# Common use of PyTorch from the basics to the advanced

Daesoo Lee

NTNU

NTNU
Norwegian University of Science and Technology

05.04.2022

NTNU

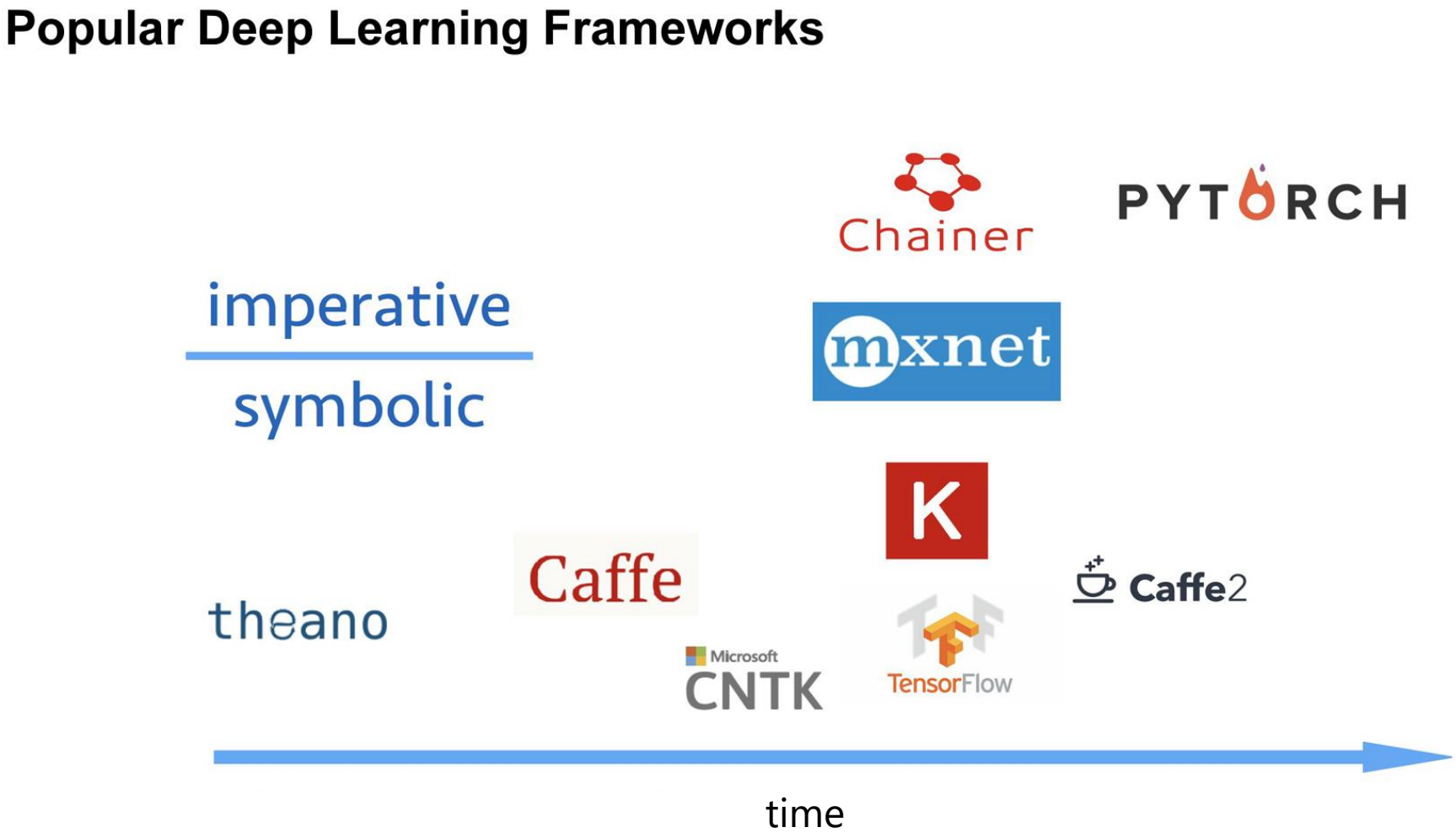
# Few Remarks

**This seminar assumes that**

- You have a basic knowledge about neural network stuff (*e.g.,* basic neural network, gradient descent).
- You've heard of PyTorch and knows what it's for.

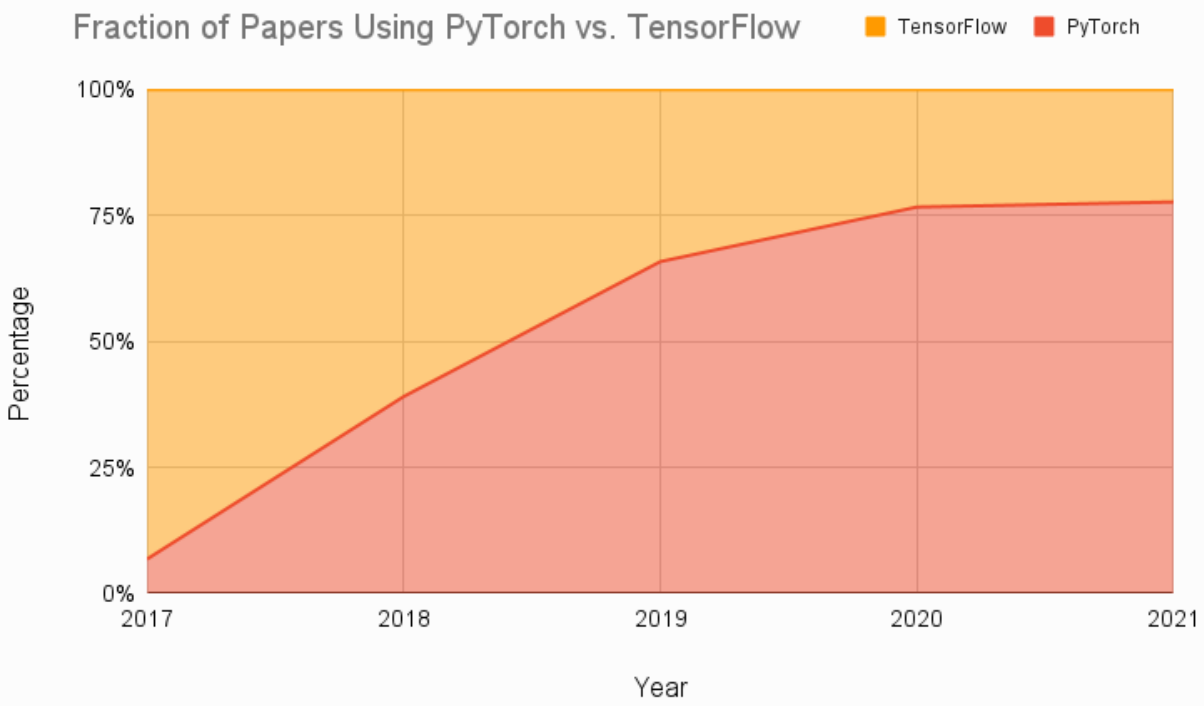
# Why PyTorch?

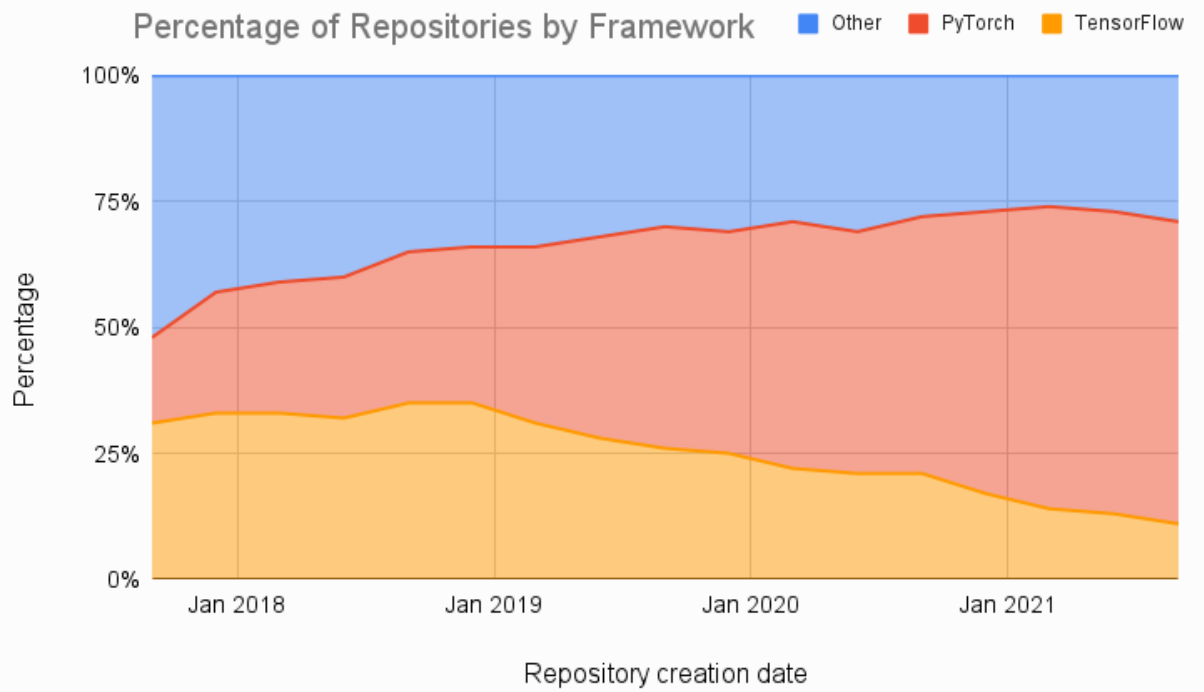
# Why PyTorch?




https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/

# Why PyTorch?

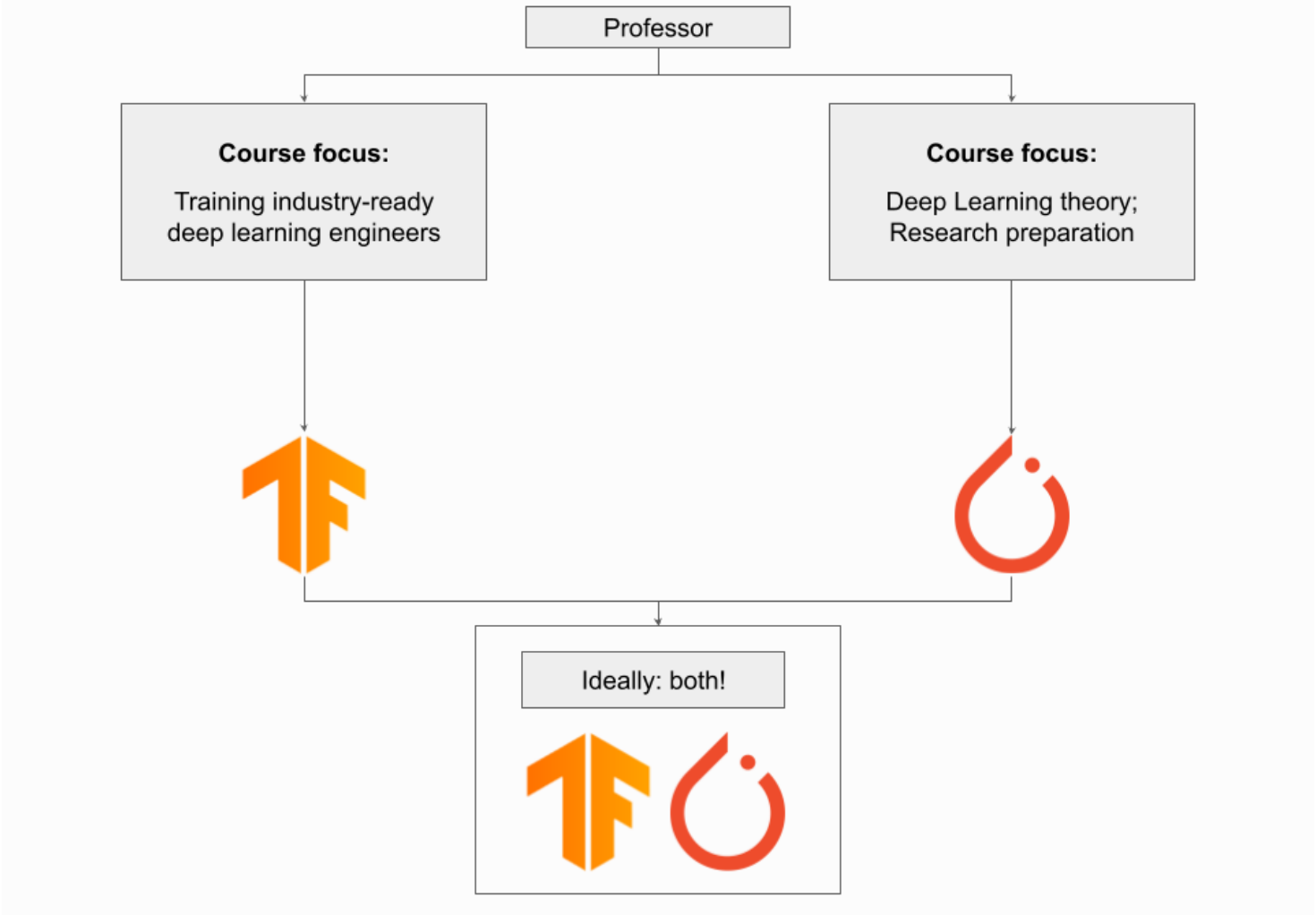



https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/

# Why PyTorch?

**What if You're a Professor? (if there's any here today)**



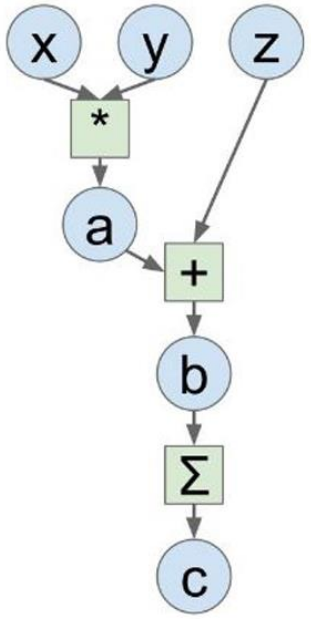

https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/

# Why PyTorch?

| | Keras | TensorFlow | PyTorch |
|---|---|---|---|
| **Level of API** | high-level API[1] | Both high & low level APIs | Lower-level API[2] |
| **Speed** | Slow | High | High |
| **Architecture** | Simple, more readable and concise | Not very easy to use | Complex[3] |
| **Debugging** | No need to debug | Difficult to debugging | Good debugging capabilities |
| **Dataset Compatibility** | Slow & Small | Fast speed & large | Fast speed & large datasets |
| **Popularity Rank** | 1 | 2 | 3 |
| **Uniqueness** | Multiple back-end support | Object Detection Functionality | Flexibility & Short Training Duration |
| **Created By** | Not a library on its own | Created by Google | Created by Facebook[4] |
| **Ease of use** | User-friendly | Incomprehensive API | Integrated with Python language |
| **Computational graphs used** | Static graphs | Static graphs | Dynamic computation graphs[5] |

# Why PyTorch?

## Computation Graph




$$\frac{\partial c}{\partial x}, \frac{\partial c}{\partial y}, \frac{\partial c}{\partial z}?$$

$$c = \sum(xy + z)$$

## Tensorflow

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

## PyTorch

```
import torch

N, D = 3, 4

x = torch.rand((N, D),requires_grad=True)
y = torch.rand((N, D),requires_grad=True)
z = torch.rand((N, D),requires_grad=True)

a =x * y
b =a + z
c=torch.sum(b)

c.backward()
```

https://www.cc.gatech.edu/classes/AY2021/cs7650_fall/slides/Introduction_to_PyTorch.pdf

# Quick Computation Speed Experiment for Matrix Multiplication

# Quick Computation Speed Experiment for Matrix Multiplication


```python
import numpy as np
import torch

# task: compute matrix multiplication C = AB
d = 3000

# using numpy
A = np.random.rand(d, d).astype(np.float32)
B = np.random.rand(d, d).astype(np.float32)
C = A.dot(B)

# using torch with gpu
A = torch.rand(d, d).cuda()
B = torch.rand(d, d).cuda()
C = torch.mm(A, B)
```

PYTORCH

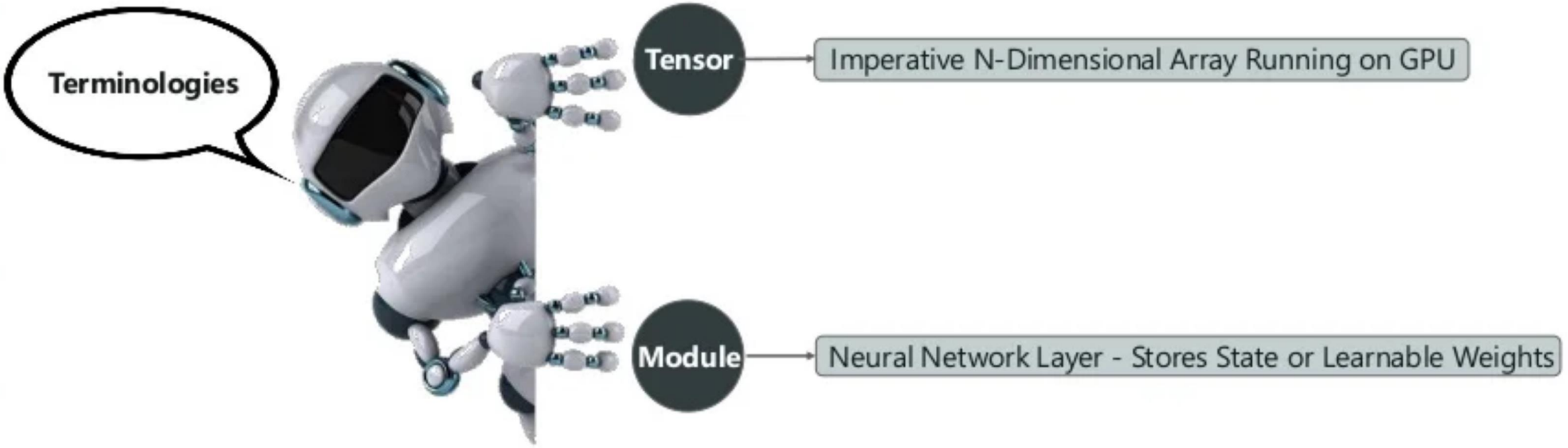
350 ms

0.1 ms

https://www.slideshare.net/EdurekaIN/pytorch-python-tutorial-deep-learning-using-pytorch-image-classifier-using-pytorch-edureka

# Major PyTorch Concepts

# Major PyTorch Concepts




https://www.slideshare.net/EdurekaIN/pytorch-python-tutorial-deep-learning-using-pytorch-image-classifier-using-pytorch-edureka

# Major PyTorch Concepts

## Tensors

Tensors are similar to NumPy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

Common operations for creation and manipulation of these Tensors are similar to those for ndarrays in NumPy. (rand, ones, zeros, indexing, slicing, reshape, transpose, cross product, matrix product, element wise multiplication)

https://www.cc.gatech.edu/classes/AY2021/cs7650_fall/slides/Introduction_to_PyTorch.pdf

# Major PyTorch Concepts

**Gradient Descent Update Rule**

$$w_{t+1} = w_t - \eta \nabla w_t$$

## Tensor Example: single variable

Let's see the following example: We're trying to compute a derivative of $y$ w.r.t $x$:

$$y(x) = x^2$$

If $x = 2$, then

$$y(2) = 2^2 = 4$$

$$\frac{dy(x)}{dx} = 2x$$

$$\frac{dy(2)}{dx} = 2(2) = 4$$

Let's see how we can compute them with pytorch:

```
import torch

x = torch.tensor(2., requires_grad=True)

y = x**2

y.backward()

x.grad
```

```
tensor(4.)
```

allows to track gradients
= making it a trainable parameter

Backpropagation = automatic differentiation

$$y(x) = x^3$$

$$\frac{dy(x)}{dx} = 3x^2$$

$$\frac{dy(2)}{dx} = 12$$

```
import torch

x = torch.tensor(2., requires_grad=True)

y = x**3

y.backward()

x.grad
```

```
tensor(12.)
```

https://daesoolee.tistory.com/123

# Major PyTorch Concepts

## Tensor Example: matrices

Attributes of a tensor 't':

- `t= torch.randn(1)`

`requires_grad`- making a trainable parameter

- By default False
- Turn on:
  - `t.requires_grad_()` or
  - `t = torch.randn(1, requires_grad=True)`
- Accessing tensor value:
  - t.data
- Accessingtensor gradient
  - t.grad

```
1  import torch
2
3  N, D = 3, 4
4
5  x = torch.rand((N, D),requires_grad=True)
6  y = torch.rand((N, D),requires_grad=True)
7  z = torch.rand((N, D),requires_grad=True)
8
9  a = x * y
10 b = a + z
11 c=torch.sum(b)
12
13 c.backward()
14
15
16 print(x.data)
17 print(x.grad)
```

```
tensor([[0.4118, 0.2576, 0.3470, 0.0240],
        [0.7797, 0.1519, 0.7513, 0.7269],
        [0.8572, 0.1165, 0.8596, 0.2636]])
tensor([[0.6855, 0.9696, 0.4295, 0.4961],
        [0.3849, 0.0825, 0.7400, 0.0036],
        [0.8104, 0.8741, 0.9729, 0.3821]])
```

https://www.cc.gatech.edu/classes/AY2021/cs7650_fall/slides/Introduction_to_PyTorch.pdf

# Major PyTorch Concepts

## Loading Data, Devices and CUDA

Numpy arrays to PyTorch tensors

- `torch.from_numpy(x_train)`
- `Returns a cpu tensor!`

PyTorchtensor to numpy

- `t.numpy()`

Using GPU acceleration

- `t.to()`
- Sends to whatever device (cudaor cpu)

Fallback to cpu if gpu is unavailable:

- `torch.cuda.is_available()`

Check cpu/gpu tensor OR numpyarray ?

- `type(t)` or `t.type()` returns
  - numpy.ndarray
  - torch.Tensor
    - CPU - torch.cpu.FloatTensor
    - GPU - torch.cuda.FloatTensor

```
x = np.random.rand(2, 2)
x
```

```
array([[0.18229139, 0.46024418],
       [0.32090798, 0.9557614 ]])
```

```
torch.cuda.is_available()
```

NVIDIA GeForce RTX 3060

```
True
```

```
device = torch.device(0)
device
```

```
device(type='cuda', index=0)
```

```
x = torch.from_numpy(x)
x
```

```
tensor([[0.1823, 0.4602],
        [0.3209, 0.9558]], dtype=torch.float64)
```

```
x.to(device)
```

```
tensor([[0.1823, 0.4602],
        [0.3209, 0.9558]], device='cuda:0', dtype=torch.float64)
```

https://www.cc.gatech.edu/classes/AY2021/cs7650_fall/slides/Introduction_to_PyTorch.pdf

# Major PyTorch Concepts

## Model

In PyTorch, a model is represented by a regular Python class that inherits from the Module class.

- Two components
  - `__init__(self)` : it defines the parts that make up the model- in our case, two parameters, a and b
  - `forward(self, x)` : it performs the actual computation, that is, it outputs a prediction, given the inputx

```
import torch.nn as nn

class ManualLinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        self.a = nn.Parameter(torch.randn(1, requires_grad=True))
        self.b = nn.Parameter(torch.randn(1, requires_grad=True))

    def forward(self, x):
        return self.a + self.b * x
```

```
reg = ManualLinearRegression()
reg
```

```
ManualLinearRegression()
```

```
for name, param in reg.named_parameters():
    print(name, param)
```

```
a Parameter containing:
tensor([0.0038], requires_grad=True)
b Parameter containing:
tensor([-0.4261], requires_grad=True)
```

```
x = torch.Tensor([1])
out = reg(x)
print(out)
```

```
tensor([-0.4223], grad_fn=<AddBackward0>)
```

https://www.cc.gatech.edu/classes/AY2021/cs7650_fall/slides/Introduction_to_PyTorch.pdf

# Major PyTorch Concepts

```
import torch.nn as nn

class ManualLinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        self.a = nn.Parameter(torch.randn(1, requires_grad=True))
        self.b = nn.Parameter(torch.randn(1, requires_grad=True))

    def forward(self, x):
        return self.a + self.b * x
```

```
reg = ManualLinearRegression()
reg
```

```
ManualLinearRegression()
```

```
for name, param in reg.named_parameters():
    print(name, param)
```

```
a Parameter containing:
tensor([0.0038], requires_grad=True)
b Parameter containing:
tensor([-0.4261], requires_grad=True)
```

```
x = torch.Tensor([1])
out = reg(x)
print(out)
```

```
tensor([-0.4223], grad_fn=<AddBackward0>)
```

```
linear = nn.Linear(1, 1)  # (in_size, out_size)

x = torch.Tensor([1])
out = linear(x)

print(out)
```

```
tensor([-1.5249], grad_fn=<AddBackward0>)
```

```
class LinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(1, 1)
        self.linear2 = nn.Linear(1, 1)

    def forward(self, x):
        return self.linear2(self.linear1(x))
```

```
reg = LinearRegression()
reg
```

```
LinearRegression(
  (linear1): Linear(in_features=1, out_features=1, bias=True)
  (linear2): Linear(in_features=1, out_features=1, bias=True)
)
```

```
x = torch.Tensor([1])
out = reg(x)
print(out)
```

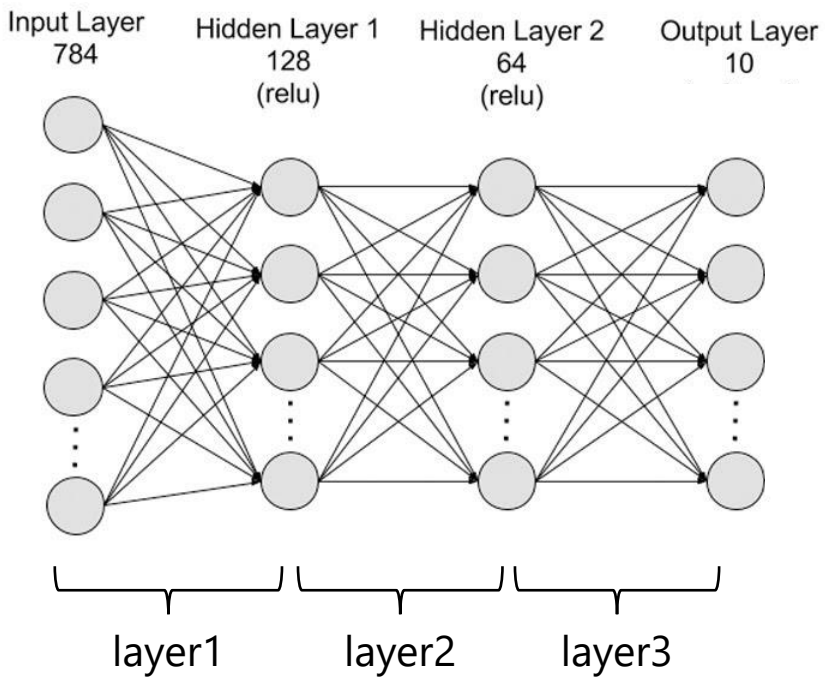
```
tensor([-0.3079], grad_fn=<AddBackward0>)
```

Essentially, “Module/Model” is a sort of wrapper.

https://www.cc.gatech.edu/classes/AY2021/cs7650_fall/slides/Introduction_to_PyTorch.pdf

# Creating Simple Neural Network (Multi-Layer Perceptron)

# Creating Simple Neural Network (Multi-Layer Perceptron)




```python
from torch import relu

class SimpleNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(784, 128)
        self.layer2 = nn.Linear(128, 64)
        self.layer3 = nn.Linear(64, 10)

    def forward(self, x):
        out = relu(self.layer1(x))
        out = relu(self.layer2(out))
        out = self.layer3(out)
        return out
```

```python
model = SimpleNN()
model
```

```
SimpleNN(
  (layer1): Linear(in_features=784, out_features=128, bias=
  (layer2): Linear(in_features=128, out_features=64, bias=T
  (layer3): Linear(in_features=64, out_features=10, bias=Tr
)
```

```python
batch_size = 32
n_predictors = 784
x = torch.rand(batch_size, n_predictors)

out = model(x)
print(out.shape)
```

```
torch.Size([32, 10])
```

```python
import torch.nn as nn

class SimpleNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(nn.Linear(784, 128), nn.ReLU(),
                                   nn.Linear(128, 64), nn.ReLU(),
                                   nn.Linear(64, 10))

    def forward(self, x):
        return self.model(x)
```

```python
model = nn.Sequential(nn.Linear(784, 128), nn.ReLU(),
                      nn.Linear(128, 64), nn.ReLU(),
                      nn.Linear(64, 10))
```

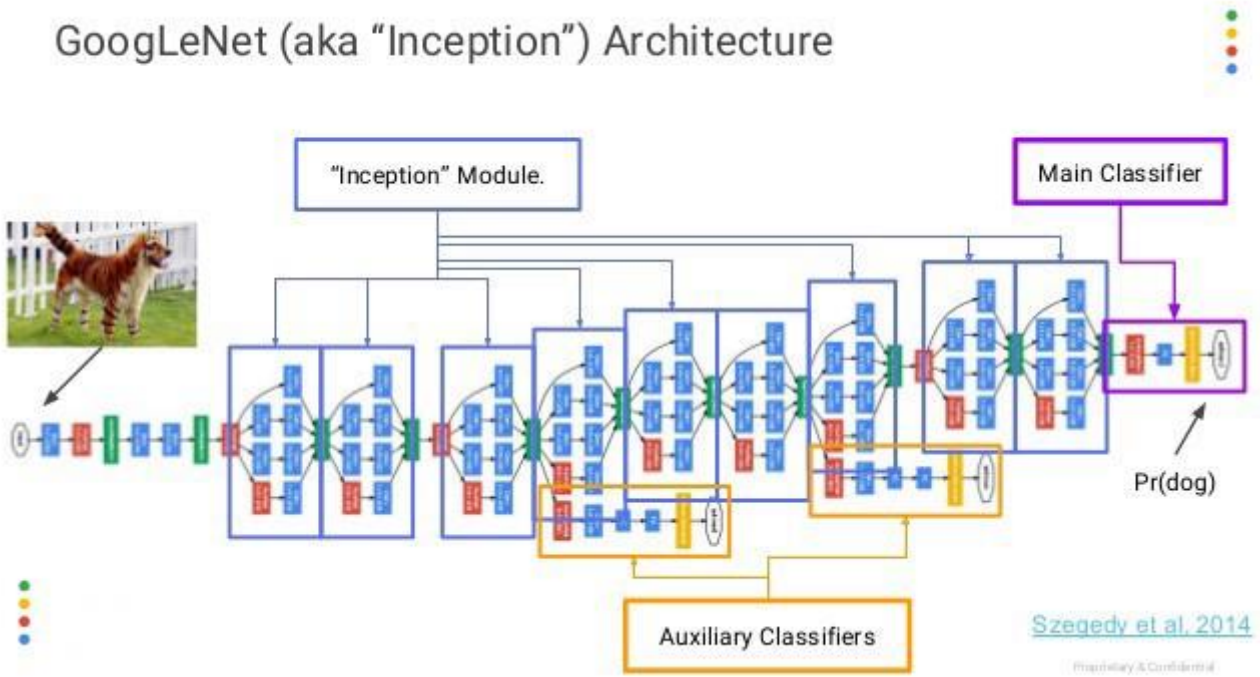
```python
out = model(x)
print(out.shape)
```

```
torch.Size([32, 10])
```

# Creating Simple Neural Network (Multi-Layer Perceptron)

**You'd need nn.Module if you're building some complex architecture**

**For instance,**



# Data Pipeline

# Data Pipeline

## Toy Dataset: Iris

- The iris dataset is a classic and very easy multi-class classification dataset.

| | |
|---|---|
| Classes | 3 |
| Samples per class | 50 |
| Samples total | 150 |
| Dimensionality | 4 |
| Features | real, positive |

```
from sklearn.datasets import load_iris

data = load_iris()
```

```
data['data']
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

```
data['target']
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

# Data Pipeline

```
from sklearn.datasets import load_iris

data = load_iris()
```

```
data['data']
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

| | |
|---|---|
| Classes | 3 |
| Samples per class | 50 |
| Samples total | 150 |
| Dimensionality | 4 |
| Features | real, positive |

```
data['target']
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
from torch.utils.data import Dataset

class ToyDataset(Dataset):
    def __init__(self):
        data = load_iris()

        self.X = data['data']
        self.Y = data['target']

    def __getitem__(self, idx):
        """
        set such that an individual sample is returned.
        """
        x = self.X[idx, :]
        y = self.Y[idx]
        return x, y

    def __len__(self):
        """
        an entire number of samples should be returned.
        """
        return self.Y.size
```

```
dataset = ToyDataset()
```

```
idx = 0
print(dataset[idx])
```

```
(array([5.1, 3.5, 1.4, 0.2
```

```
idx = 1
print(dataset[idx])
```

```
(array([4.9, 3. , 1.4, 0.2
```

**Dataset**

```
from torch.utils.data import DataLoader

dataset = ToyDataset()
data_loader = DataLoader(dataset,
                         batch_size=32,
                         shuffle=True)
```

```
for batch in data_loader:
    x, y = batch
    break
```

```
print('x.shape:', x.shape)
print('y.shape:', y.shape)
```

```
x.shape: torch.Size([32, 4])
y.shape: torch.Size([32])
```

**DataLoader**

```
# pseudo code
for batch in data_loader:          ← data pipeline
    x, y = batch

    yhat = model(x)                ← model forward propagation

    loss = loss_func(y, yhat)      ← loss function

    loss.backward()  # compute gradients / back-propagation
    update_model()  # by "gradient descent update rule"
```

# Optimizer
# (for Gradient Descent)

# Optimizer (for Gradient Descent)

```
model = nn.Sequential(nn.Linear(784, 128), nn.ReLU(),
                      nn.Linear(128, 64), nn.ReLU(),
                      nn.Linear(64, 10))
```

```
from torch import optim
optimizer = optim.SGD(model.parameters(), lr= 0.01)  # lr: learning rate
optimizer
```

Register model's parameters in the optimizer.

```
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.01
    momentum: 0
    nesterov: False
    weight_decay: 0
)
```

SGD: Stochastic Gradient Descent

## Algorithms

- Adadelta
- Adagrad
- LBFGS
- ★ Adam
- NAdam
- ★ AdamW
- RAdam
- SparseAdam
- RMSprop
- Adamax
- Rprop
- ASGD
- SGD

https://pytorch.org/docs/stable/optim.html

# Loss Functions

# Loss Functions

- Regression: Mean Squared Loss (MSE)
- Classification: Cross Entropy Loss
- etc.

## MSELOSS

CLASS `torch.nn.MSELoss(size_average=None, reduce=None, reduction='mean')` [SOURCE]

## CROSSENTROPYLOSS

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)` [SOURCE]

# Entire Training Pipeline

# Entire Training Pipeline

```python
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from sklearn.preprocessing import StandardScaler

class ToyDataset(Dataset):
    def __init__(self):
        data = load_iris()

        self.X = data['data']
        self.Y = data['target']

        # scale
        scaler = StandardScaler()
        self.X = scaler.fit_transform(self.X)

    def __getitem__(self, idx):
        """
        set such that an individual sample is returned.
        """
        x = self.X[idx, :]
        y = self.Y[idx]
        return x, y

    def __len__(self):
        """
        an entire number of samples should be returned.
        """
        return self.Y.size
```

** scaling added*

| | |
|---|---|
| Classes | 3 |
| Samples per class | 50 |
| Samples total | 150 |
| Dimensionality | 4 |
| Features | real, positive |

```python
in_size = 4  # num_predictors
h_size = 32  # hidden layer size
out_size = 3  # num_classes

model = nn.Sequential(nn.Linear(in_size, h_size), nn.ReLU(),
                      nn.Linear(h_size, h_size), nn.ReLU(),
                      nn.Linear(h_size, out_size))
```

```python
from torch import optim
optimizer = optim.Adam(model.parameters(),
                       lr= 0.001)  # lr: learning rate
optimizer
```

```python
loss_func = nn.CrossEntropyLoss()
```

*Training pipeline template*

```python
# Training
n_epochs = 20

step = 1
for epoch in range(n_epochs):

    for batch in data_loader:
        x, y = batch

        yhat = model(x.float())
        loss = loss_func(yhat, y.long())

        optimizer.zero_grad()  # reset the gradients (to zero) w
        loss.backward()  # compute gradients
        optimizer.step()  # update model's weight

        print(f'step: {step} | loss: {loss.item()}')
        step += 1
```

```
step: 1 | loss: 1.1227813959121704
step: 2 | loss: 1.1198639869689941
step: 3 | loss: 1.1341817378997803
step: 4 | loss: 1.107405424118042
step: 5 | loss: 1.1096519231796265
step: 96 | loss: 0.4173617660999298
step: 97 | loss: 0.4993354082107544
step: 98 | loss: 0.37763741612434387
step: 99 | loss: 0.4581018388271332
step: 100 | loss: 0.31296491622924805
```

# How to deal with a large dataset?

# How to deal with a large dataset?

```python
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from sklearn.preprocessing import StandardScaler

class ToyDataset(Dataset):
    def __init__(self):
        data = load_iris()

        self.X = data['data']
        self.Y = data['target']

        # scale
        scaler = StandardScaler()
        self.X = scaler.fit_transform(self.X)

    def __getitem__(self, idx):
        """
        set such that an individual sample is returned.
        """
        x = self.X[idx, :]
        y = self.Y[idx]
        return x, y

    def __len__(self):
        """
        an entire number of samples should be returned.
        """
        return self.Y.size
```

```python
class Dataset(Dataset):
  def __init__(self, csv_file, data_dir, transform=None):

    self.transform = transform

    self.data_dir=data_dir
    data_dircsv_file=os.path.join(self.data_dir,csv_file)

    self.data_name= pd.read_csv(data_dircsv_file)

    self.len=self.data_name.shape[0]

  def __len__(self):
     return self.len

  def __getitem__(self, idx):

    img_name=os.path.join(self.data_dir,self.data_name.iloc[idx, 1])
    image = Image.open(img_name)

    y = self.data_name.iloc[idx, 0]

    if self.transform:
        image = self.transform(image)
  return image, y
```

To load the following csv file :

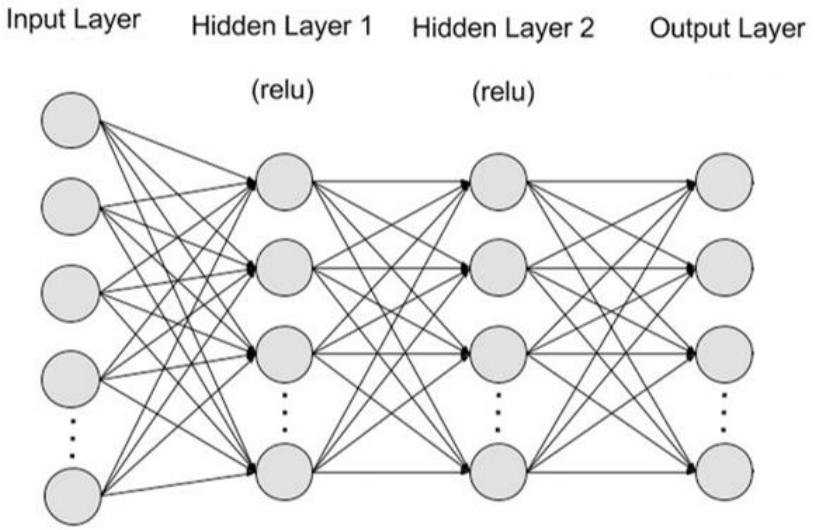
| | category | image |
|---|---|---|
| 0 | Ankle boot | img/fashion0.png |
| 1 | T-shirt | img/fashion1.png |
| 2 | T-shirt | img/fashion2.png |
| 3 | Dress | img/fashion3.png |
| 4 | T-shirt | img/fashion4.png |

This allows data to be fetched by one by one without loading all the images into the memory.

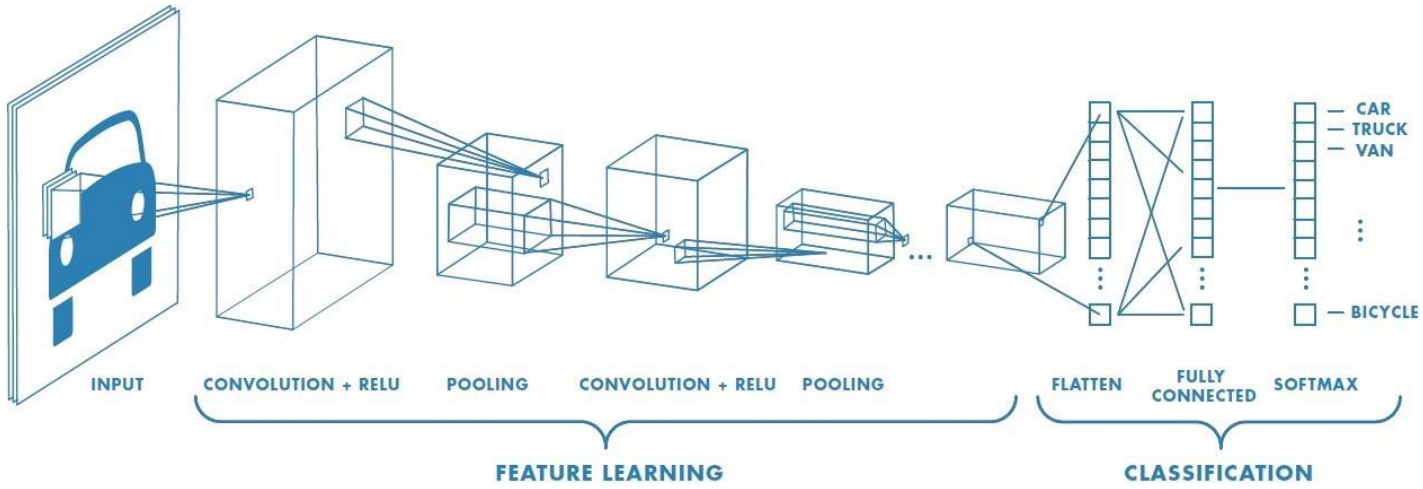
https://daesoolee.tistory.com/123

# CNN, RNN

# CNN, RNN




```
in_size = 4  # num_predictors
h_size = 32  # hidden layer size
out_size = 3  # num_classes

model = nn.Sequential(nn.Linear(in_size, h_size), nn.ReLU(),
                      nn.Linear(h_size, h_size), nn.ReLU(),
                      nn.Linear(h_size, out_size))
```




```
in_channels = 3
model = nn.Sequential(nn.Conv2D(in_channels, out_channels=32, kernel_size=3, stride=1),
                      nn.ReLU(),
                      nn.MaxPool2d(kernel_size=3, stride=2),

                      nn.Conv2D(in_channels=32, out_channels=64, kernel_size=3, stride=1),
                      nn.ReLU(),
                      nn.MaxPool2d(kernel_size=3, stride=2),

                      nn.Conv2D(in_channels=64, out_channels=128, kernel_size=3, stride=1),
                      nn.ReLU(),
                      nn.MaxPool2d(kernel_size=3, stride=2),

                      nn.Flatten(start_dim=1),

                      nn.Linear(...),
                      nn.ReLU(),
                      nn.Linear(...)
                      )
```

# CNN, RNN

## Pretrained Models

# MODELS AND PRE-TRAINED WEIGHTS

The `torchvision.models` subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection, video classification, and optical flow.

**NOTE**

Backward compatibility is guaranteed for loading a serialized `state_dict` to the model created using old PyTorch version. On the contrary, loading entire saved models or serialized `ScriptModules` (seralized using older versions of PyTorch) may not preserve the historic behaviour. Refer to the following documentation

## Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet
- EfficientNet
- RegNet
- VisionTransformer
- ConvNeXt

*Trained on lots of images*

```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet_v2 = models.mobilenet_v2(pretrained=True)
mobilenet_v3_large = models.mobilenet_v3_large(pretrained=True)
mobilenet_v3_small = models.mobilenet_v3_small(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
efficientnet_b0 = models.efficientnet_b0(pretrained=True)
efficientnet_b1 = models.efficientnet_b1(pretrained=True)
efficientnet_b2 = models.efficientnet_b2(pretrained=True)
efficientnet_b3 = models.efficientnet_b3(pretrained=True)
efficientnet_b4 = models.efficientnet_b4(pretrained=True)
efficientnet_b5 = models.efficientnet_b5(pretrained=True)
efficientnet_b6 = models.efficientnet_b6(pretrained=True)
efficientnet_b7 = models.efficientnet_b7(pretrained=True)
regnet_y_400mf = models.regnet_y_400mf(pretrained=True)
regnet_y_800mf = models.regnet_y_800mf(pretrained=True)
regnet_y_1_6gf = models.regnet_y_1_6gf(pretrained=True)
regnet_y_3_2gf = models.regnet_y_3_2gf(pretrained=True)
regnet_y_8gf = models.regnet_y_8gf(pretrained=True)
regnet_y_16gf = models.regnet_y_16gf(pretrained=True)
regnet_y_32gf = models.regnet_y_32gf(pretrained=True)
regnet_x_400mf = models.regnet_x_400mf(pretrained=True)
regnet_x_800mf = models.regnet_x_800mf(pretrained=True)
regnet_x_1_6gf = models.regnet_x_1_6gf(pretrained=True)
regnet_x_3_2gf = models.regnet_x_3_2gf(pretrained=True)
regnet_x_8gf = models.regnet_x_8gf(pretrained=True)
regnet_x_16gf = models.regnet_x_16gf(pretrainedTrue)
regnet_x_32gf = models.regnet_x_32gf(pretrained=True)
vit_b_16 = models.vit_b_16(pretrained=True)
vit_b_32 = models.vit_b_32(pretrained=True)
vit_l_16 = models.vit_l_16(pretrained=True)
vit_l_32 = models.vit_l_32(pretrained=True)
convnext_tiny = models.convnext_tiny(pretrained=True)
convnext_small = models.convnext_small(pretrained=True)
convnext_base = models.convnext_base(pretrained=True)
convnext_large = models.convnext_large(pretrained=True)
```

https://pytorch.org/vision/stable/models.html

# CNN, RNN

**RNN, LSTM, GRU**

## RNN

CLASS torch.nn.RNN(*args, **kwargs) [SOURCE]

## LSTM

CLASS torch.nn.LSTM(*args, **kwargs) [SOURCE]

## GRU

CLASS torch.nn.GRU(*args, **kwargs) [SOURCE]

```
model = nn.Sequential(nn.LSTM(...),
                      ...)
```

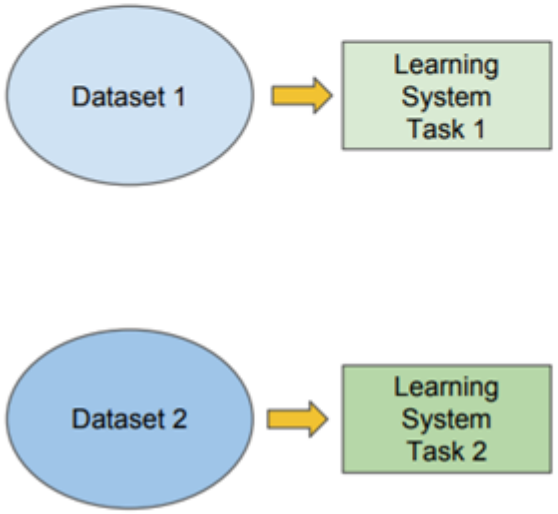
https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

# Transfer Learning

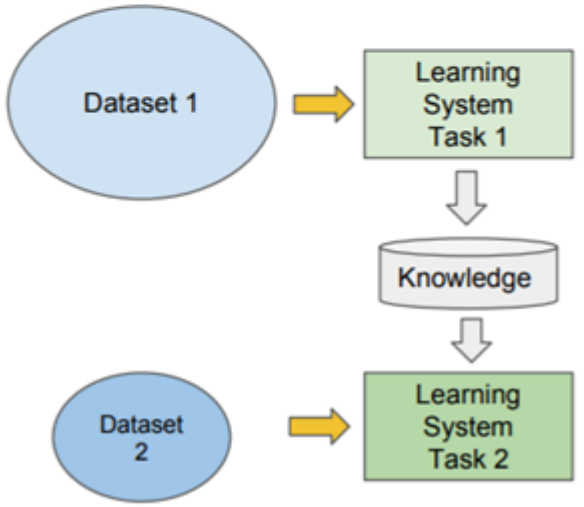
# Transfer Learning

## Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks




- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data




https://medium.com/@agarwallashivam/face-recognition-using-transfer-learning-b8e36e6a15f1

# Transfer Learning

## MODELS AND

The `torchvision.models` subp
including: image classification, p
segmentation, person keypoint

• NOTE

Backward compatibility is gu
using old PyTorch version. C
`ScriptModules` (seralized u
behaviour. Refer to the follo

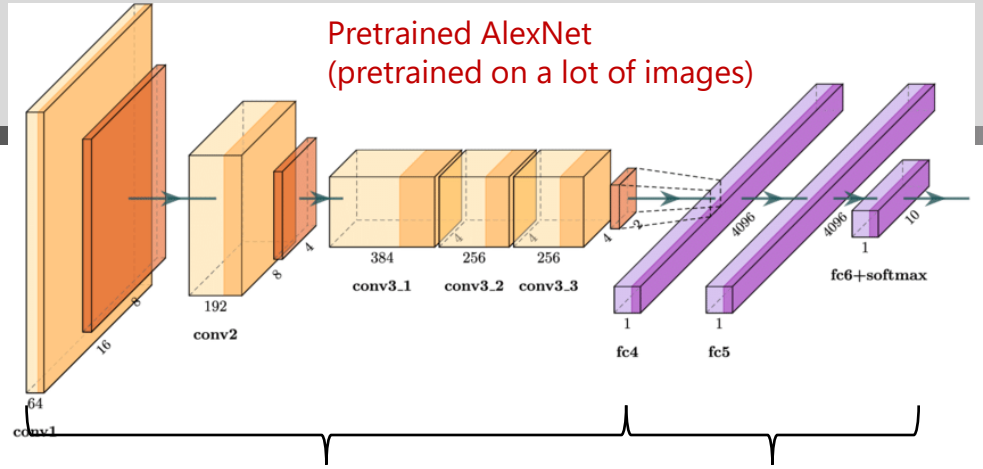
### Classification

The models subpackage contair
classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet
- EfficientNet
- RegNet
- VisionTransformer
- ConvNeXt

```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet_v2 = models.mobilenet_v2(pretrained=True)
mobilenet_v3_large = models.mobilenet_v3_large(pretrained=True)
mobilenet_v3_small = models.mobilenet_v3_small(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
efficientnet_b0 = models.efficientnet_b0(pretrained=True)
efficientnet_b1 = models.efficientnet_b1(pretrained=True)
efficientnet_b2 = models.efficientnet_b2(pretrained=True)
efficientnet_b3 = models.efficientnet_b3(pretrained=True)
efficientnet_b4 = models.efficientnet_b4(pretrained=True)
efficientnet_b5 = models.efficientnet_b5(pretrained=True)
efficientnet_b6 = models.efficientnet_b6(pretrained=True)
efficientnet_b7 = models.efficientnet_b7(pretrained=True)
regnet_y_400mf = models.regnet_y_400mf(pretrained=True)
regnet_y_800mf = models.regnet_y_800mf(pretrained=True)
regnet_y_1_6gf = models.regnet_y_1_6gf(pretrained=True)
regnet_y_3_2gf = models.regnet_y_3_2gf(pretrained=True)
regnet_y_8gf = models.regnet_y_8gf(pretrained=True)
regnet_y_16gf = models.regnet_y_16gf(pretrained=True)
regnet_y_32gf = models.regnet_y_32gf(pretrained=True)
regnet_x_400mf = models.regnet_x_400mf(pretrained=True)
regnet_x_800mf = models.regnet_x_800mf(pretrained=True)
regnet_x_1_6gf = models.regnet_x_1_6gf(pretrained=True)
regnet_x_3_2gf = models.regnet_x_3_2gf(pretrained=True)
regnet_x_8gf = models.regnet_x_8gf(pretrained=True)
regnet_x_16gf = models.regnet_x_16gf(pretrainedTrue)
regnet_x_32gf = models.regnet_x_32gf(pretrained=True)
vit_b_16 = models.vit_b_16(pretrained=True)
vit_b_32 = models.vit_b_32(pretrained=True)
vit_l_16 = models.vit_l_16(pretrained=True)
vit_l_32 = models.vit_l_32(pretrained=True)
convnext_tiny = models.convnext_tiny(pretrained=True)
convnext_small = models.convnext_small(pretrained=True)
convnext_base = models.convnext_base(pretrained=True)
convnext_large = models.convnext_large(pretrained=True)
```

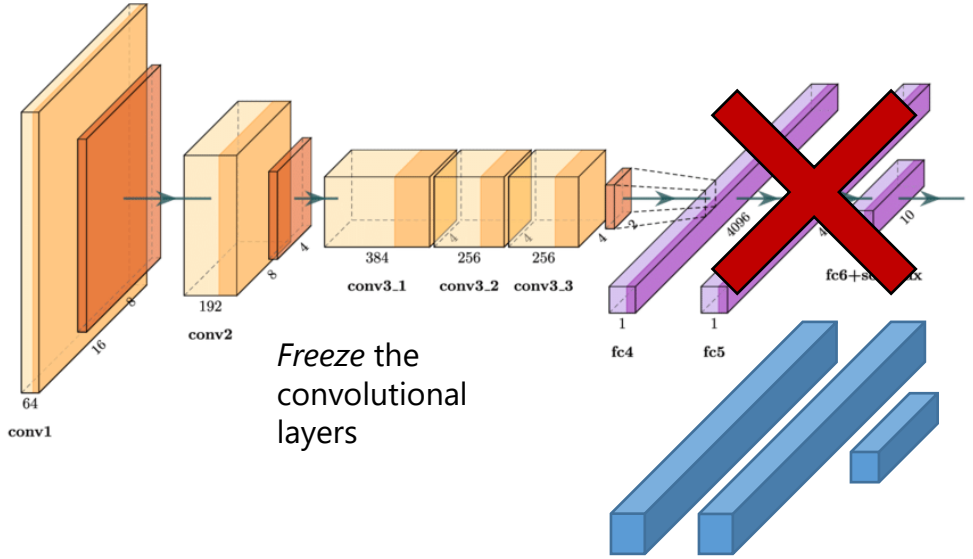
**1**




Learned to capture low and high-level semantics of images

The last linear layer(s) usually learned to perform a downstream task

**2**




*Freeze* the convolutional layers

Discard the last *N* layers.

Add new layers.

**3**

Train the newly-added layers only.

*NB! There are several ways to do it depending on your dataset size. But not getting into details.*

# Transformer

# CNN, RNN, Transformer

## Transformer

BERT ←

CLASS `torch.nn.TransformerEncoder(encoder_layer, num_layers, norm=None)` [SOURCE]

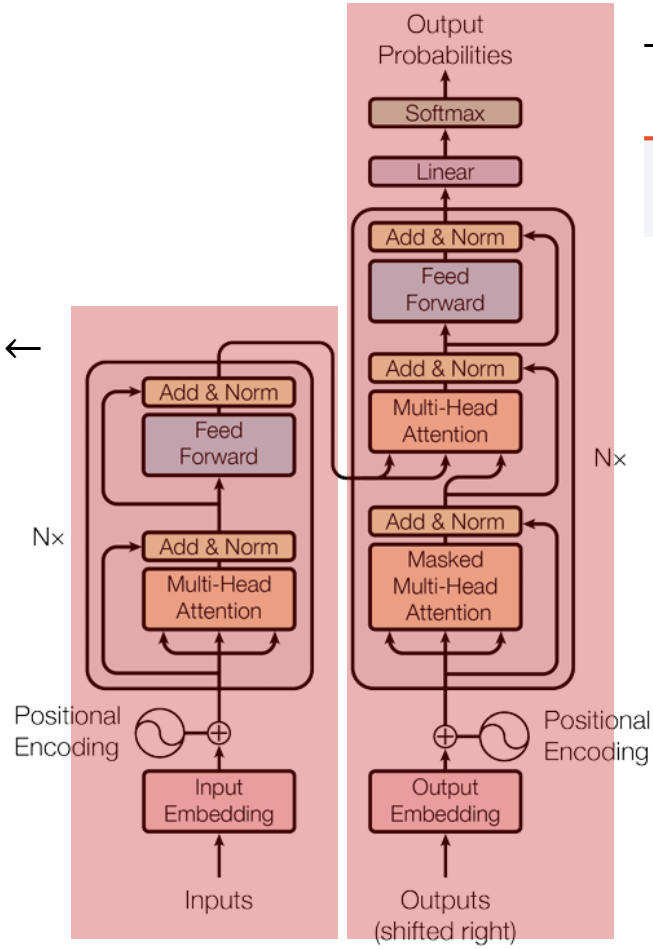



Figure 1: The Transformer - model architecture.

→ GPT

CLASS `torch.nn.TransformerDecoder(decoder_layer, num_layers, norm=None)` [SOURCE]

**Attention Is All You Need**


**Ashish Vaswani*** Google Brain avaswani@google.com

**Noam Shazeer*** Google Brain noam@google.com

**Niki Parmar*** Google Research nikip@google.com

**Jakob Uszkoreit*** Google Research usz@google.com

**Llion Jones*** Google Research llion@google.com

**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu

**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com

**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com

# CNN, RNN, Transformer

## Transformer for Image

- Transformer for Image: Vision Transformer

PyTorch

Get Started | Ecosystem | Mobile | Blog | Tutorials | Docs

0.12 ▼

Search Docs

Package Reference

Transforming and augmenting images

Models and pre-trained weights

Docs > Models and pre-trained weights

MODELS AND PRE-TRAINED WEIGHTS

The `torchvision.models` subpackage contains definitions of models for addressing differ
including: image classification, pixelwise semantic segmentation, object detection, instance
segmentation, person keypoint detection, video classification, and optical flow.

```
vit_b_16 = models.vit_b_16(pretrained=True)
vit_b_32 = models.vit_b_32(pretrained=True)
vit_l_16 = models.vit_l_16(pretrained=True)
vit_l_32 = models.vit_l_32(pretrained=True)
```

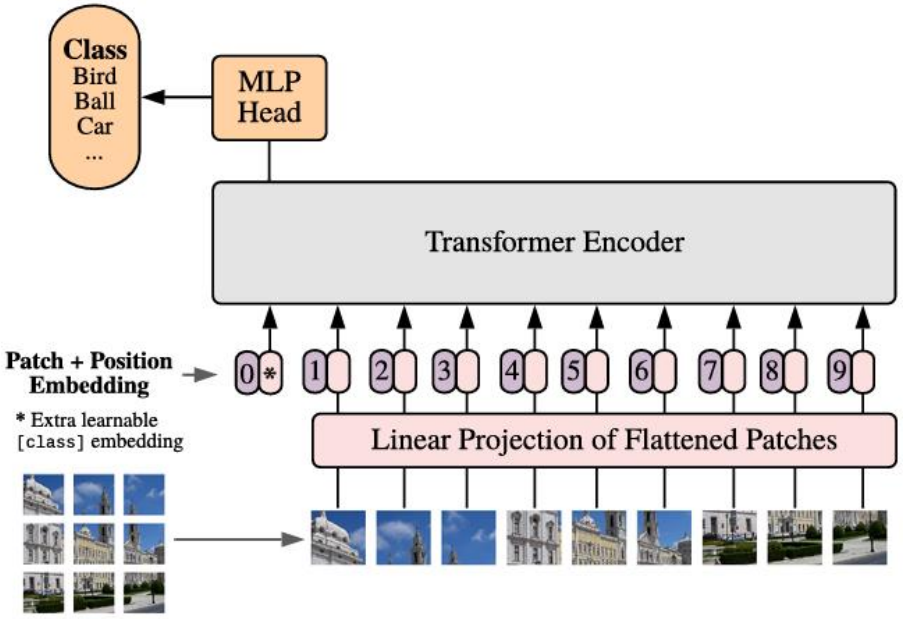
AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

**Alexey Dosovitskiy*,†, Lucas Beyer*, Alexander Kolesnikov*, Dirk Weissenborn*, Xiaohua Zhai*, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby*,†**

*equal technical contribution, †equal advising

Google Research, Brain Team

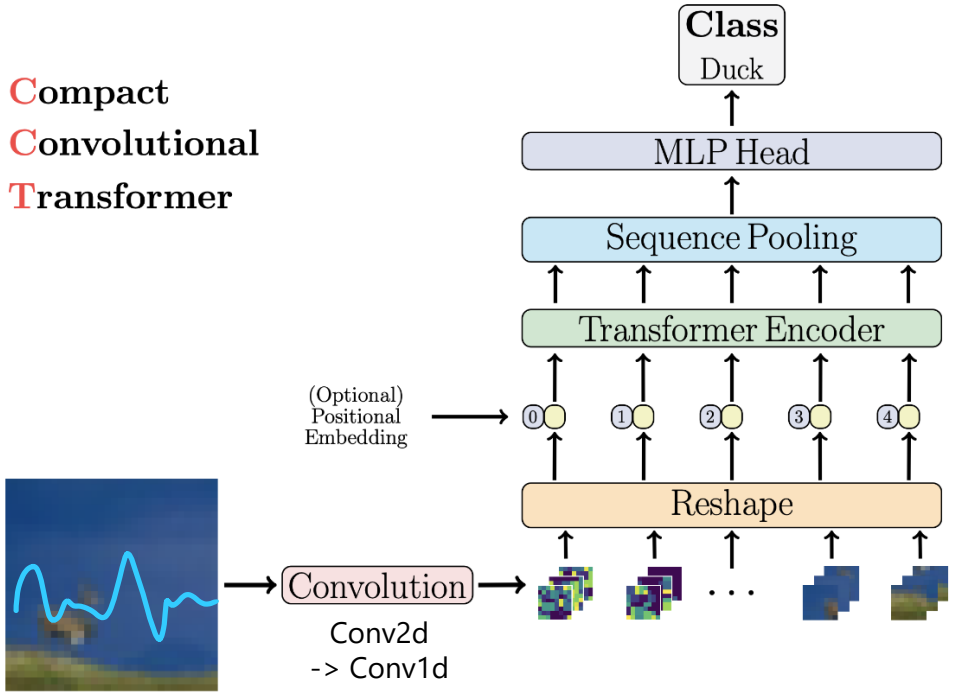
**Vision Transformer (ViT)**

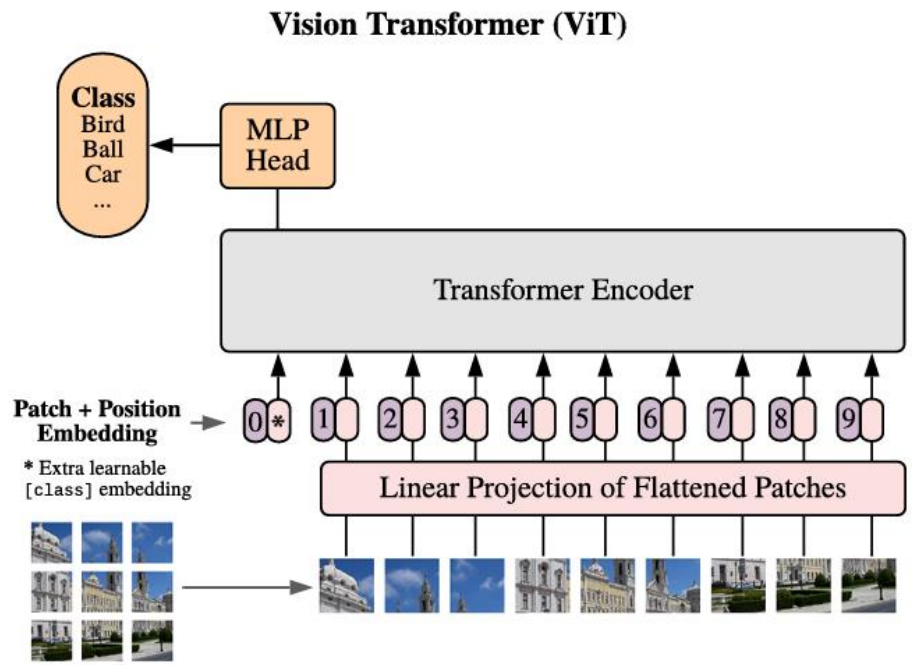


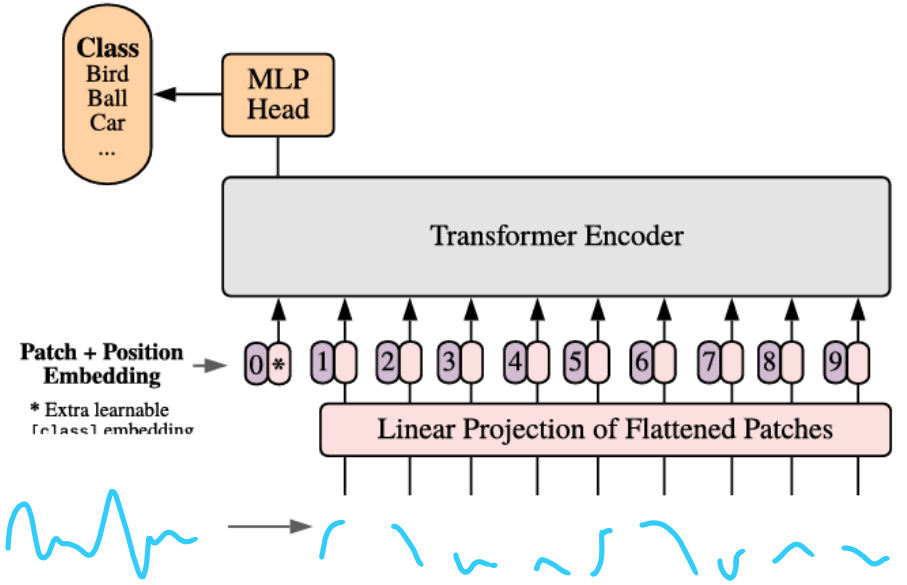
# CNN, RNN, Transformer

## Transformer for Time Series

- Not much of standardized models.
- You can build one by yourself.



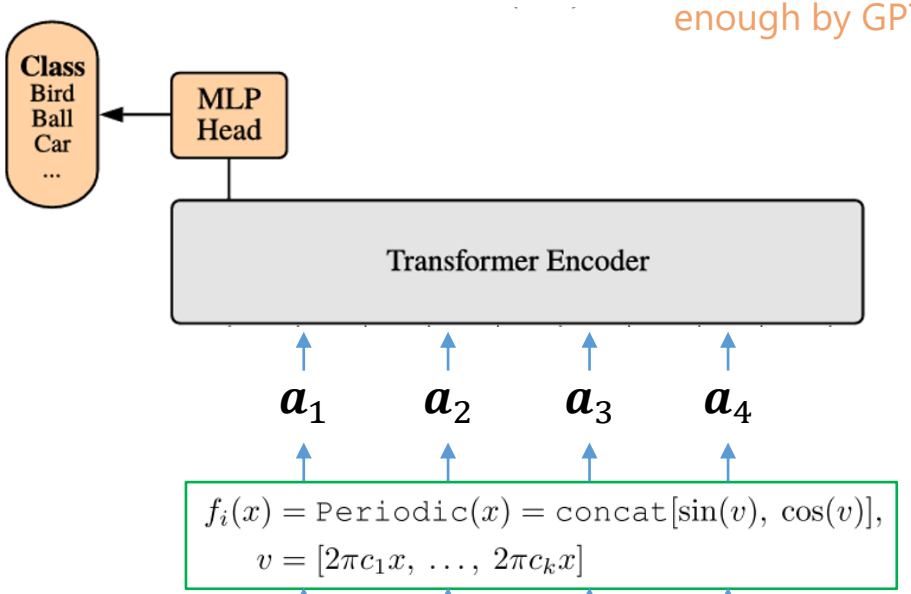
# CNN, RNN, Transformer

## Transformer for Tabular Data

- Relatively new research direction.
- Previous studies have shown that transformer models beat basic neural network models.
- Not much of standardized models.
- You can build one by yourself.

Try using the power of Transformer for your Tabular data ☺

The power of Transformer is proven enough by GPT.




$c$: trainable parameter
$x$: predictor input

| Iris | Sepal.L ($a_1$) | Sepal.W ($a_2$) | Petal.L ($a_3$) | Petal.W ($a_4$) | Class |
|---|---|---|---|---|---|
| $u_1$ | 5.5 cm | 4.2 cm | 1.4 cm | 0.2 cm | Setosa |
| $u_2$ | 5.0 cm | 3.4 cm | 1.5 cm | 0.2 cm | Setosa |
| $u_3$ | 6.1 cm | 2.9 cm | 4.7 cm | 1.4 cm | Versicolor |
| $u_4$ | 6.2 cm | 2.2 cm | 4.5 cm | 1.5 cm | Versicolor |
| $u_5$ | 6.3 cm | 2.7 cm | 4.9 cm | 1.8 cm | Virginica |
| $u_6$ | 6.0 cm | 2.2 cm | 5.0 cm | 1.5 cm | Virginica |



Gorishniy, Y., Rubachev, I., & Babenko, A. (2022). On Embeddings for Numerical Features in Tabular Deep Learning. *arXiv preprint arXiv:2203.05556*.

# Language Models: GPT, BERT

# Language Models

## Pretrained Language Models?

- *e.g.,* Pretrained GPT

https://www.crunchbase.com › organization › hugging-...

Hugging Face - Crunchbase Company Profile & Funding

**Hugging Face** is an open-source & platform provider of machine learning technologies.
**Hugging Face** was launched in 2016 and is headquartered in New York.

Quick tour

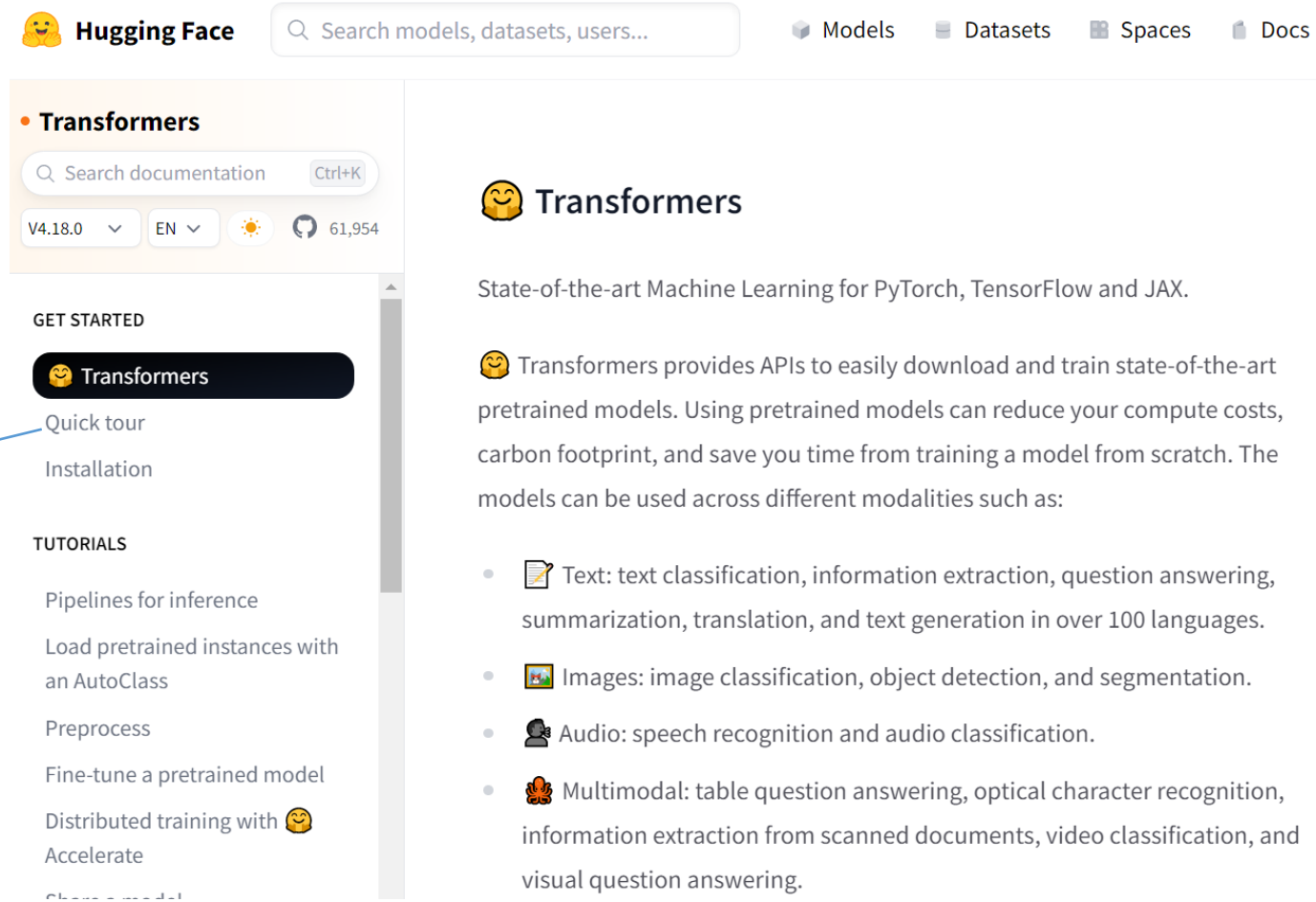
```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
```

```
classifier("I'm so happy to do PhD at NTNU.")
```

```
[{'label': 'POSITIVE', 'score': 0.999579131603241}]
```

```
classifier("I paid so much tax this year.")
```

```
[{'label': 'NEGATIVE', 'score': 0.9869840145111084}]
```

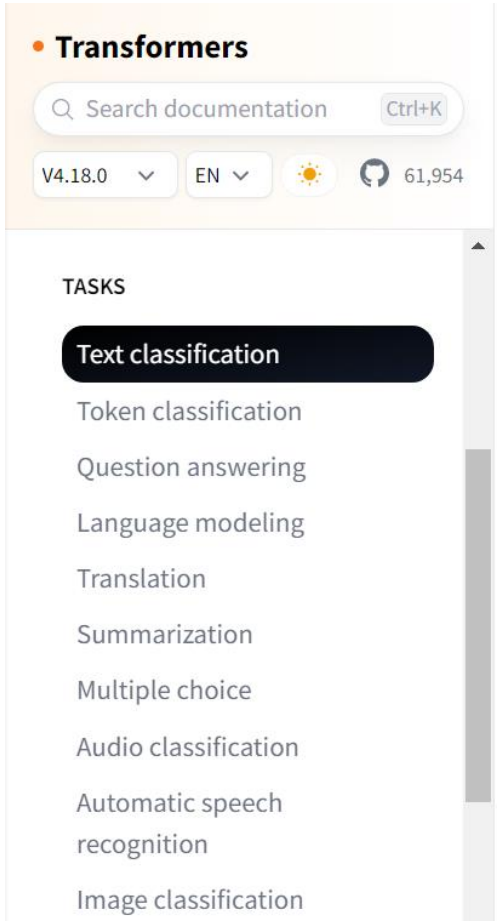



https://huggingface.co/docs/transformers/index

# Language Models

**Model for Different Tasks**

**& Different Models**




MODELS

ALBERT
Auto Classes
BART
BARThez
BARTpho
BEiT
BERT
Bertweet
BertGeneration
BertJapanese
BigBird
BigBirdPegasus
Blenderbot
Blenderbot Small
BORT
ByT5
CamemBERT
CANINE
ConvNeXT
CLIP
ConvBERT
CPM
CTRL
Data2Vec
DeBERTa
DeBERTa-v2
Decision Transformer
DeiT
DETR
DialoGPT
DistilBERT
DiT
DPR
DPT

DPT
ELECTRA
Encoder Decoder Models
FlauBERT
FNet
FSMT
Funnel Transformer
GLPN
HerBERT
I-BERT
ImageGPT
LayoutLM
LayoutLMV2
LayoutXLM
LED
Longformer
LUKE
LXMERT
MarianMT
MaskFormer
M2M100
MBart and MBart-50
MegatronBERT
MegatronGPT2
mLUKE
MobileBERT
MPNet
MT5
Nyströmformer
OpenAI GPT
OpenAI GPT2
GPT-J
GPT Neo
Hubert
Perceiver
Pegasus

PhoBERT
PLBart
PoolFormer
ProphetNet
QDQBert
RAG
REALM
Reformer
RemBERT
ResNet
RetriBERT
RoBERTa
RoFormer
SegFormer
SEW
SEW-D
Speech Encoder Decoder Models
Speech2Text
Speech2Text2
Splinter
SqueezeBERT
Swin Transformer
T5
T5v1.1
TAPAS
Transformer XL
TrOCR
UniSpeech
UniSpeech-SAT
VAN
ViLT
Vision Encoder Decoder Models
Vision Text Dual Encoder
Vision Transformer (ViT)
ViTMAE

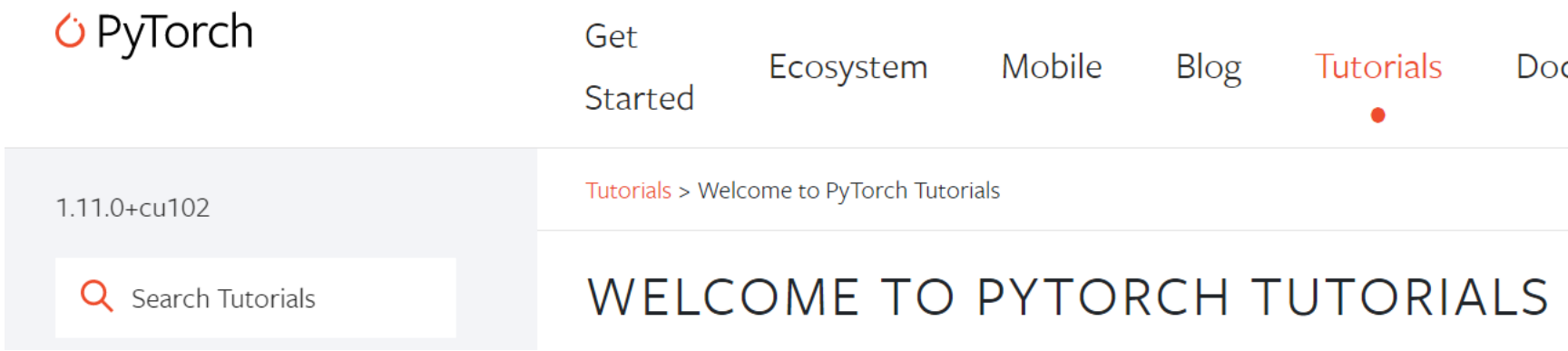
ViTMAE
VisualBERT
Wav2Vec2
Wav2Vec2Phoneme
WavLM
XGLM
XLM
XLM-ProphetNet
XLM-RoBERTa
XLM-RoBERTa-XL
XLNet
XLSR-Wav2Vec2
XLS-R
YOSO

https://huggingface.co/docs/transformers/index

# Recommend Study Materials

# Recommend Study Materials

- Deep Neural Networks with PyTorch in COURSERA
- Examples from PyTorch official documentation
- StackOverFlow
- etc.

# Highly Recommended Tools and Libraries

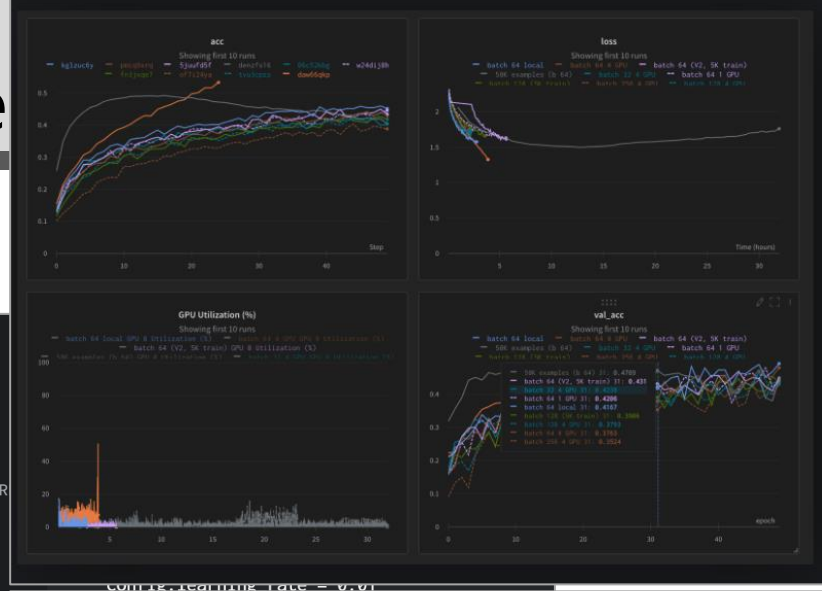
(that can make your life easier.)

# Highly Recommended Tools and Libraries

- **Weights and Biases (wandb)**: cloud-based result tracking tool
- **PyTorch lightning**: simplified version of PyTorch

# Highly Recommende

## Weights and Biases (wandb)



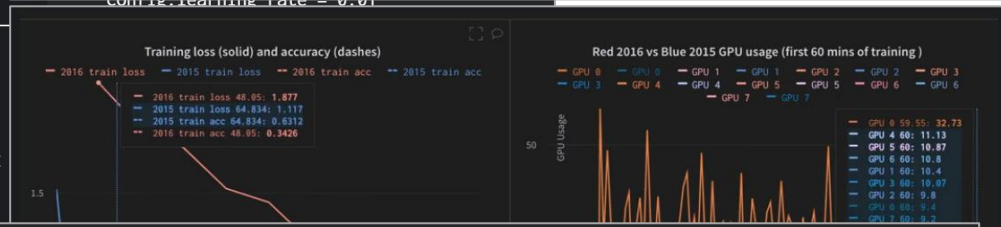

## Visualize and compare every experiment

See model metrics stream live into interactive graphs and tables. It is easy to see how your latest model is performing compared to previous experiments, no matter where you are training your models.

## Track, compare, and visualize with 5 lines of code

Add a few lines to your script to start logging results. Our lightweight integration works with any Python script.

- ANY FRAMEWOR
- TENSORFLOW
- PYTORCH
- KERAS
- SCIKIT
- HUGGING FACE
- XGBOOST

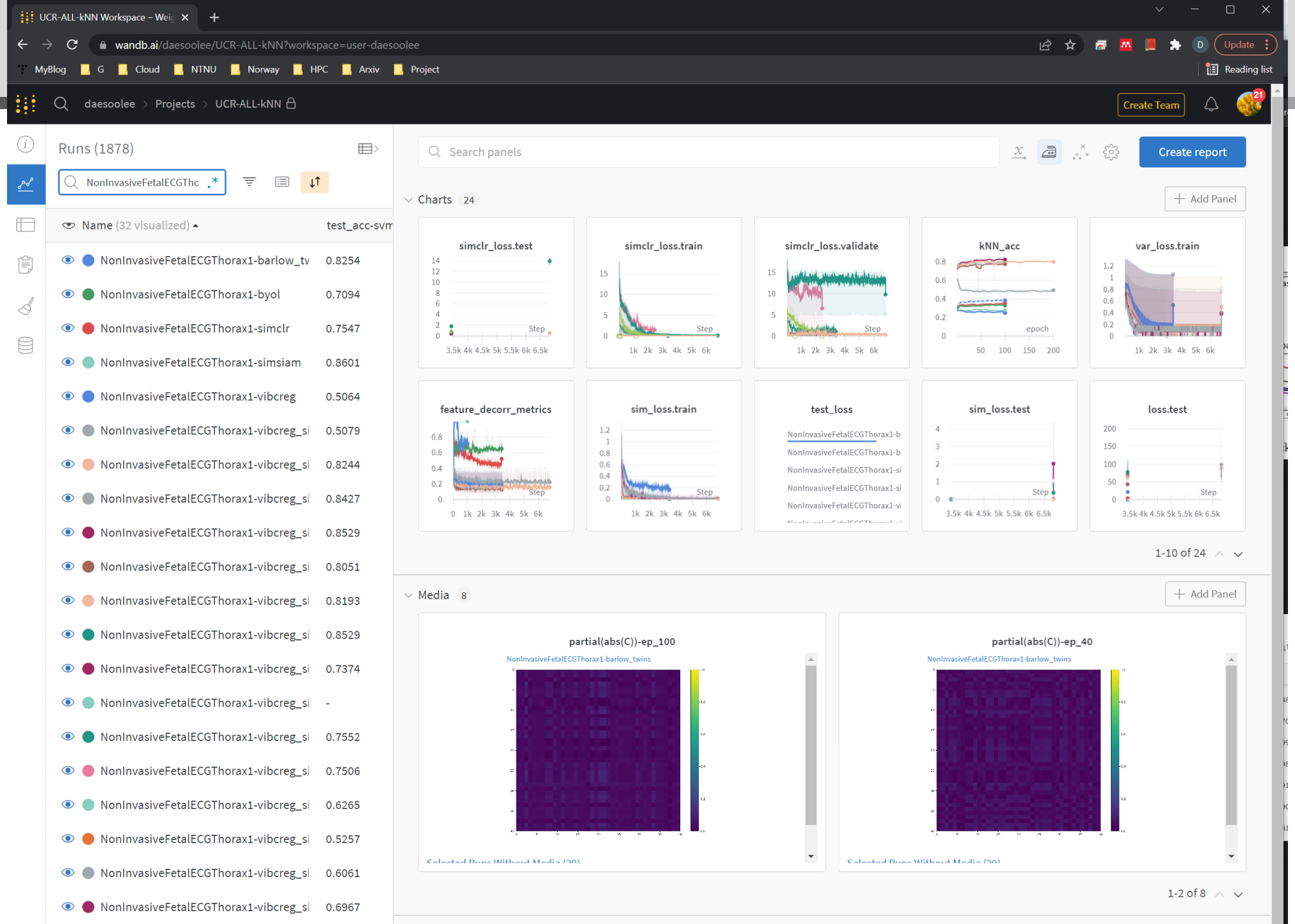



## Accessible anywhere

Check the latest training model and results on desktop and mobile. Use collaborative hosted projects to coordinate across your team.

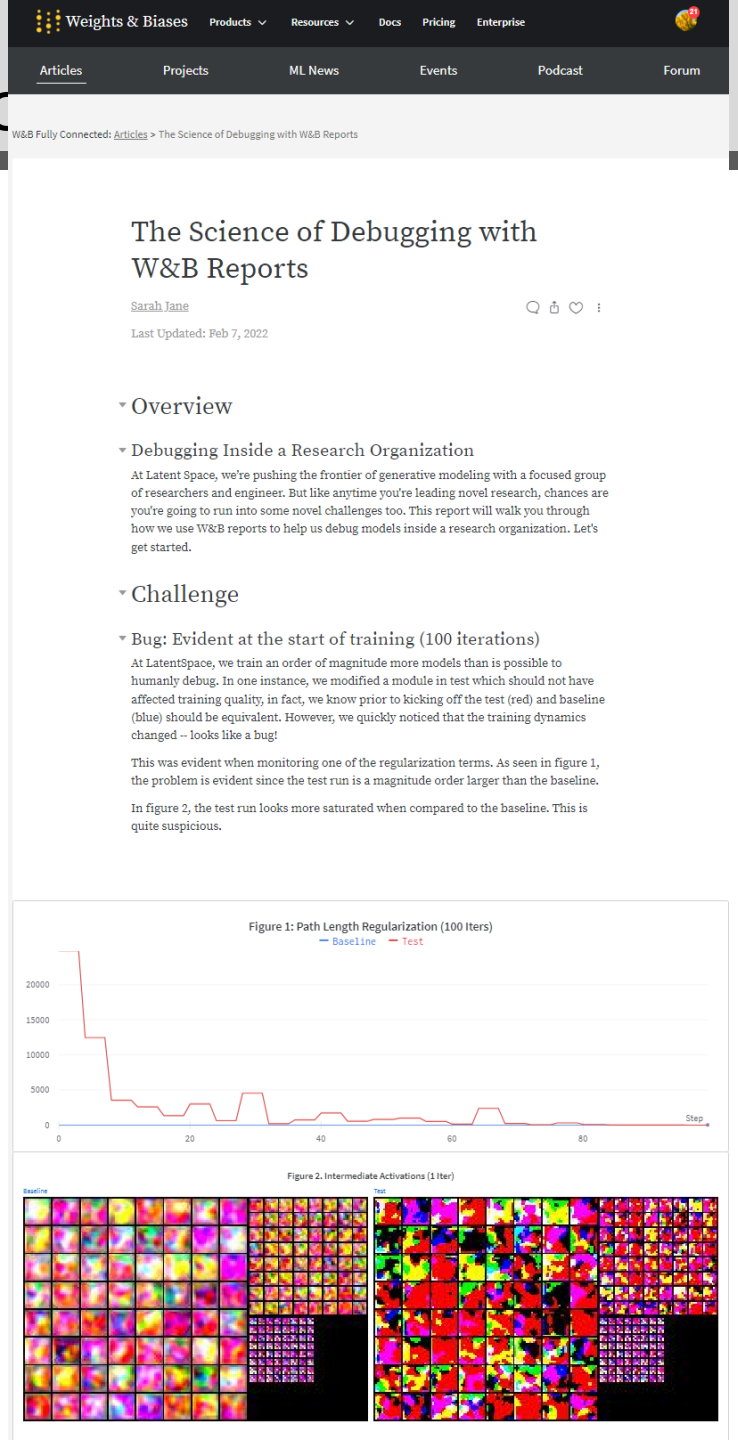

## Monitor your CPU and GPU usage

Visualize live metrics like GPU utilization to identify training bottlenecks and avoid wasting expensive resources.

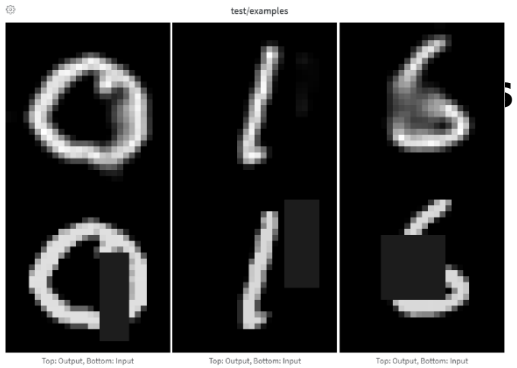

UCR-ALL-kNN Workspace – Weig
wandb.ai/daesoolee/UCR-ALL-kNN?workspace=user-daesoolee
MyBlog G Cloud NTNU Norway HPC Arxiv Project
Reading list
daesoolee > Projects > UCR-ALL-kNN
Create Team
Runs (1878)
NonInvasiveFetalECGTho
Name (32 visualized)
test_acc-svm
NonInvasiveFetalECGThorax1-barlow_tv 0.8254
NonInvasiveFetalECGThorax1-byol 0.7094
NonInvasiveFetalECGThorax1-simclr 0.7547
NonInvasiveFetalECGThorax1-simsiam 0.8601
NonInvasiveFetalECGThorax1-vibcreg 0.5064
NonInvasiveFetalECGThorax1-vibcreg_si 0.5079
NonInvasiveFetalECGThorax1-vibcreg_si 0.8244
NonInvasiveFetalECGThorax1-vibcreg_si 0.8427
NonInvasiveFetalECGThorax1-vibcreg_si 0.8529
NonInvasiveFetalECGThorax1-vibcreg_si 0.8051
NonInvasiveFetalECGThorax1-vibcreg_si 0.8193
NonInvasiveFetalECGThorax1-vibcreg_si 0.8529
NonInvasiveFetalECGThorax1-vibcreg_si 0.7374
NonInvasiveFetalECGThorax1-vibcreg_si -
NonInvasiveFetalECGThorax1-vibcreg_si 0.7552
NonInvasiveFetalECGThorax1-vibcreg_si 0.7506
NonInvasiveFetalECGThorax1-vibcreg_si 0.6265
NonInvasiveFetalECGThorax1-vibcreg_si 0.5257
NonInvasiveFetalECGThorax1-vibcreg_si 0.6061
NonInvasiveFetalECGThorax1-vibcreg_si 0.6967
Search panels
Create report
Charts 24
Add Panel
simclr_loss.test
simclr_loss.train
simclr_loss.validate
kNN_acc
var_loss.train
feature_decorr_metrics
sim_loss.train
test_loss
sim_loss.test
loss.test
1-10 of 24
Media 8
Add Panel
partial(abs(C))-ep_100
NonInvasiveFetalECGThorax1-barlow_twins
partial(abs(C))-ep_40
NonInvasiveFetalECGThorax1-barlow_twins
1-2 of 8

# Highly Recommended To

## Weights and Biases (wandb)



# Highly Recor ries

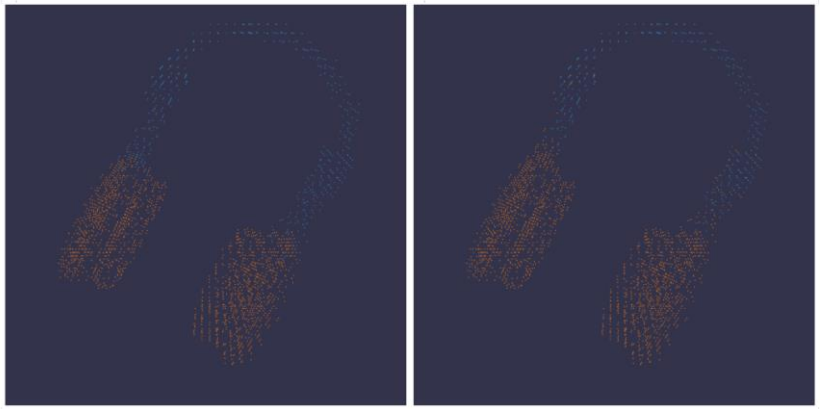



## 3D Visualizations

3D Object | Point Clouds | Molecules

Log files in the formats `'obj'`, `'gltf'`, `'glb'`, `'babylon'`, `'stl'`, `'pts.json'`, and we will render them in the UI when your run finishes.

```
wandb.log({"generated_samples":
           [wandb.Object3D(open("sample.obj")),
            wandb.Object3D(open("sample.gltf")),
            wandb.Object3D(open("sample.glb"))]})
```



## Other Media

Weights & Biases also supports logging of a variety of other media types.

Audio | Video | Text | HTML

```
wandb.log(
  {"whale songs": wandb.Audio(np_array, caption="OooOoo", sample_rate=32)})
```

The maximum number of audio clips that can be logged per step is 100.

Segmentation Masks | Bounding Boxes

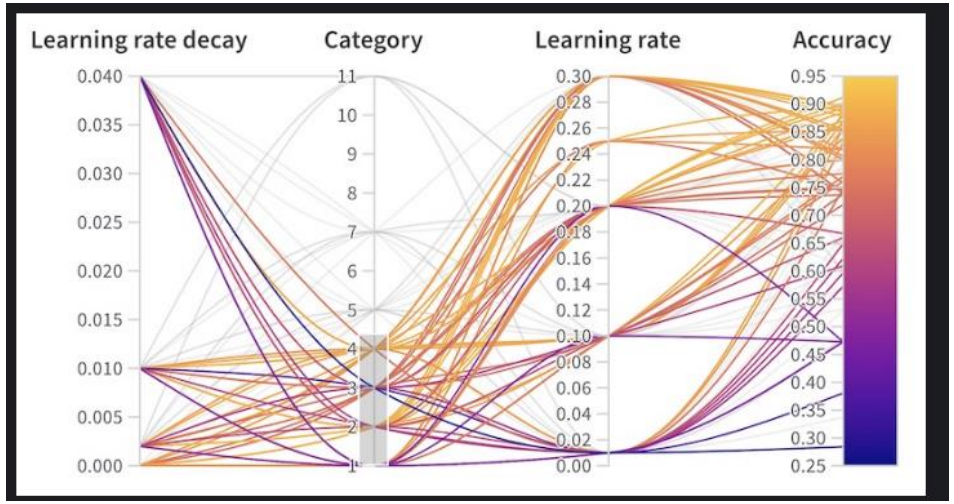
Log semantic segmentation masks and interact with them (altering opacity, viewing changes over time, and more) via the W&B UI.


Segmentation Masks | Bounding Boxes

Log bounding boxes with images, and use filters and toggles to dynamically visualize different sets of boxes in the UI.

# Highly Recommended Tools and Libraries

## Weights and Biases (wandb)



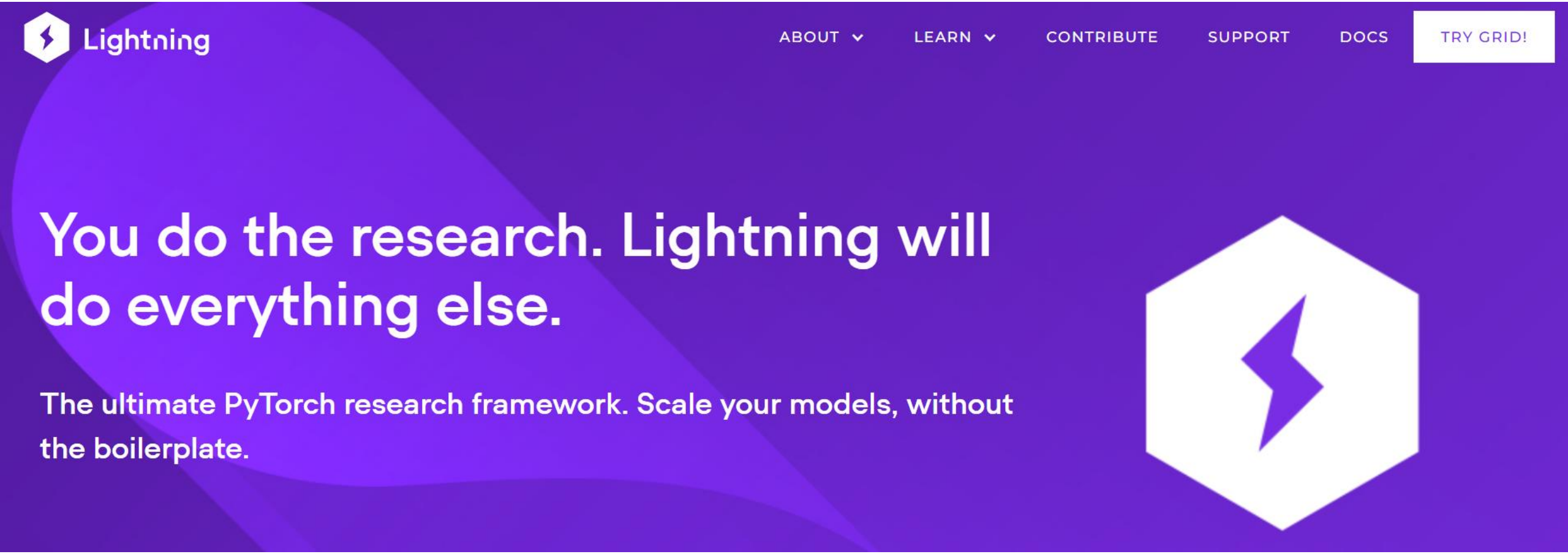


### Parameter importance

Visualize which hyperparameters affect the metrics you care about. W&B comes with default visualizations that make it easy to get started without writing custom code to compare experiments.

# Highly Recommended Tools and Libraries

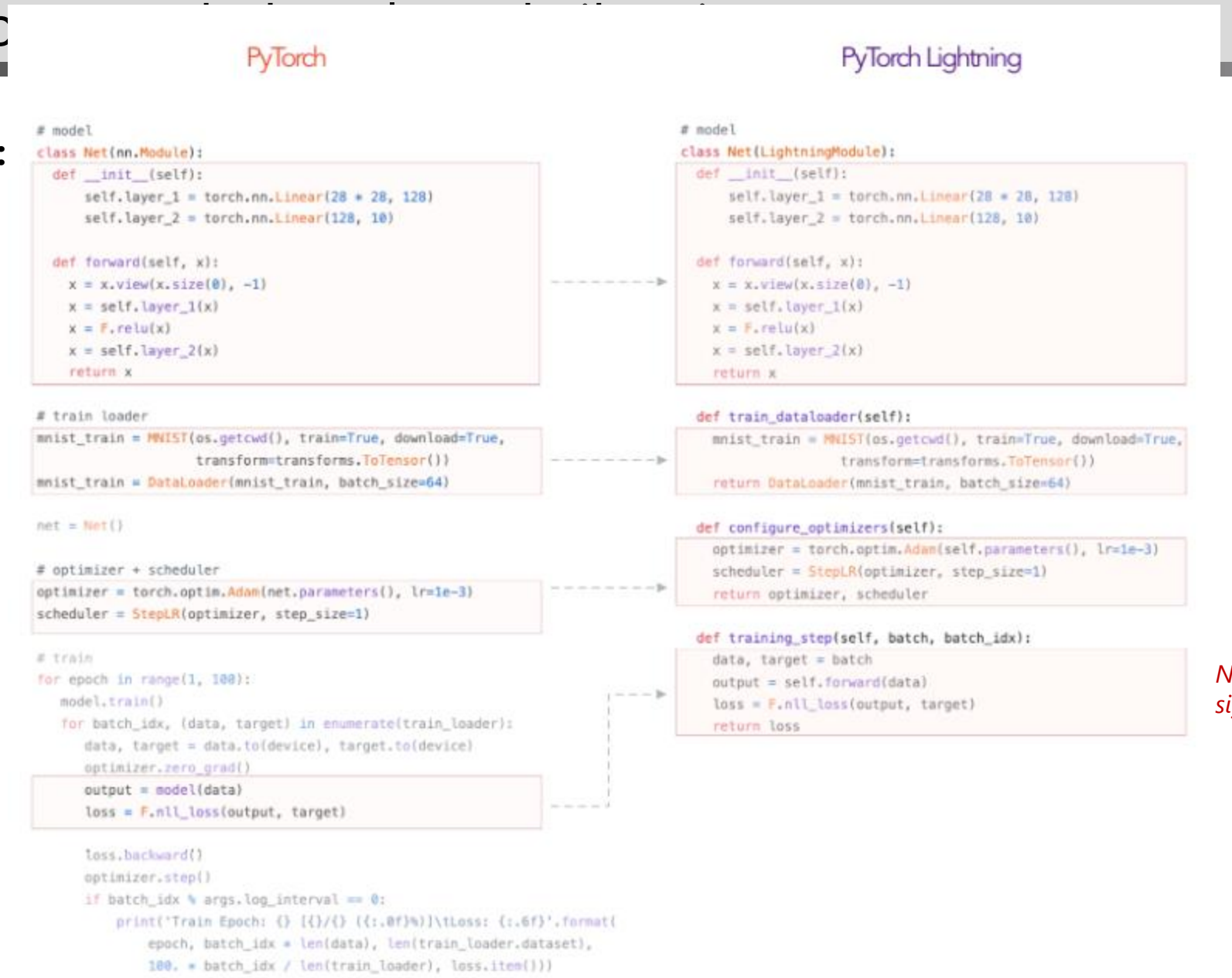
**PyTorch lightning:** simplified version of PyTorch




https://www.pytorchlightning.ai/#grid-section

# Highly Recc

**PyTorch lightning:**



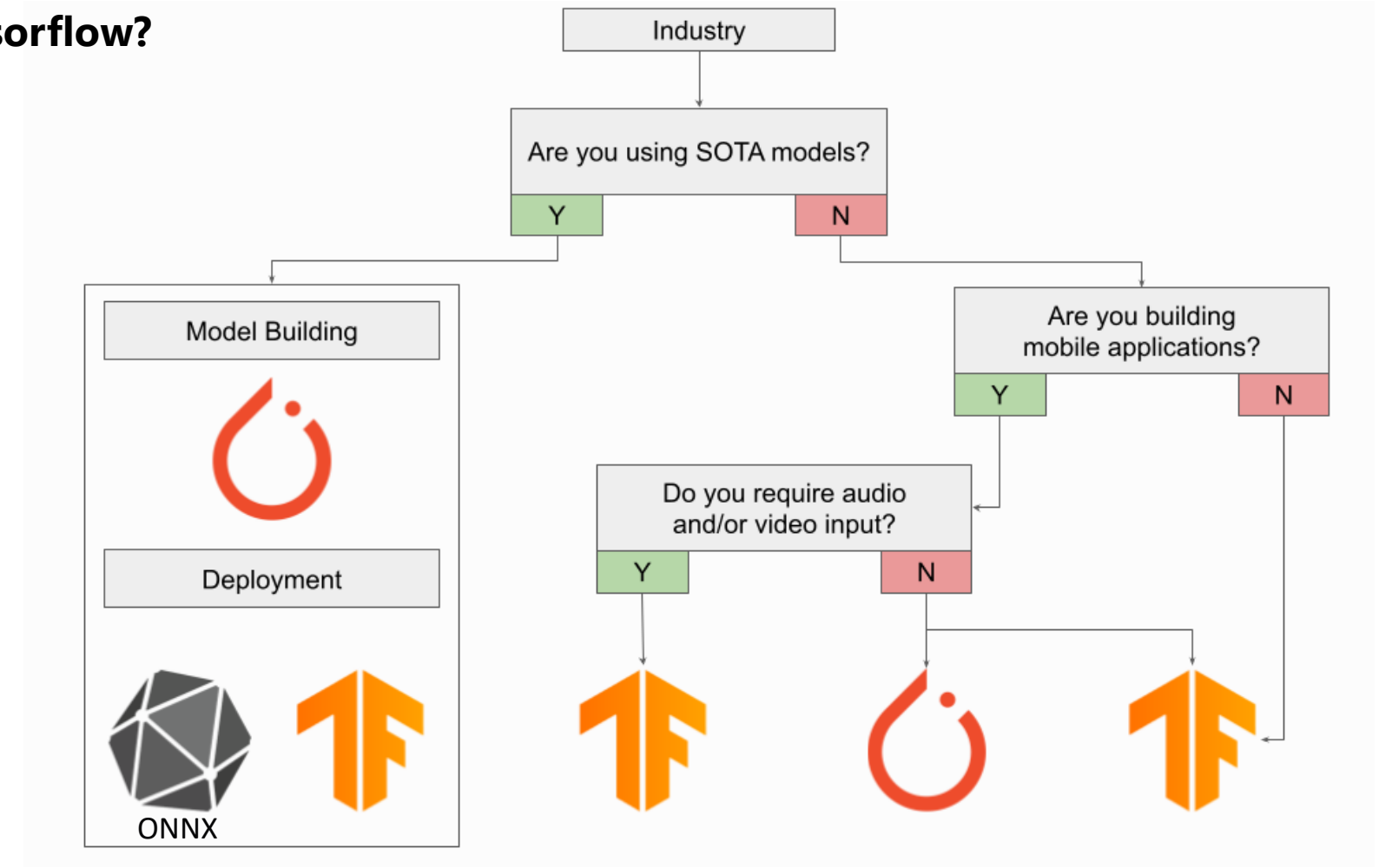

*NB! Training code becomes significantly simpler.*

# Thank you!


Norwegian University of
Science and Technology

# Why PyTorch?

**When to use Tensorflow?**




https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/