Drum Sound Classifier

Deep Learning Final Project

Group 3, P102

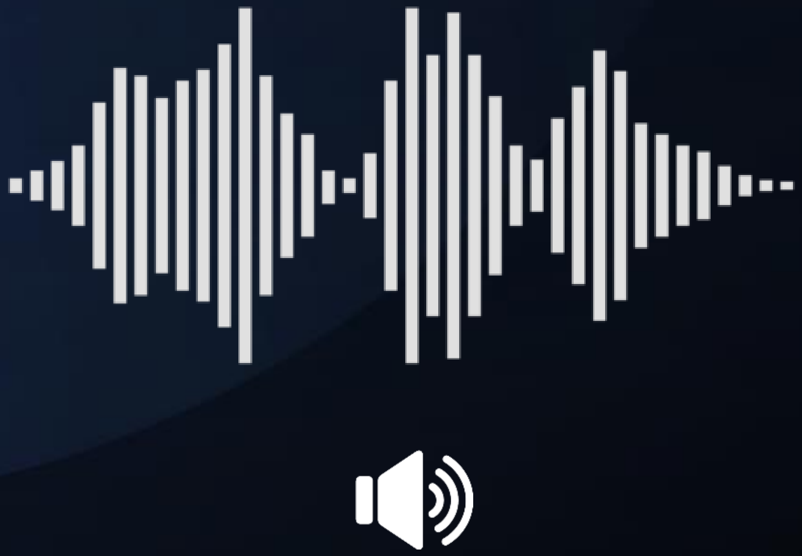
Miles Churchland, Daniel Marín & Àlex Montoya

Table Of Contents

Introduction

<u>**Topics seen in Labs sessions**</u>

- **Lab 1:** Intro Numpy/Pytorch, MLP/Gradient Descent/Linear Regression
- **Lab 2:** Recurrent Neural Networks (RNN)
- **Lab 3:** Basic and Advanced CNNs
- **Lab 4:** Generative Models GANs/VAEs



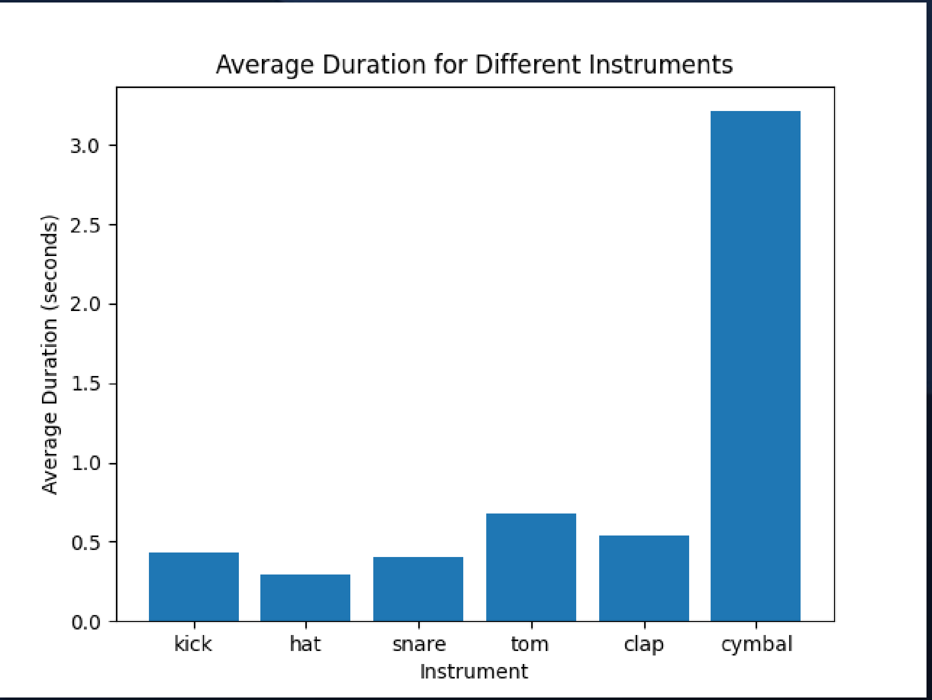
Dataset

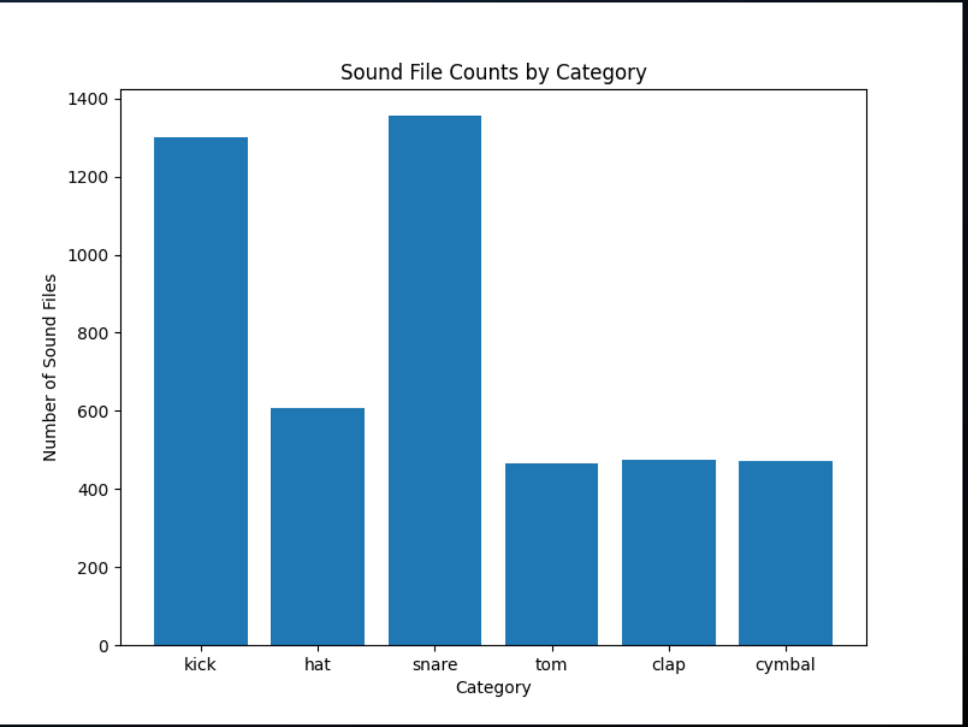
To effectively classify sounds we need a substantial dataset to train our model. We got sounds from these sources to complete our dataset. These are all copyright free and available to use for machine learning.

- Zenodo Dataset: https://zenodo.org/records/3665275
- Freewaves: freewavesamples.com
- Samples from Reddit Users: Reddit
- Live Samples from Website for Musicians: www.musicradar.com

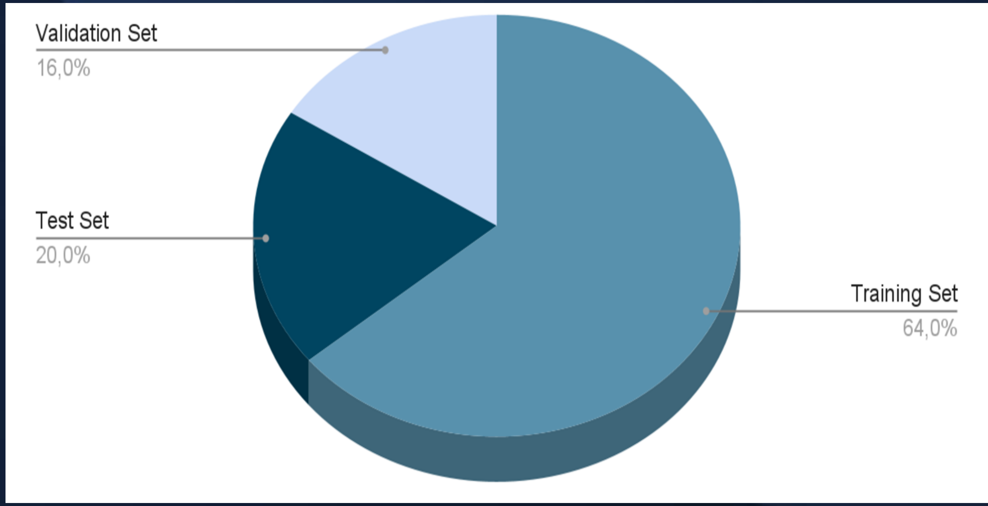
Dataset Attributes

To optimally use the dataset we must understand the characteristics of it so that we can best utilize the data contained. Our dataset is unbalanced as we have different numbers of sounds for each category. This does not affect our training. Class overlap problem: certain classes have similar features to others.





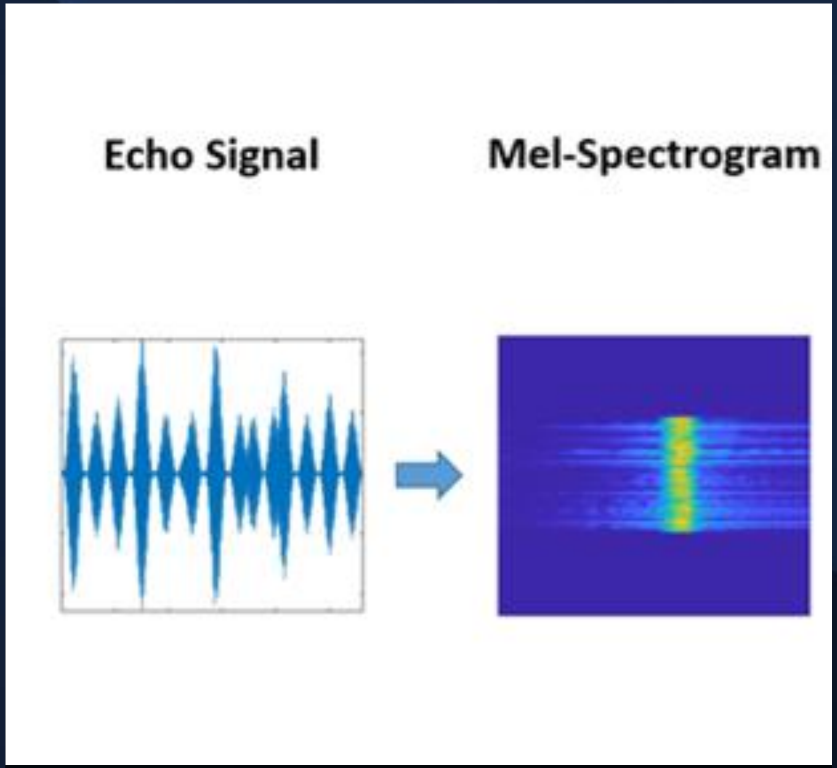
Data Splitting



- **Training Set (64%) :** Used to train the model
- **Validation Set (16%) :** Used to track model parameters and avid overfitting
- **Testing Set (20%) :** Used for checking the model's performance on new data

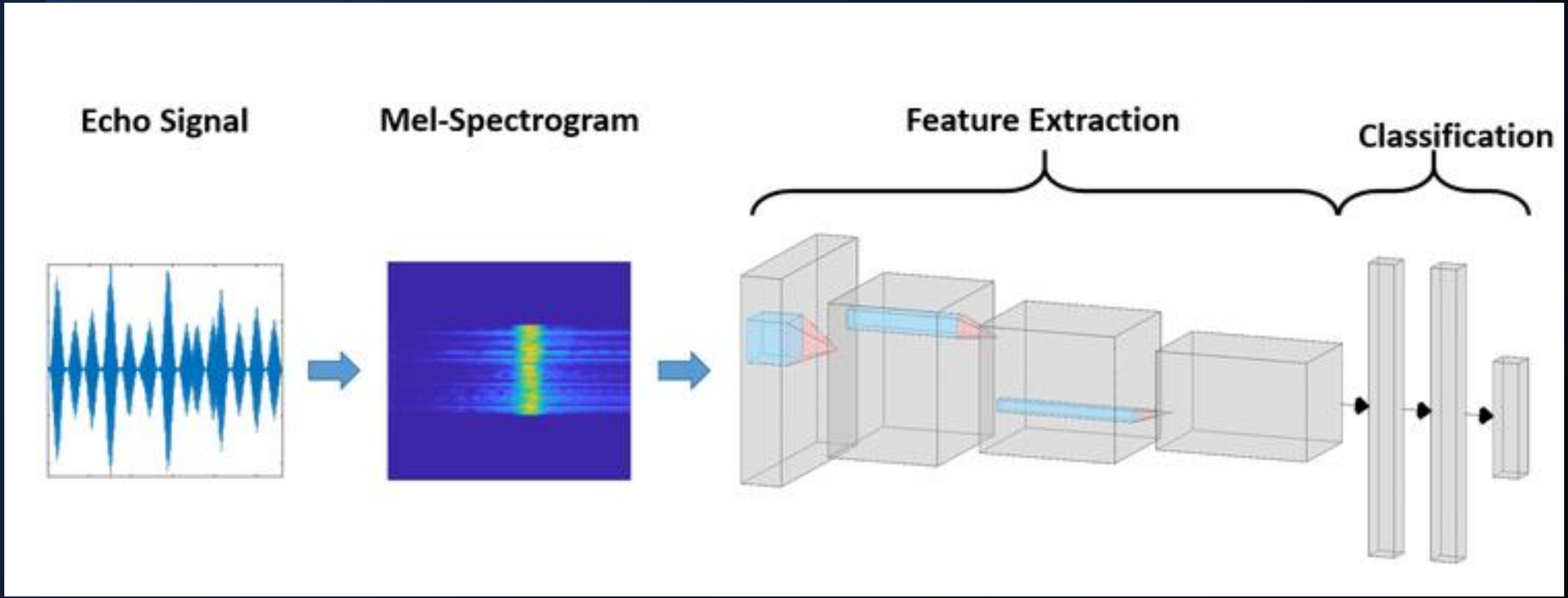
How Can We Use Deep Learning to Classify Drum Sounds?

Mel-spectrograms



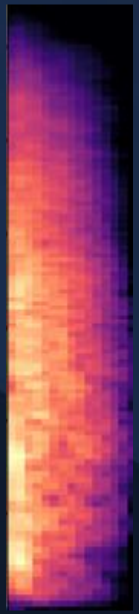
- **Convert to Mel-Spectrograms:** Use librosa to convert audio to mel spectrograms.
- **Pad Spectrograms:** Ensure a fixed width
- **Process Files:** Iterate through files, load, generate, and pad spectrograms.
- **Store in DataFrame:** Convert processed data to a pandas DataFrame.
- ***Libraries:*** *librosa, numpy, tqdm and pandas.*

Mel-spectrograms & CNN

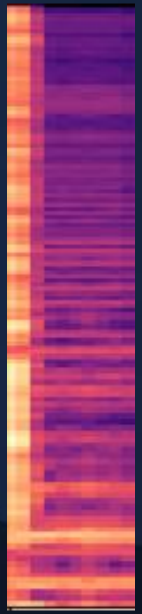


Example of Mel-spectrograms in our Dataset

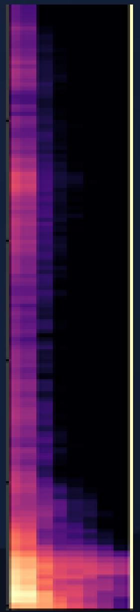
Snare-4338.wav



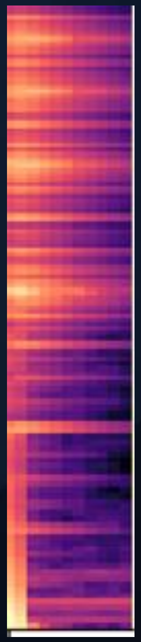
Clap-3212.wav



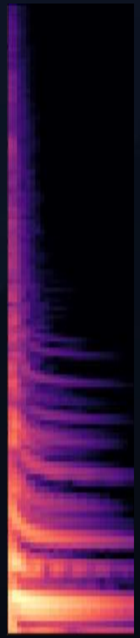
hat-414965.wav



kick-58901.wav



cymbal-231.wav



tom-909 5.wav

Defining a Basic CNN model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 32, 128, 128]             320
         MaxPool2d-2          [-1, 32, 64, 64]               0
            Conv2d-3          [-1, 64, 64, 64]          18,496
         MaxPool2d-4          [-1, 64, 32, 32]               0
            Conv2d-5         [-1, 128, 32, 32]          73,856
         MaxPool2d-6         [-1, 128, 16, 16]               0
            Linear-7                 [-1, 512]      16,777,728
            Linear-8                   [-1, 6]           3,078
================================================================
Total params: 16,873,478
Trainable params: 16,873,478
Non-trainable params: 0
----------------------------------------------------------------
```

- *Number of epochs = 20*
- *Learning Rate = 0.001*
- *Optimizer = Adam*
- *Loss = Cross Entropy*

Defining a Basic CNN model

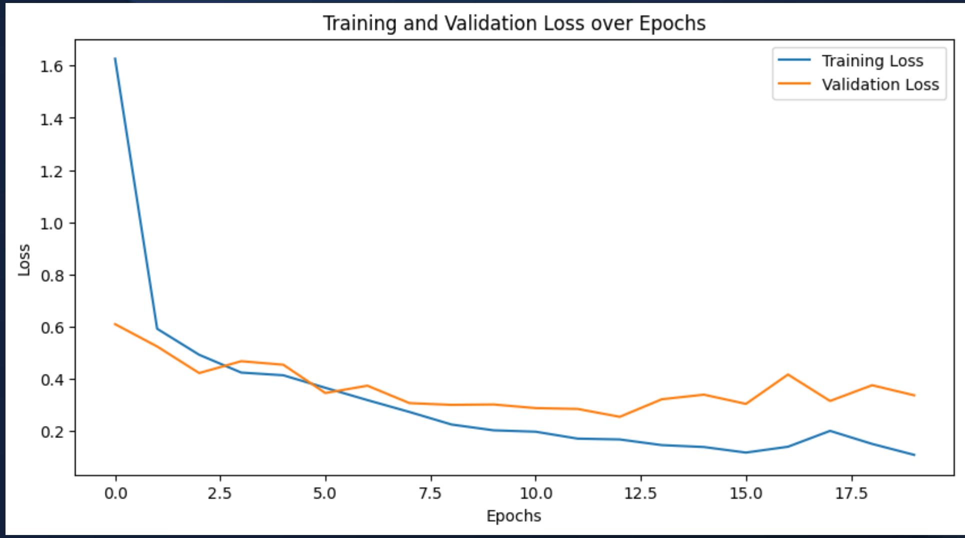
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 128, 128]             320
         MaxPool2d-2           [-1, 32, 64, 64]               0
            Conv2d-3           [-1, 64, 64, 64]          18,496
         MaxPool2d-4           [-1, 64, 32, 32]               0
            Conv2d-5          [-1, 128, 32, 32]          73,856
         MaxPool2d-6          [-1, 128, 16, 16]               0
            Linear-7                  [-1, 512]      16,777,728
            Linear-8                    [-1, 6]           3,078
================================================================
Total params: 16,873,478
Trainable params: 16,873,478
Non-trainable params: 0
----------------------------------------------------------------
```

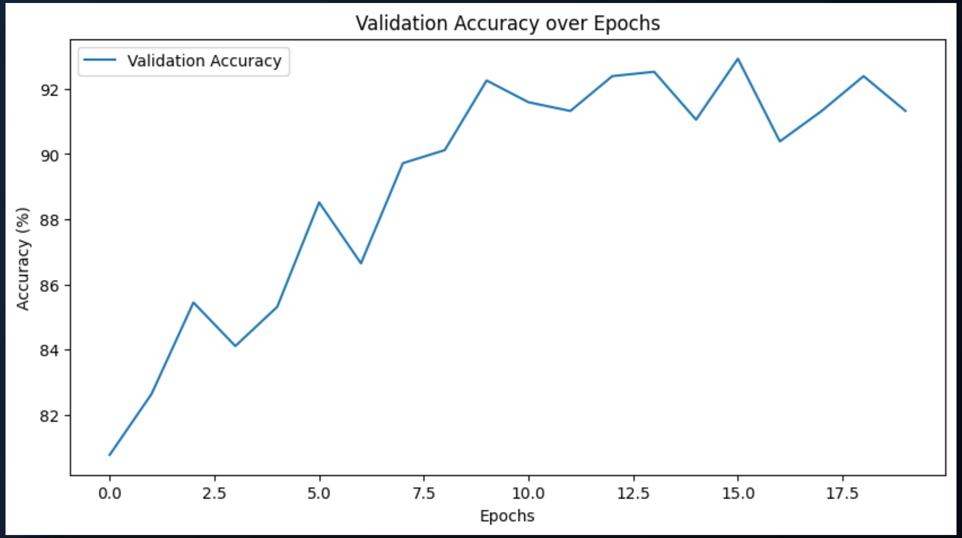
Batch Normalization & Dropout

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 128, 128]             320
       BatchNorm2d-2         [-1, 32, 128, 128]              64
         MaxPool2d-3           [-1, 32, 64, 64]               0
            Conv2d-4           [-1, 64, 64, 64]          18,496
       BatchNorm2d-5           [-1, 64, 64, 64]             128
         MaxPool2d-6           [-1, 64, 32, 32]               0
            Conv2d-7          [-1, 128, 32, 32]          73,856
       BatchNorm2d-8          [-1, 128, 32, 32]             256
         MaxPool2d-9          [-1, 128, 16, 16]               0
           Linear-10                  [-1, 512]      16,777,728
          Dropout-11                  [-1, 512]               0
           Linear-12                    [-1, 6]           3,078
================================================================
Total params: 16,873,926
Trainable params: 16,873,926
Non-trainable params: 0
----------------------------------------------------------------
```

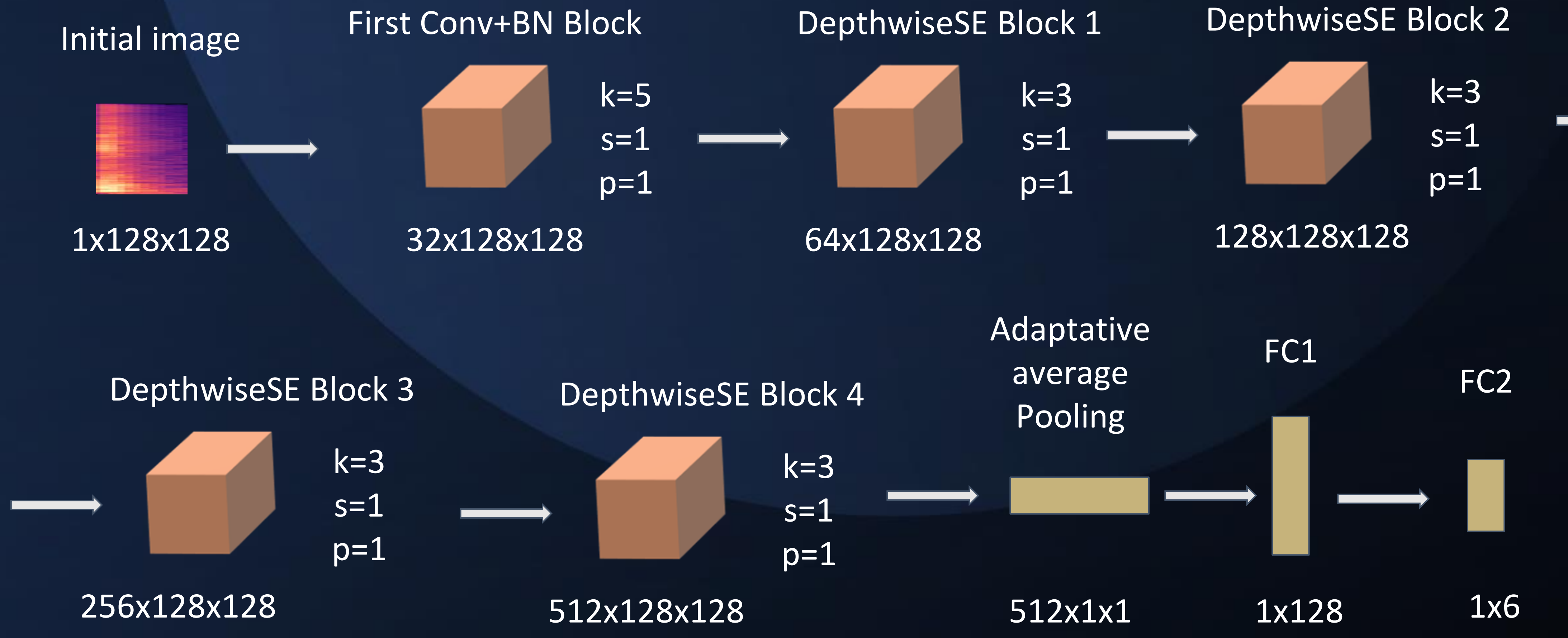
- *Number of epochs = 20*
- *Learning Rate = 0.001*
- *Optimizer = Adam*
- *Loss = Cross Entropy*

Defining a Basic CNN model





Final CNN Model



Total of 294.084 parameter
s

Depthwise SE Block

Depthwise Convolution

↓

Pointwise Convolution

↓

Batch Norm

↓

SE Block

- Applies a single convolutional filter per input channel.

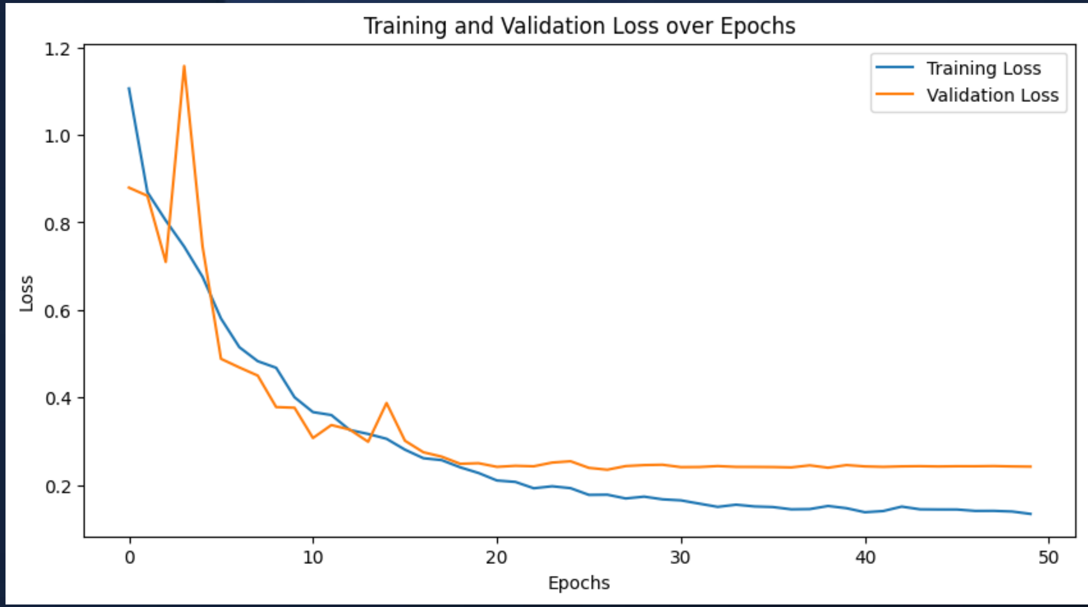
```
self.dwconv2 = nn.Conv2d(in_channels=32, out_channels=32,
kernel_size=3, stride=1, padding=1, groups=32)
```

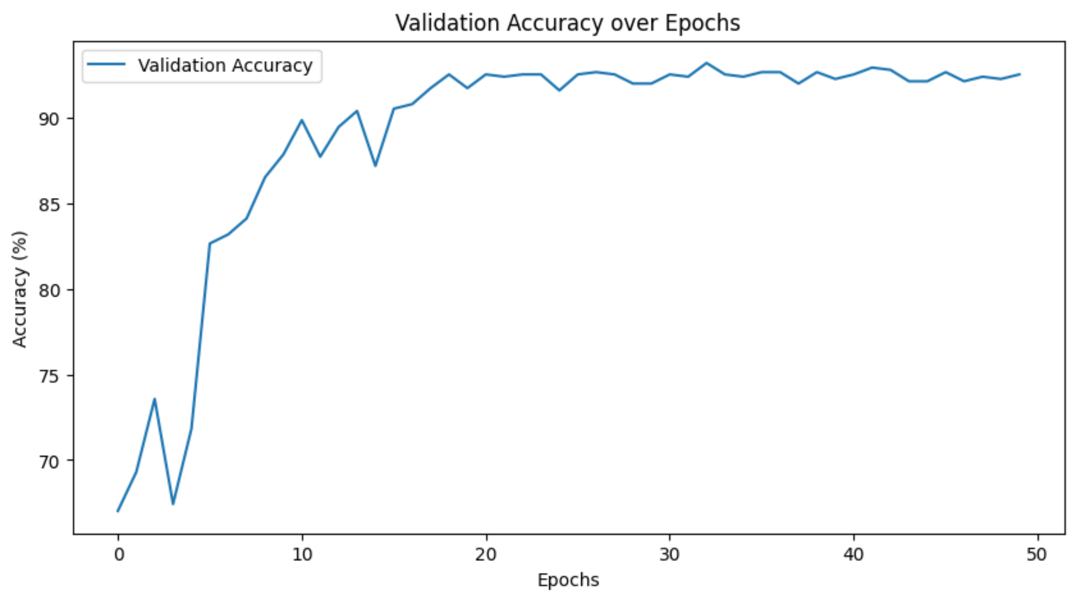
- Applies a 1x1 convolution to combine the outputs of depthwise convolution.

```
self.pwconv2 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=1, stride=1)
```

- Improve the ability of the network to focus on the most relevant features by emphasizing important channels and suppressing less useful ones.
- They adaptively recalibrate the importance of each channel based on the global context of the feature maps.

Final CNN Model





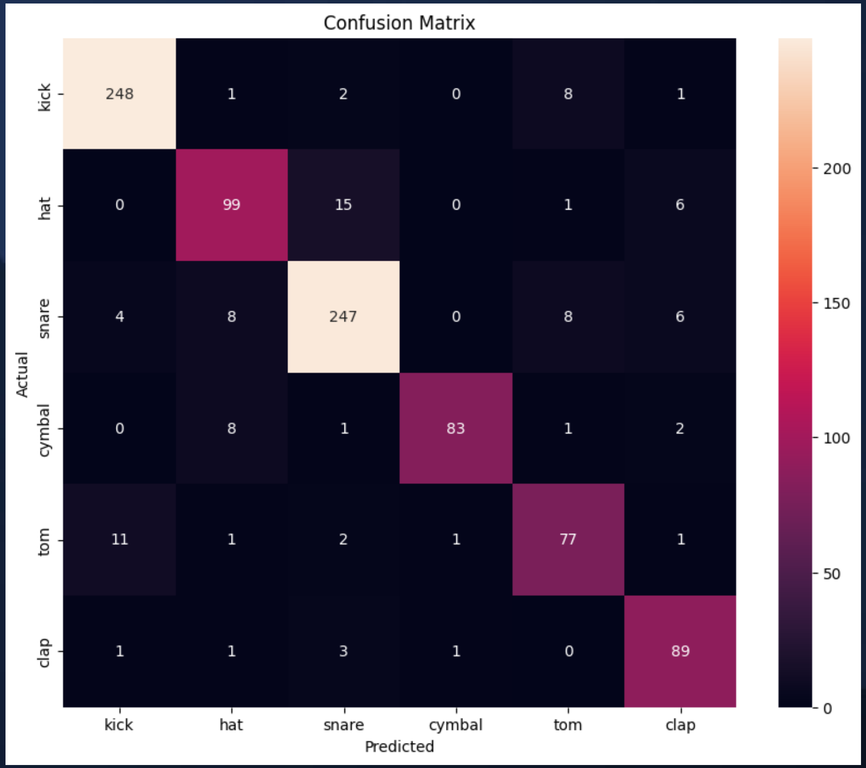
92.5% accuracy on validation set

RESULTS ANALYSIS

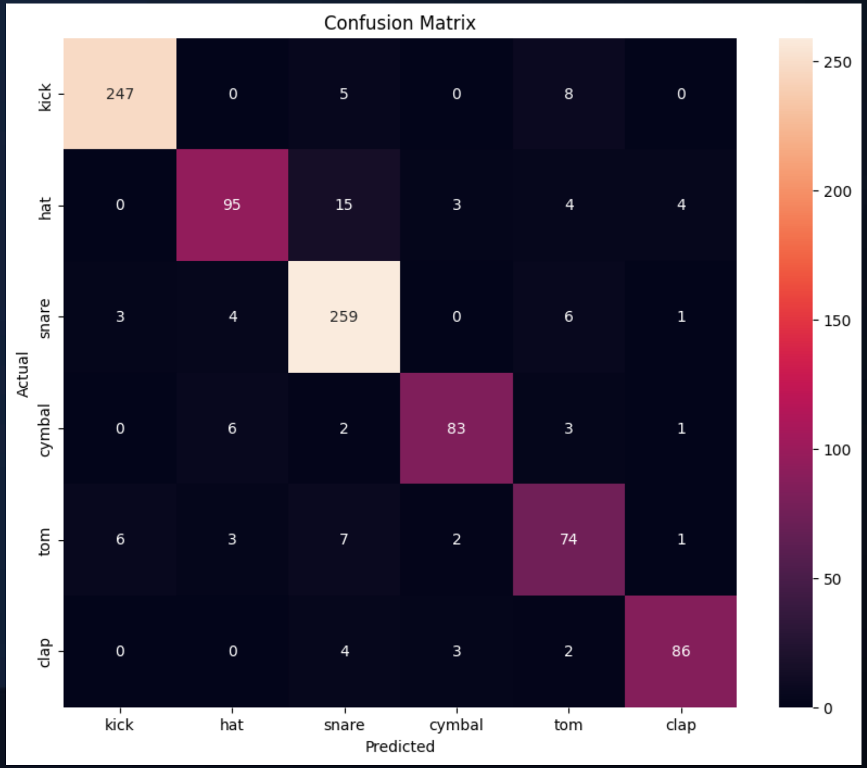
90% accuracy

(could be higher if fine tuned)

70’ training



Basic Model



Final Model

91.5% accuracy

10’ training

Sound Sorting using Trained model

- We can use our model to classify sounds that users provide.
- To do this we wrote a Python script that classifies each sound in an input folder using the trained model.
- The sounds then are moved into an folders according to the predicted class.
- Anyone can now use their own sounds with our trained model, enabling them to organize large datasets of unclassified sound files.

```
for file_name in tqdm(os.listdir(input_dir), desc="Classifying audio files"):
    file_path = os.path.join(input_dir, file_name)
    if os.path.isfile(file_path):
        try:
            # Load and preprocess the audio file
            y, sr = librosa.load(file_path, sr=None)
            mel_spectrogram = generate_mel_spectrogram(y, sr)
            mel_spectrogram = pad_spectrogram(mel_spectrogram, max_len=fixed_
            mel_spectrogram_tensor = to_tensor(mel_spectrogram)

            # Make prediction
            with torch.no_grad():
                outputs = model(mel_spectrogram_tensor)
                _, predicted = torch.max(outputs.data, 1)

            # Convert prediction to label
            predicted_label = categories[predicted.item()]

            # Create the target directory if it doesn't exist
            target_dir = os.path.join(output_dir, predicted_label)
            os.makedirs(target_dir, exist_ok=True)

            # Move the file to the target directory
            shutil.move(file_path, os.path.join(target_dir, file_name))
```

CONCLUSIONS

- Successes
    - Application and Model Usage: After converting audio to mel spectrograms, padding them, and processing the files, we can utilize our trained model to classify sounds. We found that using a CNN on spectrograms proved effective in sound classification problems.
    - New sounds are rapidly classified by the trained model, and its performance on these sounds is robust, indicating that the model generalizes well.

- Areas of Improvement
    - Class Overlap: Multiple classes share similar features in their spectrograms, the model occasionally struggles to distinguish between them accurately.

THANK YOU FOR YOUR ATTENTION

Deep Learning Final Project

Group 3, P102

Miles Churchland, Daniel Marín & Àlex Montoya