report

October 29, 2024

1 Task 1: 10-Word Speech Recognition

Student: Aleksandr J. Smoliakov, VU MIF, Data Science, year 1
Date: 22nd October 2024

In this report, we will build a speech recognition system that recognizes 10 words. The system will be based on the Mel-frequency cepstral coefficient (MFCC) features and a Convolutional Neural Network with 4 stacks of 2D convolution + max pooling, followed by 2 dense layers. Finally, we will evaluate the model on a test set and analyze the results.

1.1 Dataset

The dataset used in this assignment is sampled from the **Google's Speech Commands Dataset** version **0.02**, which consists of 105 thousands of one-second long utterances of 30 short words. The dataset is available at https://huggingface.co/datasets/google/speech_commands.

In this assignment, we will use a subset of the dataset that contains 10 words:
`zero`, `one`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, and `nine`.

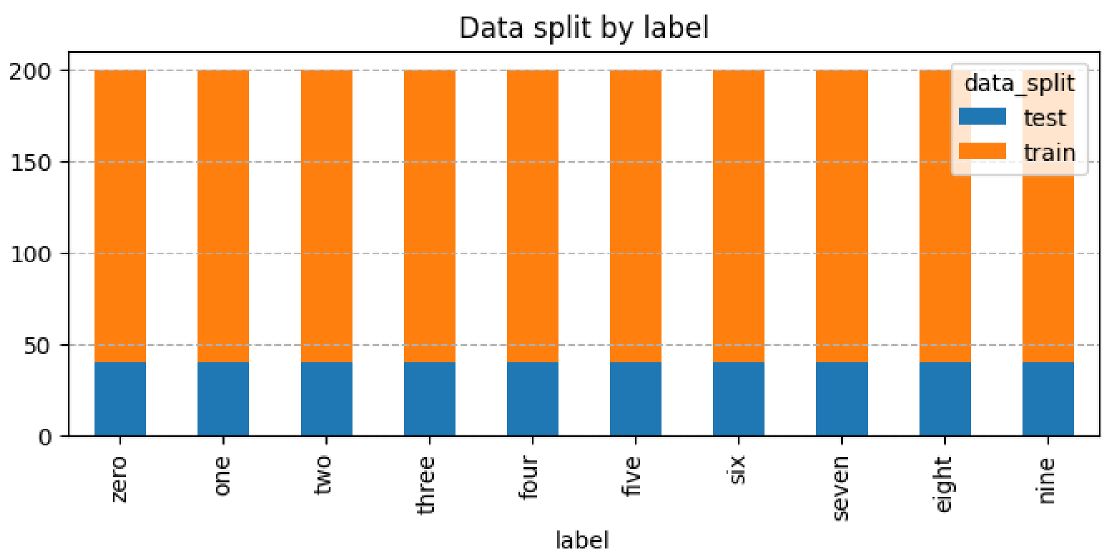
Downloading the full Speech Commands Dataset:

Filtering the 10 words defined above and randomly selecting 200 samples for each word:

Let's split the samples into training (80%) and testing (20%) sets. Since the samples were randomly shuffled by `.sample()` above, we can simply assign every 5th sample to the test set without any bias.

1.2 Exploratory Data Analysis

Let's plot the distribution of samples in the training and testing sets.



Evidently, the dataset is perfectly balanced in terms of labels, with each word having exactly 160 samples in the training set and 40 samples in the testing set, as intended.

Speakers The number of unique speakers in our dataset:

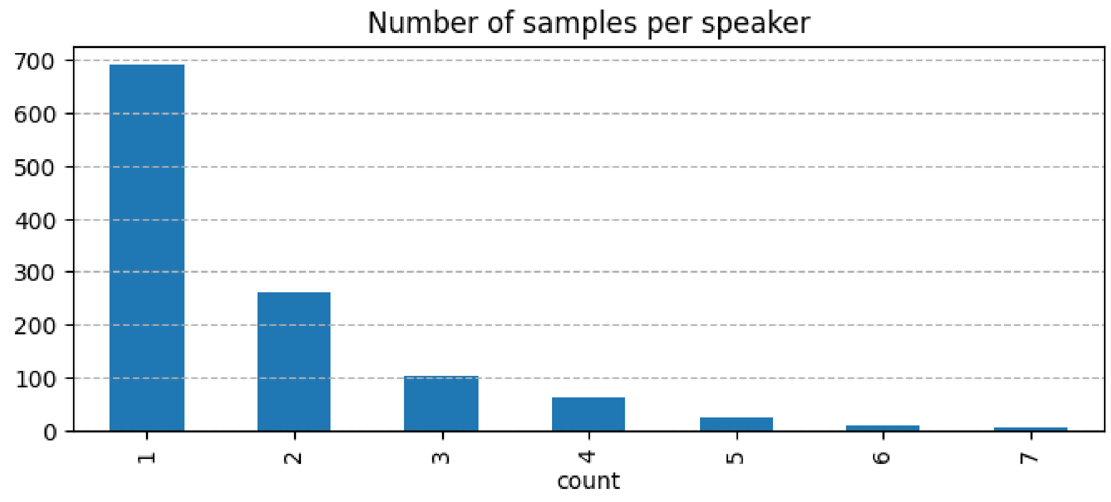
```
1160
```

Now, let's check the number of speakers of each word.

```
label     zero  one  two  three  four  five  six  seven  eight  nine
speakers   187  191  189    193   184   184  188    185    184   191
```

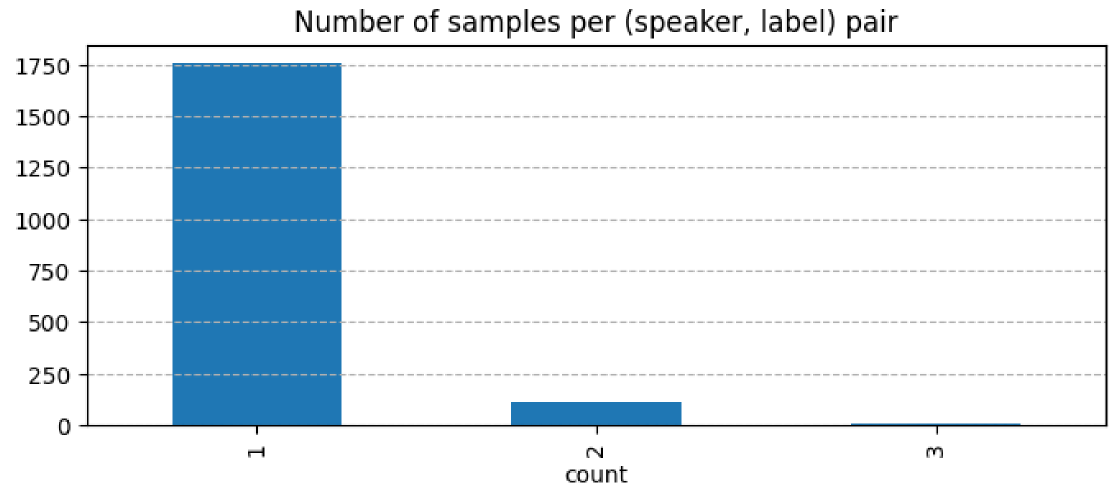
The number of unique speakers per each word is between 82-91% of the total number of samples for each word. This means that the dataset is well balanced in terms of the number of speakers.

Distribution of the number of samples per speaker:



In our dataset, over half of the speakers have recorded only one sample, and over 90% of speakers have recorded 4 or fewer samples. There are a lot of unique speakers - this is likely to increase the in-class variability and make it more challenging for the model to fit to the training data.

However, this is likely to lead to a more robust model that generalizes well to unseen speakers.



In our dataset, the vast majority of speakers only have a single utterance of a word. The maximum number of samples per (speaker, label) is 3, which is quite low.

Sample rate The sample rate of all recordings is exactly 16 kHz:

```
             count
sample_rate
```

```
16000             2000
```

1.3 Feature extraction

We will use the Mel-frequency cepstral coefficients (MFCC) as features for our speech recognition model. MFCCs are widely used in speech recognition systems because they mimic the human auditory system's response more closely than the raw audio signal. The frequency scale is scaled to the Mel scale, which is similar to the logarithmic scale.

Our parameters for the MFCC transformation:

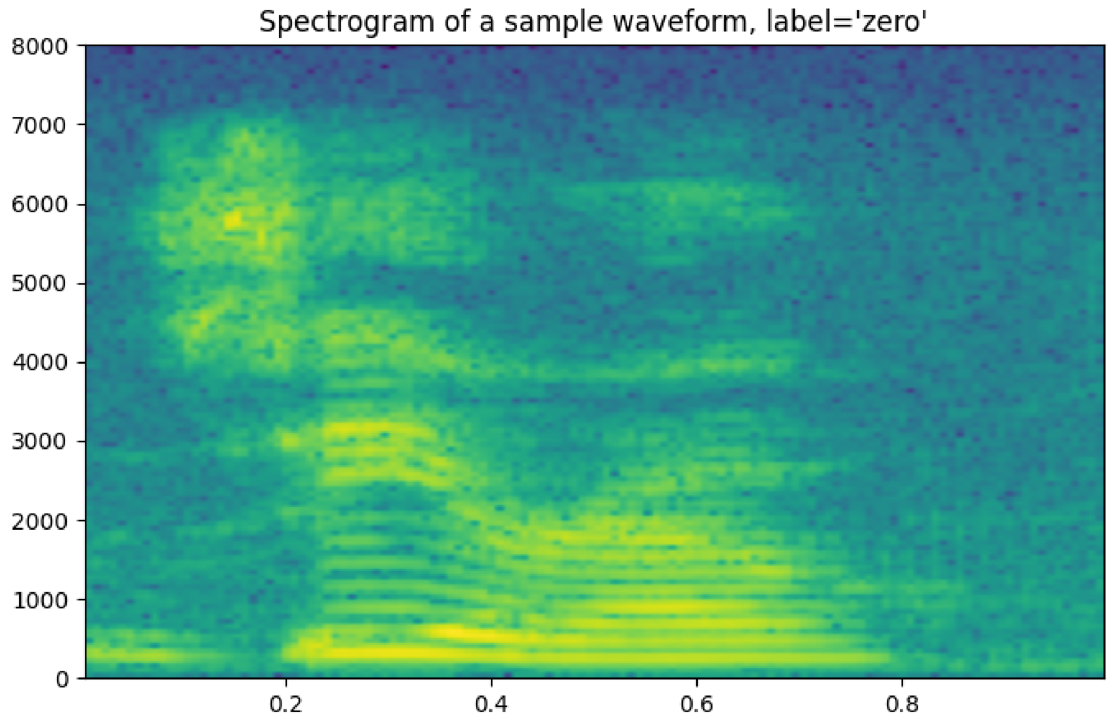
- Sample size: 16000 Hz
- Maximum frequency: 4000 Hz
- Window size: 32 ms (512 samples)
- Hop length: 8 ms (128 samples)
- Number of MFCC coefficients: 20
- Number of Mel filters: 64

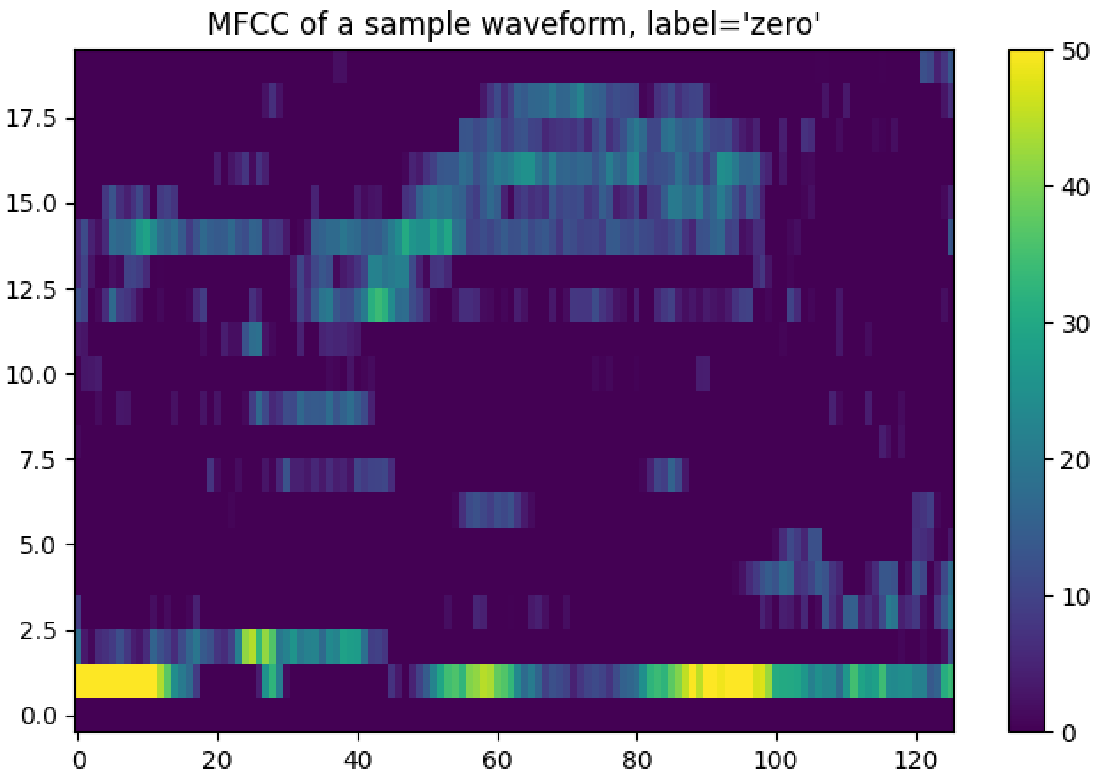
We will transform the raw audio samples into MFCC features the following way:

- Load the audio samples using `torchaudio`
- Pad/trim the audio to a fixed length of exactly 16000 samples at 16 kHz (1 second)
- Apply the MFCC transformation to the audio samples

1.3.1 Create datasets

Let's plot the raw audio waveform and the MFCC features for a sample from the training set.





MFCC features capture the essential information in the audio signal while significantly reducing the dimensionality of the input data.

1.4 Model training

1.4.1 Creating datasets and dataloaders

We will create PyTorch datasets and dataloaders for the training and testing sets. The full dataset of 5,000 samples has already been split into 4,000 samples for training and 1,000 samples for testing.

We will use a batch size of 32.

1.4.2 Model architecture

We have considered using these model architectures:

- Convolutional Neural Network (CNN)
- Long Short-Term Memory (LSTM)

While Recurrent architectures like RNN, LSTM, and GRU can also be used for speech recognition, we have chosen to use a Convolutional Neural Network (CNN) for this task. CNNs are known to be very effective in image recognition tasks, and they can also be used for speech recognition.

In `model.py` we have defined a convolutional neural network with 4 stacks of 2D convolution + max pooling, followed by 2 dense layers. The model architecture is given below:

Let's visualize the model architecture:

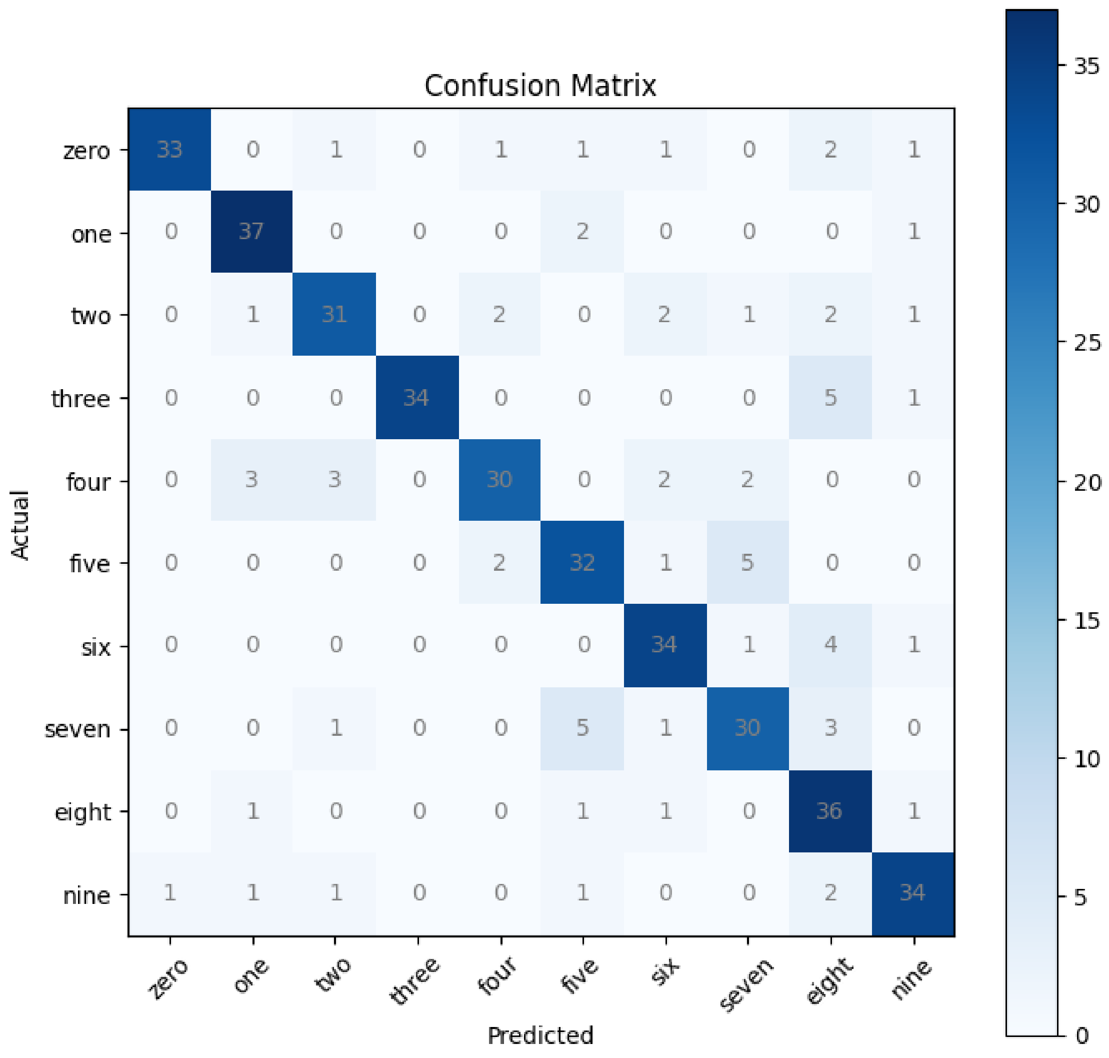
```
AudioClassifier(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Dropout(p=0.25, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Dropout(p=0.25, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Dropout(p=0.25, inplace=False)
  )
  (conv4): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc1): Linear(in_features=896, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

Let's train the model for 25 epochs and evaluate it on the test set.

```
Epoch 1/25, Loss: 2.426
Epoch 1/25, Test Loss: 2.295, Test Acc: 0.107
Epoch 2/25, Loss: 2.28
Epoch 2/25, Test Loss: 2.23, Test Acc: 0.212
Epoch 3/25, Loss: 2.147
Epoch 3/25, Test Loss: 2.042, Test Acc: 0.225
Epoch 4/25, Loss: 1.914
Epoch 4/25, Test Loss: 1.757, Test Acc: 0.36
Epoch 5/25, Loss: 1.682
Epoch 5/25, Test Loss: 1.552, Test Acc: 0.405
Epoch 6/25, Loss: 1.438
Epoch 6/25, Test Loss: 1.259, Test Acc: 0.59
Epoch 7/25, Loss: 1.192
Epoch 7/25, Test Loss: 1.288, Test Acc: 0.583
Epoch 8/25, Loss: 1.065
Epoch 8/25, Test Loss: 0.956, Test Acc: 0.69
Epoch 9/25, Loss: 0.884
Epoch 9/25, Test Loss: 0.866, Test Acc: 0.72
Epoch 10/25, Loss: 0.76
Epoch 10/25, Test Loss: 0.801, Test Acc: 0.735
Epoch 11/25, Loss: 0.69
Epoch 11/25, Test Loss: 0.784, Test Acc: 0.728
Epoch 12/25, Loss: 0.613
Epoch 12/25, Test Loss: 0.659, Test Acc: 0.782
Epoch 13/25, Loss: 0.562
Epoch 13/25, Test Loss: 0.692, Test Acc: 0.762
Epoch 14/25, Loss: 0.514
Epoch 14/25, Test Loss: 0.649, Test Acc: 0.795
Epoch 15/25, Loss: 0.494
Epoch 15/25, Test Loss: 0.718, Test Acc: 0.762
Epoch 16/25, Loss: 0.451
Epoch 16/25, Test Loss: 0.593, Test Acc: 0.81
Epoch 17/25, Loss: 0.383
Epoch 17/25, Test Loss: 0.592, Test Acc: 0.795
Epoch 18/25, Loss: 0.358
Epoch 18/25, Test Loss: 0.561, Test Acc: 0.81
Epoch 19/25, Loss: 0.3
Epoch 19/25, Test Loss: 0.589, Test Acc: 0.818
Epoch 20/25, Loss: 0.304
Epoch 20/25, Test Loss: 0.519, Test Acc: 0.825
Epoch 21/25, Loss: 0.279
Epoch 21/25, Test Loss: 0.591, Test Acc: 0.802
Epoch 22/25, Loss: 0.255
Epoch 22/25, Test Loss: 0.629, Test Acc: 0.812
Epoch 23/25, Loss: 0.278
Epoch 23/25, Test Loss: 0.605, Test Acc: 0.795
Epoch 24/25, Loss: 0.239
Epoch 24/25, Test Loss: 0.532, Test Acc: 0.835
Epoch 25/25, Loss: 0.231
Epoch 25/25, Test Loss: 0.573, Test Acc: 0.828
```

The model has been successfully trained on the training set. Its loss has decreased over time until the test loss has started to converge. The model has achieved a test accuracy of 82.8% after 25 epochs.

Let's evaluate the model on the test set and plot the confusion matrix.

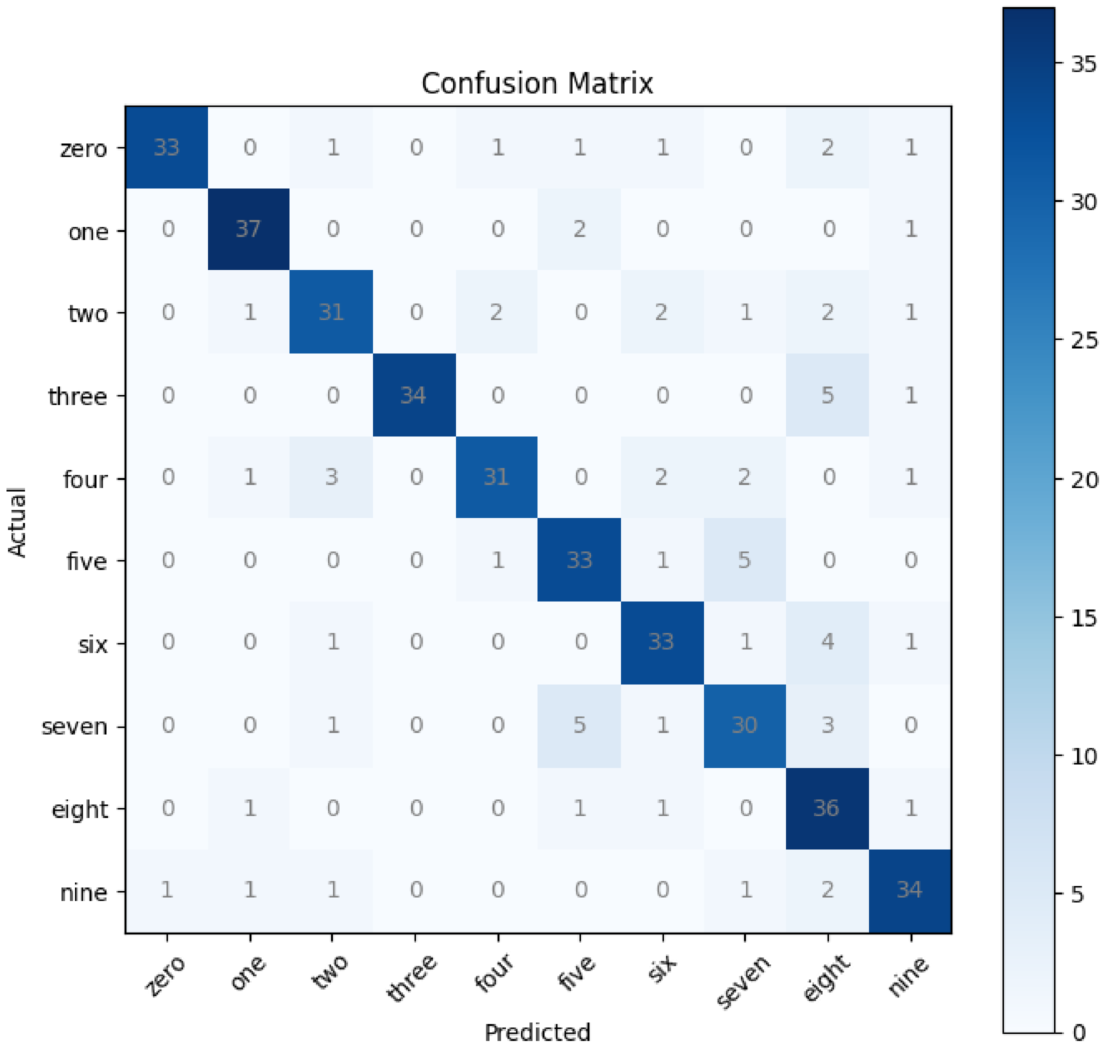


```
Accuracy: 0.8275
```

1.4.3 Adding noise to the training data

Let's test our model on noisy data. We will add 40 dB, then 20 dB of white noise to the test data and evaluate the model's performance.

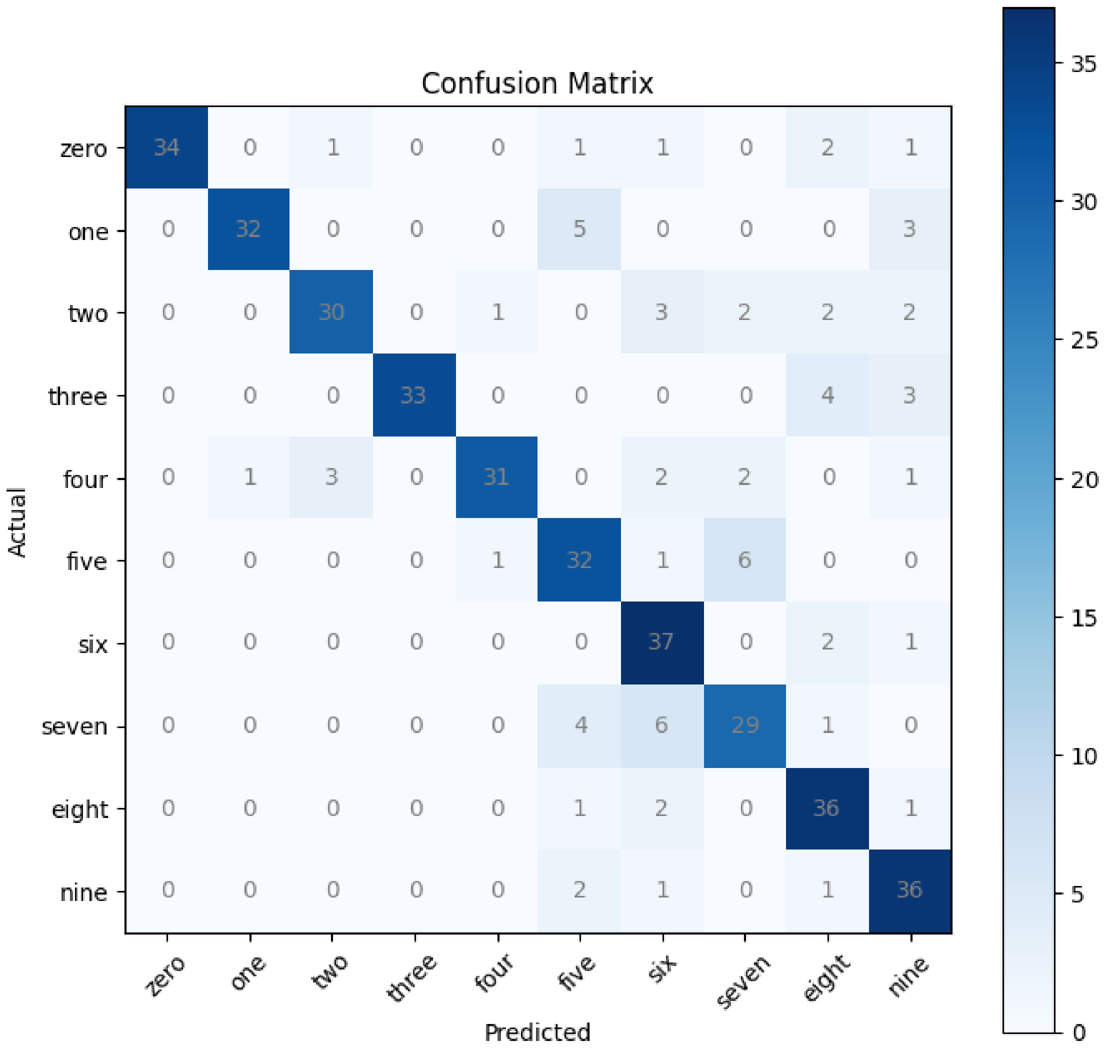
1.4.4 40 dB white noise



```
Accuracy with 40db SNR: 0.83
```

The accuracy of the model on the test set with 40 dB white noise is 83.0%, which negligibly (by 0.2%) higher than the accuracy on the clean test set. This indicates that the model is robust to 40 dB white noise.

1.4.5 20 dB white noise



```
Accuracy with 20db SNR: 0.825
```

The accuracy of the model on the test set with 20 dB white noise is 82.5%, which is by 0.3% lower than the accuracy on the clean test set. This indicates that the model is still quite robust to 20 dB white noise.