Implementation Description

In this section, you will focus on experimenting with various types of recurrent neural networks (RNNs) to implement the speech recognition task. The dataset for this exercise, which contains several audio files along with their corresponding text, is available at this link.

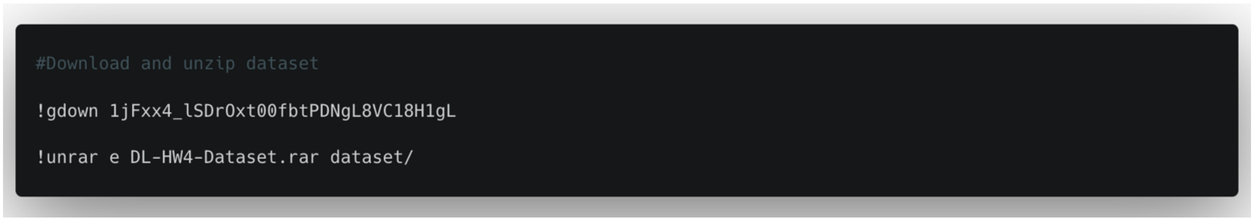
First, perform the necessary preprocessing steps to prepare the dataset as input for the model.

Next, consider three models: LSTM, RNN, and GRU. The architecture and number of layers for each model are arbitrary, but note that the structure of the models should be such that they can be compared with each other. This includes using the same activation function, loss function, and optimizer across all three models.

To evaluate the models, use the metrics of Loss, Accuracy, and F1-score and plot the learning curve of each model based on Accuracy. Finally, after evaluating the models and analyzing the results, announce the best-performing model and report some examples of the top network outputs, which include the predicted text for each audio file in the evaluation dataset.

Implementation Details:

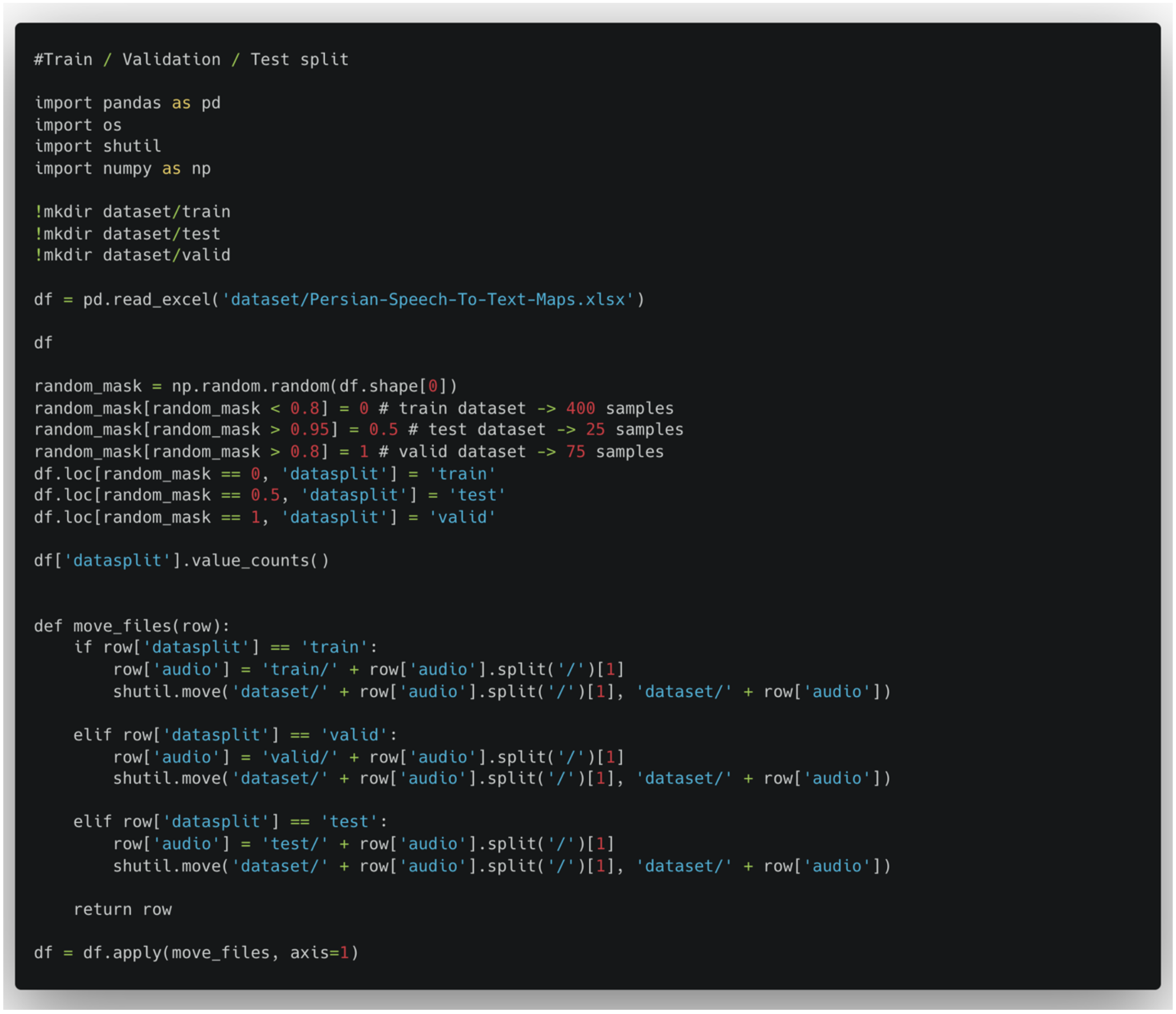
1. Download the dataset:



```
#Download and unzip dataset

!gdown 1jFxx4_lSDrOxt00fbtPDNgL8VC18H1gL

!unrar e DL-HW4-Dataset.rar dataset/
```

Initially, we download the dataset using the 'gdown' command and extract it from the compressed format using the 'unrar' command so that we have access to the audio files and their associated labels in the programming environment.

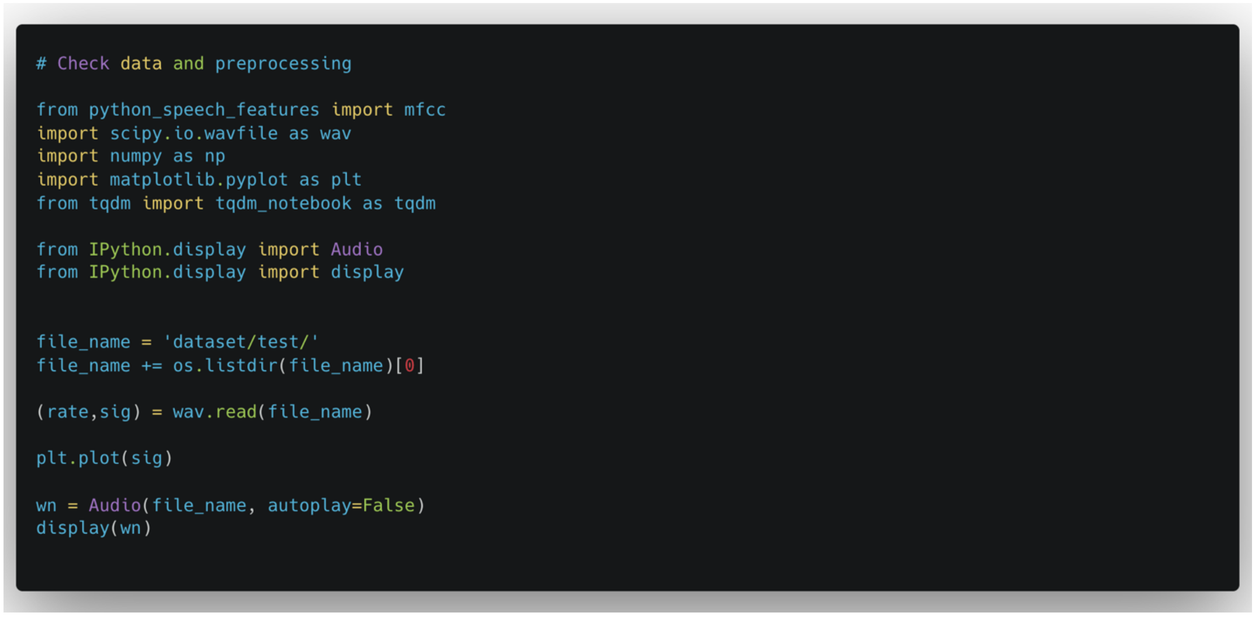
2. Data Splitting into Training, Validation, and Evaluation Sets:



```
#Train / Validation / Test split

import pandas as pd
import os
import shutil
import numpy as np

!mkdir dataset/train
!mkdir dataset/test
!mkdir dataset/valid

df = pd.read_excel('dataset/Persian-Speech-To-Text-Maps.xlsx')

df

random_mask = np.random.random(df.shape[0])
random_mask[random_mask < 0.8] = 0 # train dataset -> 400 samples
random_mask[random_mask > 0.95] = 0.5 # test dataset -> 25 samples
random_mask[random_mask > 0.8] = 1 # valid dataset -> 75 samples
df.loc[random_mask == 0, 'datasplit'] = 'train'
df.loc[random_mask == 0.5, 'datasplit'] = 'test'
df.loc[random_mask == 1, 'datasplit'] = 'valid'

df['datasplit'].value_counts()


def move_files(row):
    if row['datasplit'] == 'train':
        row['audio'] = 'train/' + row['audio'].split('/')[1]
        shutil.move('dataset/' + row['audio'].split('/')[1], 'dataset/' + row['audio'])

    elif row['datasplit'] == 'valid':
        row['audio'] = 'valid/' + row['audio'].split('/')[1]
        shutil.move('dataset/' + row['audio'].split('/')[1], 'dataset/' + row['audio'])

    elif row['datasplit'] == 'test':
        row['audio'] = 'test/' + row['audio'].split('/')[1]
        shutil.move('dataset/' + row['audio'].split('/')[1], 'dataset/' + row['audio'])

    return row

df = df.apply(move_files, axis=1)
```

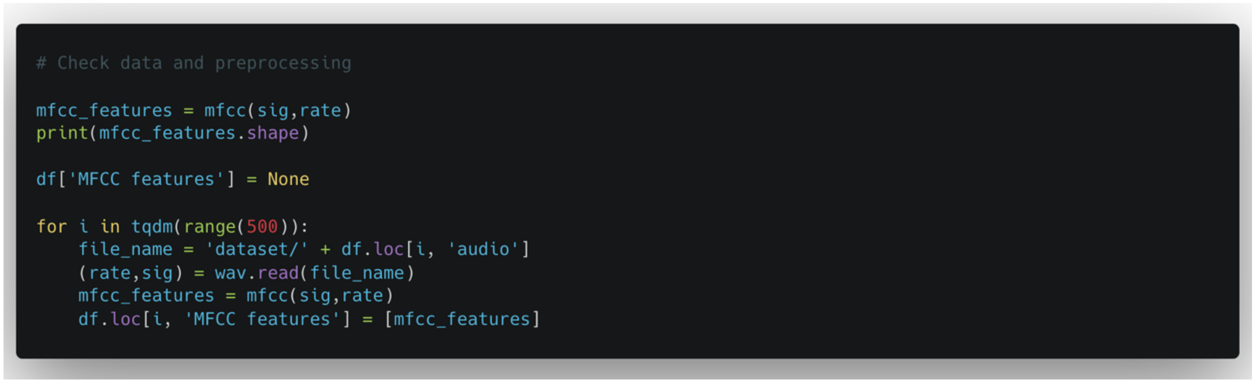
Using this code snippet, we separate the training, validation, and evaluation data for use in the model training and evaluation process. Specifically, we use 400 data points for training, 75 data points for validation, and 25 data points for model evaluation. The data is split and written in their respective rows based on the category they belong to. Finally, three folders named 'training,' 'validation,' and 'evaluation' are created, and the data corresponding to each section is moved to their respective folders.

3- Data Preprocessing



```python
# Check data and preprocessing

from python_speech_features import mfcc
import scipy.io.wavfile as wav
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook as tqdm

from IPython.display import Audio
from IPython.display import display


file_name = 'dataset/test/'
file_name += os.listdir(file_name)[0]

(rate,sig) = wav.read(file_name)

plt.plot(sig)

wn = Audio(file_name, autoplay=False)
display(wn)
```

Initially, we read a random data from the evaluation dataset and plot its signal chart. Then, we add the ability to play the audio file.

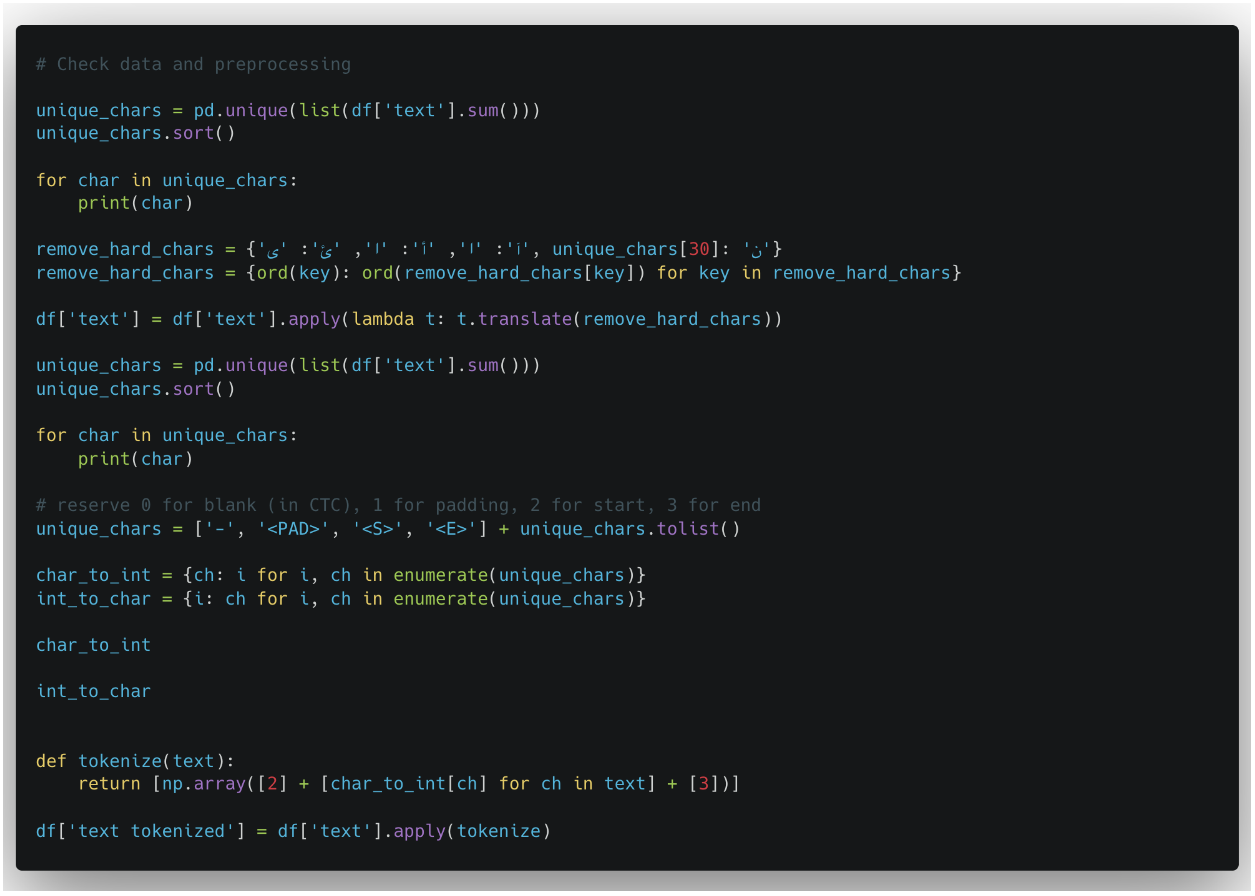


```python
# Check data and preprocessing

mfcc_features = mfcc(sig,rate)
print(mfcc_features.shape)

df['MFCC features'] = None

for i in tqdm(range(500)):
    file_name = 'dataset/' + df.loc[i, 'audio']
    (rate,sig) = wav.read(file_name)
    mfcc_features = mfcc(sig,rate)
    df.loc[i, 'MFCC features'] = [mfcc_features]
```

To extract features from audio data, we use the MFCC feature extraction method. This method is one of the most commonly used methods in the field of speech processing or speech-to-text conversion. Feature extraction from speech signals is an important step in speech processing, which enables us to extract important information from speech signals and use them for speech recognition and differentiation. One of the popular methods for feature extraction from speech signals is the use of Mel Frequency Cepstral Coefficients (MFCC).

In the MFCC method, the speech signal is first divided into short-time frames. Then, using continuous Mel filters, the spectral audio signal is transformed into the Mel-Frequency space. The Mel Frequency space is designed based on the auditory characteristics of the human ear, and the difference between it and the typical frequency space is that higher frequencies are less important in the Mel-Frequency space. Then, using the Cepstral transform, the Cepstral coefficients are calculated for each frame. The Cepstral transform is highly effective due to its ability to model the behavior of the filters inside the human ear and its ability to reduce the dimensions of features. Finally, the Cepstral coefficients are used as signal features. The use of Mel filters and Cepstral transform in the MFCC method makes the extracted features compatible with human hearing and highly effective for speech processing applications such as speaker recognition, speech differentiation, speech-to-text conversion, etc. Therefore, good features for speech processing or speech-to-text conversion are obtained in the frequency range and application of frequency filter banks on them. Using this method, 13 features are extracted every 10 milliseconds.

Finally, in this code section, for each row of the dataset or each audio file, we extract a feature vector and store each in its corresponding dataframe and row.



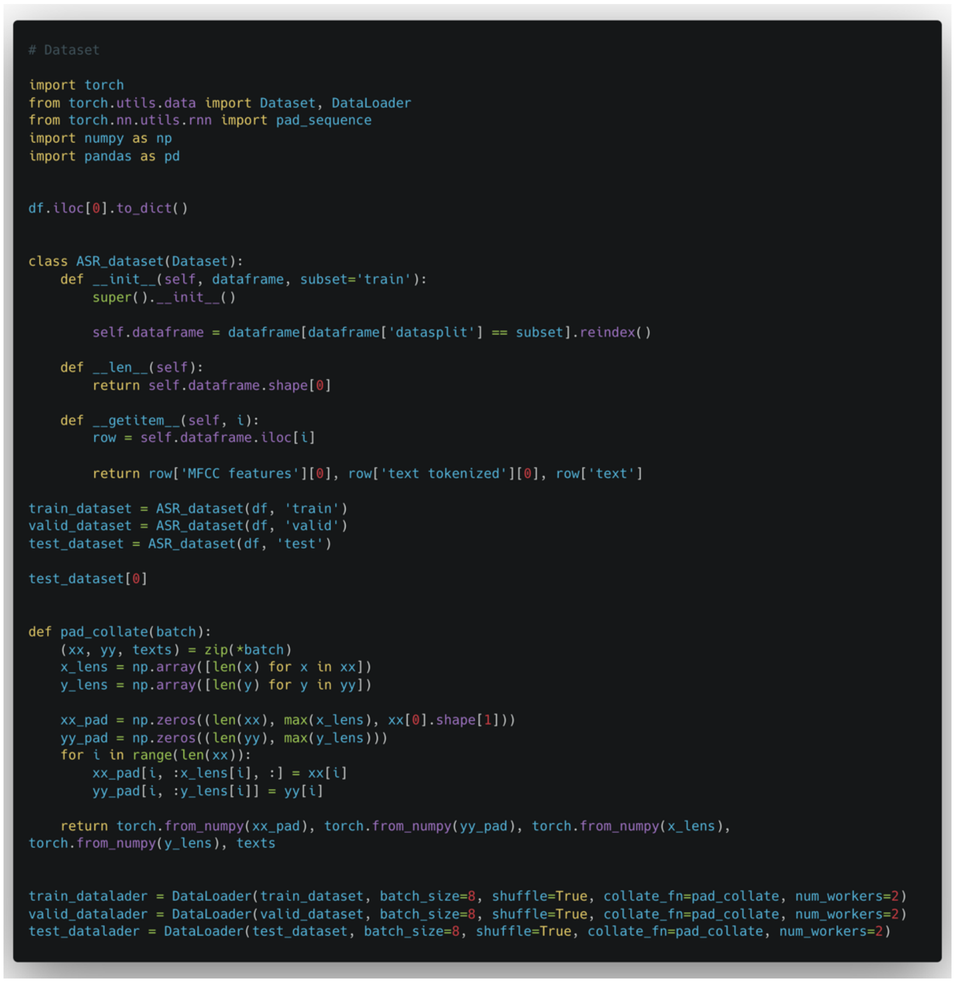
```
# Check data and preprocessing

unique_chars = pd.unique(list(df['text'].sum()))
unique_chars.sort()

for char in unique_chars:
    print(char)

remove_hard_chars = {'ی' :'ئ' ,'ا' :'أ' ,'ا' :'آ', unique_chars[30]: 'ن'}
remove_hard_chars = {ord(key): ord(remove_hard_chars[key]) for key in remove_hard_chars}

df['text'] = df['text'].apply(lambda t: t.translate(remove_hard_chars))

unique_chars = pd.unique(list(df['text'].sum()))
unique_chars.sort()

for char in unique_chars:
    print(char)

# reserve 0 for blank (in CTC), 1 for padding, 2 for start, 3 for end
unique_chars = ['-', '<PAD>', '<S>', '<E>'] + unique_chars.tolist()

char_to_int = {ch: i for i, ch in enumerate(unique_chars)}
int_to_char = {i: ch for i, ch in enumerate(unique_chars)}

char_to_int

int_to_char


def tokenize(text):
    return [np.array([2] + [char_to_int[ch] for ch in text] + [3])]

df['text tokenized'] = df['text'].apply(tokenize)
```

This code section is implemented to examine special characters present in the dataset. Initially, we print the characters present in the dataset. Then, after performing the necessary checks, we replace the more special characters with simpler characters. For example, we replace "آ" with "ا" and "ي" with "ی". This improves the model's accuracy in speech-to-text conversion.

Then we define special characters. The character "-" is defined to determine the unknown output of the model. This character is helpful in the CTC error function for detecting output errors. The "<pad>" character is related to the Padding operation. Additionally, two special characters are used to identify the beginning and end of the sentence and the label.

Finally, using the tokenize function, we tokenize the sentence or label of each audio data. So that we replace the beginning and end of the sentence with special characters and each regular character with its corresponding numerical substitute.

4. Dataset



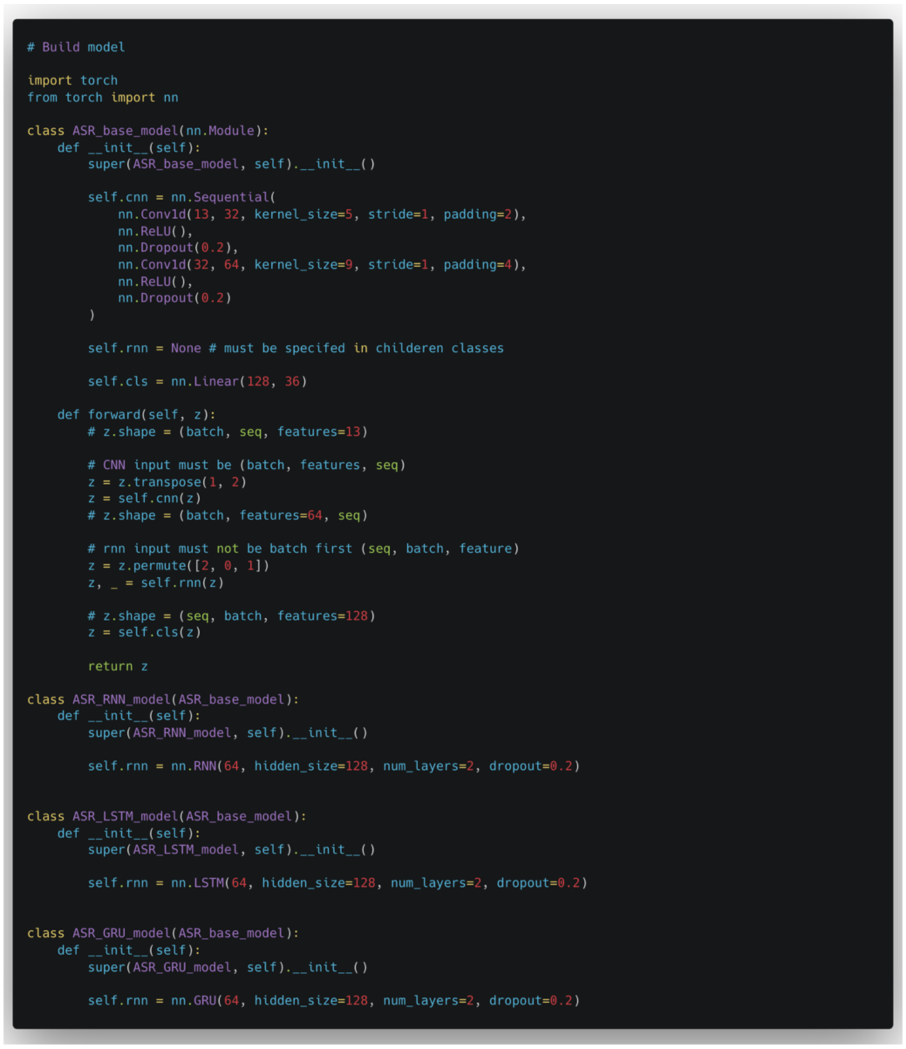
```python
# Dataset

import torch
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
import numpy as np
import pandas as pd


df.iloc[0].to_dict()


class ASR_dataset(Dataset):
    def __init__(self, dataframe, subset='train'):
        super().__init__()

        self.dataframe = dataframe[dataframe['datasplit'] == subset].reindex()

    def __len__(self):
        return self.dataframe.shape[0]

    def __getitem__(self, i):
        row = self.dataframe.iloc[i]

        return row['MFCC features'][0], row['text tokenized'][0], row['text']

train_dataset = ASR_dataset(df, 'train')
valid_dataset = ASR_dataset(df, 'valid')
test_dataset = ASR_dataset(df, 'test')

test_dataset[0]


def pad_collate(batch):
    (xx, yy, texts) = zip(*batch)
    x_lens = np.array([len(x) for x in xx])
    y_lens = np.array([len(y) for y in yy])

    xx_pad = np.zeros((len(xx), max(x_lens), xx[0].shape[1]))
    yy_pad = np.zeros((len(yy), max(y_lens)))
    for i in range(len(xx)):
        xx_pad[i, :x_lens[i], :] = xx[i]
        yy_pad[i, :y_lens[i]] = yy[i]

    return torch.from_numpy(xx_pad), torch.from_numpy(yy_pad), torch.from_numpy(x_lens),
torch.from_numpy(y_lens), texts


train_datalader = DataLoader(train_dataset, batch_size=8, shuffle=True, collate_fn=pad_collate, num_workers=2)
valid_datalader = DataLoader(valid_dataset, batch_size=8, shuffle=True, collate_fn=pad_collate, num_workers=2)
test_datalader = DataLoader(test_dataset, batch_size=8, shuffle=True, collate_fn=pad_collate, num_workers=2)
```

According to the code section above, the ASR_dataset class is implemented to separate the train_dataset, valid_dataset, and test_dataset variables from the DataFrame consisting of the dataset. Inside each of these variables, each row of the dataset is stored as feature vectors extracted as input data and the tokenized sentence as the label. Then, using the Pad_collate() function, the data inside each batch is padded to be of the same length and entered into the model. Inside each batch, the data is stored as a tuple or a 3-tuple variable. These are feature vectors extracted, tokenized labels, and complete labels.

The variable xx contains all the feature vectors related to the input data, and the variable yy contains all their corresponding labels. Moreover, the length of both of these variables is entered as input into the CTC Loss.

Finally, using the DataLoader function, we create batches of 8 and store them in the corresponding variables after padding.

5. Network Architecture



```python
# Build model

import torch
from torch import nn

class ASR_base_model(nn.Module):
    def __init__(self):
        super(ASR_base_model, self).__init__()

        self.cnn = nn.Sequential(
            nn.Conv1d(13, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Conv1d(32, 64, kernel_size=9, stride=1, padding=4),
            nn.ReLU(),
            nn.Dropout(0.2)
        )

        self.rnn = None # must be specifed in childeren classes

        self.cls = nn.Linear(128, 36)

    def forward(self, z):
        # z.shape = (batch, seq, features=13)

        # CNN input must be (batch, features, seq)
        z = z.transpose(1, 2)
        z = self.cnn(z)
        # z.shape = (batch, features=64, seq)

        # rnn input must not be batch first (seq, batch, feature)
        z = z.permute([2, 0, 1])
        z, _ = self.rnn(z)

        # z.shape = (seq, batch, features=128)
        z = self.cls(z)

        return z

class ASR_RNN_model(ASR_base_model):
    def __init__(self):
        super(ASR_RNN_model, self).__init__()

        self.rnn = nn.RNN(64, hidden_size=128, num_layers=2, dropout=0.2)


class ASR_LSTM_model(ASR_base_model):
    def __init__(self):
        super(ASR_LSTM_model, self).__init__()

        self.rnn = nn.LSTM(64, hidden_size=128, num_layers=2, dropout=0.2)


class ASR_GRU_model(ASR_base_model):
    def __init__(self):
        super(ASR_GRU_model, self).__init__()

        self.rnn = nn.GRU(64, hidden_size=128, num_layers=2, dropout=0.2)
```

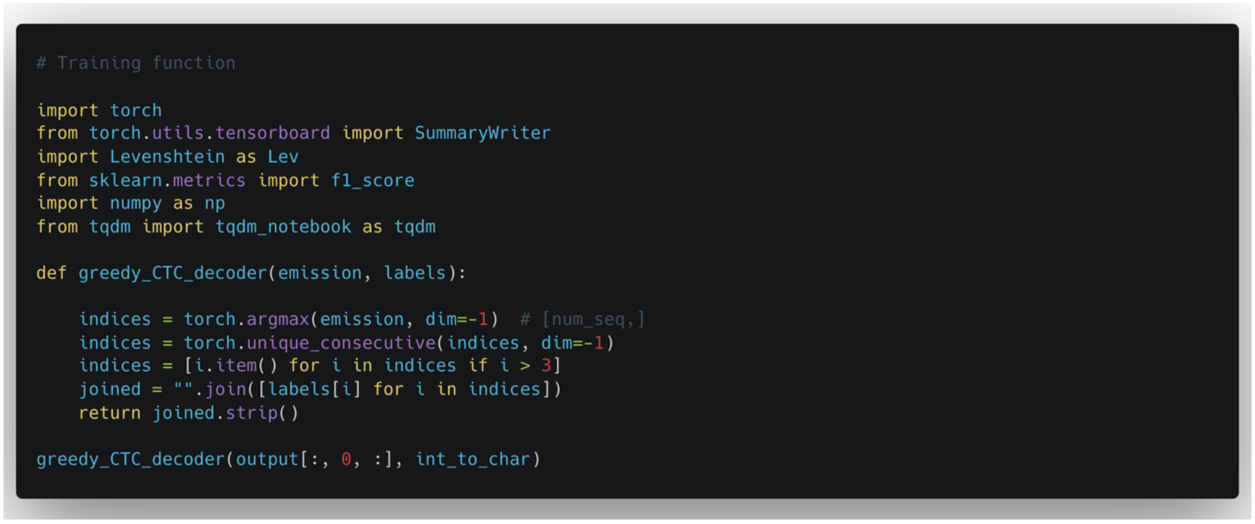
First, we implement the ASR_base_model class. This class initially consists of two one-dimensional convolutional layers. In this network, a convolutional layer with 13 channels, a kernel size of 5, and padding of 2 are defined. Then, a ReLU activation function is used as a non-linear function. After that, a dropout layer with a value of 0.2 is defined to prevent overfitting. Then, another convolutional layer with 32 channels, a kernel size of 9, and padding of 4 is defined. Then, a ReLU activation function and another dropout layer with a value of 0.2 are added in sequence. Then, in this class, we consider the recurrent layers as inactive so that the classes related to each of the recurrent models, after inheriting from their ASR_base_model class, consider the recurrent layers exclusively. In the forward section, the dimensions of the output of the convolutional layers and the input of the recurrent layers are adjusted.

Then, in the end, three classes related to the RNN, LSTM, and GRU recurrent models are created by inheriting from the base class. The difference is that in each of them, the recurrent layers are considered in a specific way.

The input of the recurrent layers is the feature vector extracted by the convolutional network. The number of neurons in the hidden layer is 128, the number of recurrent layers is two, and finally, the dropout value is also 0.2.

Note that the feature and architecture of these three classes of recurrent models are exactly the same, and this feature makes it easier to compare the results of these three models.

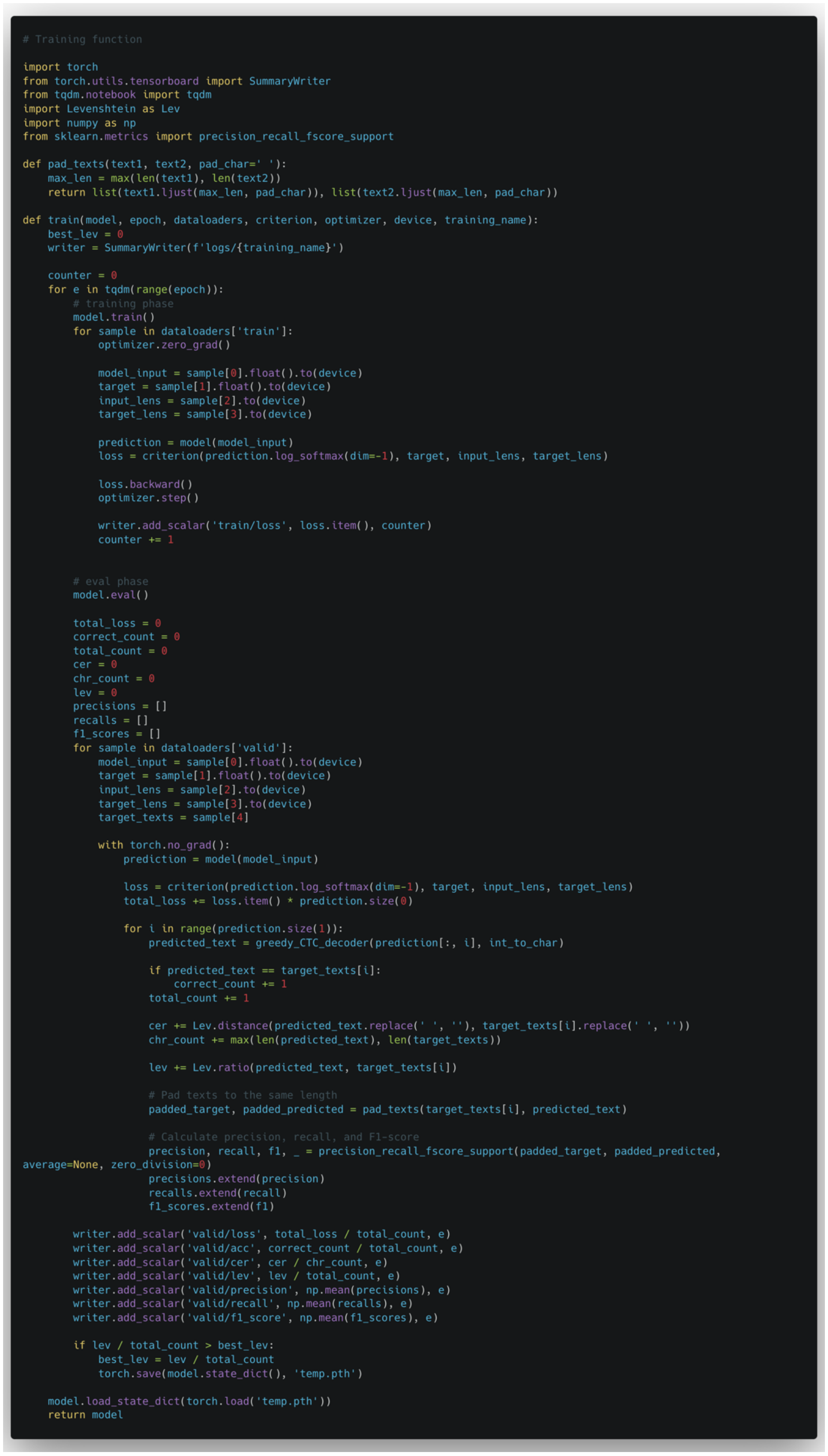
6 - Training Function



```
# Training function

import torch
from torch.utils.tensorboard import SummaryWriter
import Levenshtein as Lev
from sklearn.metrics import f1_score
import numpy as np
from tqdm import tqdm_notebook as tqdm

def greedy_CTC_decoder(emission, labels):

    indices = torch.argmax(emission, dim=-1)  # [num_seq,]
    indices = torch.unique_consecutive(indices, dim=-1)
    indices = [i.item() for i in indices if i > 3]
    joined = "".join([labels[i] for i in indices])
    return joined.strip()

greedy_CTC_decoder(output[:, 0, :], int_to_char)
```

This function has been implemented to arrange the output of the implemented model. Since the model output may include special characters such as - ,<Pad>, <Start>, <End> or several consecutive normal characters, this function has been designed to arrange the model output. At first, this function calculates the highest score and repetition of each character at each time, and then removes consecutive repeating characters.

Then, it removes special characters and converts the characters from numerical to real characters to form the output text.



```python
# Training function

import torch
from torch.utils.tensorboard import SummaryWriter
from tqdm.notebook import tqdm
import Levenshtein as Lev
import numpy as np
from sklearn.metrics import precision_recall_fscore_support

def pad_texts(text1, text2, pad_char=' '):
    max_len = max(len(text1), len(text2))
    return list(text1.ljust(max_len, pad_char)), list(text2.ljust(max_len, pad_char))

def train(model, epoch, dataloaders, criterion, optimizer, device, training_name):
    best_lev = 0
    writer = SummaryWriter(f'logs/{training_name}')

    counter = 0
    for e in tqdm(range(epoch)):
        # training phase
        model.train()
        for sample in dataloaders['train']:
            optimizer.zero_grad()

            model_input = sample[0].float().to(device)
            target = sample[1].float().to(device)
            input_lens = sample[2].to(device)
            target_lens = sample[3].to(device)

            prediction = model(model_input)
            loss = criterion(prediction.log_softmax(dim=-1), target, input_lens, target_lens)

            loss.backward()
            optimizer.step()

            writer.add_scalar('train/loss', loss.item(), counter)
            counter += 1


        # eval phase
        model.eval()

        total_loss = 0
        correct_count = 0
        total_count = 0
        cer = 0
        chr_count = 0
        lev = 0
        precisions = []
        recalls = []
        f1_scores = []
        for sample in dataloaders['valid']:
            model_input = sample[0].float().to(device)
            target = sample[1].float().to(device)
            input_lens = sample[2].to(device)
            target_lens = sample[3].to(device)
            target_texts = sample[4]

            with torch.no_grad():
                prediction = model(model_input)

                loss = criterion(prediction.log_softmax(dim=-1), target, input_lens, target_lens)
                total_loss += loss.item() * prediction.size(0)

                for i in range(prediction.size(1)):
                    predicted_text = greedy_CTC_decoder(prediction[:, i], int_to_char)

                    if predicted_text == target_texts[i]:
                        correct_count += 1
                    total_count += 1

                    cer += Lev.distance(predicted_text.replace(' ', ''), target_texts[i].replace(' ', ''))
                    chr_count += max(len(predicted_text), len(target_texts))

                    lev += Lev.ratio(predicted_text, target_texts[i])

                    # Pad texts to the same length
                    padded_target, padded_predicted = pad_texts(target_texts[i], predicted_text)

                    # Calculate precision, recall, and F1-score
                    precision, recall, f1, _ = precision_recall_fscore_support(padded_target, padded_predicted,
average=None, zero_division=0)
                    precisions.extend(precision)
                    recalls.extend(recall)
                    f1_scores.extend(f1)

        writer.add_scalar('valid/loss', total_loss / total_count, e)
        writer.add_scalar('valid/acc', correct_count / total_count, e)
        writer.add_scalar('valid/cer', cer / chr_count, e)
        writer.add_scalar('valid/lev', lev / total_count, e)
        writer.add_scalar('valid/precision', np.mean(precisions), e)
        writer.add_scalar('valid/recall', np.mean(recalls), e)
        writer.add_scalar('valid/f1_score', np.mean(f1_scores), e)

        if lev / total_count > best_lev:
            best_lev = lev / total_count
            torch.save(model.state_dict(), 'temp.pth')

    model.load_state_dict(torch.load('temp.pth'))
    return model
```

The function pad_texts is used to pad two strings with characters added at the end to make both strings of equal length.

The Train function is responsible for training the model. It divides the training data into two parts train and valid. During each training epoch, the model is first put in train mode and for each training data, it is trained once on that data. The process of performing the training has no special notes and is done as usual. But it is noteworthy that the inputs of the CTC loss function are: model output, data label, model output length and actual length of label to consider Padding operations.

Later, the model is placed in evaluation mode and for each evaluation data, the model output is calculated and converted into text using the greedy_CTC_decoder algorithm. Then, using various criteria including error function, accuracy, CER, and F1-score, the model performance is evaluated and graphs related to evaluation criteria such as accuracy are plotted on training and evaluation data. Dropout has also been used in the model to prevent overfitting. Also, Tensorboard library has been used to monitor the performance of the model over time.

Due to the inefficiency of the accuracy evaluation criterion relative to the volume of training data, two other evaluation criteria named Levenshtein Distance and CER (Character Error Rate) have been used .

The Levenshtein Distance and CER evaluation criteria are very important in speech to text problems and are used as evaluation criteria for the performance of speech to text models.

Levenshtein distance or Edit Distance is a criterion used to measure the distance between two text strings. This criterion calculates the minimum number of operations (deletion, substitution and adding character) required to transform one string into another. In this criterion, each of the characters of the two strings are compared and if they are different, one of the three operations of deletion, substitution or addition is performed. The Levenshtein distance is equal to the total number of operations performed to transform one string into another.

CER or Character Error Rate measures the extent of error in converting a generated text string by the model into an actual text string. In this criterion, the number of letters that differ in the string produced by the model from the actual string is calculated and then divided by the total number of letters in the actual string. As a result, this criterion indicates what percentage of the letters of the string produced by the model differ from the actual letters.

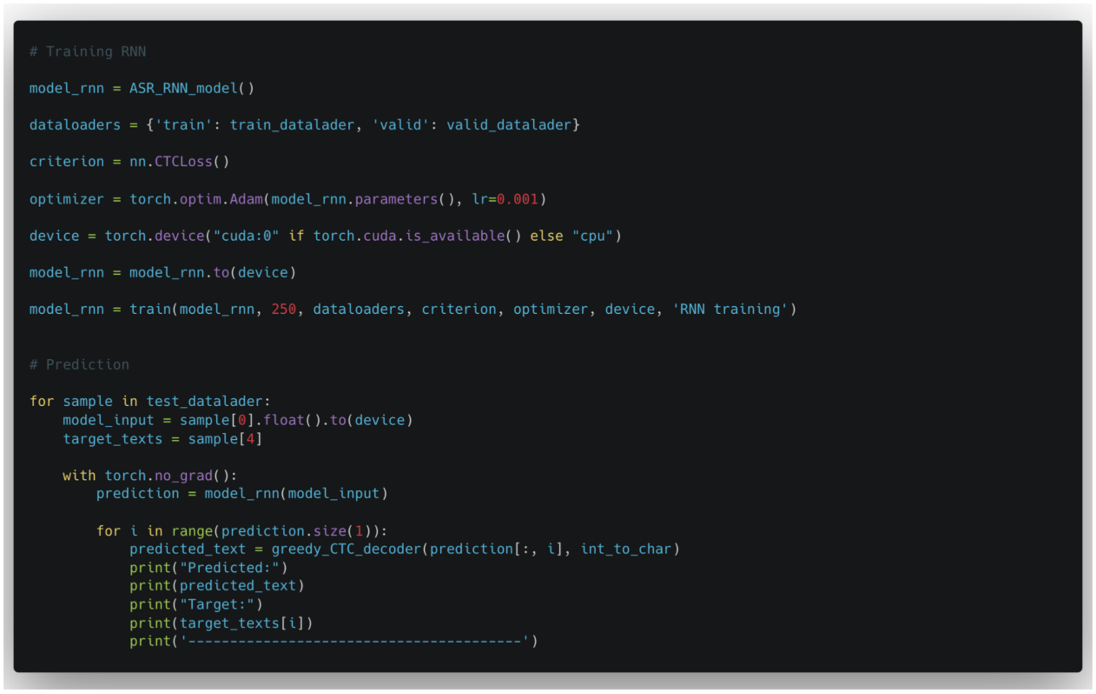
Both these criteria are used in evaluating the performance of speech to text models. In fact, Levenshtein distance is used to calculate word errors and CER is used to calculate character errors. Using these two criteria, the performance of speech to text models can be accurately determined with high precision.

In the review of the results section, we will interpret the effectiveness or ineffectiveness of each of the evaluation criteria.

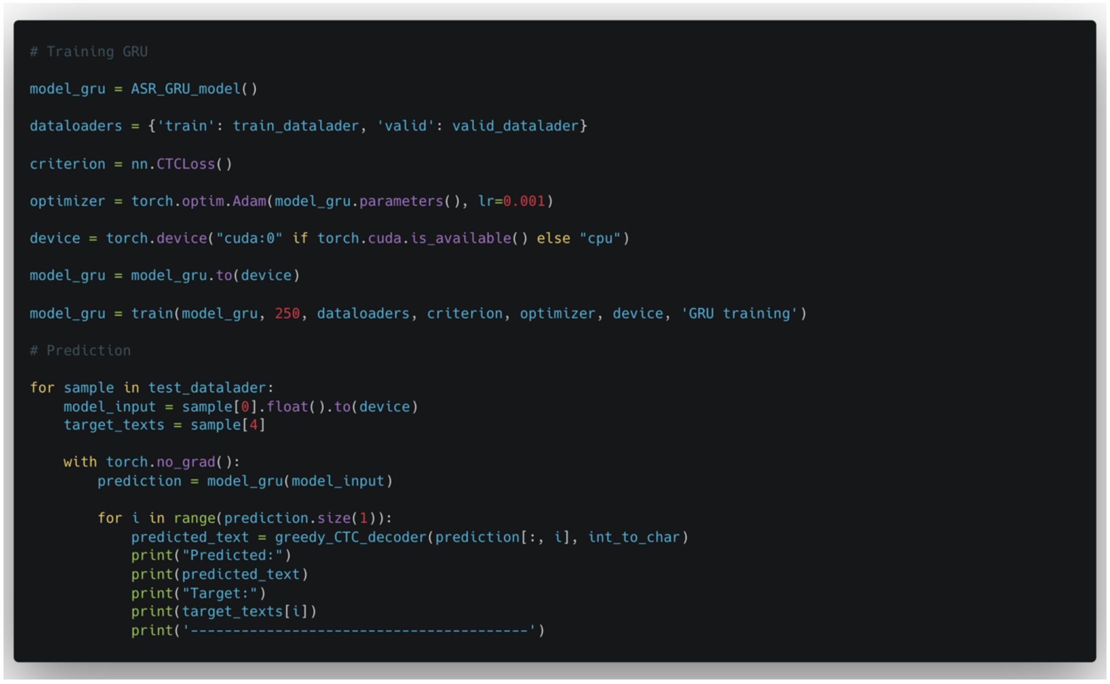
7 - Model Training

Using the following code snippets, we train the RNN, GRU and LSTM models and then display the output of each of them along with their respective labels to the user. In each of these code snippets, the training and validation data are specified. The error function, the CTC error function has been considered. Adam has been used as the optimizer and the training process takes place using the graphics processor.

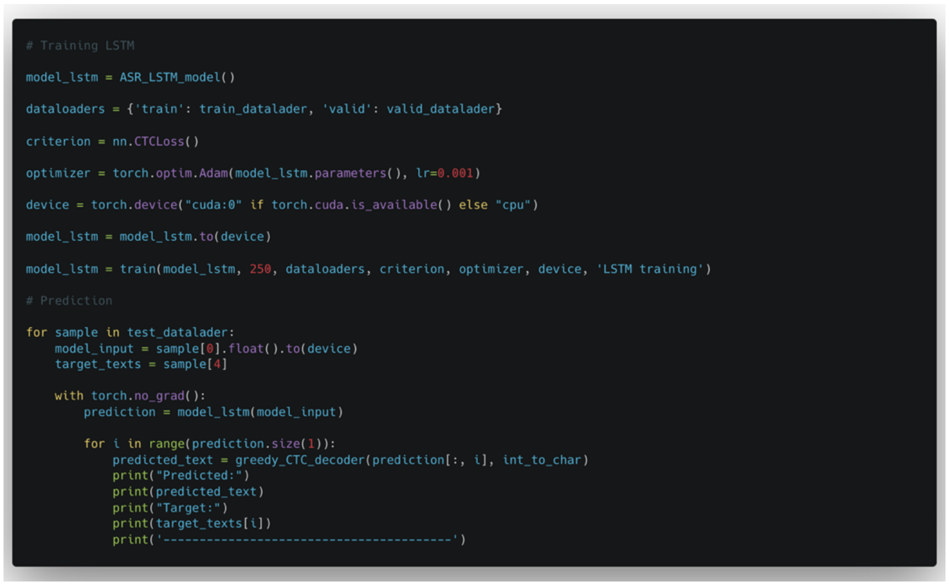
The CTCloss (Connectionist Temporal Classification Loss) function is an error function used to train speech processing models. This error function has been proposed for models that have recurrent neural networks (RNN) and helps the neural network to solve the problem of temporal labels in speech data. In speech recognition, each speech may contain more than one word and there are gaps between words. Therefore, the model is expected to be able to identify the temporal labels corresponding to each word. Using the CTCloss error function, these temporal labels can be extracted automatically without the need for manual labeling. The CTCloss function, by receiving the neural network output, the actual labels of the training data and the length of each data, calculates the error related to the network's prediction. For this purpose, all possible strings that are consistent with the actually labeled string are considered as a word. Then the probability of each of these strings is calculated using the forward-backward algorithm and the Baum-Welch (Baum-Welch) equation. Finally, the error related to the network's prediction is calculated using these probabilities. Using the CTCloss error function has many advantages. Among them is that there is no need for manual labeling of training data and the model can be trained with speech data without any kind of labeled information. It also helps the model to solve the problem of temporal labeling and be able to extract temporal labels corresponding to words automatically.



```
# Training RNN

model_rnn = ASR_RNN_model()

dataloaders = {'train': train_datalader, 'valid': valid_datalader}

criterion = nn.CTCLoss()

optimizer = torch.optim.Adam(model_rnn.parameters(), lr=0.001)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model_rnn = model_rnn.to(device)

model_rnn = train(model_rnn, 250, dataloaders, criterion, optimizer, device, 'RNN training')


# Prediction

for sample in test_datalader:
    model_input = sample[0].float().to(device)
    target_texts = sample[4]

    with torch.no_grad():
        prediction = model_rnn(model_input)

        for i in range(prediction.size(1)):
            predicted_text = greedy_CTC_decoder(prediction[:, i], int_to_char)
            print("Predicted:")
            print(predicted_text)
            print("Target:")
            print(target_texts[i])
            print('----------------------------------------')
```

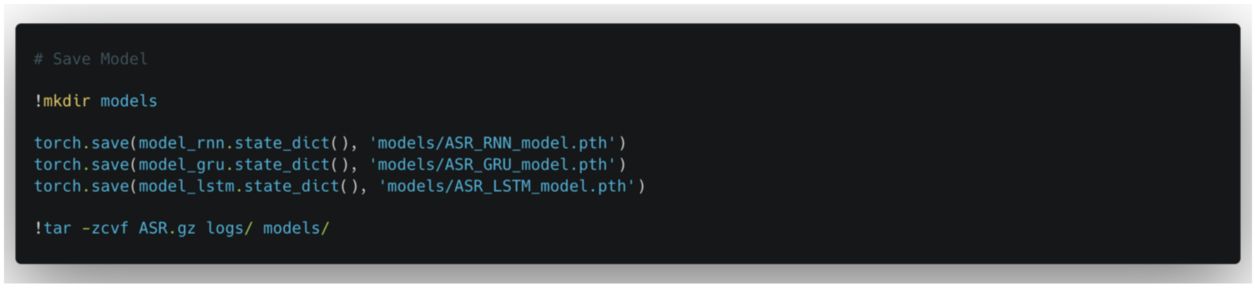


```
# Training GRU

model_gru = ASR_GRU_model()

dataloaders = {'train': train_datalader, 'valid': valid_datalader}

criterion = nn.CTCLoss()

optimizer = torch.optim.Adam(model_gru.parameters(), lr=0.001)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model_gru = model_gru.to(device)

model_gru = train(model_gru, 250, dataloaders, criterion, optimizer, device, 'GRU training')

# Prediction

for sample in test_datalader:
    model_input = sample[0].float().to(device)
    target_texts = sample[4]

    with torch.no_grad():
        prediction = model_gru(model_input)

        for i in range(prediction.size(1)):
            predicted_text = greedy_CTC_decoder(prediction[:, i], int_to_char)
            print("Predicted:")
            print(predicted_text)
            print("Target:")
            print(target_texts[i])
            print('----------------------------------------')
```



```
# Training LSTM

model_lstm = ASR_LSTM_model()

dataloaders = {'train': train_datalader, 'valid': valid_datalader}

criterion = nn.CTCLoss()

optimizer = torch.optim.Adam(model_lstm.parameters(), lr=0.001)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model_lstm = model_lstm.to(device)

model_lstm = train(model_lstm, 250, dataloaders, criterion, optimizer, device, 'LSTM training')

# Prediction

for sample in test_datalader:
    model_input = sample[0].float().to(device)
    target_texts = sample[4]

    with torch.no_grad():
        prediction = model_lstm(model_input)

        for i in range(prediction.size(1)):
            predicted_text = greedy_CTC_decoder(prediction[:, i], int_to_char)
            print("Predicted:")
            print(predicted_text)
            print("Target:")
            print(target_texts[i])
            print('----------------------------------------')
```

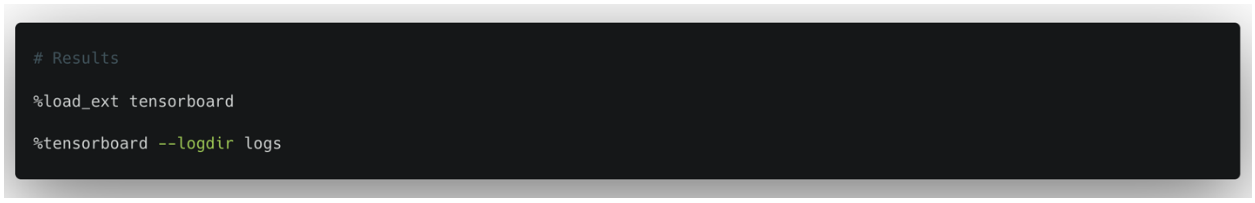
8 - Saving Models



```
# Save Model

!mkdir models

torch.save(model_rnn.state_dict(), 'models/ASR_RNN_model.pth')
torch.save(model_gru.state_dict(), 'models/ASR_GRU_model.pth')
torch.save(model_lstm.state_dict(), 'models/ASR_LSTM_model.pth')

!tar -zcvf ASR.gz logs/ models/
```

Using this code, we save the trained models so that we later have access to the final weights of each model.

9 - Displaying Results



```
# Results

%load_ext tensorboard

%tensorboard --logdir logs
```

Using this code, we run Tensorboard to display accuracy and error charts on training and validation data.

The results obtained from this model can be seen in section 3 or in the review of results.

Conclusion and Analysis of Results

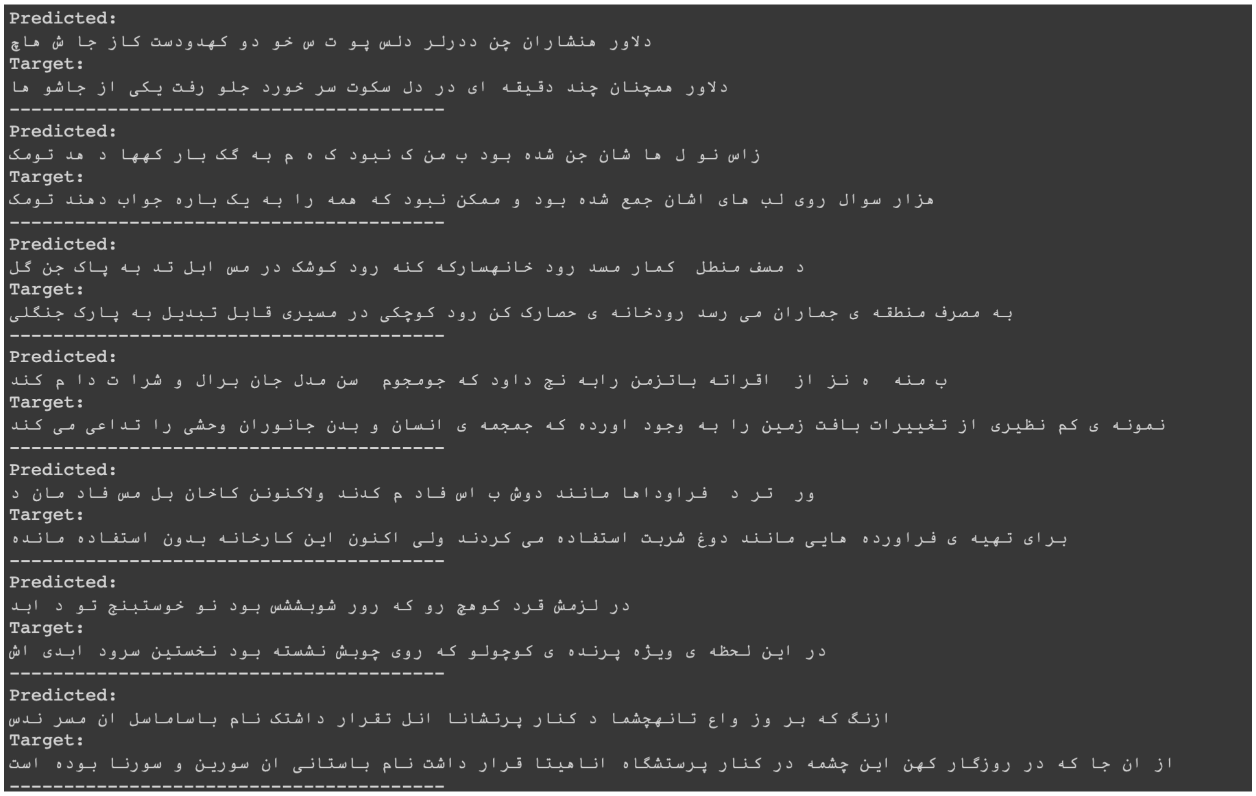
In this report, we initially reviewed the concepts of recurrent neural networks and then explained the implemented functions. Now we come to the conclusion and analysis of results section, where we present the results of the implemented models and analyze them. In this section, we reiterate the issues asked and will review the results of implementing each part.

In order to evaluate the models, we used the Loss, Accuracy and F1-score evaluation metrics and plotted the learning curve of each model based on Accuracy. In the final step, after evaluating the models with analysis of results and stating the reason, declare the superior model and report a few samples of the network's outputs that include predicted text for each audio file in evaluation dataset.

Initially, let us have a look at a few outputs from the three models related to the RNN, GRU and LSTM recurrent neural networks:

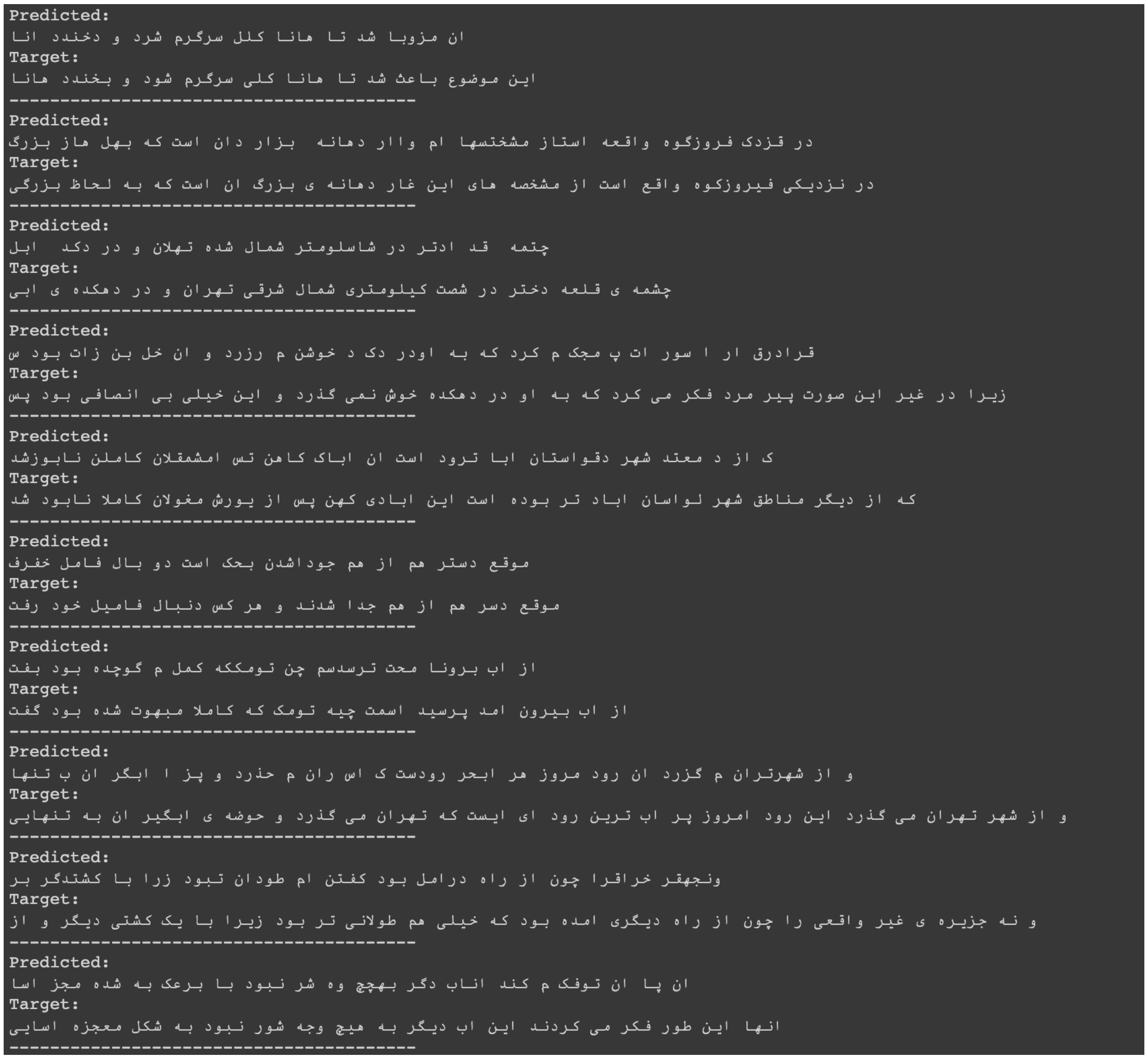
It should be noted that due to the very little data in the dataset, the accuracy recorded by these three models is far from desirable.

A few outputs from the RNN network model:



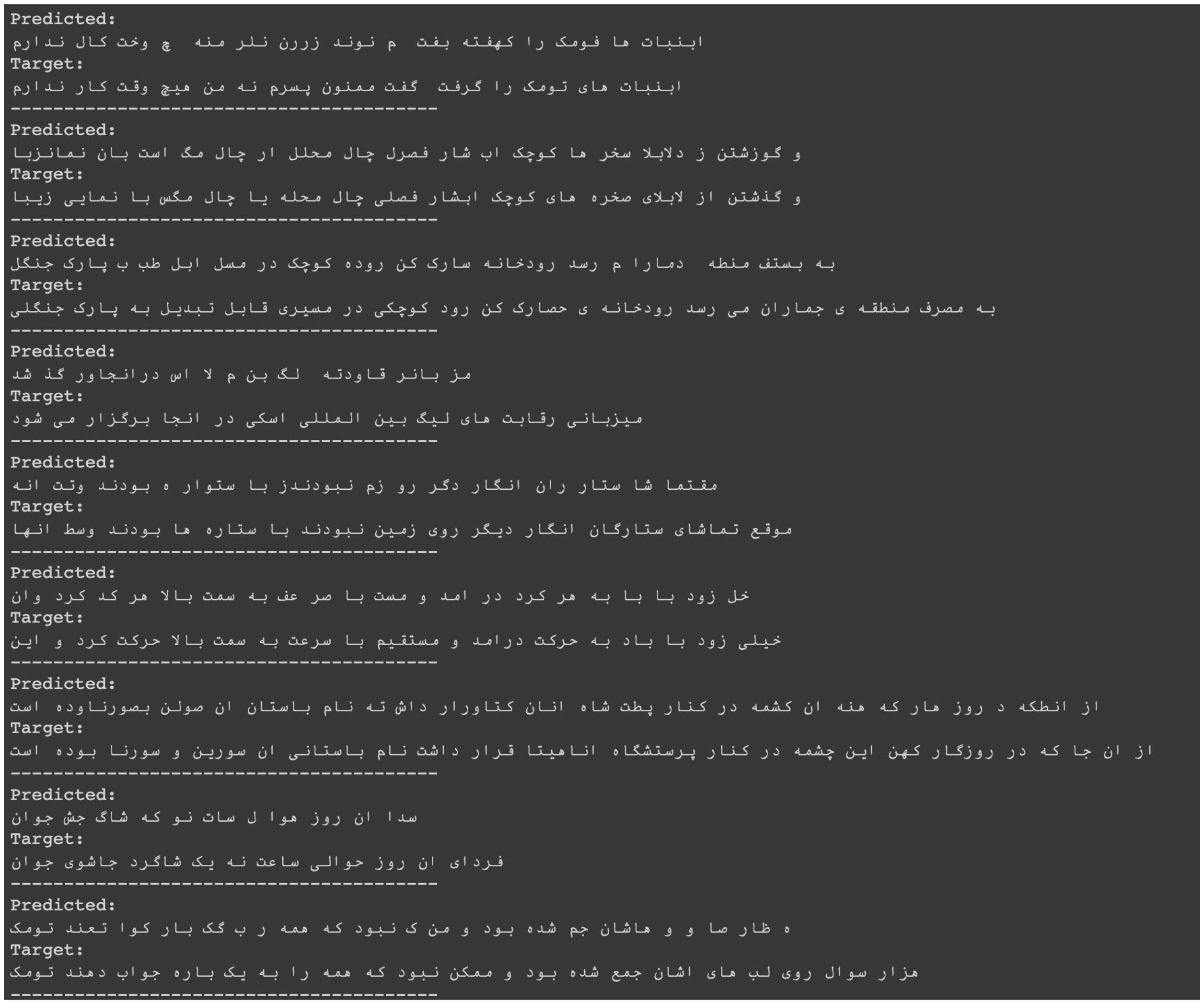
```
Predicted:
دلاور هنشاران چن ددرلر دلس پو ت س خو دو کهدودست کاز جا ش هاچ
Target:
دلاور همچنان چند دقیقه ای در دل سکوت سر خورد جلو رفت یکی از جاشو ها
----------------------------------------
Predicted:
زاس نو ل ها شان جن شده بود ب من ک نبود ک ه م به گک بار کهها د هد تومک
Target:
هزار سوال روی لب های اشان جمع شده بود و ممکن نبود که همه را به یک باره جواب دهند تومک
----------------------------------------
Predicted:
د مسف منطل  کمار مسد رود خانهسارکه کنه رود کوشک در مس ابل تد به پاک جن گل
Target:
به مصرف منطقه ی جماران می رسد رودخانه ی حصارک کن رود کوچکی در مسیری قابل تبدیل به پارک جنگلی
----------------------------------------
Predicted:
ب منه  ه نز از  اقراته باتزمن رابه نج داود که جومجوم  سن مدل جان برال و شرا ت دا م کند
Target:
نمونه ی کم نظیری از تغییرات بافت زمین را به وجود اورده که جمجمه ی انسان و بدن جانوران وحشی را تداعی می کند
----------------------------------------
Predicted:
ور  تر د  فراوداها مانند دوش ب اس فاد م کدند ولاکنونن کاخان بل مس فاد مان د
Target:
برای تهیه ی فراورده هایی مانند دوغ شربت استفاده می کردند ولی اکنون این کارخانه بدون استفاده مانده
----------------------------------------
Predicted:
در لزمش قرد کوهچ رو که رور شوبششس بود نو خوستبنج تو د ابد
Target:
در این لحظه ی ویژه پرنده ی کوچولو که روی چوبش نشسته بود نخستین سرود ابدی اش
----------------------------------------
Predicted:
ازنگ که بر وز واع تانهچشما د کنار پرتشانا  انل تقرار داشتک نام باساماسل ان مسر ندس
Target:
از ان جا که در روزگار کهن این چشمه در کنار پرستشگاه اناهیتا قرار داشت نام باستانی ان سورین و سورنا بوده است
----------------------------------------
```

A few outputs from the GRU network model:



```
Predicted:
ان مزوبا شد تا هانا کلل سرگرم شرد و دخندد انا
Target:
این موضوع باعث شد تا هانا کلی سرگرم شود و بخندد هانا
----------------------------------------
Predicted:
در قزدک فروزگوه واقعه استاز مشختسها ام واار دهانه  بزار دان است که بهل هاز بزرگ
Target:
در نزدیکی فیروزکوه واقع است از مشخصه های این غار دهانه ی بزرگ ان است که به لحاظ بزرگی
----------------------------------------
Predicted:
چتمه  قد ادتر در شاسلومتر شمال شده تهلان و در دکد  ابل
Target:
چشمه ی قلعه دختر در شصت کیلومتری شمال شرقی تهران و در دهکده ی ابی
----------------------------------------
Predicted:
قرادرق ار ا سور ات پ مجک م کرد که به اودر دک د خوشن م رزرد و ان خل بن زات بود س
Target:
زیرا در غیر این صورت پیر مرد فکر می کرد که به او در دهکده خوش نمی گذرد و این خیلی بی انصافی بود پس
----------------------------------------
Predicted:
ک از د معتد شهر دقواستان ابا ترود است ان اباک کاهن تس امشمقلان کاملن نابوزشد
Target:
که از دیگر مناطق شهر لواسان اباد تر بوده است این ابادی کهن پس از یورش مغولان کاملا نابود شد
----------------------------------------
Predicted:
موقع دستر هم از هم جوداشدن بحک است دو بال فامل خفرف
Target:
موقع دسر هم از هم جدا شدند و هر کس دنبال فامیل خود رفت
----------------------------------------
Predicted:
از اب برونا محت ترسدسم چن تومککه کمل م گوچده بود بفت
Target:
از اب بیرون امد پرسید اسمت چیه تومک که کاملا مبهوت شده بود گفت
----------------------------------------
Predicted:
و از شهرتران م گزرد ان رود مروز هر ابحر رودست ک اس ران م حذرد و پز ا ابگر ان ب تنها
Target:
و از شهر تهران می گذرد این رود امروز پر اب ترین رود ای ایست که تهران می گذرد و حوضه ی ابگیر ان به تنهایی
----------------------------------------
Predicted:
ونجهقر خراقرا چون از راه درامل بود کفتن ام طودان تبود زرا با کشتدگر بر
Target:
و نه جزیره ی غیر واقعی را چون از راه دیگری امده بود که خیلی هم طولانی تر بود زیرا با یک کشتی دیگر و از
----------------------------------------
Predicted:
ان پا ان توفک م کند اناب دگر بهچچ وه شر نبود با برعک به شده مجز اسا
Target:
انها این طور فکر می کردند این اب دیگر به هیچ وجه شور نبود به شکل معجزه اسایی
----------------------------------------
```

A few outputs from the LSTM network model:



```
Predicted:
ابنبات ها فومک را کهفته بفت  م نوند زررن نلر منه  چ وخت کال ندارم
Target:
ابنبات های تومک را گرفت  گفت ممنون پسرم نه من هیچ وقت کار ندارم
----------------------------------------
Predicted:
و گوزشتن ز دلابلا سخر ها کوچک اب شار فصرل چال محلل ار چال مگ است بان نمانزبا
Target:
و گذشتن از لابلای صخره های کوچک ابشار فصلی چال محله یا چال مگس با نمایی زیبا
----------------------------------------
Predicted:
به بستف منطه  دمارا م رسد رودخانه سارک کن روده کوچک در مسل ابل طب ب پارک جنگل
Target:
به مصرف منطقه ی جماران می رسد رودخانه ی حصارک کن رود کوچکی در مسیری قابل تبدیل به پارک جنگلی
----------------------------------------
Predicted:
مز بانر قاودته  لگ بن م لا اس درانجاور گذ شد
Target:
میزبانی رقابت های لیگ بین المللی اسکی در انجا برگزار می شود
----------------------------------------
Predicted:
مقتما شا ستار ران انگار دگر رو زم نبودندز با ستوار ه بودند وتت انه
Target:
موقع تماشای ستارگان انگار دیگر روی زمین نبودند با ستاره ها بودند وسط انها
----------------------------------------
Predicted:
خل زود با با به هر کرد در امد و مست با صر عف به سمت بالا هر کد کرد وان
Target:
خیلی زود با باد به حرکت درامد و مستقیم با سرعت به سمت بالا حرکت کرد و این
----------------------------------------
Predicted:
از انطکه د روز هار که هنه ان کشمه در کنار پطت شاه انان کتاورار داش ته نام باستان ان صولن بصورناوده است
Target:
از ان جا که در روزگار کهن این چشمه در کنار پرستشگاه اناهیتا قرار داشت نام باستانی ان سورین و سورنا بوده است
----------------------------------------
Predicted:
سدا ان روز هوا ل سات نو که شاگ جش جوان
Target:
فردای ان روز حوالی ساعت نه یک شاگرد جاشوی جوان
----------------------------------------
Predicted:
ه ظار صا و و هاشان جم شده بود و من ک نبود که همه ر ب گک بار کوا تعند تومک
Target:
هزار سوال روی لب های اشان جمع شده بود و ممکن نبود که همه را به یک باره جواب دهند تومک
----------------------------------------
```

Finally, based on the results recorded by the assessment criteria, we compare these three models. It should be noted that due to the very little data in the dataset, the accuracy recorded by these three models is far from desirable. As a result, the model will not be able to accurately predict and produce an entire sentence exactly like the label without any error. Therefore, evaluation metrics that will assess the model's output based on the output of the entire sentence will not be effective and suitable for evaluating these three models. For this reason, the Accuracy evaluation metric will be complete without any useful information for us.

Therefore, other criteria like Precision and Recall have been calculated based on the predicted characters and characters in the label

As mentioned in the implementation section, two new evaluation metrics were predicted to better evaluate the predicted models. Levenshtein distance measures the similarity of two output strings and the label, and CER or character-level accuracy is a good metric for evaluating these models.

The results of each model are shown in colors specific to them in the charts below:

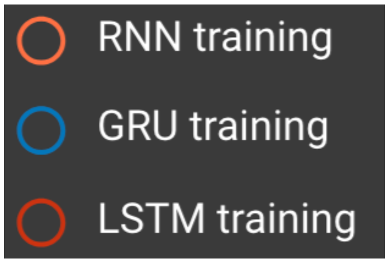


Chart of model error on training data:

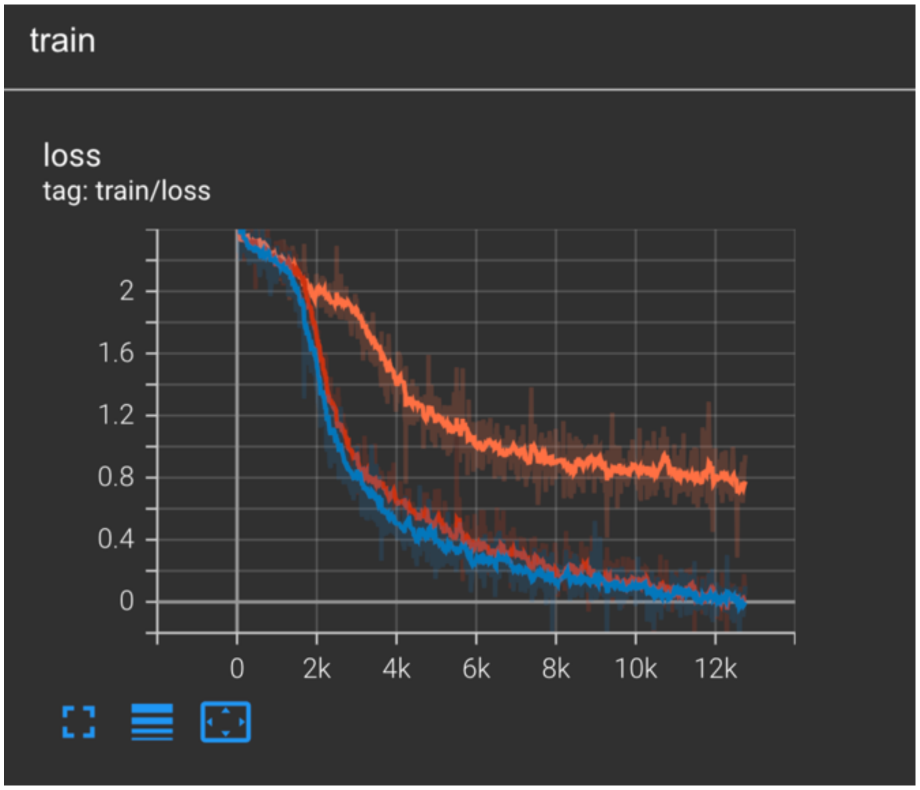
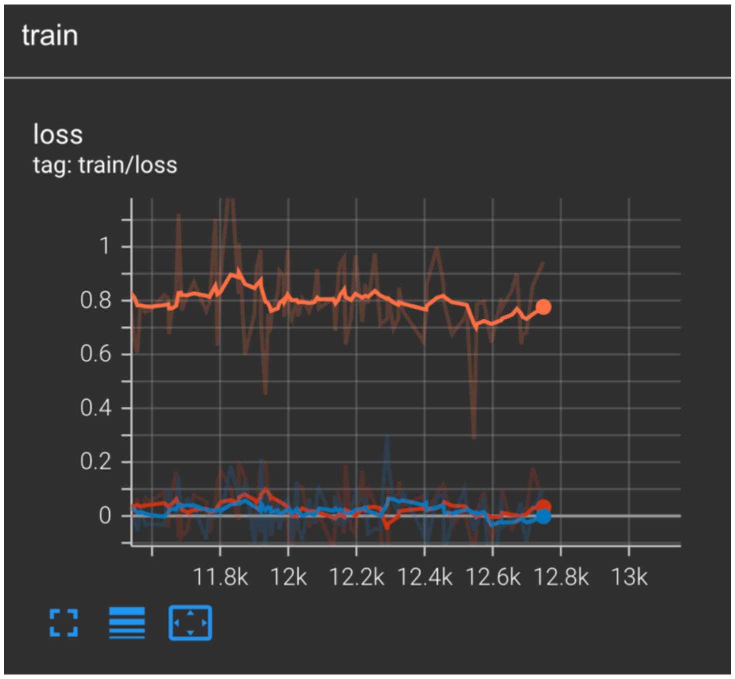
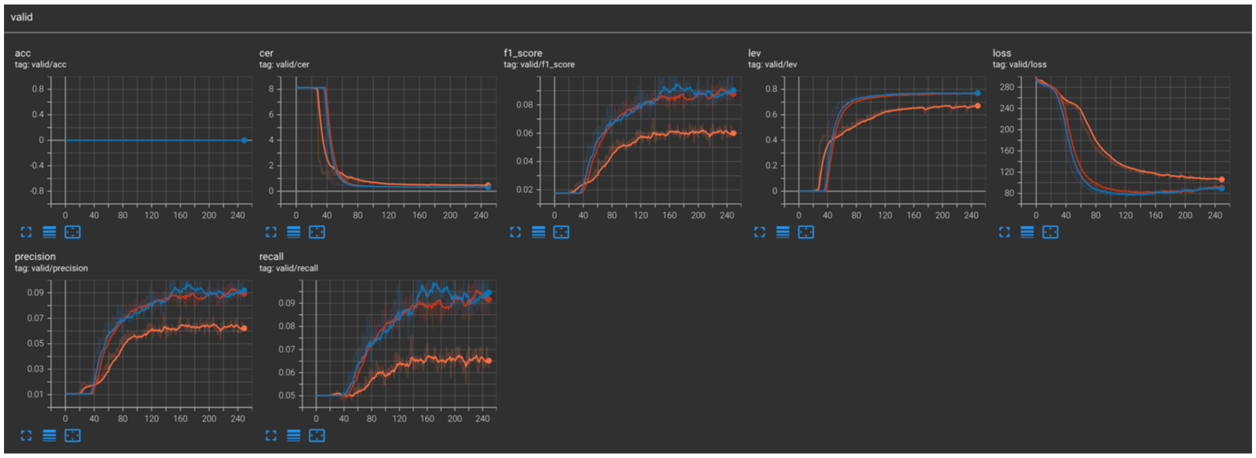


Chart of model error on training data in larger dimensions:

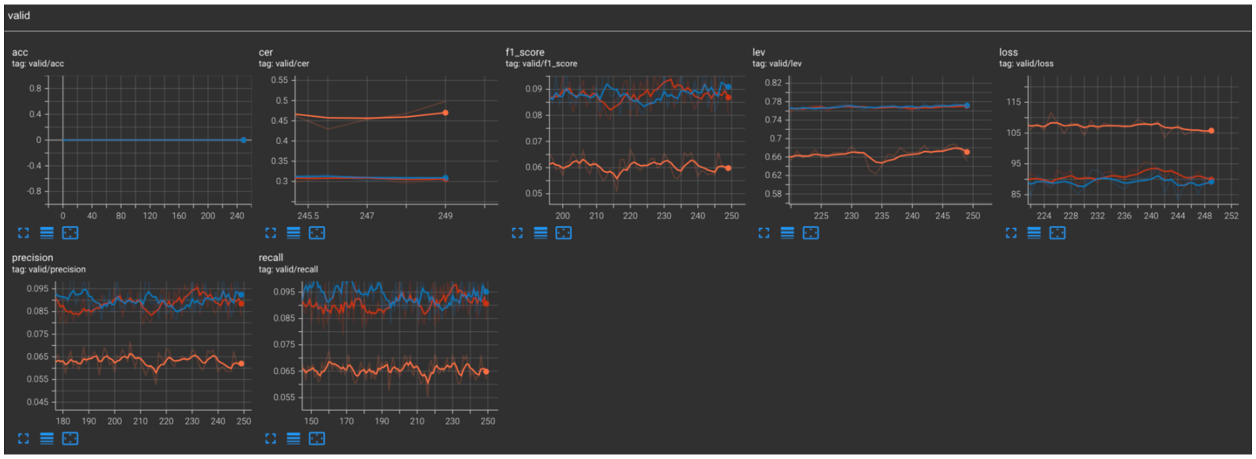


As you can see in the charts above, the RNN model error is much higher than the GRU and LSTM models. The results recorded by the GRU and LSTM models are also very close to each other.

Charts related to each of the evaluation criteria for models on validation data:



Charts related to each of the evaluation criteria for models on validation data in larger dimensions:



As predicted in advance and shown in the charts above, the model related to the RNN network performed worse than the other models and the other two models strongly recorded very close results.

Comparison of RNN and GRU and LSTM models:

Recurrent RNN networks will perform much weaker due to their simpler structure and fewer parameters or weights compared to the other two competitors. In other words, due to being simpler, the model will not be able to learn long-term dependencies and will record weak results.

Comparison of GRU and LSTM models:

The GRU and LSTM models, due to having more parameters compared to RNN, recorded better results. The LSTM recurrent neural networks also have more parameters than GRU for learning long-term dependencies, which has made the LSTM models much more complex. This greater complexity and number of parameters require more training data to tune the model weights more precisely. If we take a look at the charts we get, we see that these two models strongly recorded very close results, but in some criteria, the GRU model managed to get ahead of LSTM. This superiority can be due to the fact that the GRU model is simpler compared to LSTM and according to the little data we have for solving this problem, the fewer weights of the GRU model have been trained a little better than the many and complex weights of the LSTM model and they recorded slightly better results.