deep Learning & Machine Learning

Machine Learning Algorithms vs Traditional Programming:



Why use machine learning and deep learning?

Reason: For a complex problem, can you think of all the rules? (Probably not)

"If you can build a simple rule-based system that doesn't require machine learning, do that."

(Google machine learning handbook)

What deep learning is good for?

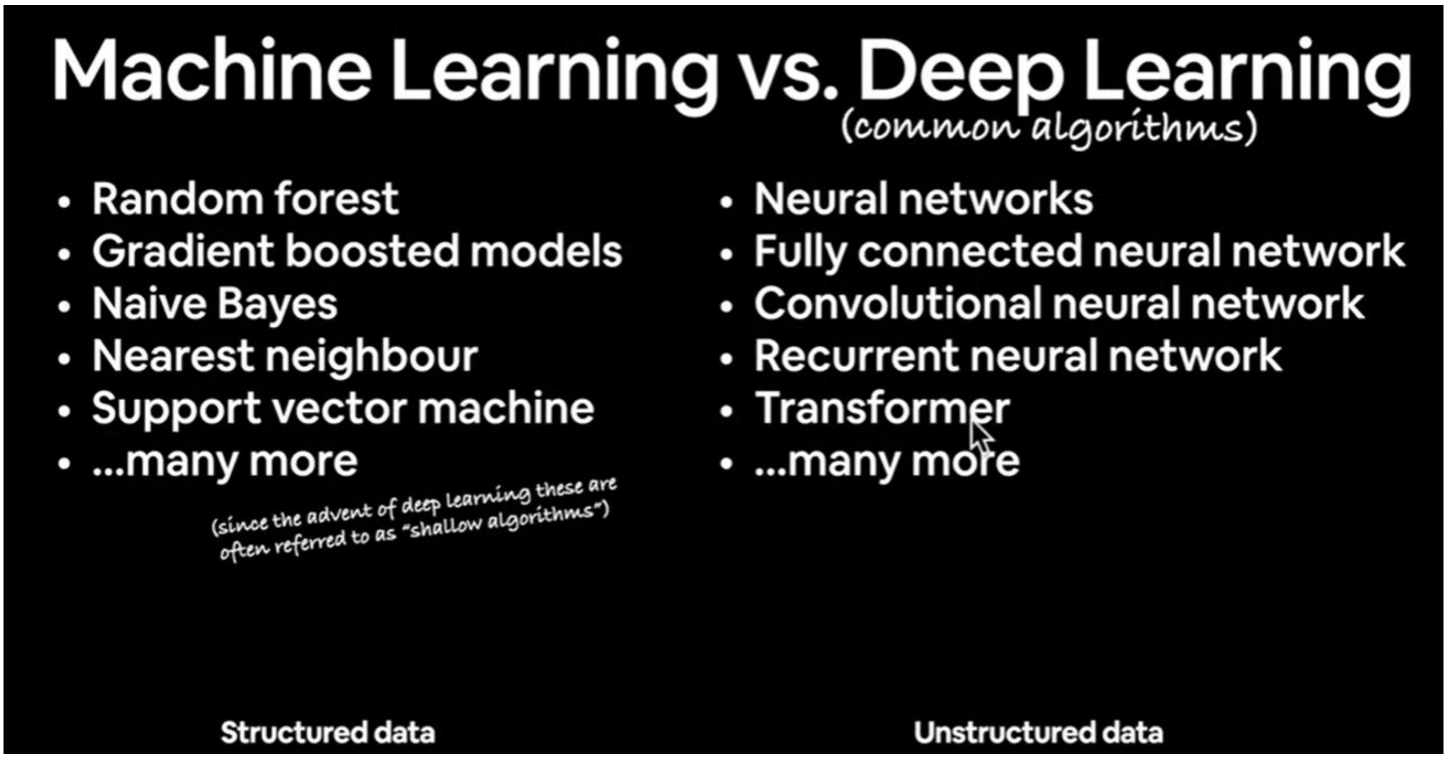
- **Problems with long lists of rules:** When the traditional approach fails, machine learning/deep learning may help.
- **Continually changing environments:** Deep learning can adapt (learn) to new scenarios.
- **Discovering insights within large collections of data:** Can you imagine trying to hand-craft rules for what 101 different kinds of food look like?

What deep learning is not good for?

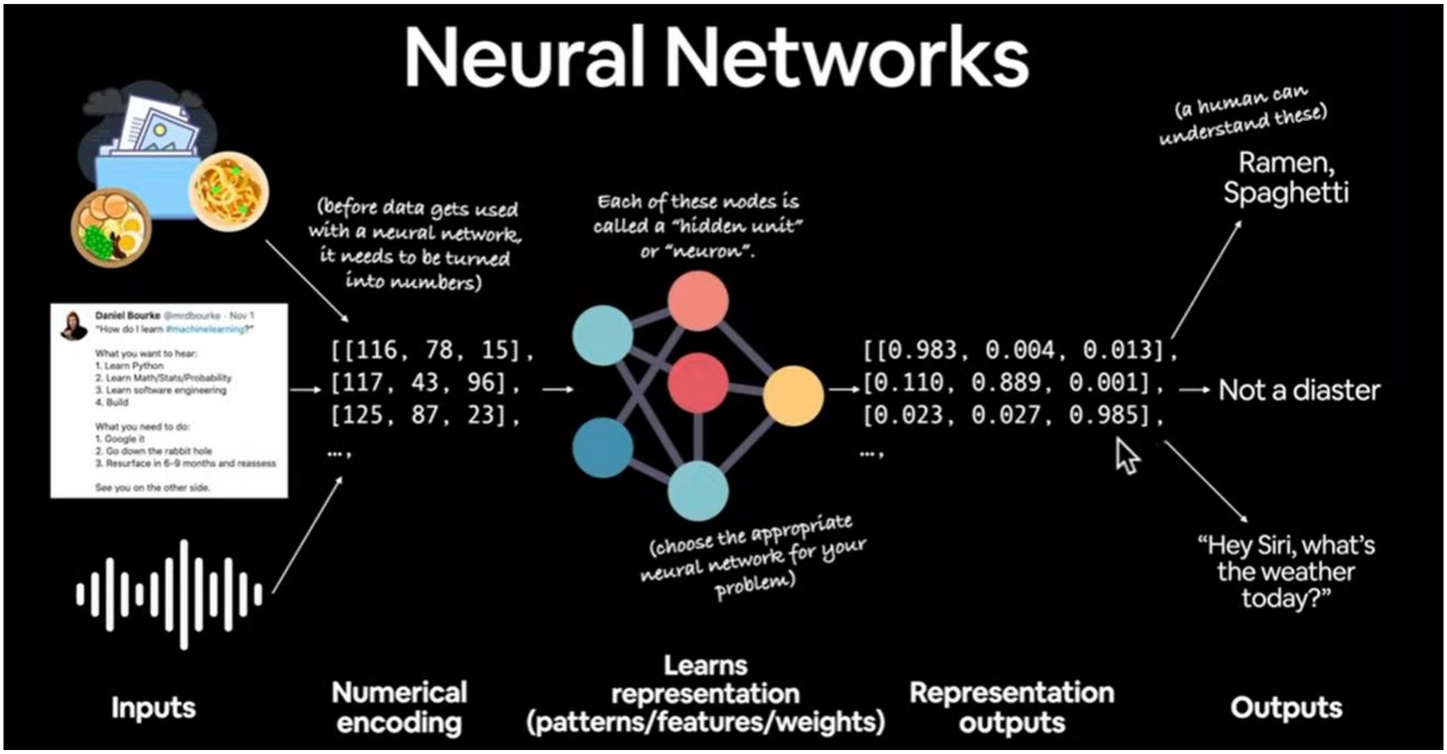
- **When you need explainability:** The patterns learned by a deep learning model are typically uninterpretable by a human.
- **When the traditional approach is a better option:** If you can accomplish what you need with a simple rule-based system.

- **When errors are unacceptable:** Since the outputs of deep learning model aren't always predictable.
- **When you don't have much data:** Deep learning models usually require a fairly large amount of data to produce great results.

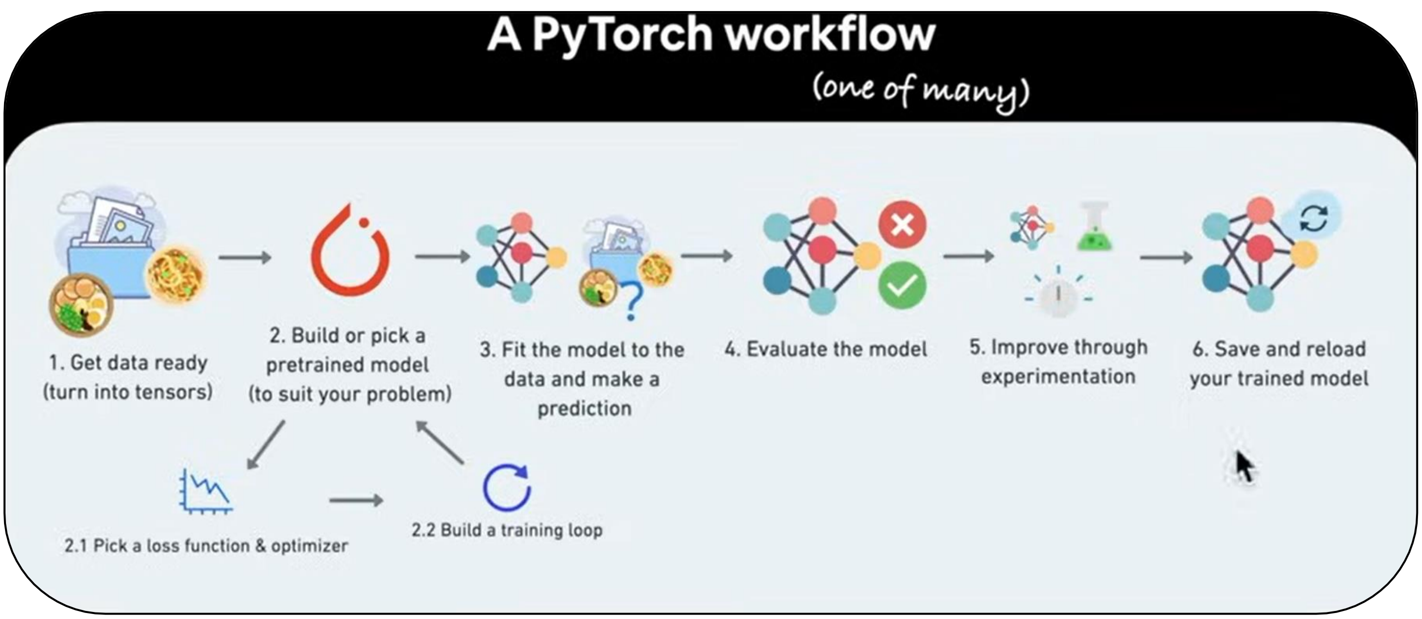
Machine learning vs Deep learning:



Neural Networks:



PyTorch



What is PyTorch?

- Most popular research deep learning framework
- Write fast deep learning code in Python (able to run on a GPU/many GPUs)
- Able to access many pre-built deep learning models (Torch Hub / torchvision.models)
- **Whole stack:** Preprocess data, model data, deploy model in your application/cloud.

- Originally designed and used in-house by Facebook/Meta (now open-source and used by companies such as Tesla, Microsoft, OpenAI)

Why PyTorch?

- It is research favorite.

What is a GPU/TPU?

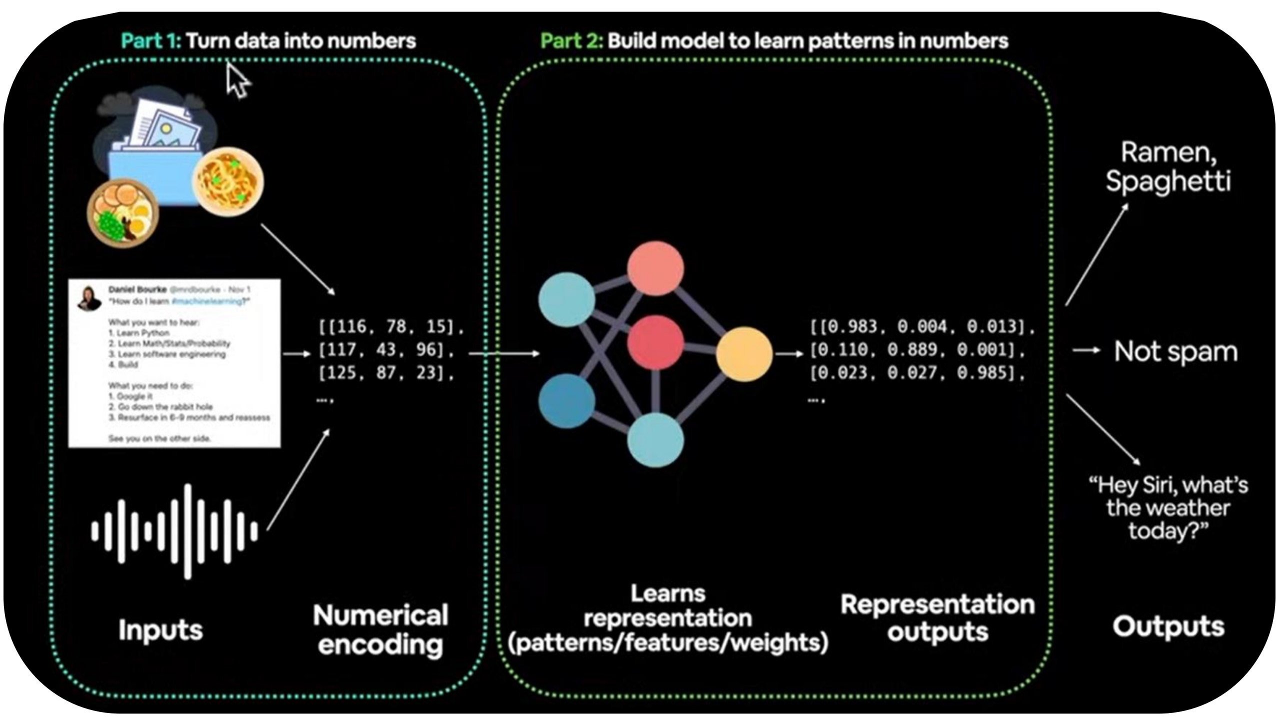
- **Graphic Processing Unit (GPU):** A specialized processor designed to accelerate graphics rendering and complex calculations in parallel.
- **Tensor Processing Unit (TPU):** A custom-designed chip by Google optimized for machine learning and artificial intelligence applications, particularly for handling tensor computations.
- **CUDA (Compute Unified Device Architecture):** A parallel computing platform and programming model developed by NVIDIA that enables general-purpose processing on GPUs, allowing software to perform complex computations efficiently.

Machine Learning is a game of two parts:

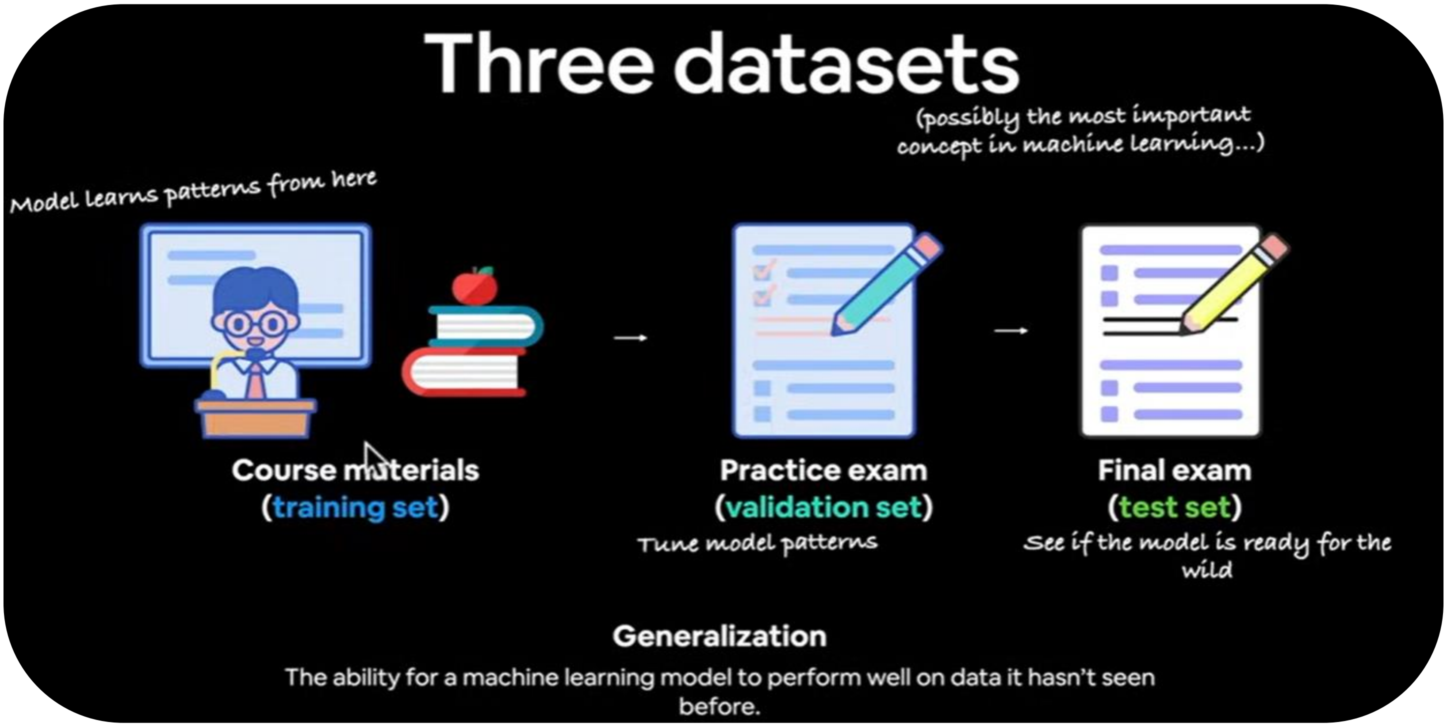
1. Get data into a numerical representation.
2. Build a model to learn pattern in that numerical representation.

PyTorch `permute()` function:

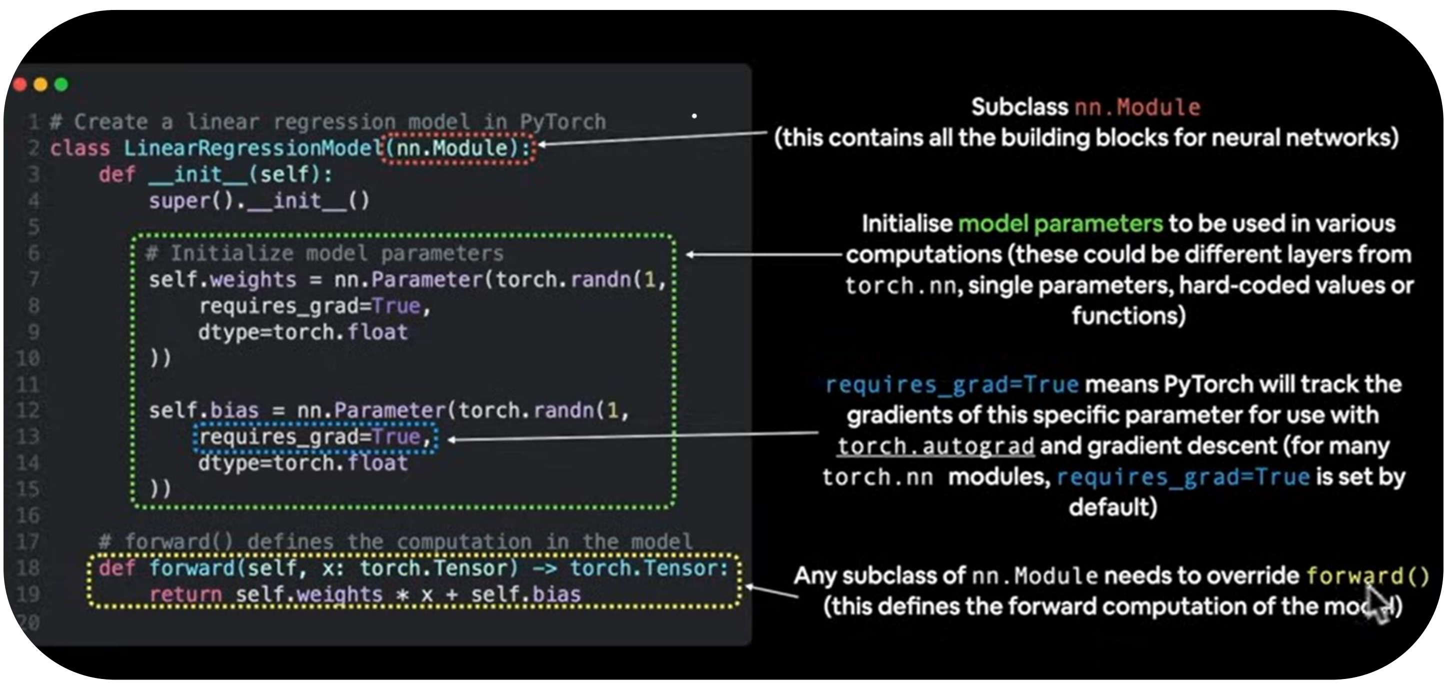
- **Purpose**: Reorders dimensions of a tensor.
- **Syntax**: `tensor.permute(dims)`
- **Key:** Does not modify original tensor, returns a new one.
- **Common use:** Adjusting tensor shape for operations (e.g., `from [batch, channels, height, width]` to `[batch, height, width, channels]`



Three Datasets



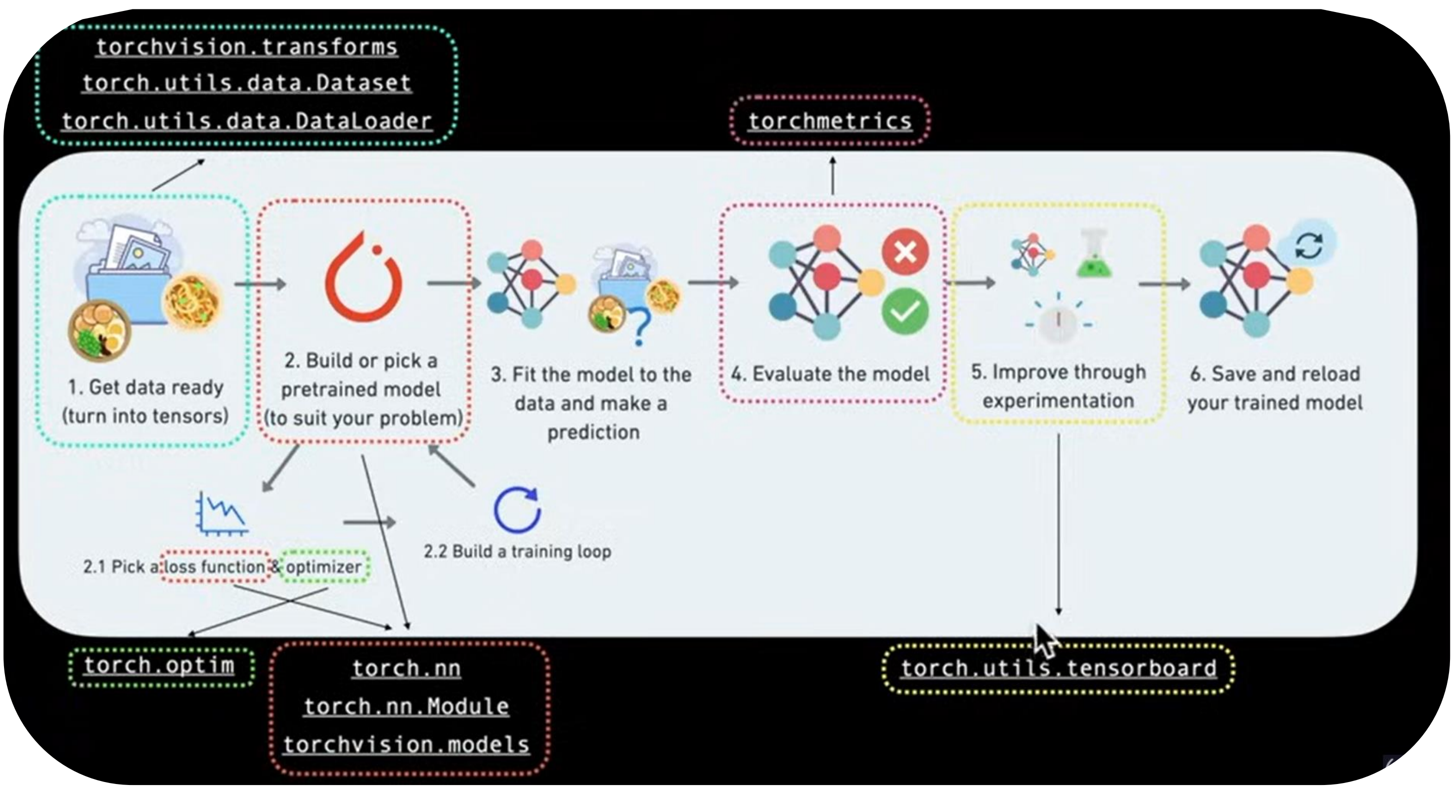
Build Model:



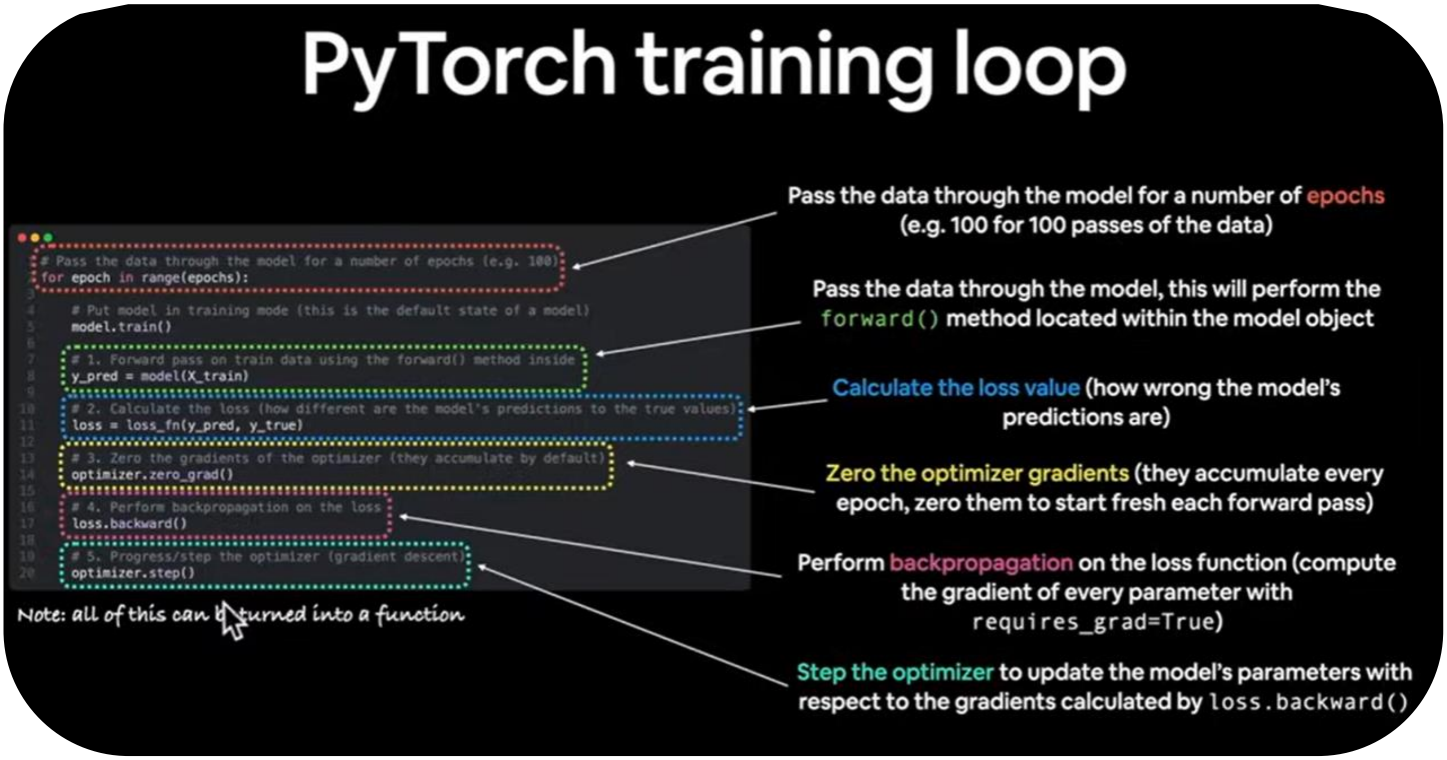
PyTorch Essential Neural Networks Building Modules:

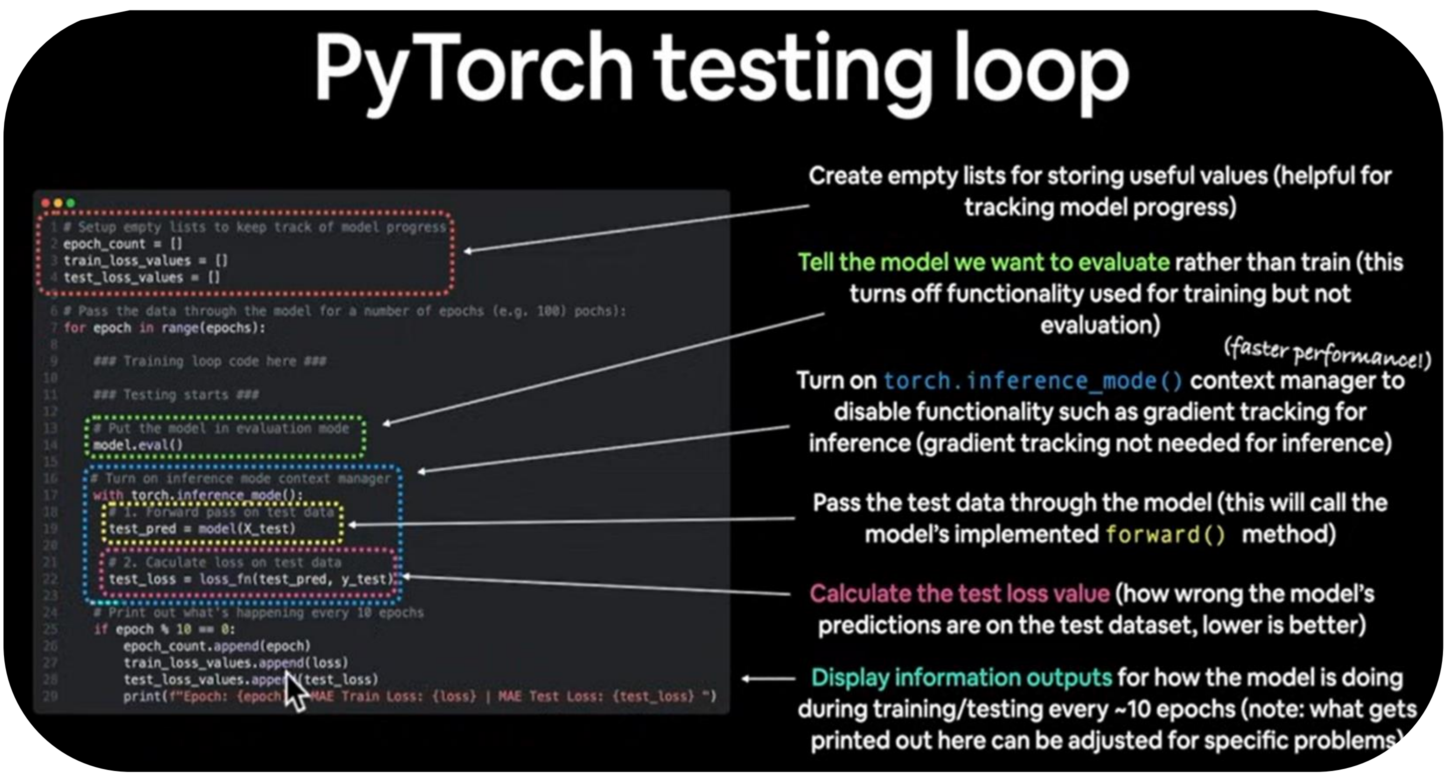
PyTorch essential neural network building modules

PyTorch module	What does it do?
torch.nn	Contains all of the building blocks for computational graphs (essentially a series of computations executed in a particular way).
torch.nn.Module	The base class for all neural network modules, all the building blocks for neural networks are subclasses. If you're building a neural network in PyTorch, your models should subclass nn.Module. Requires a forward() method be implemented.
torch.optim	Contains various optimization algorithms (these tell the model parameters stored in nn.Parameter how to best change to improve gradient descent and in turn reduce the loss).
torch.utils.data.Dataset	Represents a map between key (label) and sample (features) pairs of your data. Such as images and their associated labels.
torch.utils.data.DataLoader	Creates a Python iterable over a torch Dataset (allows you to iterate over your data).

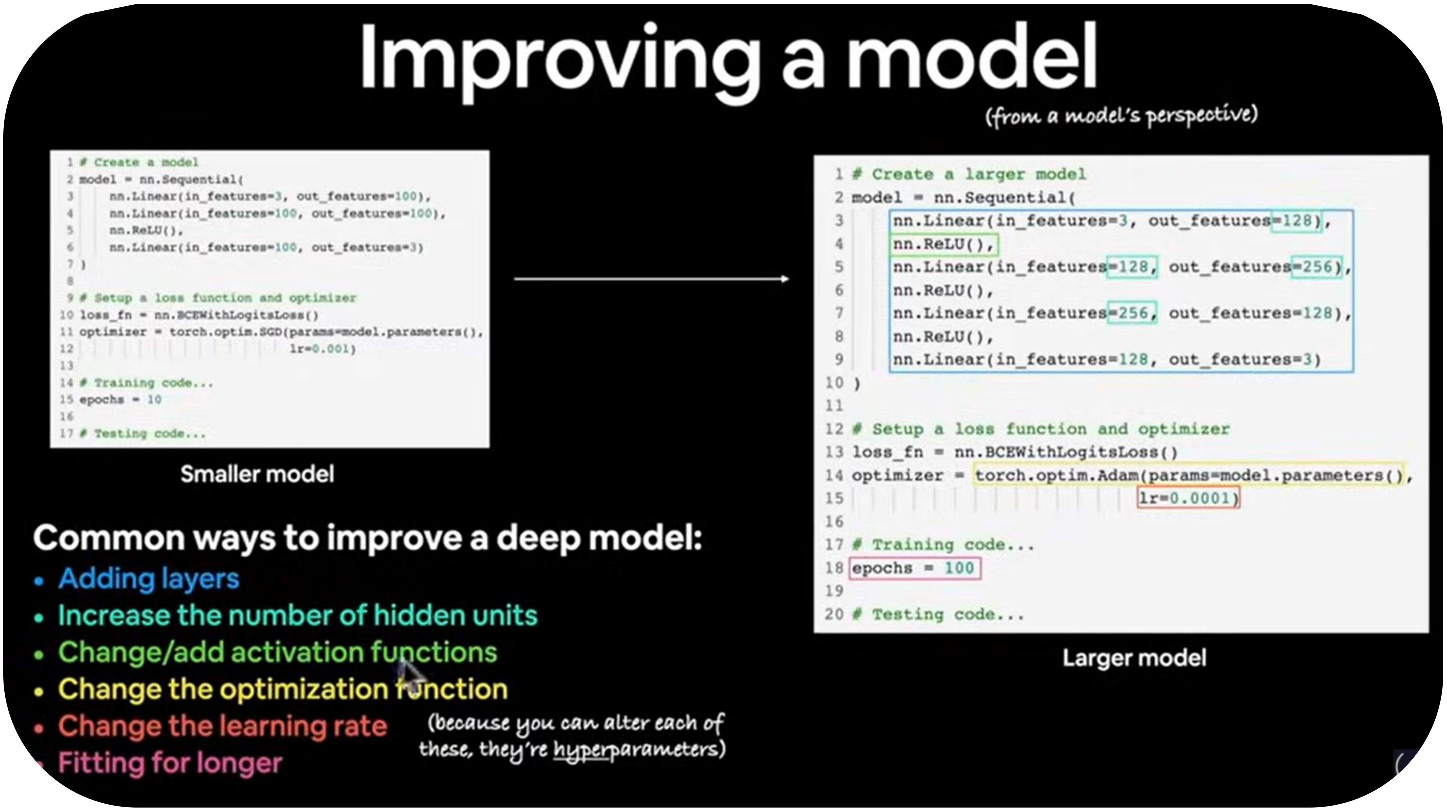


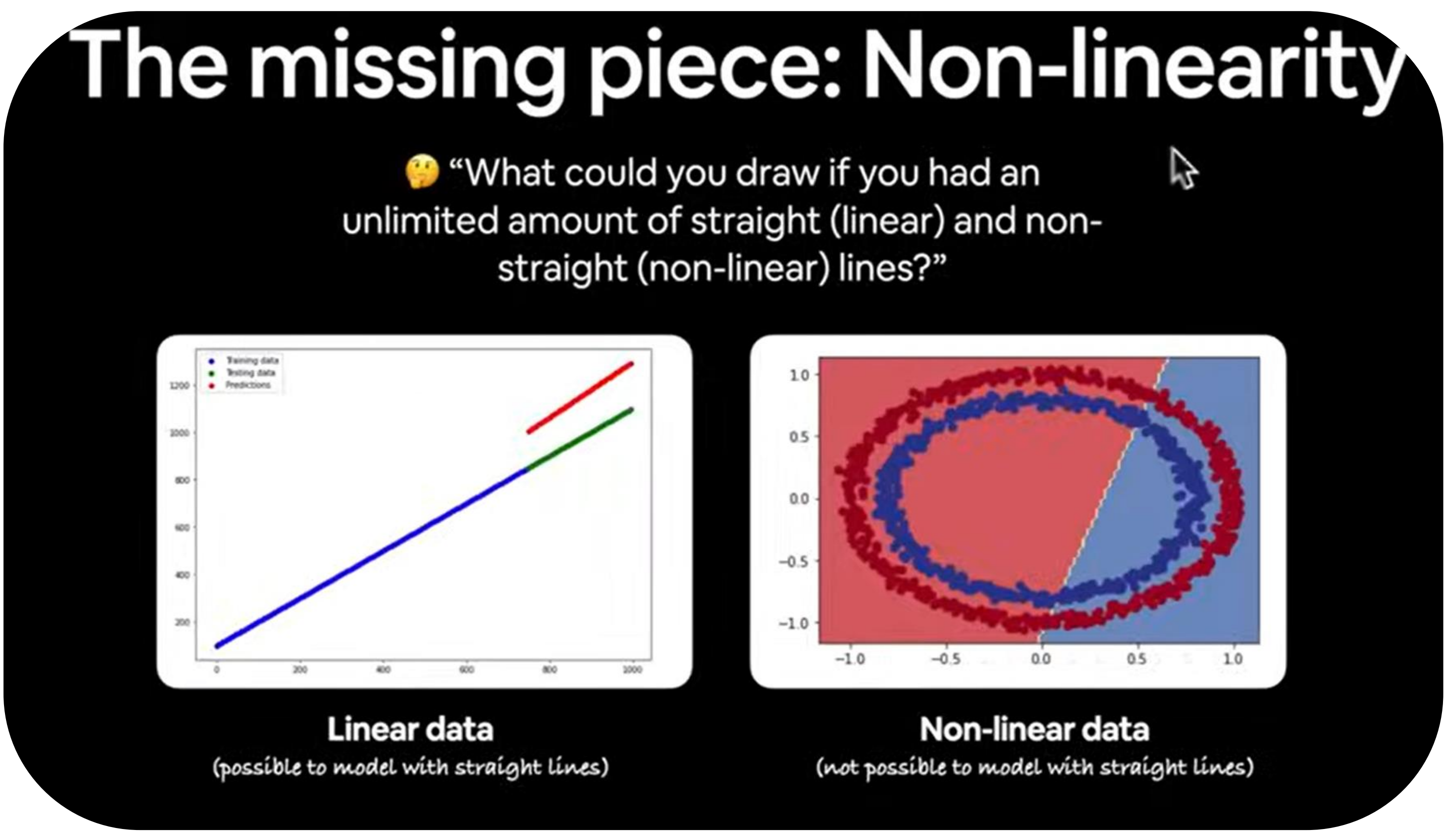
PyTorch Training Loop:



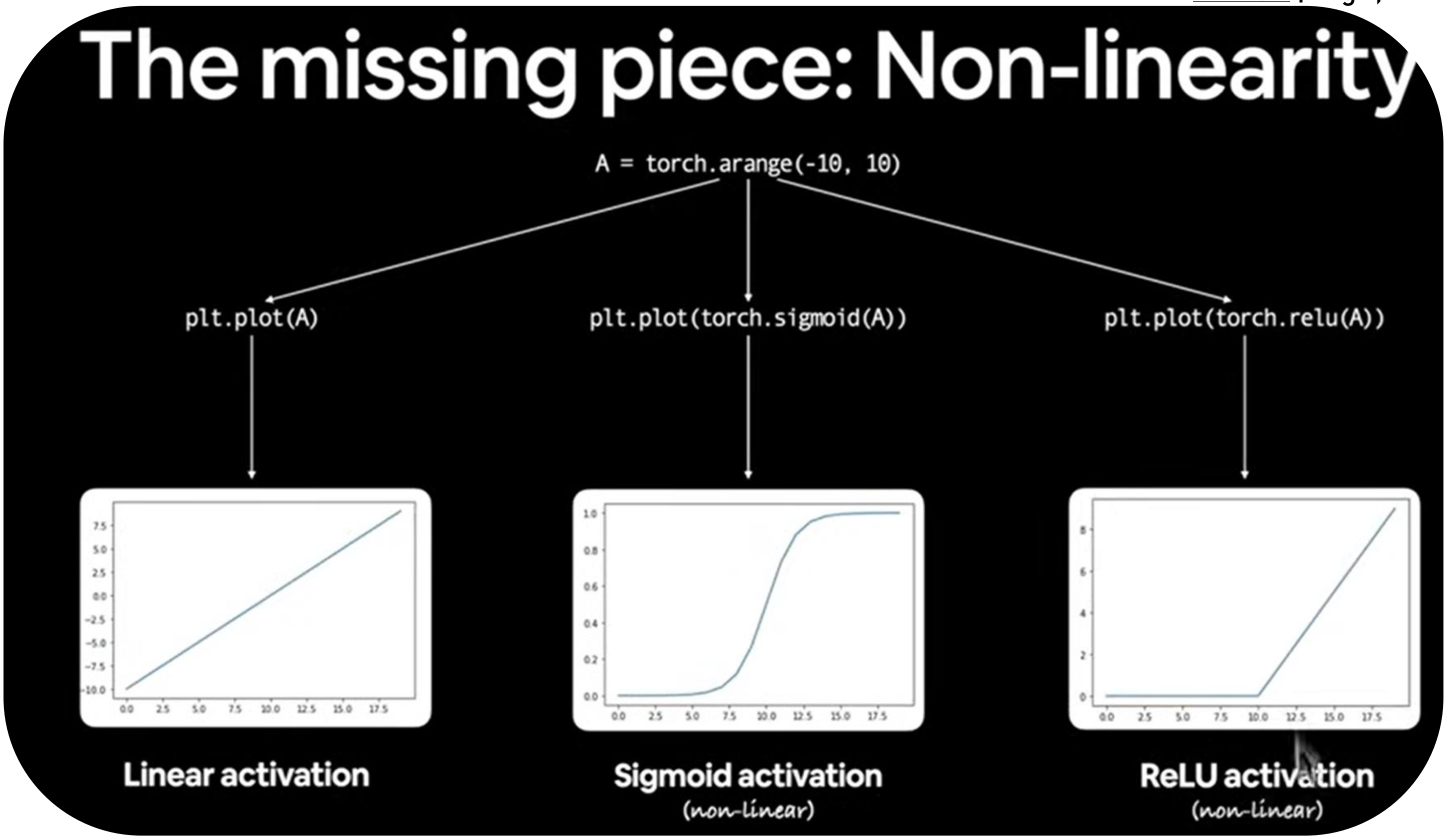


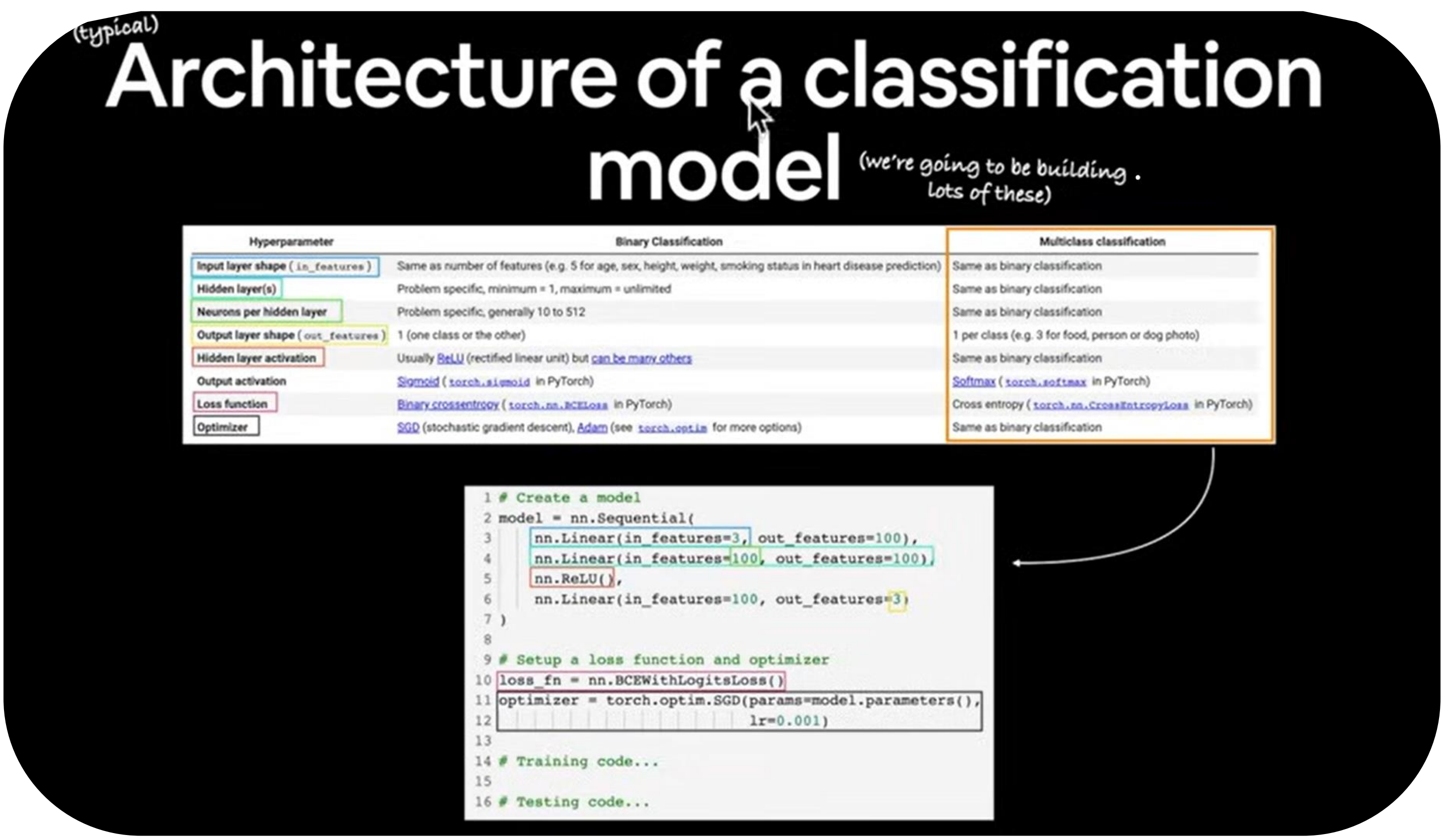
Improving a Model:





Imran Nawar | DIP Lab | Islamia College, Peshawar
LinkedIn | Aug 5, 2024





Hyperparameter	Binary Classification	Multiclass classification
Input layer shape (`in_features`)	Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction)	Same as binary classification
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited	Same as binary classification
Neurons per hidden layer	Problem specific, generally 10 to 512	Same as binary classification
Output layer shape (`out_features`)	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden layer activation	Usually ReLU (rectified linear unit) but can be many others	Same as binary classification
Output activation	Sigmoid (`torch.sigmoid` in PyTorch)	Softmax (`torch.softmax` in PyTorch)
Loss function	Binary crossentropy (`torch.nn.BCELoss` in PyTorch)	Cross entropy (`torch.nn.CrossEntropyLoss` in PyTorch)
Optimizer	SGD (stochastic gradient descent), Adam (see `torch.optim` for more options)	Same as binary classification

```
# Create a model
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=100),
    nn.Linear(in_features=100, out_features=100),
    nn.ReLU(),
    nn.Linear(in_features=100, out_features=3)
)

# Setup a loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(params=model.parameters(),
                            lr=0.001)

# Training code...

# Testing code...
```

(some common)

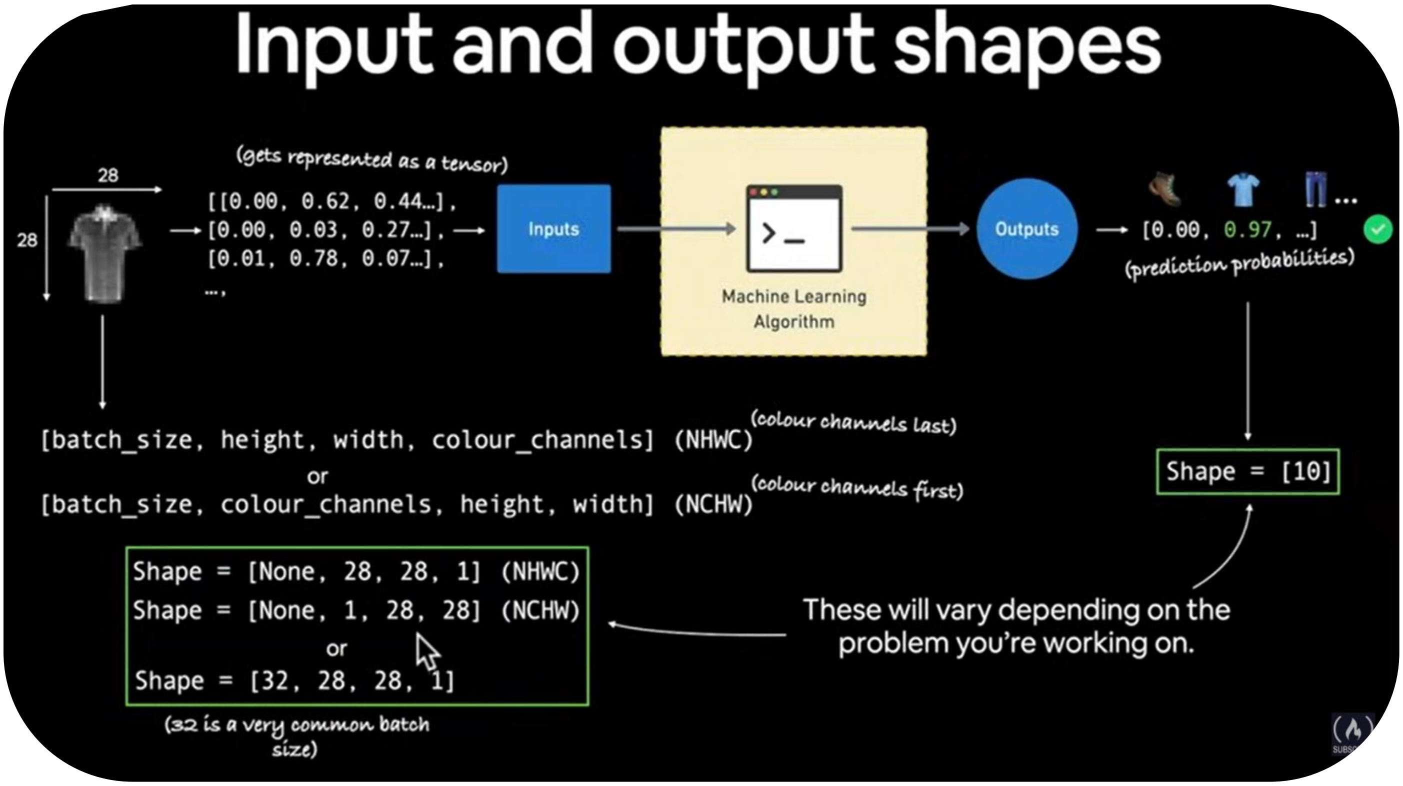
Classification evaluation methods

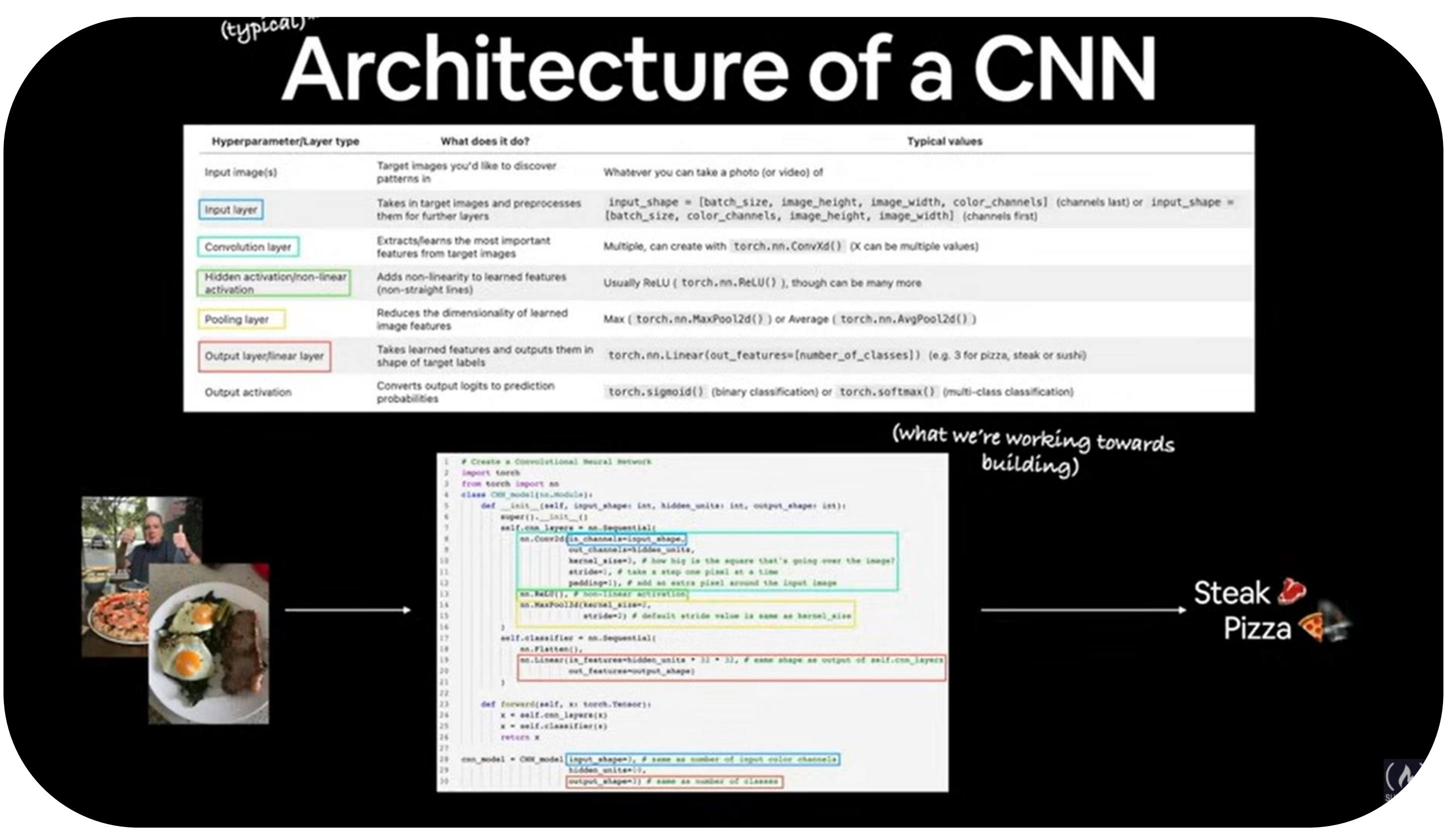
Key: **tp** = True Positive, **tn** = True Negative, **fp** = False Positive, **fn** = False Negative

Metric Name	Metric Forumla	Code	When to use
Accuracy	**Accuracy** $= \frac{tp + tn}{tp + tn + fp + fn}$	`torchmetrics.Accuracy()` or `sklearn.metrics.accuracy_score()`	Default metric for classification problems. Not the best for imbalanced classes.
Precision	**Precision** $= \frac{tp}{tp + fp}$	`torchmetrics.Precision()` or `sklearn.metrics.precision_score()`	Higher precision leads to less false positives.
Recall	**Recall** $= \frac{tp}{tp + fn}$	`torchmetrics.Recall()` or `sklearn.metrics.recall_score()`	Higher recall leads to less false negatives.
F1-score	**F1-score** $= 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	`torchmetrics.F1Score()` or `sklearn.metrics.f1_score()`	Combination of precision and recall, usually a good overall metric for a classification model.
Confusion matrix	NA	`torchmetrics.ConfusionMatrix()`	When comparing predictions to truth labels to see where model gets confused. Can be hard to use with large numbers of classes.

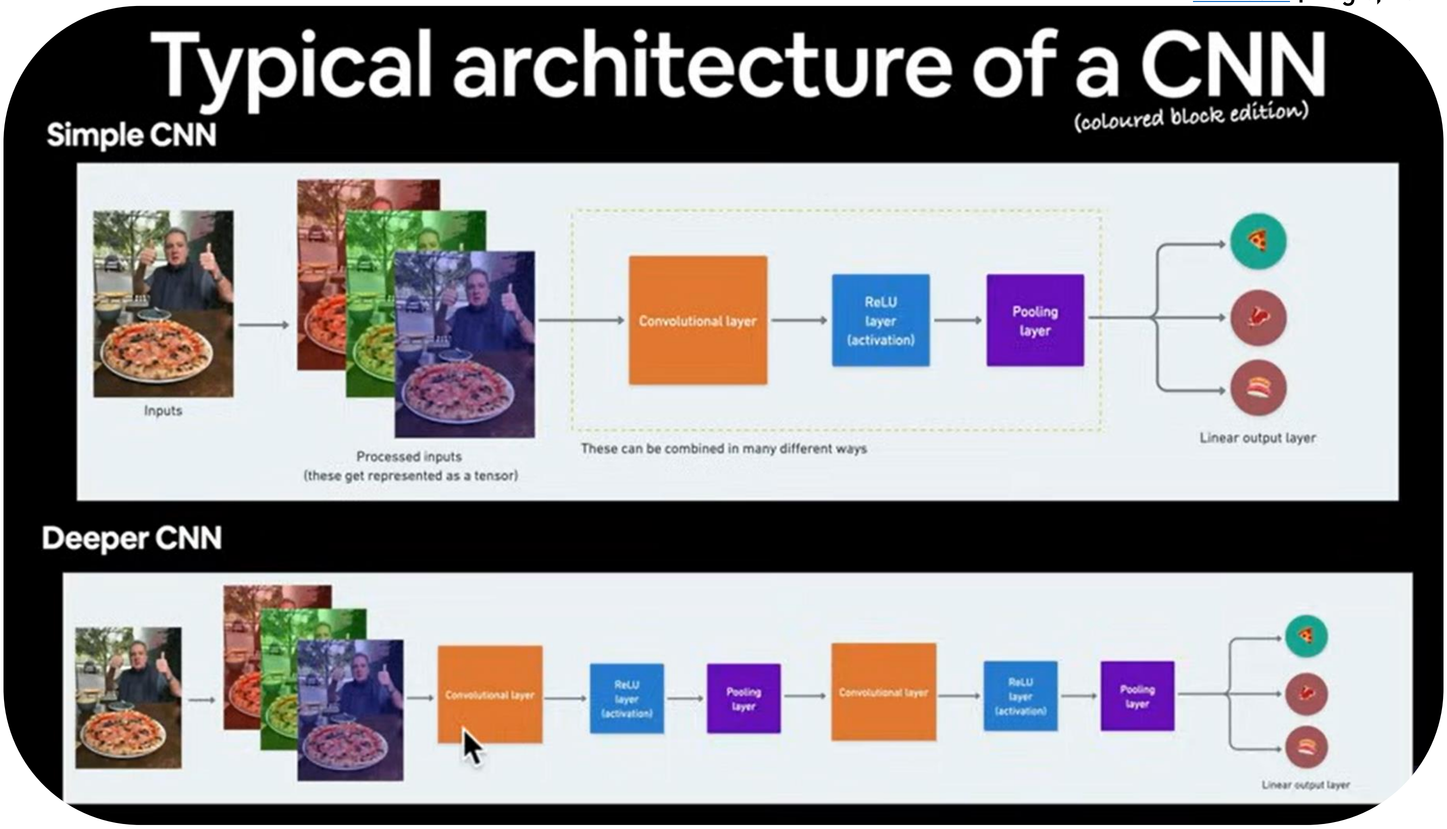
Imran Nawar | DIP Lab | Islamia College, Peshawar
LinkedIn | Aug 5, 2024

Computer Vision:





Imran Nawar | DIP Lab | Islamia College, Peshawar
[LinkedIn] | Aug 5, 2024

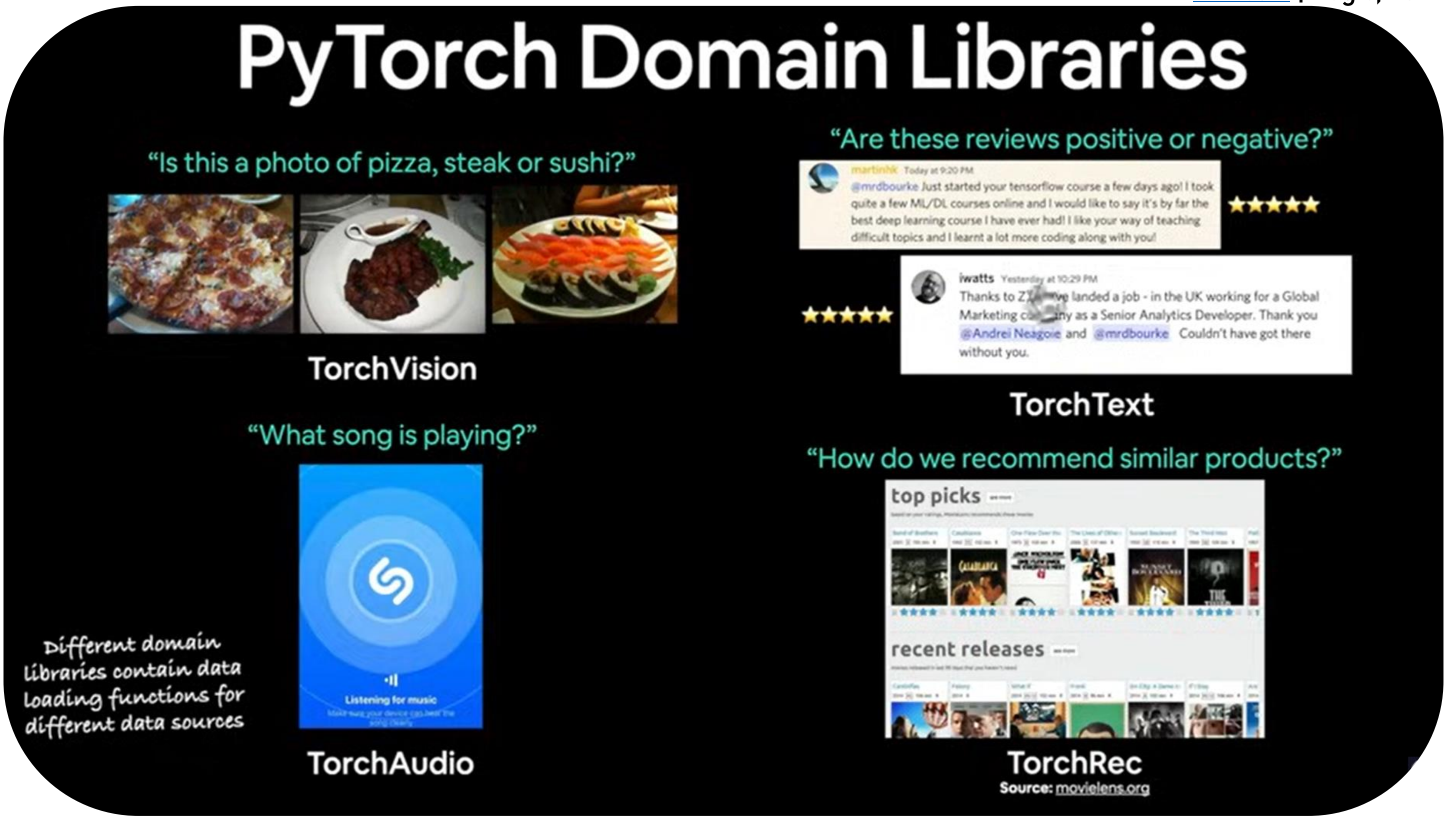


Breakdown of torch.nn.Conv2d layer

Example code: `torch.nn.Conv2d(in_channels=3, out_channels=10, kernel_size=(3, 3), stride=(1, 1), padding=0)`
Example 2 (same as above): `torch.nnConv2d(in_channels=3, out_channels=10, kernel_size=3, stride=1, padding=0)`

Hyperparameter name	What does it do?	Typical values
`in_channels`	Defines the number of input channels of the input data.	`1 (grayscale), 3 (RGB color images)`
`out_channels`	Defines the number output channels of the layer (could also be called hidden units).	`10, 128, 256, 512`
`kernel_size` (also referred to as filter size)	Determines the shape of the kernel (sliding windows) over the input.	`3, 5, 7 (lowers values learn smaller features, higher values learn larger features)`
`stride`	The number of steps a filter takes across an image at a time (e.g. if `strides=1`, a filter moves across an image 1 pixel at a time).	`1 (default), 2`
`padding`	Pads the target tensor with zeroes (if "`same`") to preserve input shape. Or leaves in the target tensor as is (if "`valid`"), lowering output shape.	`0, 1, "same", "valid"`

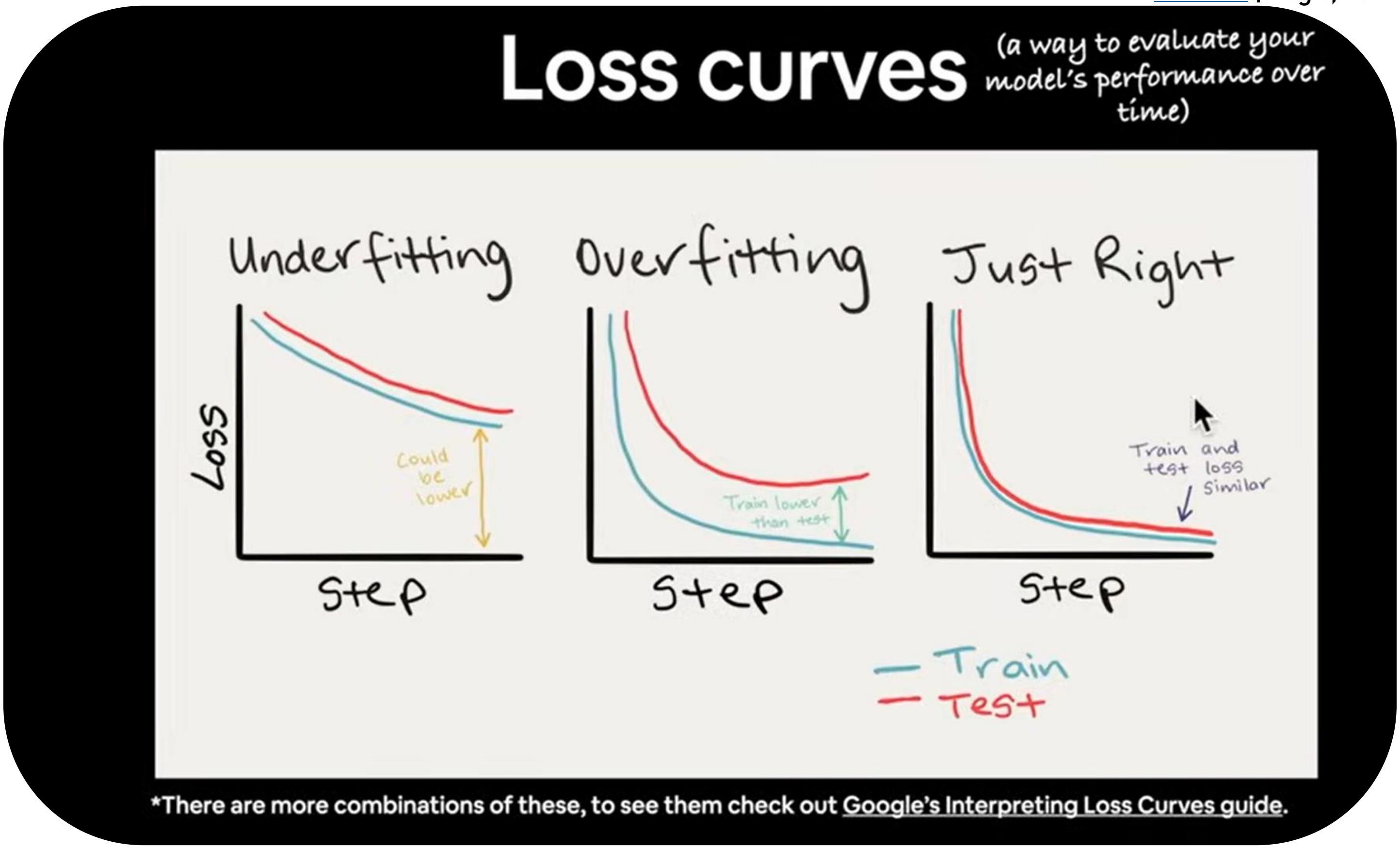
📖 **Resource:** For an interactive demonstration of the above hyperparameters, see the CNN Explainer website.



PyTorch Domain Libraries

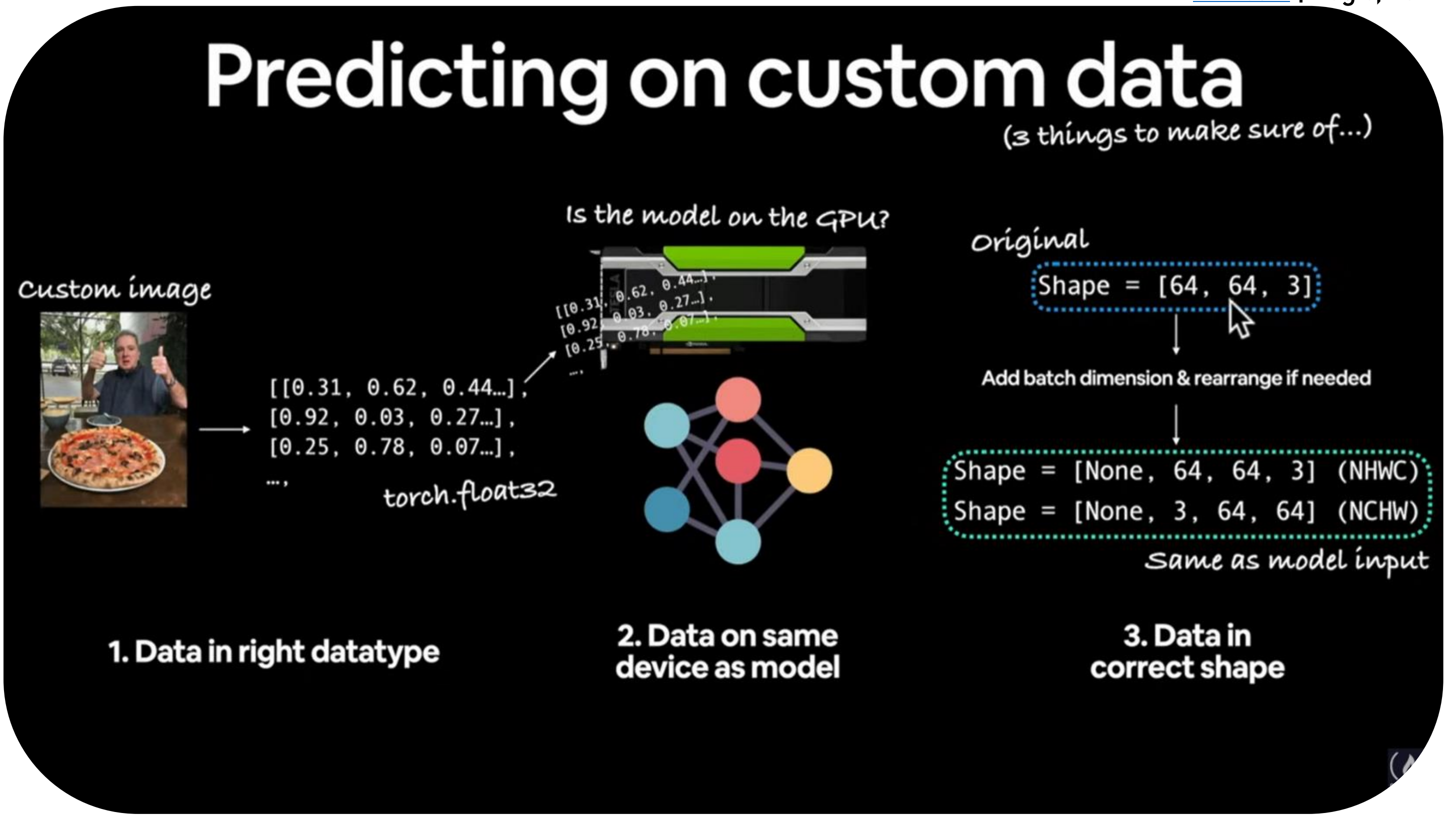
Problem Space	Pre-built Datasets and Fuctions
Vision	`torchvision.datasets`
Text	`torchtext.datasets`
Audio	`torchaudio.datasets`
Recommendation system	`torchrec.datasets`
Bonus	`TorchData`*

*`TorchData` contains many different helper functions for loading data and is currently in beta as of April 2022.



Dealing with overfitting

Method to improve a model (reduce overfitting)	What does it do?
Get more data	Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).
Data augmentation	Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.
Better data	Not all data samples are created equally. Removing poor samples from or adding better samples to your dataset can improve your model's performance.
Use transfer learning	Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.
Simplify your model	If the current model is already overfitting the training data, it may be too complicated of a model. This means it's learning the patterns of the data too well and isn't able to generalize well to unseen data. One way to simplify a model is to reduce the number of layers it uses or to reduce the number of hidden units in each layer.
Use learning rate decay	The idea here is to slowly decrease the learning rate as a model trains. This is akin to reaching for a coin at the back of a couch. The closer you get, the smaller your steps. The same with the learning rate, the closer you get to convergence, the smaller you'll want your weight updates to be.
Use early stopping	Early stopping stops model training *before* it begins to overfit. As in, say the model's loss has stopped decreasing for the past 10 epochs (this number is arbitrary), you may want to stop the model training here and go with the model weights that had the lowest loss (10 epochs prior).



Why GPUs are efficient for Deep Learning:

1. **Memory bandwidth:**
   - GPUs are optimized for high bandwidth, capable of fetching large amounts of memory at once (up to 750GB/s vs 50GB/s for CPUs).
2. **Latency hiding:**
   - GPUs use thread parallelism to hide their higher latency, effectively providing high bandwidth without significant latency drawbacks for large data operations.
3. **Register and cache advantage:**
   - GPUs have larger aggregate register memory (up to 14MB) operating at higher speeds (80TB/s) compared to CPUs (64-128KB at 10-20TB/s). This allows for efficient storage and reuse of data tiles for matrix operations.
4. **Parallelism:**
   - GPUs have many processing units (stream processors) each with their own small, fast register pack, allowing for efficient parallel computations.
5. **Memory hierarchy utilization:**
   - GPUs can effectively use their memory hierarchy (main memory, L1 cache, registers) to split large matrices into smaller tiles for fast computations.

These factors combine to make GPUs particularly well-suited for the large matrix operations common in deep learning, offering significant performance advantages over CPUs for these tasks.

Acknowledgement:

These notes were created based on insights gained from a course on YouTube. The images used have been sourced from the corresponding video lectures.

Reference:

1. https://www.youtube.com/watch?v=V_xro1bcAuA