VERY DEEP CONVOLUTIONAL NEURAL NETWORKS FOR RAW WAVEFORMS

Presented by: Mohamed Anwar

Overview

- Classifies raw Audio waveform into 10 classes.
- Uses only Conv/Pool Layers.
- Uses Batch Normalization & Glorot initialization to avoid exploding or vanishing gradients.
- Builds models from scratch (no pre-training).
- Proposes five different Models.

Data

- The dataset consists of 8732 audio clips (7GB).
- Each audio clip is a wav file (≈ 4 seconds).
- Sample rate: 44100 HZ
- Channels: Stereo
- Total duration: 9.7 hours.

Data Classes

0) air_conditioner 

1) Car_horn 

2) children_playing 

3) dog_bark 

4) drilling 

5) engine_idling 

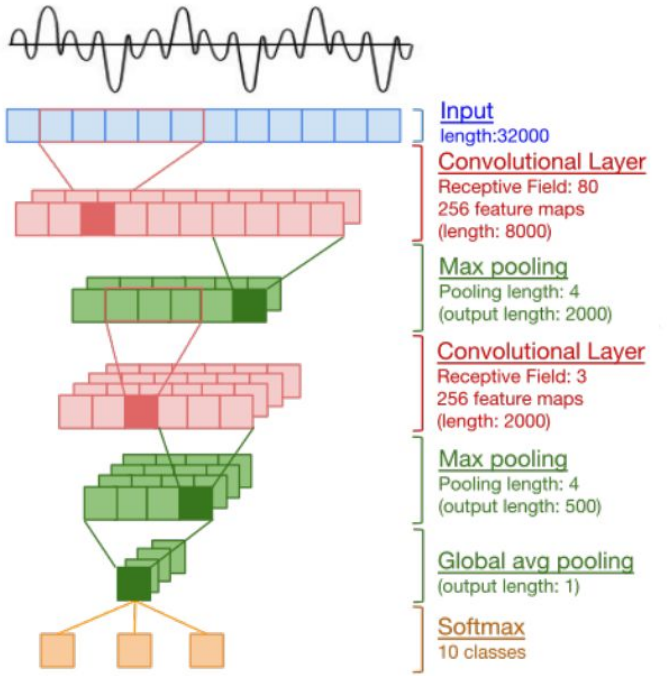
6) gun_shot 

7) jackhammer 

8) siren 

9) street_music 

Models



M3 (0.2M)	M5 (0.5M)	M11 (1.8M)	M18 (3.7M)	M34-res (4M)
Input: 32000x1 time-domain waveform				
[80/4, 256]	[80/4, 128]	[80/4, 64]	[80/4, 64]	[80/4, 48]
Maxpool: 4x1 (output: 2000 × n)				
[3, 256]	[3, 128]	[3, 64] × 2	[3, 64] × 4	$\begin{bmatrix} 3, 48 \\ 3, 48 \end{bmatrix} \times 3$
Maxpool: 4x1 (output: 500×n)				
	[3, 256]	[3, 128] × 2	[3, 128] × 4	$\begin{bmatrix} 3, 96 \\ 3, 96 \end{bmatrix} \times 4$
	Maxpool: 4x1 (output: 125 × n)			
	[3, 512]	[3, 256] × 3	[3, 256] × 4	$\begin{bmatrix} 3, 192 \\ 3, 192 \end{bmatrix} \times 6$
	Maxpool: 4x1 (output: 32 × n)			
		[3, 512] × 2	[3, 512] × 4	$\begin{bmatrix} 3, 384 \\ 3, 384 \end{bmatrix} \times 3$
Global average pooling (output: 1 × n)				
Softmax				

My Implementation

My implementation is divided into three different parts:

1. Preprocessing the data.
2. Creating the models.
3. Replicating the results.

Data Preprocessing

```
class ToMono(torch.nn.Module):
    def forward(self, waveform: torch.Tensor) -> torch.Tensor:
        return torch.mean(waveform, dim=0, keepdim=True)

class Normalize(torch.nn.Module):
    def forward(self, waveform: torch.Tensor) -> torch.Tensor:
        return (waveform-waveform.mean()) / waveform.std()

class Pad(torch.nn.Module):
    def __init__(self, value: float, size: int):
        super(Pad, self).__init__()
        self.value = value
        self.size = size

    def forward(self, waveform: torch.Tensor) -> torch.Tensor:
        return torch.nn.functional.pad(waveform, (0, self.size-max(waveform.shape)), "constant", self.value)

audio_transform = torch.nn.Sequential(*[
    ToMono(), #converts audio channels to mono
    torchaudio.transforms.Resample(orig_freq=441000, new_freq=8000), # downsamples audio signal to 8000 HZ
    Normalize(), # normalize audio signal to have mean=0 & std=1
    Pad(value=0, size=32000),
])
```

Models

Two base classes:

1. CNN
2. CCNRes

M3 (0.2M)	M5 (0.5M)	M11 (1.8M)	M18 (3.7M)	M34-res (4M)
Input: 32000x1 time-domain waveform				
[80/4, 256]	[80/4, 128]	[80/4, 64]	[80/4, 64]	[80/4, 48]
Maxpool: 4x1 (output: 2000 × n)				
[3, 256]	[3, 128]	[3, 64] × 2	[3, 64] × 4	$\begin{bmatrix} 3, 48 \\ 3, 48 \end{bmatrix} \times 3$
Maxpool: 4x1 (output: 500×n)				
	[3, 256]	[3, 128] × 2	[3, 128] × 4	$\begin{bmatrix} 3, 96 \\ 3, 96 \end{bmatrix} \times 4$
	Maxpool: 4x1 (output: 125 × n)			
	[3, 512]	[3, 256] × 3	[3, 256] × 4	$\begin{bmatrix} 3, 192 \\ 3, 192 \end{bmatrix} \times 6$
	Maxpool: 4x1 (output: 32 × n)			
		[3, 512] × 2	[3, 512] × 4	$\begin{bmatrix} 3, 384 \\ 3, 384 \end{bmatrix} \times 3$
Global average pooling (output: 1 × n)				
Softmax				

C N N

CNNRes

CNN Base Class

```
class CNN(torch.nn.Module):
    def __init__(self, channels, conv_kernels, conv_strides, conv_padding, pool_padding, num_classes=10):
        assert len(conv_kernels) == len(channels) == len(conv_strides) == len(conv_padding)
        super(CNN, self).__init__()
        # create conv blocks
        self.conv_blocks = torch.nn.ModuleList()
        prev_channel = 1
        for i in range(len(channels)):
            # add stacked conv layer
            block = []
            for j, conv_channel in enumerate(channels[i]):
                block.append( torch.nn.Conv1d(in_channels = prev_channel, out_channels = conv_channel, kernel_size = conv_kernels[i], stride = conv_strides[i], padding = conv_padding[i]) )
                prev_channel = conv_channel
                # add batch norm layer
                block.append( torch.nn.BatchNorm1d(prev_channel) )
                # adding ReLU
                block.append( torch.nn.ReLU() )
            self.conv_blocks.append( torch.nn.Sequential(*block) )

        # create pool blocks
        self.pool_blocks = torch.nn.ModuleList()
        for i in range(len(pool_padding)):
            # adding Max Pool (drops dims by a factor of 4)
            self.pool_blocks.append( torch.nn.MaxPool1d(kernel_size = 4, stride = 4, padding = pool_padding[i]) )

        # global pooling
        self.global_pool = torch.nn.AdaptiveAvgPool1d(1)
        self.linear = torch.nn.Linear(prev_channel, num_classes)


    def forward(self, inwav):
        for i in range(len(self.conv_blocks)):
            # apply conv layer
            inwav = self.conv_blocks[i](inwav)
            # apply max_pool
            if i < len(self.pool_blocks): inwav = self.pool_blocks[i](inwav)
        # apply global pooling
        out = self.global_pool(inwav).squeeze()
        out = self.linear(out)
        return out.squeeze()
```

CNN Derived classes

```
m3 = CNN(channels = [[256], [256]],
         conv_kernels = [80, 3],
         conv_strides = [4, 1],
         conv_padding = [38, 1],
         pool_padding = [0, 0])

m5 = CNN(channels = [[128], [128], [256], [512]],
         conv_kernels = [80, 3, 3, 3],
         conv_strides = [4, 1, 1, 1],
         conv_padding = [38, 1, 1, 1],
         pool_padding = [0, 0, 0, 2])

m11 = CNN(channels = [[64], [64]*2, [128]*2, [256]*3, [512]*2],
          conv_kernels = [80, 3, 3, 3, 3],
          conv_strides = [4, 1, 1, 1, 1],
          conv_padding = [38, 1, 1, 1, 1],
          pool_padding = [0, 0, 0, 2])

m18 = CNN(channels = [[64], [64]*4, [128]*4, [256]*4, [512]*4],
          conv_kernels = [80, 3, 3, 3, 3],
          conv_strides = [4, 1, 1, 1, 1],
          conv_padding = [38, 1, 1, 1, 1],
          pool_padding = [0, 0, 0, 2])
```

CNNRes Base Class

```
class CNNRes(torch.nn.Module):

    def __init__(self, channels, conv_kernels, conv_strides, conv_padding, pool_padding, num_classes=10):
        assert len(conv_kernels) == len(channels) == len(conv_strides) == len(conv_padding)
        super(CNNRes, self).__init__()

        # create conv block
        prev_channel = 1
        self.conv_block = torch.nn.Sequential(
            torch.nn.Conv1d(in_channels = prev_channel, out_channels = channels[0][0], kernel_size = conv_kernels[0], stride = conv_strides[0], padding = conv_padding[0]),
            # add batch norm layer
            torch.nn.BatchNorm1d(channels[0][0]),
            # adding ReLU
            torch.nn.ReLU(),
            # adding max pool
            torch.nn.MaxPool1d(kernel_size = 4, stride = 4, padding = pool_padding[0]),
        )

        # create res
        prev_channel = channels[0][0]
        self.res_blocks = torch.nn.ModuleList()
        for i in range(1, len(channels)):
            # add stacked res layer
            block = []
            for j, conv_channel in enumerate(channels[i]):
                block.append( ResBlock(prev_channel, conv_channel, conv_kernels[i], conv_strides[i], conv_padding[i]) )
                prev_channel = conv_channel
            self.res_blocks.append( torch.nn.Sequential(*block) )

        # create pool blocks
        self.pool_blocks = torch.nn.ModuleList()
        for i in range(1, len(pool_padding)):
            # adding Max Pool (drops dims by a factor of 4)
            self.pool_blocks.append( torch.nn.MaxPool1d(kernel_size = 4, stride = 4, padding = pool_padding[i]) )

        # global pooling
        self.global_pool = torch.nn.AdaptiveAvgPool1d(1)
        self.linear = torch.nn.Linear(prev_channel, num_classes)


    def forward(self, inwav):
        inwav = self.conv_block(inwav)
        for i in range(len(self.res_blocks)):
            # apply conv layer
            inwav = self.res_blocks[i](inwav)
            # apply max_pool
            if i < len(self.pool_blocks): inwav = self.pool_blocks[i](inwav)
        # apply global pooling
        out = self.global_pool(inwav).squeeze()
        out = self.linear(out)
        return out.squeeze()
```

M32-Res

```
m32 = CNNRes(channels = [[48], [48]*3, [96]*4, [192]*6, [384]*3],
             conv_kernels = [80, 3, 3, 3, 3],
             conv_strides = [4, 1, 1, 1, 1],
             conv_padding = [38, 1, 1, 1, 1],
             pool_padding = [0, 0, 0, 2])
```

Results

Model	Test
M3	56.12%
M5	63.42%
M11	69.07%
M18	71.68%
M34-res	63.47%

M3: 35% (15 epochs)

M5: 45% (50 epochs)

M11: 35% (10 epochs)

M18: 32% (8 epochs)

M34-res = 37% (10 epochs)

Code

Check the GitHub Repository:

https://github.com/Anwarvic/CNN-for-Raw-Waveforms

Questions ?