MUSIC APPLICATION OF MACHINE LEARNING
Tutorial Session

Seungheon Doh

KAIST MACLab
Music and Audio Computing Lab

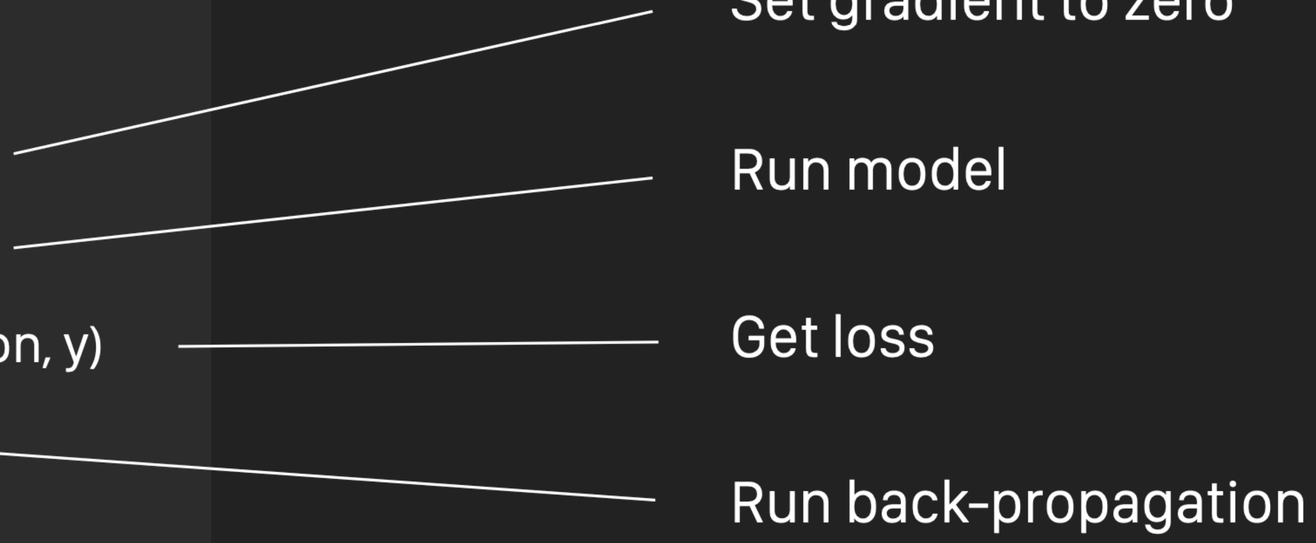
Pytorch

PyTorch	Numpy
torch.tensor([[1, 2], [3, 4])	np.array([[1, 2], [3, 4]])
torch.zeros()	np.zeros()
torch.ones()	np.ones()
torch.size()	np.shape[]
torch.view()	np.reshape()
torch.transpose()	np.transpose()
torch.cat()	np.concatenate()
torch.sum()	np.sum()
torch.load()	np.load()

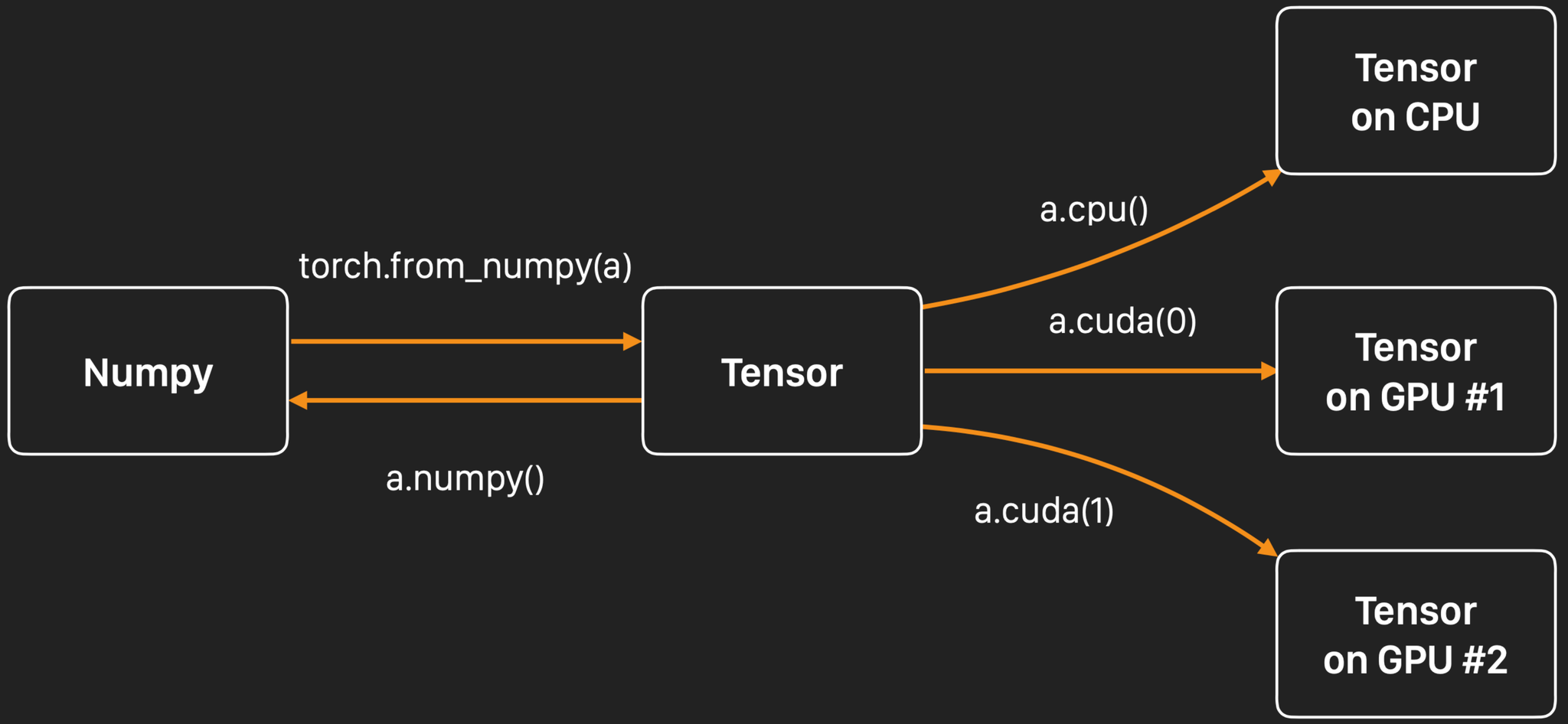
https://github.com/wkentaro/pytorch-for-numpy-users

Pytorch

```
for epoch in num_epoch:
    total_loss = 0
    for x, y in train_loader:
        optimizer.zero_grad()
        prediction = model(x)
        loss = criterion(prediction, y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
```



Pytorch


Tensor
on CPU
a.cpu()
torch.from_numpy(a)
Numpy
Tensor
a.cuda(0)
Tensor
on GPU #1
a.numpy()
a.cuda(1)
Tensor
on GPU #2

Pytorch

AUTOGRAD

- Automatic differentiation for all operations
- Automatic back-propagation

```
x = torch.ones(2, 2, requires_grad=True)
y = x + 2
print(y)
```

```
tensor([[3., 3.], [3., 3.]],
       grad_fn=<AddBackward0>)
```

```
x.detach()
```

Stopping tensor history tracking

```
loss.backward()
```

Equivalent to loss.backward(torch.tensor(1.0))
Calculating back-propagation

Pytorch

BUILDING A NETWORK

- Building a neural network based on nn.Module
- Automatically defined backward function using autograd

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16*5*5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Pytorch

DATASET & DATALOADER

- Dataset - Class for loading data

  init() - initial process like reading index file
  getitem() - return data at an index
  len() - return the size of entire data

- Dataloader - Wrapping up to batch

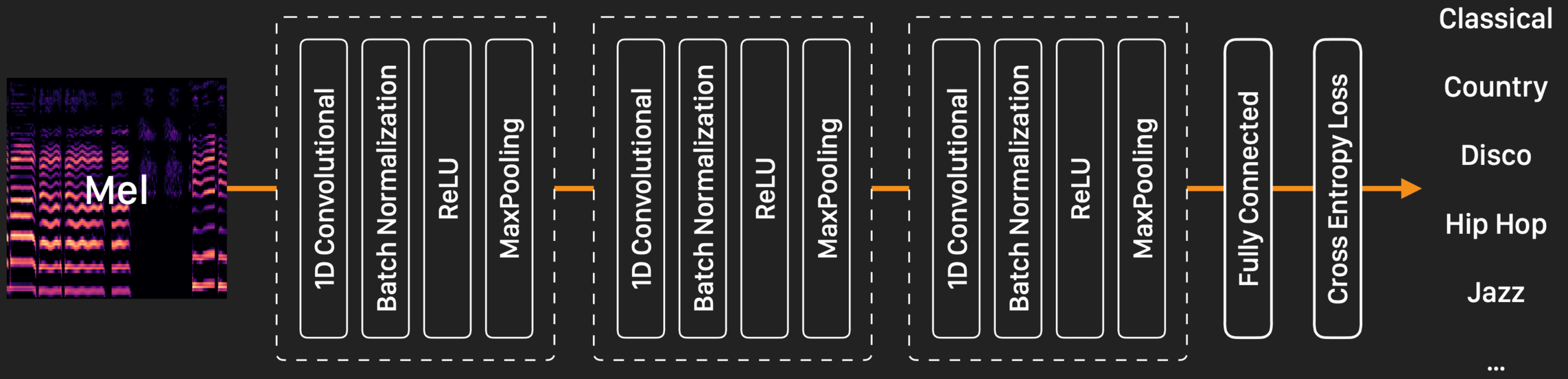
```
class GTZANDataset(Dataset):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __getitem__(self, index):
        return self.x[index], self.y[index]

    def __len__(self):
        return self.x.shape[0]
```

```
train_loader = DataLoader(dataset=train_set,
                          batch_size=32,
                          shuffle=True,
                          drop_last=False)
```

HomeWork!

- AutoTagging
- Metric Learning

[Q1,2] Automatic Music Tagging

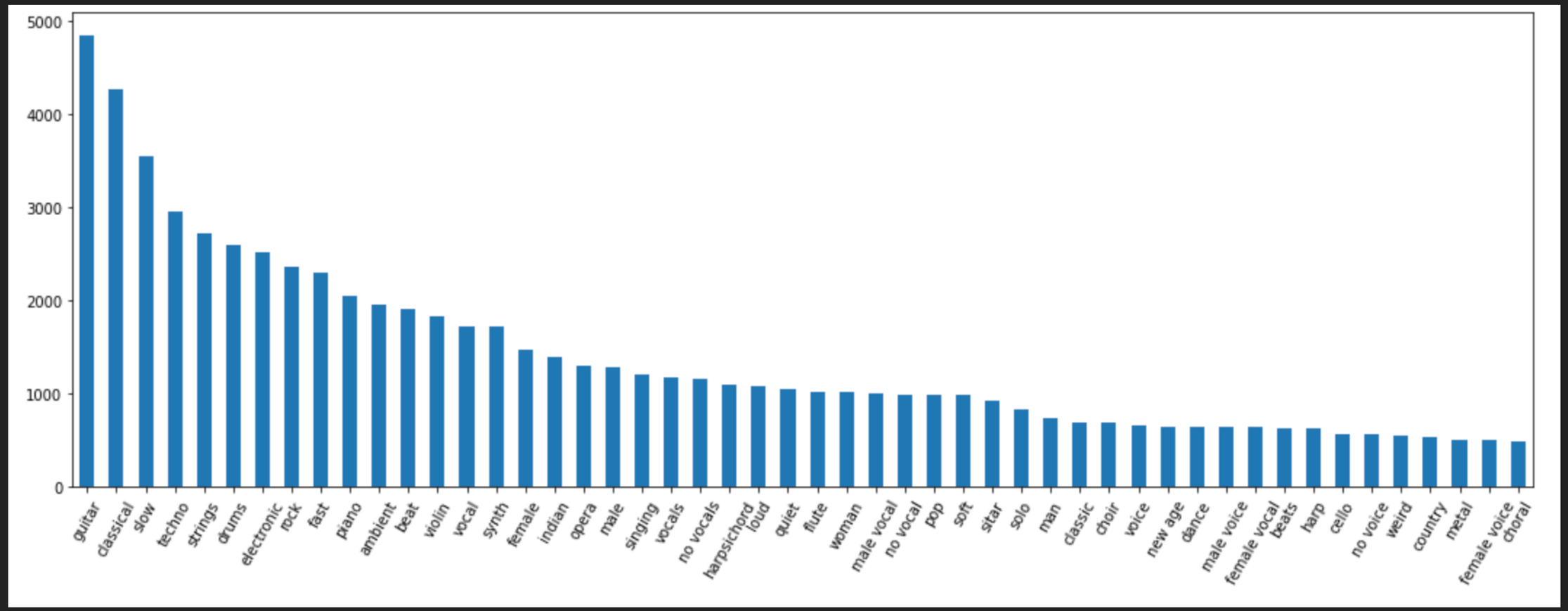


Baseline ROCAUC 0.61

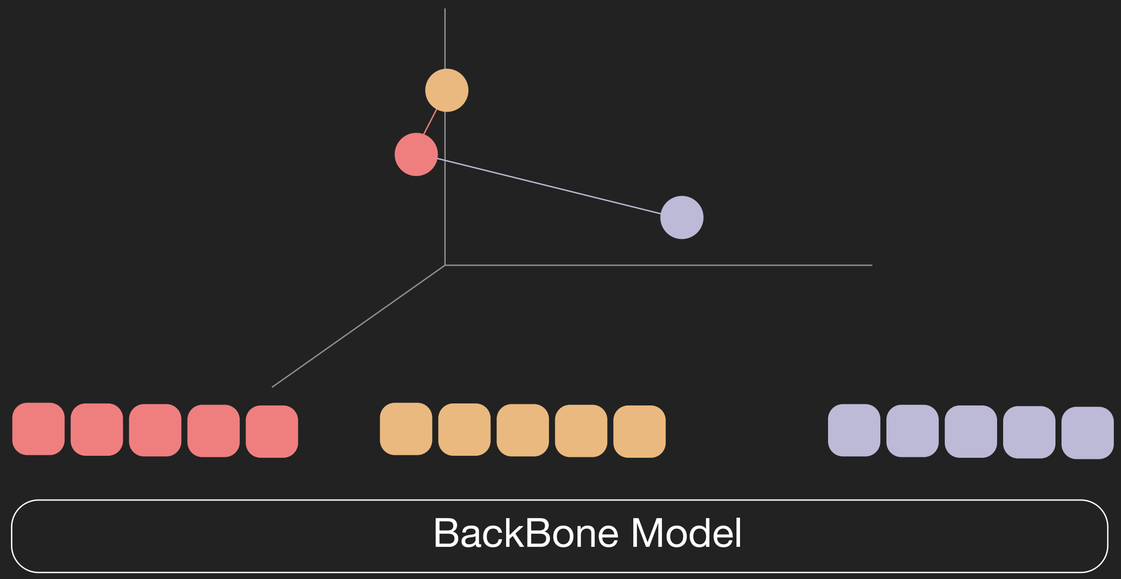
[Q1,2] Automatic Music Tagging

Dataset

- We use subset of magnatagatune dataset (9074 samples x 8 sec) with 50 Tags
- We use a relatively small and imbalanced dataset.



[Q3,4] Metric Learning

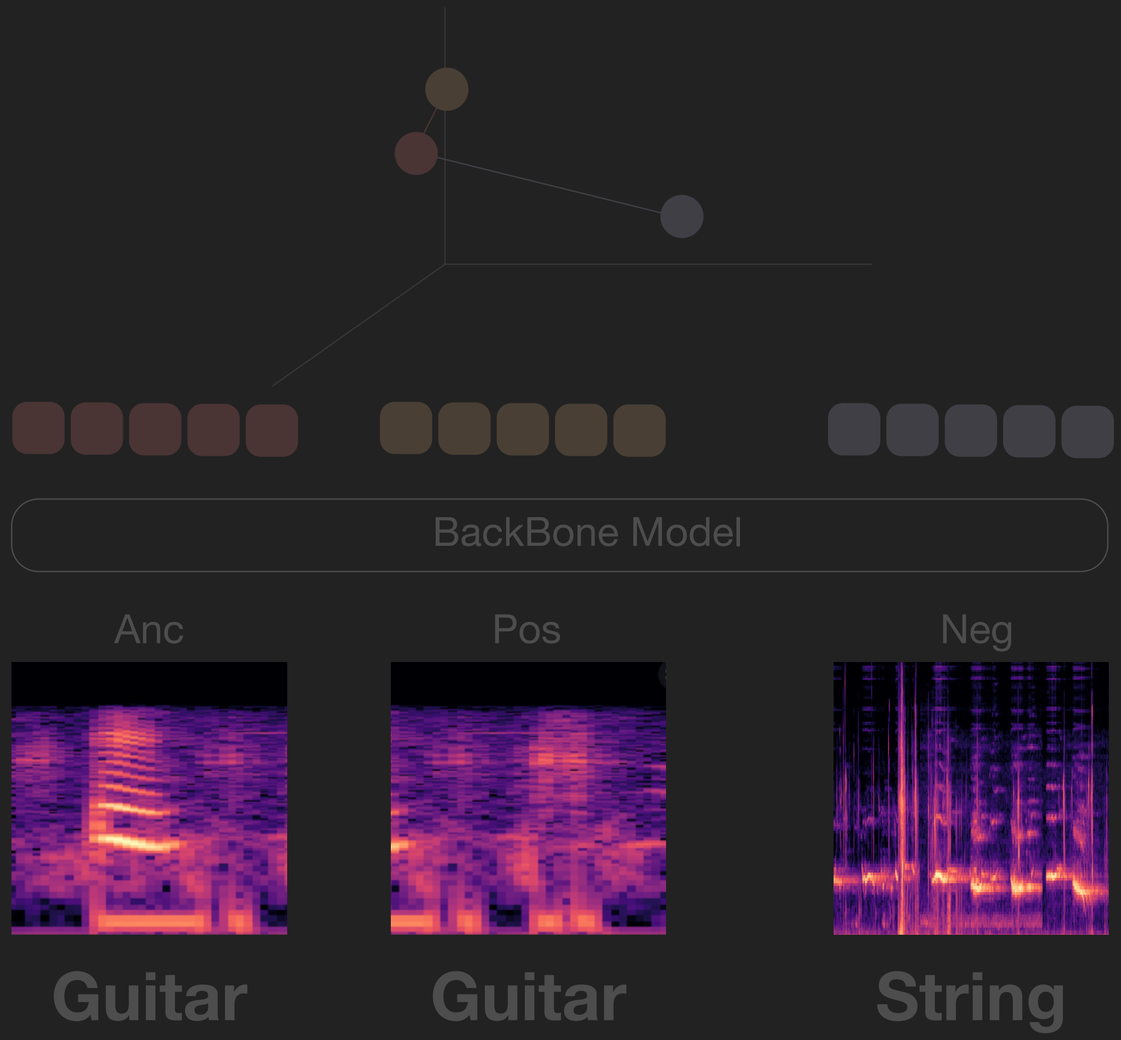


Triplet Loss

$$L = [D(E_a, E_p) - D(E_a, E_n) + \delta]_+,$$

E : Embedding from backbone model

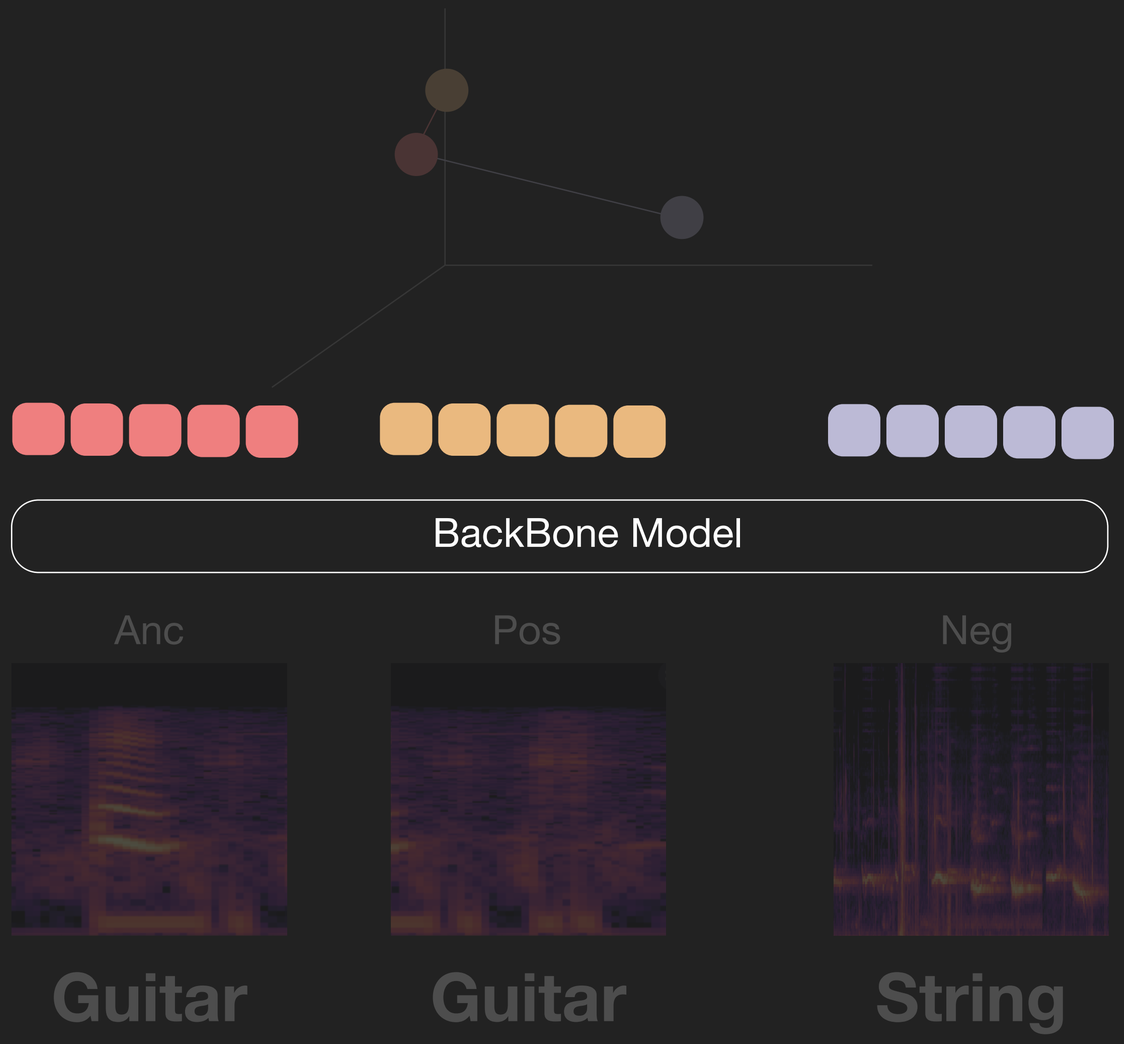
[Q1,2.3,4] Improvement



Audio Representation

- Change FFT Size & Mel Bin
- Change Input Seuqence
- MFCC
- CQT
- Time domain Representation

[Q1,2.3,4] Improvement



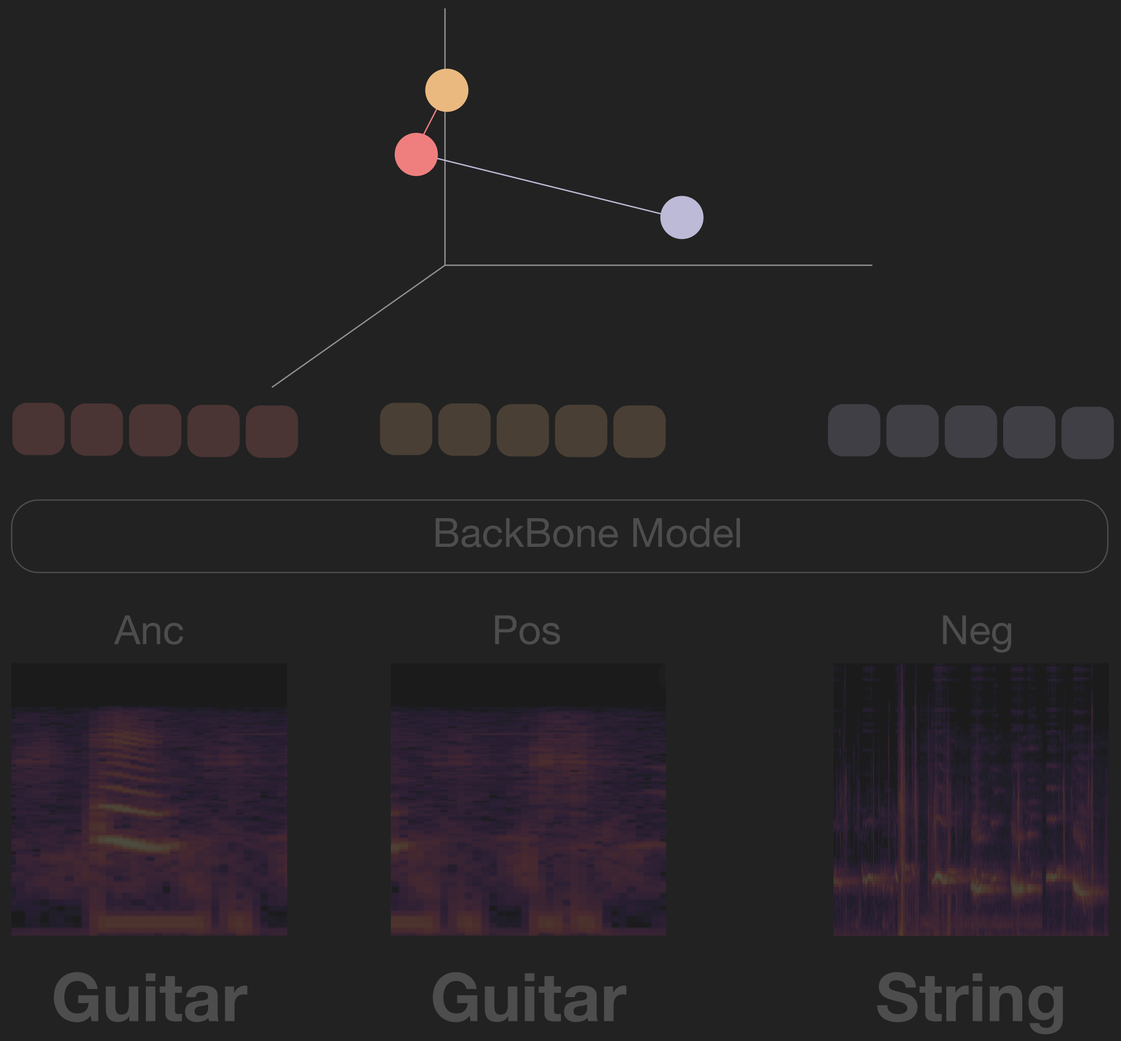
Backbone Model

- model parameters
  - Filter size
  - Pooling size
  - Stride size
  - Number of filters
  - Model depth
  - Regularization: L2/L1 and Dropout

- hyperparameters
  - Learning rate
  - Model depth
  - Optimizers: SGD (with Nesterov momentum), Adam, RMSProp, ...

- use pretrained model (openl3, vggish)

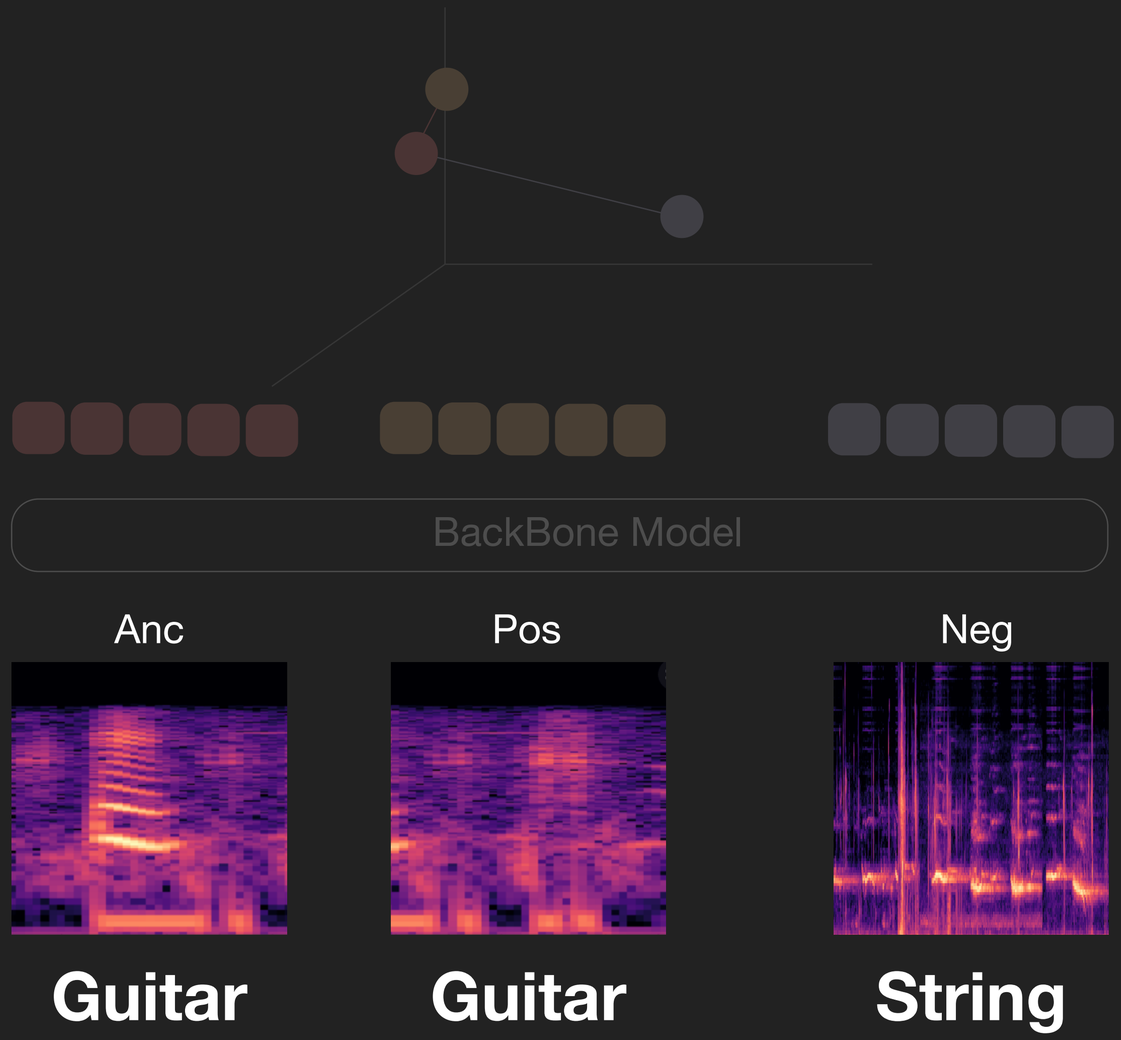
[Q3,4] Improvement



Change Distance Mesure

- Euclidean Space
- Cosine Space
- Hyperbolic Space

[Q3,4] Improvement



Sampling

- Tag based sampling
- Timbre, Beat based sampling
- Latent Space based sampling
  (Distance in SVD or Tag's Word Embedding)
- Instance based sampling (Self-Supervised)