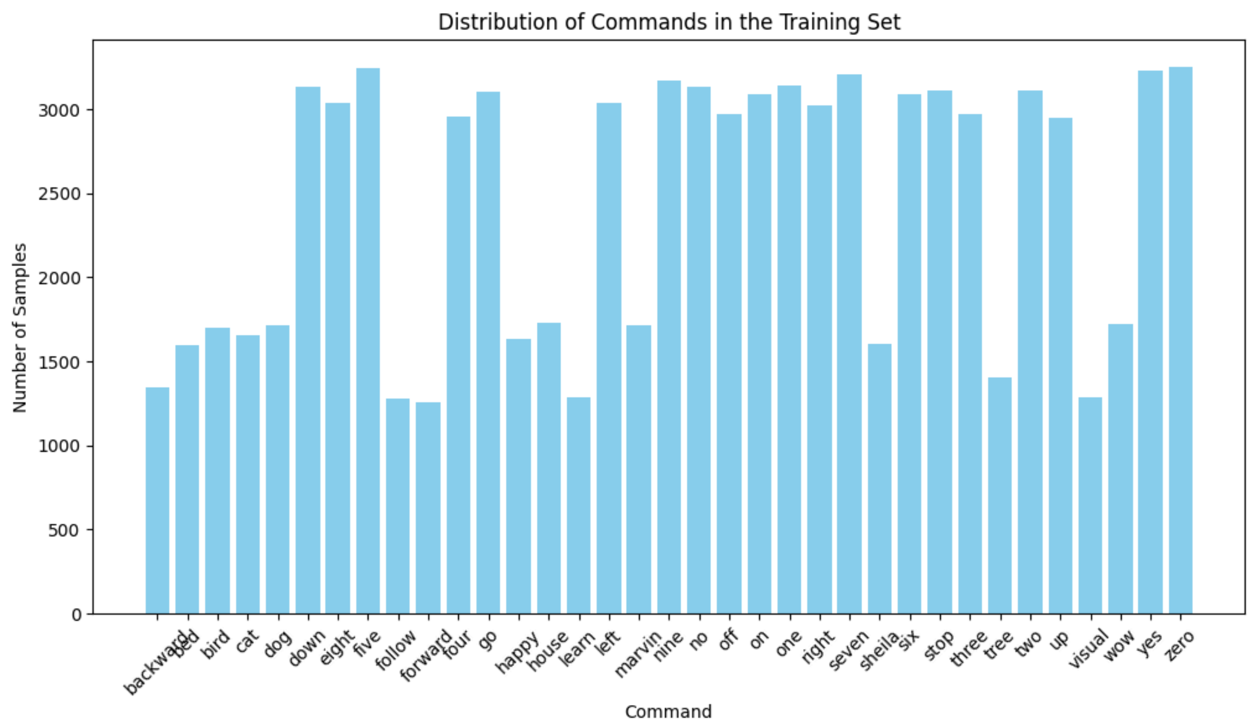
PROJECT REPORT:

In my statistical analysis of the Speech Commands dataset, I focused on examining the distribution and characteristics of the data. I explored the number of recordings per command, identifying class imbalances that could affect model performance. By visualizing data distributions and calculating key statistics, I uncovered trends and potential biases, which guided my approach to model training and fine-tuning.



During the training phase, I carefully fine-tuned the CRNN model using the Speech Commands dataset. I adjusted hyperparameters like learning rate and batch size, monitored loss and accuracy metrics, and iterated on the training process to optimize performance. By leveraging subsampling and ensuring consistent data formatting, I balanced the dataset to handle class imbalances effectively.

```
import os
import random
import torch
import torch.nn as nn
import torch.optim as optim
```

```python
import torch.nn.functional as F
from torch.utils.data import DataLoader, Subset
from torchaudio.datasets import SPEECHCOMMANDS
import torchaudio.transforms as transforms

class CRNN(nn.Module):
   def __init__(self, num_classes):
       super(CRNN, self).__init__()
       self.cnn = nn.Sequential(
           nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1),
           nn.BatchNorm2d(16),
           nn.ReLU(),
           nn.MaxPool2d(2, 2),

           nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1),
           nn.BatchNorm2d(32),
           nn.ReLU(),
           nn.MaxPool2d(2, 2),

           nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
           nn.BatchNorm2d(64),
           nn.ReLU(),
           nn.MaxPool2d(2, 2)
       )
       self.rnn = nn.GRU(input_size=64, hidden_size=128, num_layers=2,
batch_first=True, bidirectional=True)
       self.fc = nn.Linear(128 * 2, num_classes)

   def forward(self, x):
       x = self.cnn(x)
       x = x.permute(0, 2, 3, 1)
       batch_size, height, width, channels = x.shape
       x = x.view(batch_size, height * width, channels)
       x, _ = self.rnn(x)
       x = self.fc(x[:, -1, :])
       return x

class SubsetSC(SPEECHCOMMANDS):
   def __init__(self, subset: str = None):
       super().__init__("./", download=True)
```

```python
        def load_list(filename):
            filepath = os.path.join(self._path, filename)
            with open(filepath) as fileobj:
                return [os.path.normpath(os.path.join(self._path,
line.strip())) for line in fileobj]

        if subset == "validation":
            self._walker = load_list("validation_list.txt")
        elif subset == "testing":
            self._walker = load_list("testing_list.txt")
        elif subset == "training":
            excludes = load_list("validation_list.txt") +
load_list("testing_list.txt")
            excludes = set(excludes)
            self._walker = [w for w in self._walker if w not in excludes]

def subsample_dataset(dataset, fraction=0.3):
    """Subsample a fraction of the dataset."""
    total_samples = len(dataset)
    subsample_size = int(total_samples * fraction)
    indices = random.sample(range(total_samples), subsample_size)
    return Subset(dataset, indices)

transform = transforms.MelSpectrogram(sample_rate=16000, n_mels=64)

def collate_fn(batch):
    desired_height = 64
    desired_width = 81

    waveforms, labels = [], []
    for waveform, _, label, _, _ in batch:

        spec = transform(waveform).squeeze(0)
        spec = ensure_size(spec, desired_height, desired_width)

        waveforms.append(spec.unsqueeze(0))

        labels.append(label)
```

```
   waveforms = torch.stack(waveforms)
   label_map = {label: idx for idx, label in
enumerate(sorted(set(labels)))}
   targets = torch.tensor([label_map[label] for label in labels],
dtype=torch.long)

   return waveforms, targets, label_map

def ensure_size(tensor, desired_height, desired_width):
   height, width = tensor.shape
   if height < desired_height:
       padding = (0, 0, 0, desired_height - height)
       tensor = F.pad(tensor, padding, mode='constant', value=0)
   elif height > desired_height:
       tensor = tensor[:desired_height, :]
   if width < desired_width:
       padding = (0, desired_width - width)
       tensor = F.pad(tensor, padding, mode='constant', value=0)
   elif width > desired_width:
       tensor = tensor[:, :desired_width]

   return tensor
train_set = SubsetSC("training")
test_set = SubsetSC("testing")
train_set_subsampled = subsample_dataset(train_set, fraction=0.1)
test_set_subsampled = subsample_dataset(test_set, fraction=0.1)
train_loader = DataLoader(train_set_subsampled, batch_size=32,
shuffle=True, collate_fn=collate_fn)
test_loader = DataLoader(test_set_subsampled, batch_size=32,
shuffle=False, collate_fn=collate_fn)

num_classes = len(set([label for _, _, label, _, _ in train_set]))

model = CRNN(num_classes=num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Dataset Creation

To enhance my model's performance, I created a personalized dataset by recording 30 samples of each command word in my voice, ensuring a diverse range of pronunciations. I organized these recordings systematically, naming them according to command words and user IDs, and verified their consistency with the existing dataset format. This careful curation allowed me to incorporate my voice data seamlessly into the model, making the training set more representative and improving the model's adaptability to real-world usage.

Fine-Tuning

Fine-tuning was a crucial step in my model improvement process. Using the newly created dataset, I further trained the CRNN model to adapt specifically to my voice characteristics. I adjusted the learning rate and the number of epochs, and re-evaluated the loss and accuracy metrics with each pass. By iterating on this fine-tuning process, I was able to significantly enhance the model's performance, making it more responsive and precise in recognizing commands, especially when spoken by me. This personalized approach ensured the model was not just accurate but also tailored to real-life application scenarios.