**Classification of Animal Sounds Using Multi-Layer Perceptron**

**Hint:**

The Data has been split into training, test/validation sets.

**Imports:**

Make sure to download `torch`, `torchaudio`, `librosa` and `boto3`.

We are going to use `torchadio` for this task.

**Step:**

* (Base path) This path contains the dataset, which is basically a folder which contains two folders in it the `train` and `test` folders.
* (Train and test path) The train and test folders contain two subfolders `cat` and `dog`.
* Getting all the paths for the dataset.
* Loading the sample audio of a cat.
* Helper function `play\_audio()` takes in the `waveform` and the `sample\_rate` as arguments and displays an inline audio player in `ipynb`.
* `load\_audio` This function takes in a path and loops through all the `.wav` files and load them using th3 `touchaudio.load()` function which returns the waveform in a torch tensor and a sample rate.
* `load\_data` This helper function will take in the path and load all the audio and give them labels. It returns a list of list where each list contains `waveform`, `sample\_rate`, `class\_name`, `class\_index`.
* Creating a train and test data We are going to load all the data from the `cats` folder and the `dogs` folder from each set (train, test) and join them to their respective set. We are going to take an extra step of shuffling these lists so that we won't have a dataset with labels like [1, 1, 1, 1, ...., 0, 0, 0].
* Down sampling the audio Our audios has a sample rate of `16000`, which is very huge and it is not efficient to train the network on such a sample rate. We can down sample the sample rate so that we will be using the `touchaudio.transforms.Resample` method and change the sample rate to `8000`.
* Creating a dataset: We are going to create a dataset using the `Dataset` class from `torch.utils.data` we are going to call our dataset `CatsDogsAudio`, This dataset class will take in data, and transform as it's argument.
* data
  + This is essentially a list of loaded data. example is `train\_data`
* transform
  + These are transforms that we are going to apply on our features and labels.
* We are going to create a `ToTensor` custom transform that will convert features and labels to tensors.
* Now we can be able to create our train and test data using the `CatsDogsAudio` dataset.
* Creating loaders: We are then going to create two loaders, the train and the test loaders for each set. Each loader will have a `collate\_fn` which is more like a preprocessing function that is applied to the data by the `DataLoader` class from `torch.utils.data`
* pad\_sequence
  + The audio's that we care working with are of different length, so we need to use the `pad\_sequences` from `torch.nn.utils.rnn` so that we make sure that short audios are padded with `0's`. in the `torch.nn.utils.rnn.pad\_sequences` function we are going to set `batch\_first` to true, this is because we are going to use `Conv1` layers and `Conv` layers expect the batch\_size to be first.
* collate\_fn
  + In this function we are going to `transform` our audio waveforms and then apply the `pad\_sequence` function.
* Creating a Model (`M5`) : We are going to use a Convolutional neural network to process the raw audio data. Usually more advanced transforms are applied to the audio data, however CNNs can be used to accurately process the raw data.
* Optimizer: We are going to use the `Adam` optimizer with the default parameters
* Criterion: We are going to use the `BCEWithLogitsLoss` as our loss function since this is a binary classification task.
* Accuracy Function: We are going to compute the accuracy of the predicted values with the actual labels or targets using the following function.
* Train and evaluation functions
* train
  + This is a function that takes in the `model`, `optimizer`, `iterator` and `criterion` and return the train loss and train accuracy.
  + first we put the model in the `train` mode by calling `model.train()`
  + We iterate over an `iterator` and put features and labels to the device.
  + We restore the gradients by calling the `optimizer.zero\_grad()` function
  + We make predictions and calculate the loss and accuracy for each iterator.
  + After the loop we return the loss of each epoch and the accuracy
* evaluate
  + This is a function that takes in the `model`, `iterator` and `criterion` and returns the train loss and train accuracy.
  + We call the `model.eval()` so that the model will be in the evaluation mode.
  + We don't need to compute the gradients during evaluation so we wrap our iteration with `with torch.no\_grad()` function
  + We iterate over an `iterator` and put features and labels to the device.
  + We make predictions and calculate the loss and accuracy for each iterator.
  + After the loop we return the loss of each epoch and the accuracy
* Predict label function: Our `predict\_label` function will take in the `model` and the `waveform` and return the following `json` as a prediction.
* Prediction a dog sound.
* The Confusion Matrix: We can now make the Confusion matrix for this model.
* Classification report.