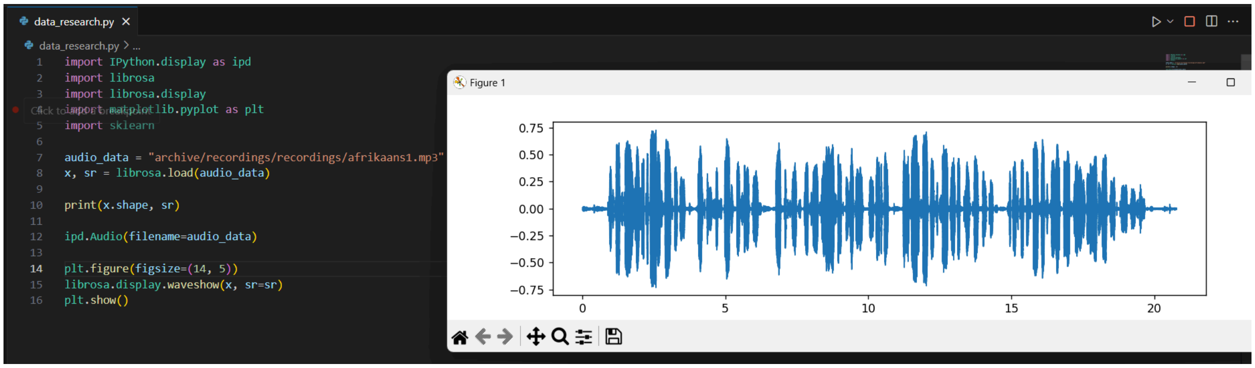
Biologically inspired artificial intelligence

Accent Detection

Patryk Hećko
Krzysztof Wojtyś

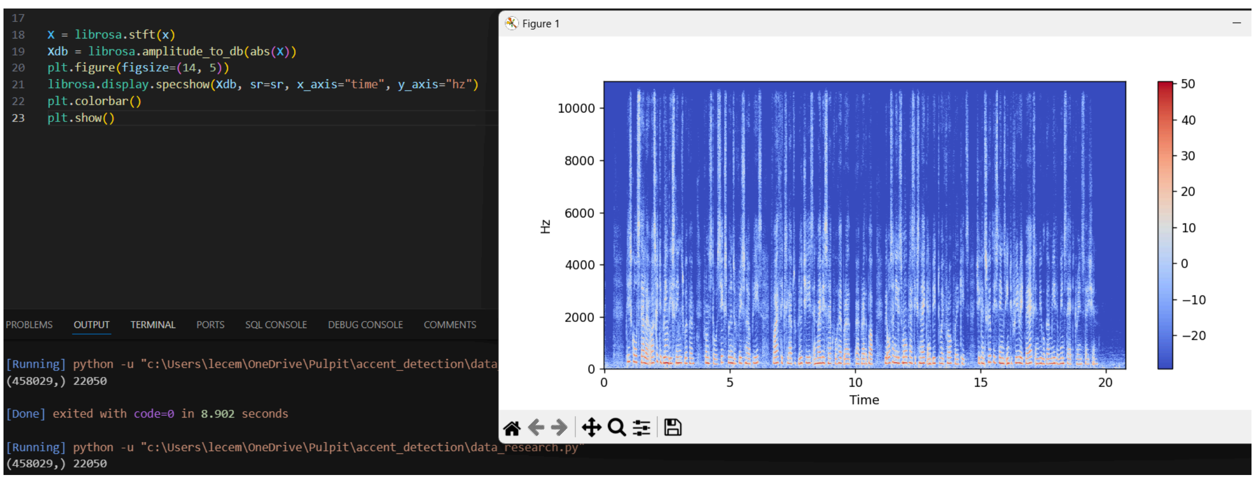
Informatyka sem. VI
Katowice

The first part of the project is examining the input audio recordings using librosa library to load the audio file and matplotlib to display the recording data.



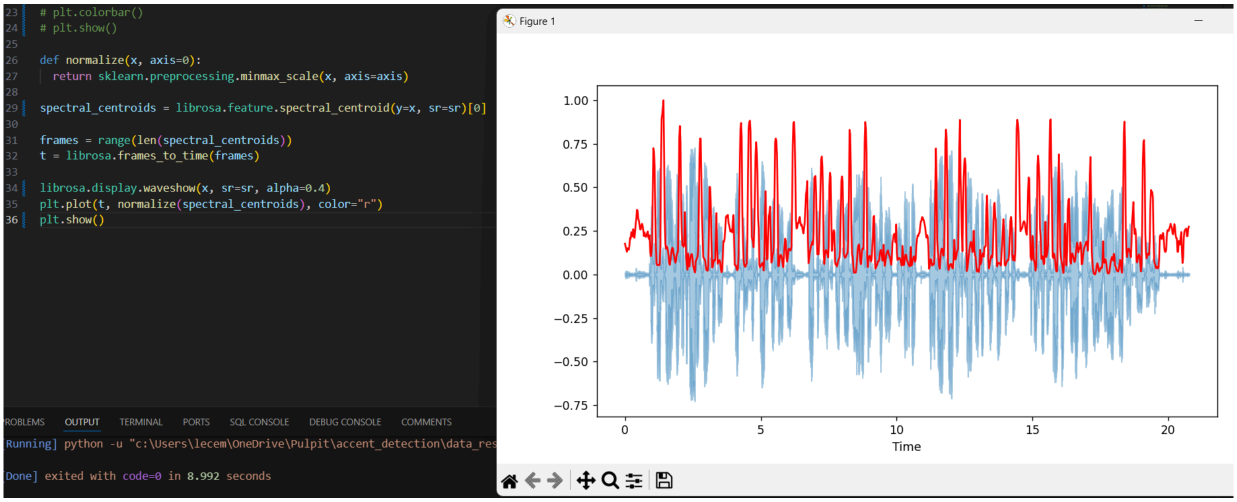
The given audio sample is sampled at 22050 Hz, dimensionality of 458029, it has 1 channel and duration of 21 s.

Next step was to plot a signal spectrogram to see the signal over time at various frequencies present in a particular waveform.

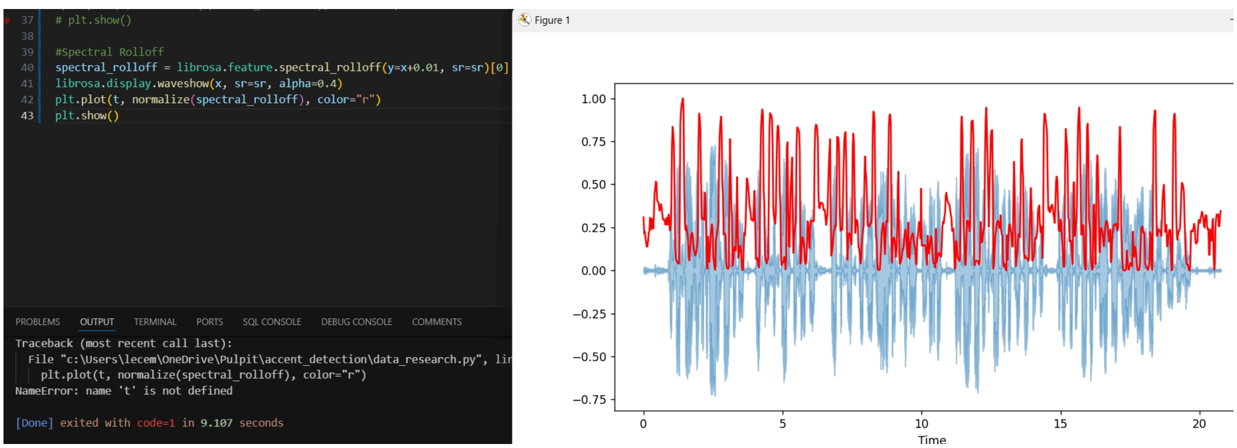


To best know what audio features will help us with our task, we extracted a few from the given audio signal.

First feature to extract is spectral centroid, it indicates where the center of mass of the spectrum is located.

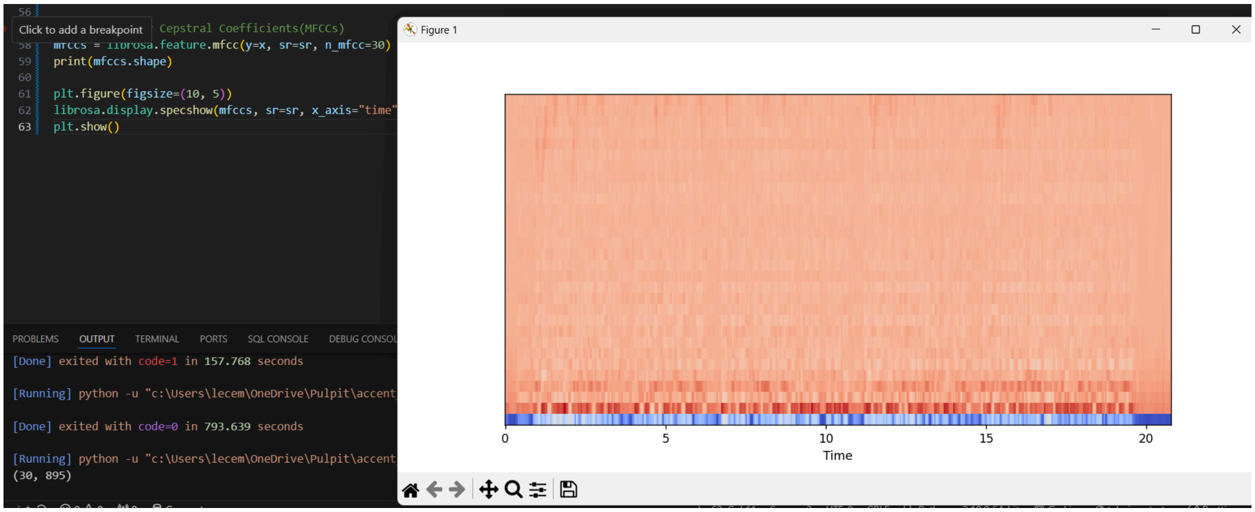


Next feature is spectral rolloff, which is a measure of the shape of the signal. It represents the frequency at which high frequencies decline to 0.



Next feature is spectral bandwidth, which is defined as the width of the band of light at one-half the peak maximum.

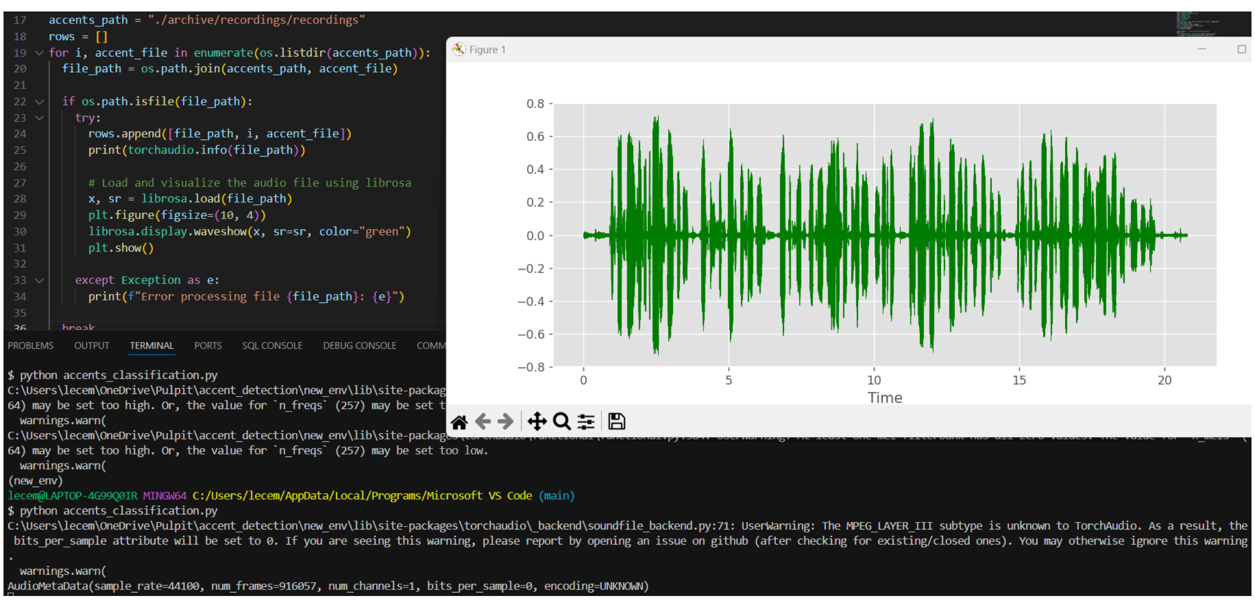
Last feature we extract is Mel-Frequency Cepstral Coefficients(MFCCs). The Mel frequency cepstral coefficients of a signal are a small set of features which concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice.



Mfcc computed 30 MFCCs over 30 frames

Because our dataset is collected human speech The Mel frequency cepstral coefficients suit it best and we will use these types of features for our further work.

After examining the audio inputs we started implementing accents_classfication file. First thing to implement was loading and displaying all recordings data.



After successfully retrieving recordings data we moved on to implement SpeechDataset class and plot_spectogram function in order to create easy to distinguish frequency(frame) images.

```python
class SpeechDataset(Dataset):
    def __init__(self, data_fr, data_path):
        self.data_fr = data_fr
        self.data_path = str(data_path)

    def __len__(self):
        return len(self.data_fr)

    def __getitem__(self, idx):
        audio_file = self.data_path + '/' + self.data_fr.loc[idx, "file_name"]
        class_id = self.data_fr.loc[idx, "class_id"]
        audio = preprocessor.load_audio(audio_file)
        rechannel = preprocessor.double_channel(audio)
        downsample = preprocessor.downsample(rechannel)
        specgram = preprocessor.spectro_mfcc(downsample)
        return specgram, class_id

speech_dataset = SpeechDataset(df, accents_path)

num_items = len(speech_dataset)
num_train = round(num_items * 0.7)
train_sample = num_items - num_train
train_ds, val_ds = random_split(speech_dataset, [num_train, train_sample])
# Training and validation data loaders
train_dl = DataLoader(train_ds, batch_size=10, shuffle=True)
val_dl = DataLoader(val_ds, batch_size=10, shuffle=False)
```

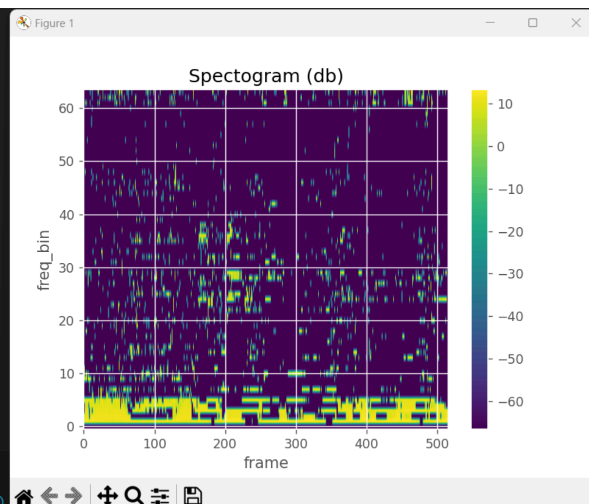
```python
num_train = round(num_items * 0.7)
train_sample = num_items - num_train
train_ds, val_ds = random_split(speech_dataset, [num_train, train_sample])
# Training and validation data loaders
train_dl = DataLoader(train_ds, batch_size=10, shuffle=True)
val_dl = DataLoader(val_ds, batch_size=10, shuffle=False)

def plot_spectogram(spec, ylabel="freq_bin", aspect="auto"):
  fig, axs = plt.subplots(1, 1)
  axs.set_title("Spectogram (db)")
  axs.set_xlabel("frame")
  axs.set_ylabel(ylabel)
  im = axs.imshow(librosa.power_to_db(spec), origin="lower", aspect=aspect)
  fig.colorbar(im, ax=axs)
  plt.show()

for i in range(5):
  plot_spectogram(train_ds[i][0][0])
```

PROBLEMS OUTPUT TERMINAL PORTS SQL CONSOLE DEBUG CONSOLE COMMENTS

```
lecem@LAPTOP-4G99Q0IR MINGW64 C:/Users/lecem/AppData/Local/Programs/Microsoft VS Code (main)
$ python accents_classification.py
```



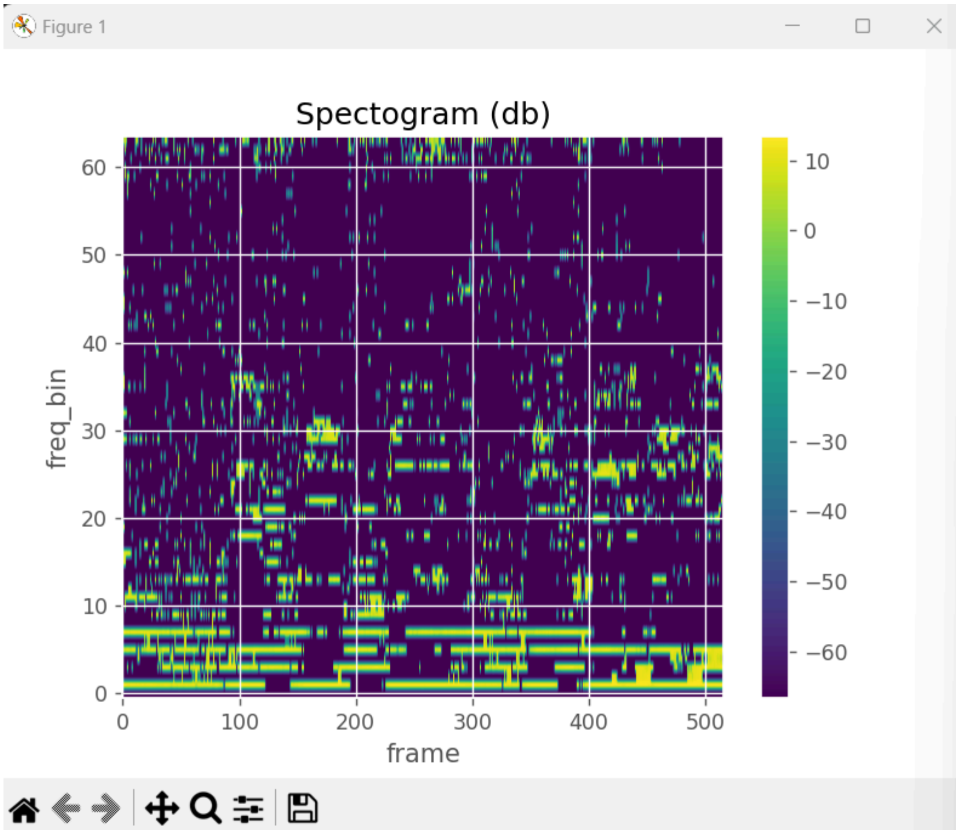
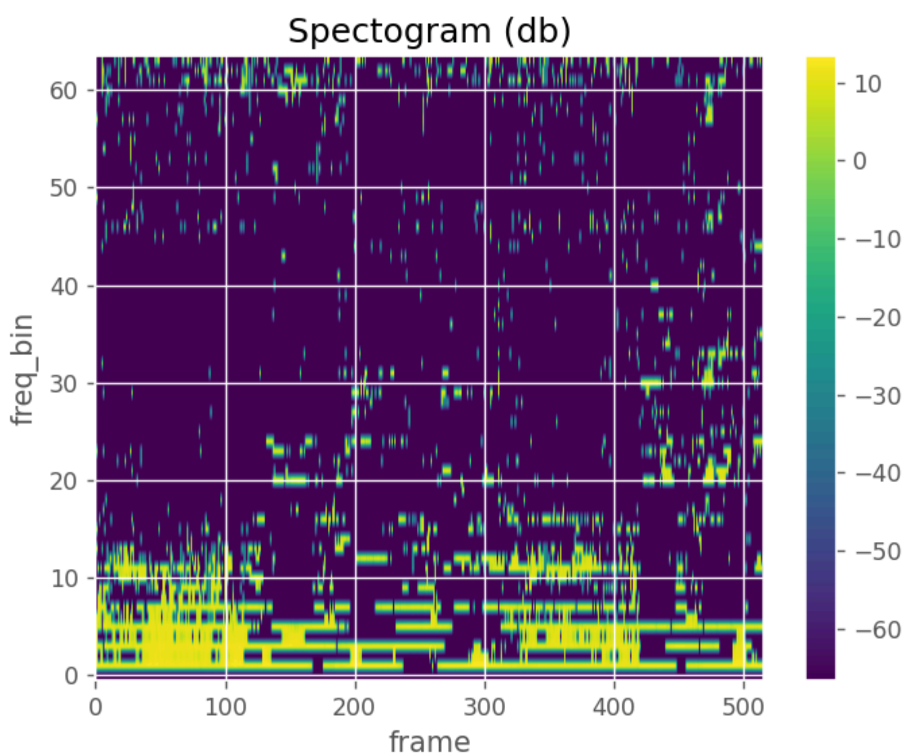


Figure 1
Spectogram (db)
freq_bin
frame
0
10
20
30
40
50
60
0
100
200
300
400
500
10
0
−10
−20
−30
−40
−50
−60


Spectogram (db)
freq_bin
frame
0
10
20
30
40
50
60
0
100
200
300
400
500
10
0
−10
−20
−30
−40
−50
−60

As we can see the images are different based on the speaker.

After successfully displaying input audio as spectrograms we implemented a classifier model specifying our neural network data such as Convolution Layers, activation function and normalization. We decided to use the ReLu activation function simply because it is easier to compute than the sigmoid function and gives very satisfactory results.

```python
class AudioClassifier(nn.Module):
  def __init__(self):
    super(AudioClassifier, self).__init__()
     # Neural Network shape
    self.conv = nn.Sequential(
      nn.Conv2d(2, 8, kernel_size=(3, 3), stride=(2,2), padding=(1, 1)), # 2D Convolution layer
      nn.ReLU(),  # Rectified Linear Unit activation function
      nn.BatchNorm2d(8), #normalization
      nn.Conv2d(8, 16, kernel_size=(3, 3), stride=(2,2), padding=(1, 1)),
      nn.ReLU(),
      nn.BatchNorm2d(16),
      nn.Conv2d(16, 32, kernel_size=(3, 3), stride=(2,2), padding=(1, 1)),
      nn.ReLU(),
      nn.BatchNorm2d(32),
    )
    # Linear Classifier
    self.ap = nn.AdaptiveAvgPool2d(output_size=1)
    self.dropout = nn.Dropout(0.5)
    # To establish in and out features
    self.lin = nn.Linear(in_features=32, out_features=9)

    # Forward propagation
  def forward(self, inp_x):
    inp_x = self.conv(inp_x)
    inp_x = self.ap(inp_x)
    inp_x = inp_x.view(inp_x.shape[0], -1)
    inp_x = self.dropout(inp_x)
    inp_x = self.lin(inp_x)
    return inp_x
```

Given the model we displayed its data. We use PyTorch on gpu (cuda) rather than cpu because it is much faster given thousands of cores rather than just a few.

```python
model = AudioClassifier()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
next(model.parameters()).device
summary(model, (2, 64, 258), 11)
```

PROBLEMS OUTPUT TERMINAL PORTS SQL CONSOLE DEBUG CONSOLE COMMENTS

```
lecem@LAPTOP-4G99Q0IR MINGW64 C:/Users/lecem/AppData/Local/Programs/Microsoft VS Code (main)
$ python accents_classification.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [11, 8, 32, 129]             152
              ReLU-2          [11, 8, 32, 129]               0
       BatchNorm2d-3          [11, 8, 32, 129]              16
            Conv2d-4          [11, 16, 16, 65]           1,168
              ReLU-5          [11, 16, 16, 65]               0
       BatchNorm2d-6          [11, 16, 16, 65]              32
            Conv2d-7           [11, 32, 8, 33]           4,640
              ReLU-8           [11, 32, 8, 33]               0
       BatchNorm2d-9           [11, 32, 8, 33]              64
AdaptiveAvgPool2d-10            [11, 32, 1, 1]               0
           Dropout-11                  [11, 32]              0
            Linear-12                   [11, 9]            297
================================================================
Total params: 6,369
Trainable params: 6,369
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.39
Forward/backward pass size (MB): 14.64
Params size (MB): 0.02
Estimated Total Size (MB): 16.05
----------------------------------------------------------------
(new_env)
```

We also defined Accuracy Metric class in order for neural network to know the accuracy and be able to improve in the following propagations

```python
class AccuracyMetric:
    def __init__(self):
        self.correct, self.total = None, None
        self.reset()

    def update(self, y_pred, y_true):
        self.correct += torch.sum(y_pred.argmax(-1) == y_true).item()
        self.total += y_true.size(0)

    def compute(self):
        return self.correct / self.total

    def reset(self):
        self.correct = 0
        self.total = 0
```

We then tried to train our model, but results were very dissatisfactory and iterations were way too slow, given our model works on gpu.

```python
for epoch in range(epochs):
  print(f"[INFO] Epoch: {epoch + 1}")
  model.train()

  batch_train_loss = []
  batch_valid_loss = []

  for X_batch, y_batch in tqdm(train_dl):
    # single training step
    model.zero_grad()
    X_batch, y_batch = X_batch.to(device), y_batch.to(device)
    y_predicted = model(X_batch)
    y_predicted = y_predicted.to(torch.float)
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE COMMENTS SQL CONSOLE

```
  return Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
 99%|                                                  | 99/100 [01:50<00:01,  1.13s/it]C:\Users\lecem\OneDrive\Pulpit\a
erWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cu
rc\ATen\native\cudnn\Conv_v8.cpp:919.)
  return Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
100%|                                                  | 100/100 [01:51<00:00,  1.11s/it]
100%|                                                  | 43/43 [00:51<00:00,  1.20s/it]
Train loss:  4.8609, Train accuracy:  0.2011 Validation loss:  4.1362, Validation accuracy:  0.2871
[INFO] Epoch: 2
100%|                                                  | 100/100 [01:50<00:00,  1.11s/it]
100%|                                                  | 43/43 [00:46<00:00,  1.09s/it]
Train loss:  4.0944, Train accuracy:  0.2645 Validation loss:  3.8365, Validation accuracy:  0.2855
[INFO] Epoch: 3
100%|                                                  | 100/100 [01:52<00:00,  1.13s/it]
100%|                                                  | 43/43 [00:47<00:00,  1.10s/it]
Train loss:  3.8929, Train accuracy:  0.2632 Validation loss:  3.7309, Validation accuracy:  0.2839
[INFO] Epoch: 4
:  0.2855
[INFO] Epoch: 5
100%|                                                  | 100/100 [01:54<00:00,  1.14s/it]
100%|                                                  | 43/43 [00:48<00:00,  1.12s/it]
Train loss:  3.7576, Train accuracy:  0.2632 Validation loss:  3.6917, Validation accuracy:  0.2839
(new_env)
```

https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504