Course Project Report

Task - ES17

- Group: D14
- Tushar Sharma (B21CS095), Niraj Patel (B21CS053)

Aim:

Implementation of Deep Learning Algorithm on PYNQ-Z2 Board

Introduction:

In this project, we implemented a deep learning algorithm using a hardware software interface platform, specifically the PYNQ-Z2 board. The objective was to leverage the capabilities of the board to accelerate the execution of the algorithm. The task involved training a deep neural network (DNN) model to classify speech commands from the mini speech commands dataset.

Dataset used: Speech Dataset

Methodology:

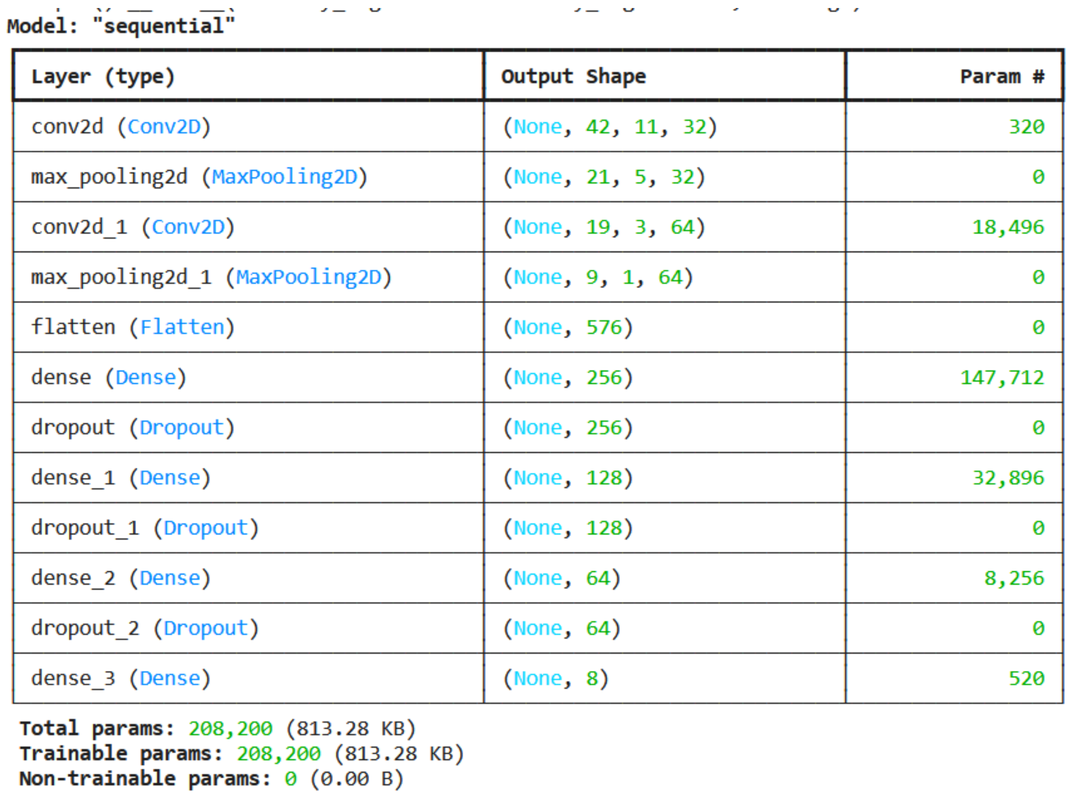
- We utilized TensorFlow and Keras to build and train a DNN model for speech command classification.
- The dataset was preprocessed to extract MFCC features from audio files.
- MFCC (Mel-Frequency Cepstral Coefficients) features are extracted from audio files as part of the preprocessing step. These features capture the spectral characteristics of the audio signals and are commonly used in speech and audio processing tasks.
- The model architecture consisted of convolutional and dense layers, followed by softmax activation for classification.
- We experimented with different hyperparameters such as learning rate, batch size, epochs, and patience value for early stopping.

Code Explanation:

- Importing Libraries:The code starts by importing necessary libraries such as TensorFlow, librosa for audio processing, pathlib for handling file paths, and json for data serialization.
- Loading Data: The dataset containing audio files of speech commands is loaded from the specified path. If the dataset does not exist locally, it is downloaded from a remote source using TensorFlow's utility function.

```
Dataset loaded.
Train set shape: (4593, 44, 13, 1)
Validation set shape: (1149, 44, 13, 1)
Test set shape: (1436, 44, 13, 1)
```

- MFCC Feature Extraction: The `extract_features` function extracts Mel-Frequency Cepstral Coefficients (MFCCs) from each audio file. It uses the librosa library to load the audio file, extract MFCC features, and return them as a list.
- Data Preprocessing: The `preprocess_data` function iterates through the directories of the dataset, extracts MFCC features for each audio file, and stores them along with their labels and file paths in a JSON file. This JSON file serves as a structured representation of the dataset.
- Training and Testing Data Split: The `prepare_dataset` function loads the dataset from the JSON file, splits it into training, validation, and testing sets, and reshapes the arrays to include a channel dimension. The train-validation split further splits the training data for hyperparameter tuning.
- Model Building: The `build_dnn` function defines the architecture of a Deep Neural Network (DNN) model using TensorFlow's Sequential API. It consists of convolutional and dense layers, with softmax activation for classification. The model is compiled with an Adam optimizer and sparse categorical cross-entropy loss.

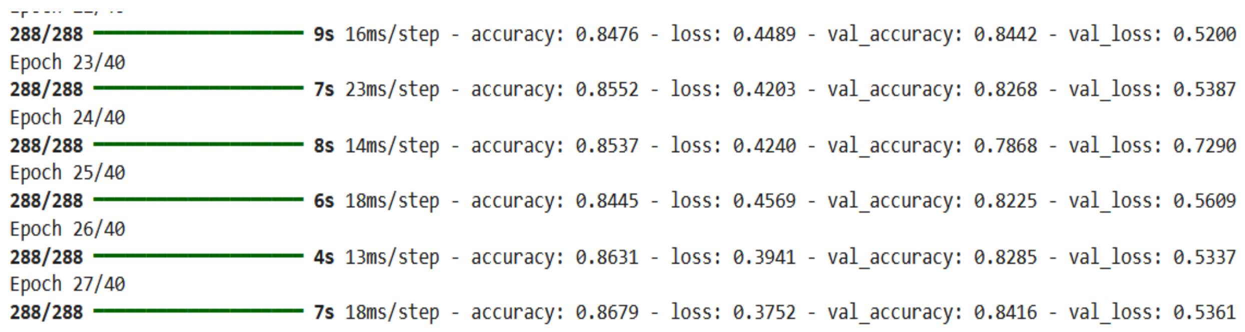


Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 42, 11, 32)	320
max_pooling2d (MaxPooling2D)	(None, 21, 5, 32)	0
conv2d_1 (Conv2D)	(None, 19, 3, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 9, 1, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 256)	147,712
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 8)	520

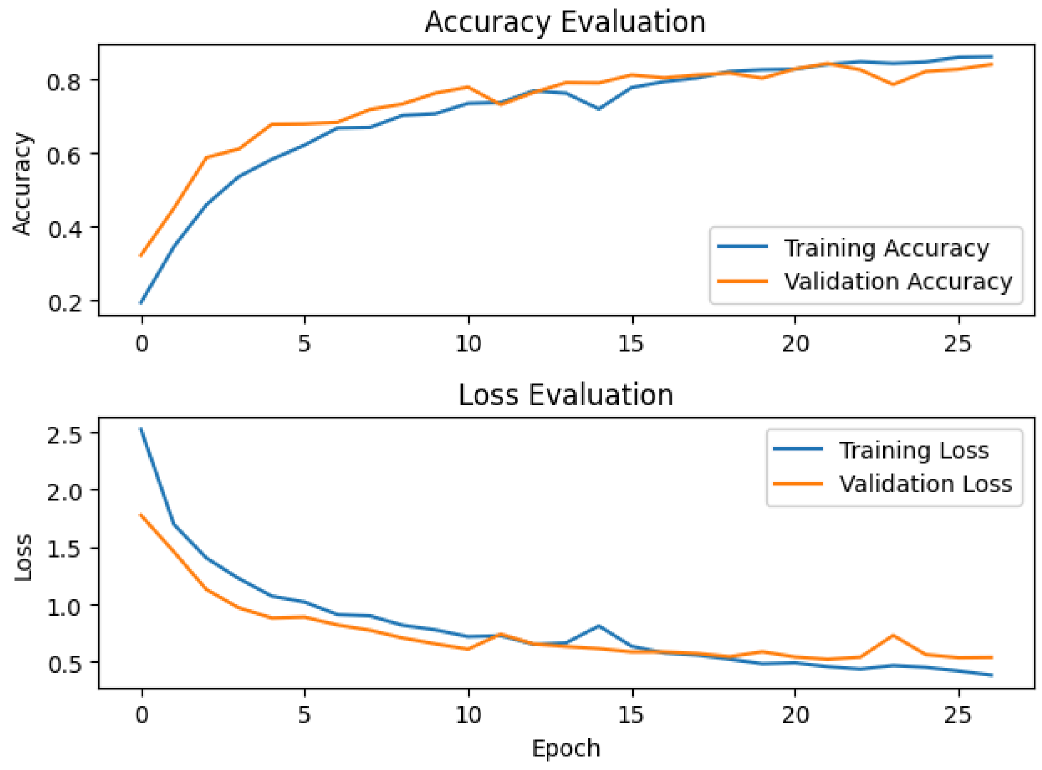
```
Total params: 208,200 (813.28 KB)
Trainable params: 208,200 (813.28 KB)
Non-trainable params: 0 (0.00 B)
```

- Model Training: The `train` function trains the DNN model on the training data for a specified number of epochs. It uses early stopping with a patience parameter to prevent overfitting and restore the best weights.



```
288/288 ━━━━━━━━━━━━━━━━━━━━ 9s 16ms/step - accuracy: 0.8476 - loss: 0.4489 - val_accuracy: 0.8442 - val_loss: 0.5200
Epoch 23/40
288/288 ━━━━━━━━━━━━━━━━━━━━ 7s 23ms/step - accuracy: 0.8552 - loss: 0.4203 - val_accuracy: 0.8268 - val_loss: 0.5387
Epoch 24/40
288/288 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - accuracy: 0.8537 - loss: 0.4240 - val_accuracy: 0.7868 - val_loss: 0.7290
Epoch 25/40
288/288 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - accuracy: 0.8445 - loss: 0.4569 - val_accuracy: 0.8225 - val_loss: 0.5609
Epoch 26/40
288/288 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8631 - loss: 0.3941 - val_accuracy: 0.8285 - val_loss: 0.5337
Epoch 27/40
288/288 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.8679 - loss: 0.3752 - val_accuracy: 0.8416 - val_loss: 0.5361
```

- Evaluation: The trained model is evaluated on the test set using the `evaluate_model` function. It calculates the test loss and accuracy of the model.



```
Test Loss: 0.6217619776725769
Test Accuracy: 80.8495819568634 %
```

- Hyperparameter Experiments: The code performs experiments with different hyperparameters such as learning rate, batch size, epochs, and patience value for early stopping. It trains and evaluates the model with each set of hyperparameters and records the results.

Results:

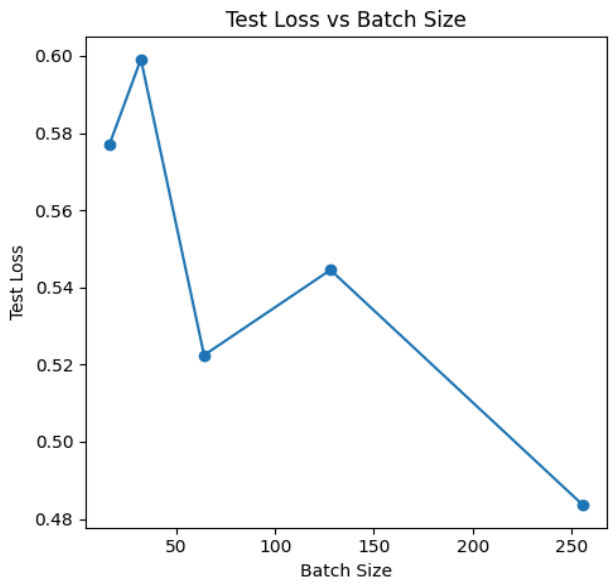
- The DNN model achieved different test accuracy on the mini_speech_commands dataset for different learning rates.
- We observed variations in performance with different hyperparameter settings.

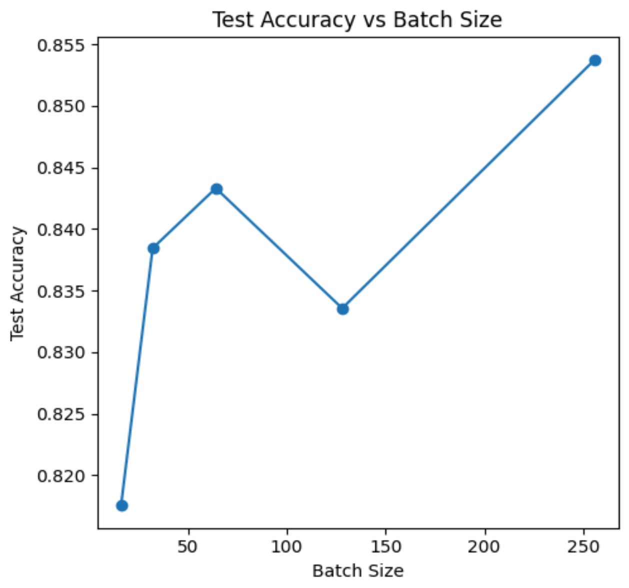
1. Learning rate

```
+-----------------+-------------+-----------------+
|   Learning Rate |   Test Loss |   Test Accuracy |
+=================+=============+=================+
|          1e-05  |    1.70427  |        0.360028 |
+-----------------+-------------+-----------------+
|          0.0001 |    0.761211 |        0.725627 |
+-----------------+-------------+-----------------+
|          0.001  |    0.582424 |        0.83844  |
+-----------------+-------------+-----------------+
|          0.01   |    2.08079  |        0.120474 |
+-----------------+-------------+-----------------+
|          0.1    |    2.08922  |        0.138579 |
+-----------------+-------------+-----------------+
```

2. Batch Size

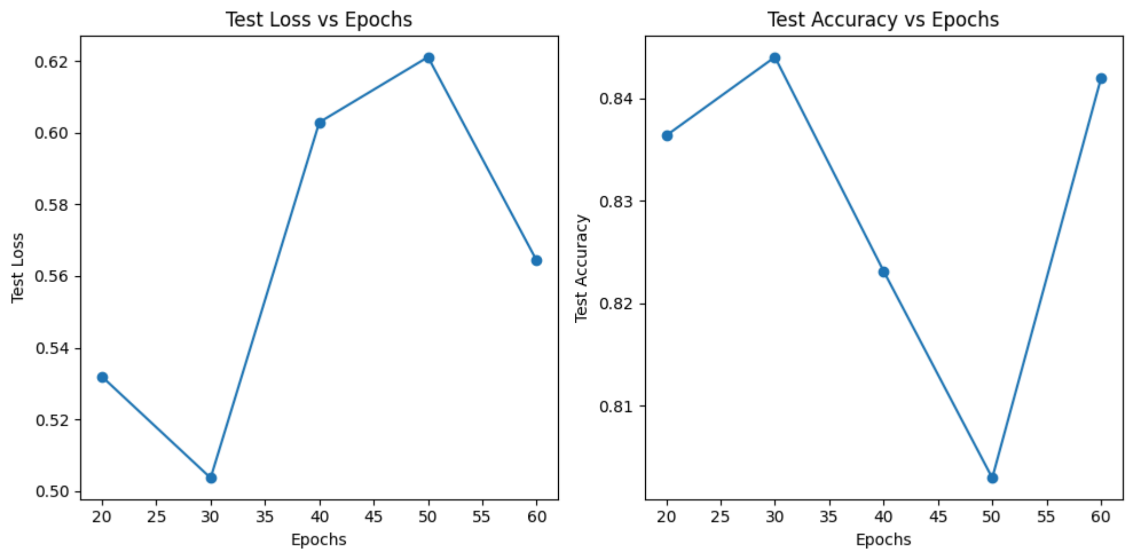
```
+--------------+-------------+-----------------+
|   Batch Size |   Test Loss |   Test Accuracy |
+==============+=============+=================+
|           16 |    0.576966 |        0.817549 |
+--------------+-------------+-----------------+
|           32 |    0.599138 |        0.83844  |
+--------------+-------------+-----------------+
|           64 |    0.522454 |        0.843315 |
+--------------+-------------+-----------------+
|          128 |    0.544541 |        0.833565 |
+--------------+-------------+-----------------+
|          256 |    0.48354  |        0.85376  |
+--------------+-------------+-----------------+
```





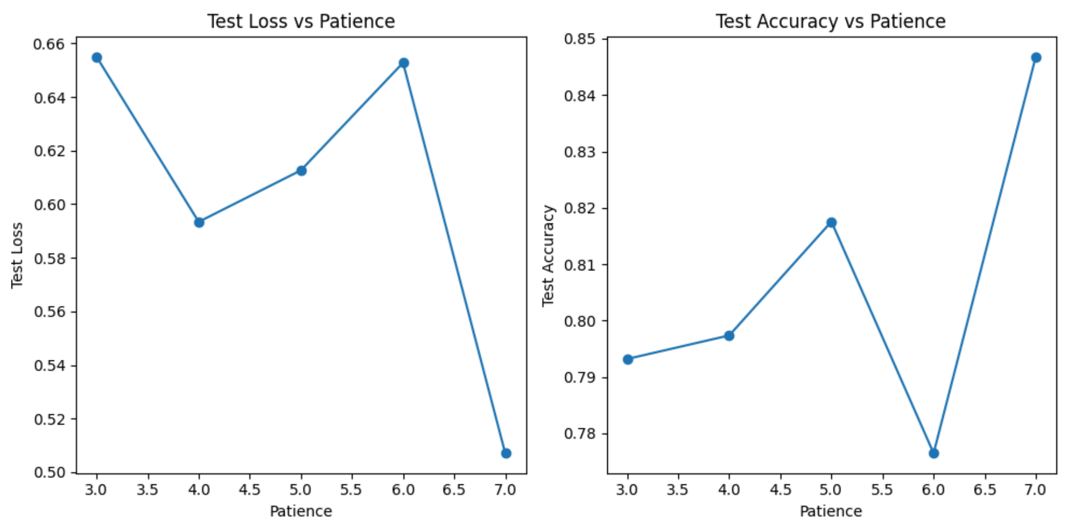
3. Epochs:

```
+---------+------------+-----------------+
|   Epoch |  Test Loss |   Test Accuracy |
+=========+============+=================+
|      20 |   0.531922 |        0.836351 |
+---------+------------+-----------------+
|      30 |   0.503626 |        0.844011 |
+---------+------------+-----------------+
|      40 |   0.602929 |        0.82312  |
+---------+------------+-----------------+
|      50 |   0.621142 |        0.802925 |
+---------+------------+-----------------+
|      60 |   0.56432  |        0.841922 |
+---------+------------+-----------------+
```



4. Patience Value:

```
+------------+------------+-----------------+
|   Patience |  Test Loss |   Test Accuracy |
+============+============+=================+
|          3 |   0.655007 |        0.793175 |
+------------+------------+-----------------+
|          4 |   0.593394 |        0.797354 |
+------------+------------+-----------------+
|          5 |   0.612605 |        0.817549 |
+------------+------------+-----------------+
|          6 |   0.65275  |        0.776462 |
+------------+------------+-----------------+
|          7 |   0.507069 |        0.846797 |
+------------+------------+-----------------+
```



- The hardware software interfacing with the PYNQ-Z2 board facilitated faster execution of the algorithm compared to running it solely on CPU.

Discussion:

- The experimentation with different hyperparameters provided insights into their impact on model performance and training dynamics.
- Leveraging the PYNQ-Z2 board for hardware acceleration demonstrated potential for real-time or low-latency applications.
- Further optimizations and fine-tuning could potentially improve both accuracy and efficiency.

Conclusion:

- The implementation of the deep learning algorithm on the PYNQ-Z2 board showcased the feasibility of using FPGA-based platforms for accelerating machine learning tasks.
- The results obtained indicate promising prospects for deploying such systems in resource-constrained environments or applications requiring low latency.

Future Innovations:

- One can explore additional optimizations for further improving the efficiency and speed of the algorithm on the PYNQ-Z2 board.
- Investigate other speech recognition datasets or applications to assess the generalizability and scalability of the approach.
- Consider integrating additional hardware accelerators or custom IP cores to enhance performance and versatility.

Steps to Reproduce:

1. Set up the PYNQ-Z2 board and install necessary dependencies.
2. Download the mini_speech_commands dataset and preprocess it to extract MFCC features.

3. Implement the DNN model architecture using TensorFlow and Keras.
4. Train the model with different hyperparameters, such as learning rate, batch size, epochs, and patience value.
5. Evaluate the model performance on a test set and analyze the results.
6. Experiment with hardware software interfacing to leverage the PYNQ-Z2 board for acceleration.

References:

[1] Xilinx Pynq-overlay document

[2] Pynq-implementation github repository for reference

[3] Dataset Link

[4] Tensorflow Documentation