Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Course No. : CSE 4238
Course Name : Soft Computing Lab

Assignment No. : 03

Submitted By:

Name : Emdadul Haque
ID No. : 17 01 04 028
Session : Fall - 2020
Section : A (A2)

Table of Contents

1. Dataset Creation

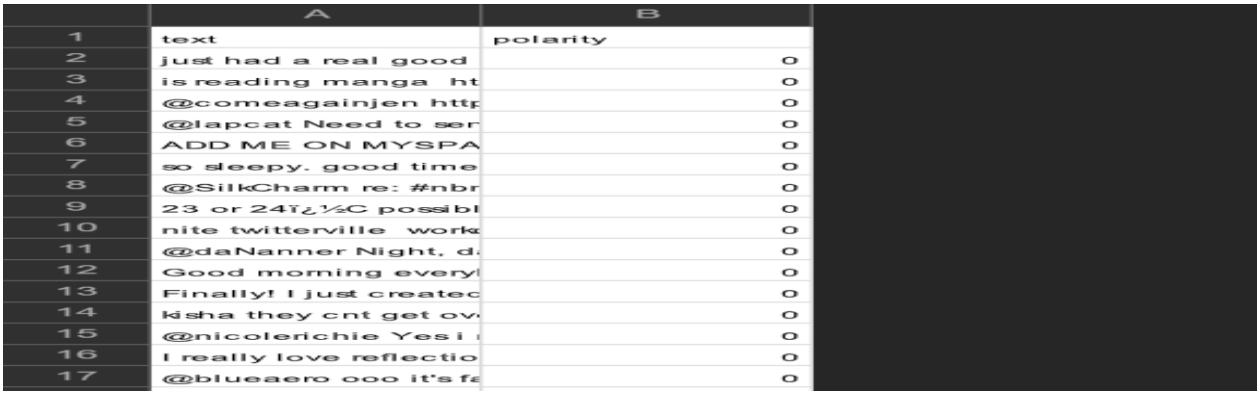
In this section, we know about the creation process of the dataset.

For the data creation process, we divide it into two different sub-processes.

1. Data Download
2. Creating Own dataset
    1. Read the raw data
    2. Dataset splitting
    3. Text cleaning
    4. Saving the new data

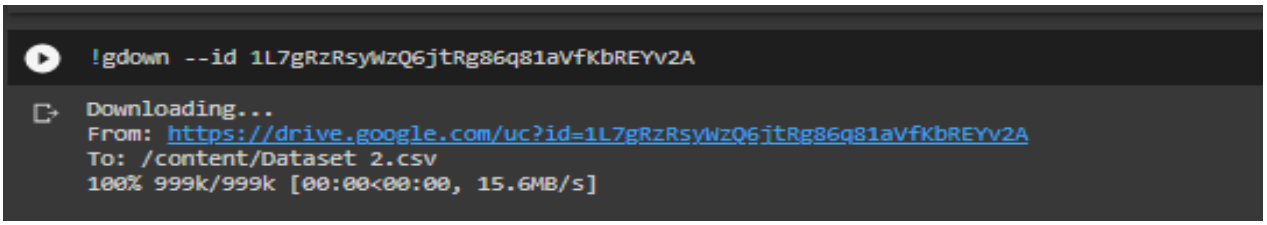
1.1. Data Downloading:

In the data downloading process, first, we need to download file based on my id. So, my id is 028. So, 28%3 = 1. So, my dataset number is 2. I need to download Dataset 2. The dataset looks like-



	A	B
1	text	polarity
2	just had a real good	0
3	is reading manga ht	0
4	@comeagainjen http	0
5	@lapcat Need to ser	0
6	ADD ME ON MYSPA	0
7	so sleepy. good time	0
8	@SilkCharm re: #nbr	0
9	23 or 24ï¿½C possibl	0
10	nite twitterville work	0
11	@daNanner Night, d	0
12	Good morning every	0
13	Finally! I just createc	0
14	kisha they cnt get ov	0
15	@nicolerichie Yes i	0
16	I really love reflectio	0
17	@blueaero ooo it's fa	0

For download the dataset I used gdown package. Because this is google drive link. For downloading I used given code-



```
!gdown --id 1L7gRzRsyWzQ6jtRg86q81aVfKbREYv2A

Downloading...
From: https://drive.google.com/uc?id=1L7gRzRsyWzQ6jtRg86q81aVfKbREYv2A
To: /content/Dataset 2.csv
100% 999k/999k [00:00<00:00, 15.6MB/s]
```

1.2. Creating Own Dataset

After Collecting the raw data. We need to generate and pre-process dataset.

1.2.1. Read the raw data:

For read the csv file I use pandas dataframe. Because it is easy to use. For read data properly. I also need the encoding. For reading the raw data I use this code –

```
df = pd.read_csv('/content/Dataset 2.csv',encoding='unicode_escape')
```

1.2.2. Dataset splitting

After reading data I need splitting data training and testing. Also, I will maintain 80% and 20% data. Also, I try to maintain positive and negative class properly. The code looks like –

```
## split dataset based on the class
traning_split_size = 0.8
df_class_1 = df[df['polarity'] == 1]
df_class_0 = df[df['polarity'] == 0]

trainSize = int(len(df_class_0) * traning_split_size)
Traning_class_0 = df_class_0[:trainSize]
Test_class_0 = df_class_0[trainSize:]

trainSize = int(len(df_class_1) * traning_split_size)
Traning_class_1 = df_class_1[:trainSize]
Test_class_1 = df_class_1[trainSize:]
```

```
[ ] ## traning dataset create
li = [Traning_class_0,Traning_class_1]
frame = pd.concat(li, axis=0, ignore_index=True)
frame = shuffle(frame)
frame.reset_index(inplace=True, drop=True)
frame.to_csv('Train-Dataset.csv',index=False)
frame
```

```
## testing dataset create
li = [Test_class_0,Test_class_1]
frame = pd.concat(li, axis=0, ignore_index=True)
frame = shuffle(frame)
frame.reset_index(inplace=True, drop=True)
frame.to_csv('Test-Dataset.csv',index=False)
frame
```

1.2.3. Text cleaning

In our dataset we used text-based dataset. We know text preprocessing is an important part of data of NLP. That's why we need to preprocess dataset properly. For text cleaning we used this code.

```
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers. removibng miltiple full stop'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub(r'\.+', ".", text)
    return text

def replace_text(text):
    text = str(text).lower()
    text = text.encode('ascii', 'ignore').decode('utf-8')
    return text


for dta in [traning_df,testing_df]:
  dta['text_cleaning'] = dta.text.apply(lambda x: x.strip().lower() )
  dta['text_cleaning'] = dta.text_cleaning.apply(lambda x : " ".join(x.split()) )
  dta['text_cleaning'] = dta.text_cleaning.apply(lambda x: contractions.fix(x) )
  dta['text_cleaning'] = dta.text_cleaning.apply(lambda x: clean_text(x) )
  dta['text_cleaning'] = dta.text_cleaning.apply(lambda x: replace_text(x) )
  dta['number_of_word'] = dta.text_cleaning.apply(lambda x: len(word_tokenize(x)) )
  dta['number_of_letter'] = dta.text_cleaning.apply(lambda x: len(x) )
  del dta['text']
  dta['text'] = dta['text_cleaning']
  del dta['text_cleaning']



for dta in [traning_df,testing_df]:
  word_count_zero = dta[dta['number_of_word'] == 0]
  dta.drop(word_count_zero.index,inplace=True)
  letter_count_zero = dta[dta['number_of_letter'] == 0]
  dta.drop(letter_count_zero.index,inplace=True)
```

1.2.4. Saving the new data

After doing those process, we must need to save the data. For saving the data we used this code –

```
[ ] traning_df.to_csv('/content/Train-Dataset-prcessed.csv',index=False)
    testing_df.to_csv('/content/Test-Dataset-prcessed.csv',index=False)
```

2. Training Process

For training process, I follow those steps –

1. Load dataset
2. Model Creation

2.1. Load Dataset

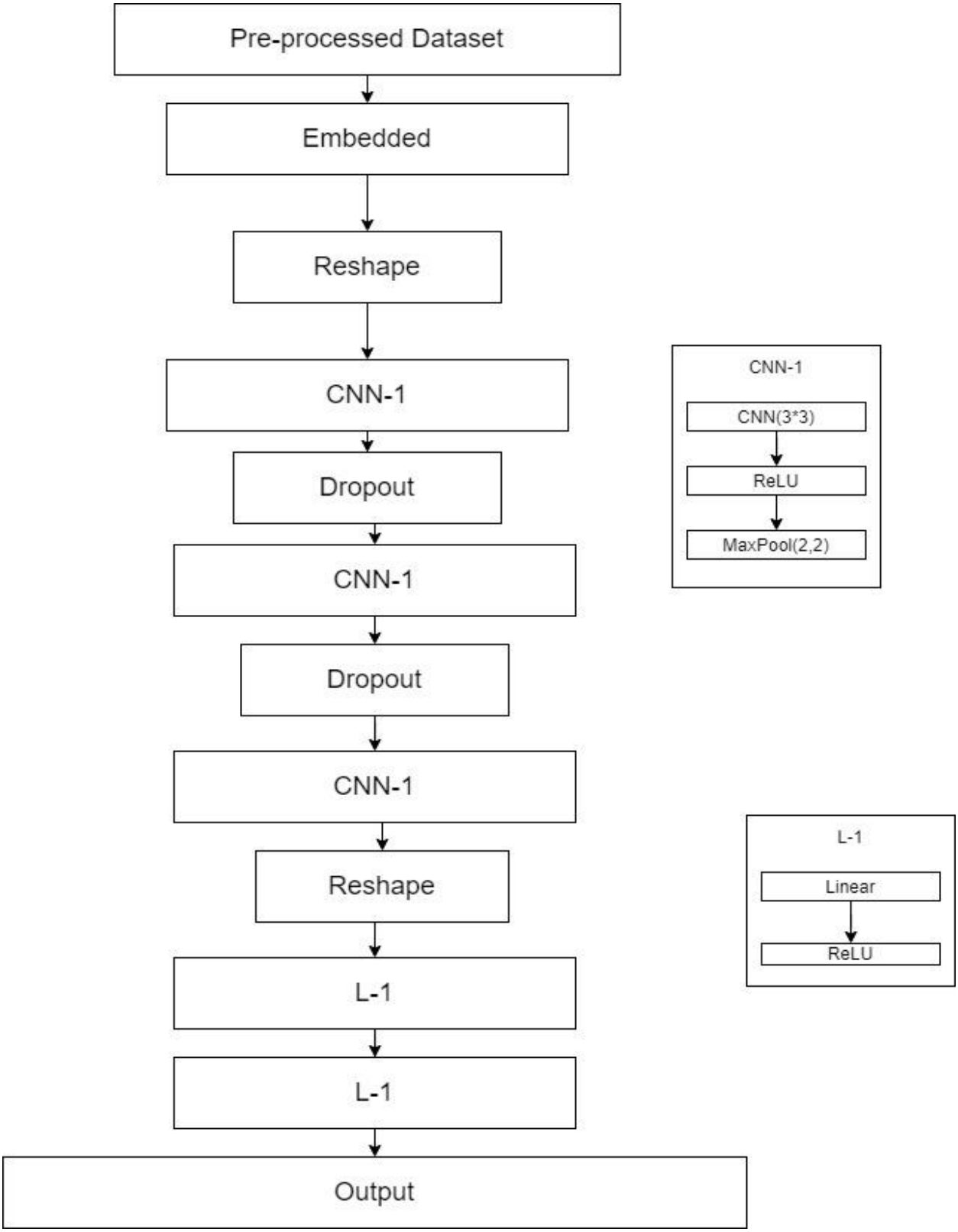
For loading the dataset I used TabularDataset and BucketIterator. TabularDataset is use to read data from saved and pre-processed. The dataset loaded code-

```
label_field = Field(sequential=False, use_vocab=False, batch_first=True, dtype=torch.float)
text_field = Field(tokenize='spacy', lower=True, include_lengths=True, batch_first=True)
fields = [('polarity', label_field),  ('text', text_field)]


train_data = TabularDataset(path="/content/Train-Dataset-prcessed.csv",
                                format="csv",
                                 fields=fields,
                                skip_header=True)

valid_data = TabularDataset(path="/content/Test-Dataset-prcessed.csv",
                                format="csv",
                                 fields=fields,
                                skip_header=True)

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

train_iter = BucketIterator(train_data, batch_size=32, sort_key=lambda x: len(x.text),
                            device=device, sort=True, sort_within_batch=True)
valid_iter = BucketIterator(valid_data, batch_size=32, sort_key=lambda x: len(x.text),
                            device=device, sort=True, sort_within_batch=True)


text_field.build_vocab(train_data,) # min_freq=3,vectors = "glove.6B.100d"
label_field.build_vocab(train_data)
```

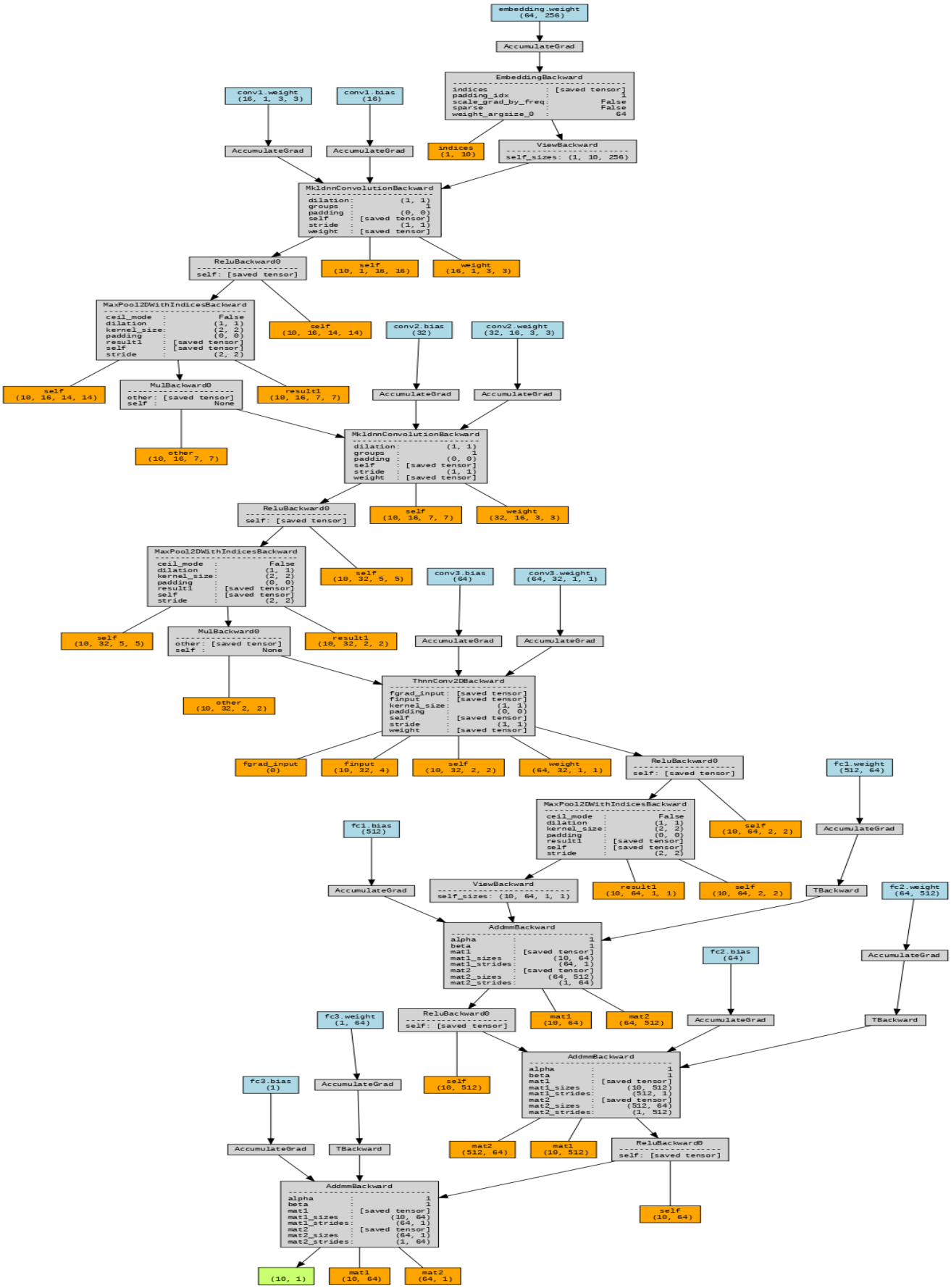
```
#No. of unique tokens in text
print("Size of TEXT vocabulary:",len(text_field.vocab))

#No. of unique tokens in label
print("Size of LABEL vocabulary:",len(label_field.vocab))

#Commonly used words
print(text_field.vocab.freqs.most_common(10))

#Word dictionary
print(text_field.vocab.stoi)
```

2.2. Model Creation

For the question, I need to make the CNN based model. My model architected look like.



The flow of model is –



The model parameter is –

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      --                        --
├─Embedding: 1-1                         [1, 10, 256]              16,384
├─Conv2d: 1-2                            [10, 16, 14, 14]          160
├─MaxPool2d: 1-3                         [10, 16, 7, 7]            --
├─Dropout: 1-4                           [10, 16, 7, 7]            --
├─Conv2d: 1-5                            [10, 32, 5, 5]            4,640
├─MaxPool2d: 1-6                         [10, 32, 2, 2]            --
├─Dropout: 1-7                           [10, 32, 2, 2]            --
├─Conv2d: 1-8                            [10, 64, 2, 2]            2,112
├─MaxPool2d: 1-9                         [10, 64, 1, 1]            --
├─Linear: 1-10                           [10, 512]                 33,280
├─Linear: 1-11                           [10, 64]                  32,832
├─Linear: 1-12                           [10, 1]                   65
==========================================================================================
Total params: 89,473
Trainable params: 89,473
Non-trainable params: 0
Total mult-adds (M): 2.24
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.40
Params size (MB): 0.36
Estimated Total Size (MB): 0.76
==========================================================================================
```

Also, the code of model is-

```
class CNN(torch.nn.Module):
    def __init__(self , vocab_size, embedding_dim, hidden_dim, output_dim, n_lay
                 bidirectional, dropout, pad_idx):
        super(CNN, self).__init__()

        self.embedding = torch.nn.Embedding(vocab_size, embedding_dim, padding_

        #initializing convolution layer
        self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size
        self.conv2 = torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_si
        self.conv3 = torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_si

        #initializing dropout
        self.dropout = torch.nn.Dropout(0.2)

        #initializing MaxPool2d
        self.pool= torch.nn.MaxPool2d(2,2)

        #initializing linear
        self.fc1 = torch.nn.Linear(64* 1* 1, 512)
        self.fc2 = torch.nn.Linear(512,64)
        self.fc3 = torch.nn.Linear(64,output_dim)


    def forward(self, text):
        embedded = self.embedding(text)
        x = embedded.view(-1,1,16,16)
        x = self.pool(torch.nn.functional.relu(self.conv1(x)))
        x = self.dropout(x)
        x = self.pool(torch.nn.functional.relu(self.conv2(x)))
        x = self.dropout(x)
        x = self.pool(torch.nn.functional.relu(self.conv3(x)))
        x = x.view(-1, 64* 1* 1)
        # x = self.pool(torch.nn.functional.relu(x))
        # x = x.view(-1, 32* 1* 1)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

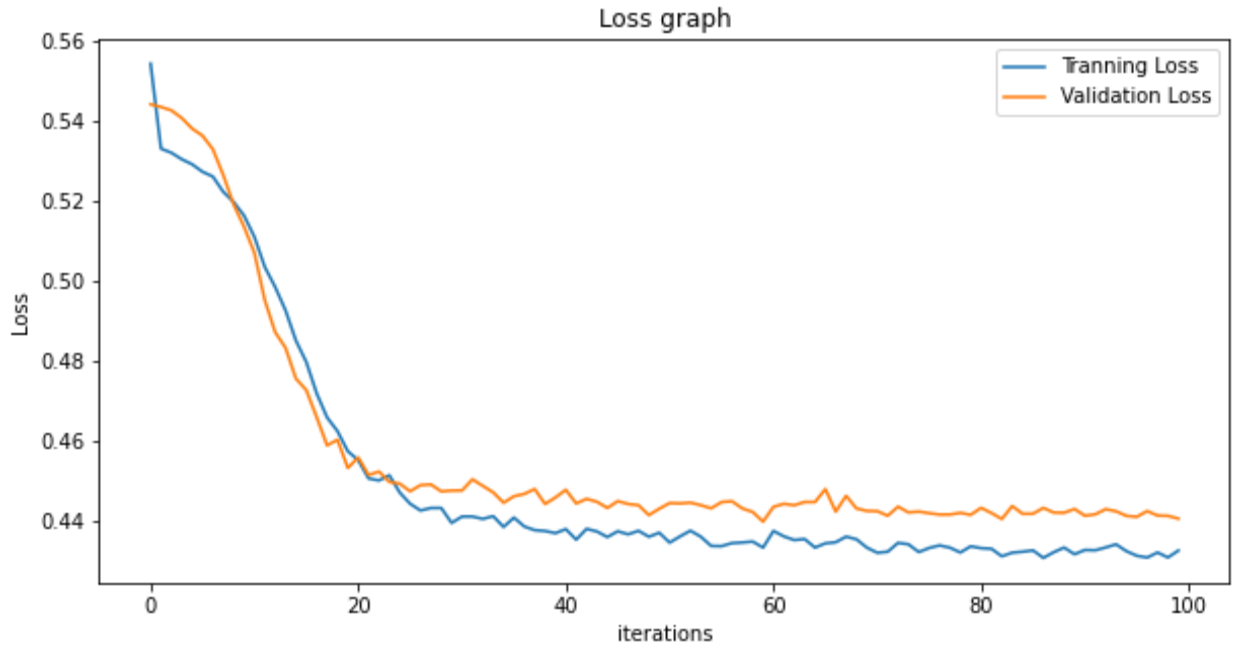
3. Result

For training the data I will use Colab. Because my personal computer is not enough to load this high performance. Below I was showing all the training result.

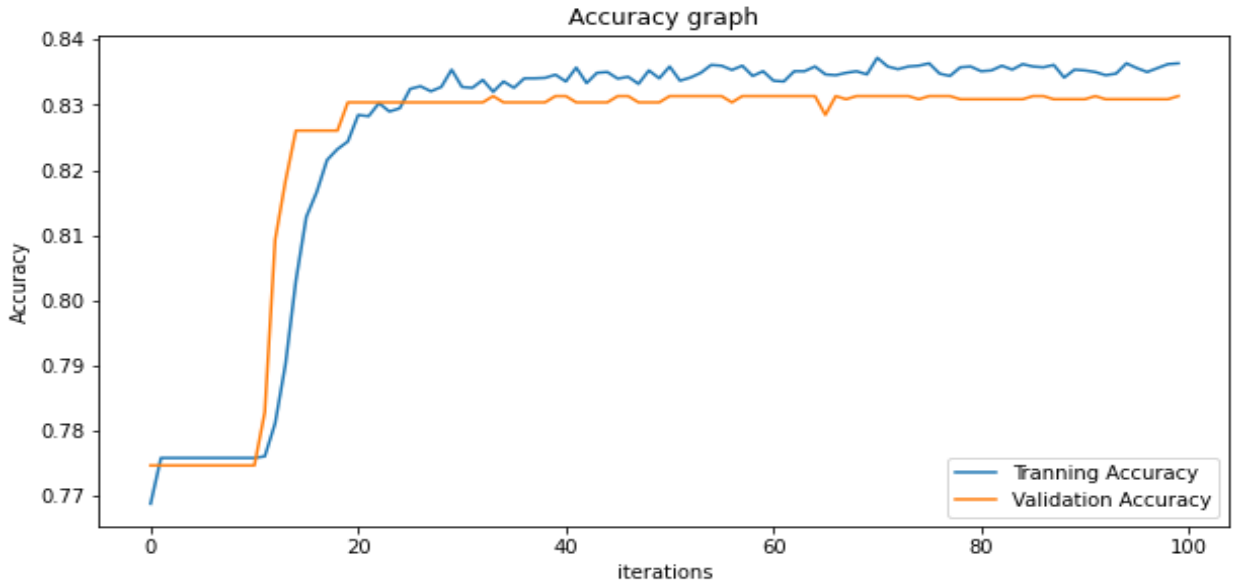
3.1.1. Parameter

Hyper Parameter	Value of parameter
K(factor)	10
iteration	100
Platform run	Colab

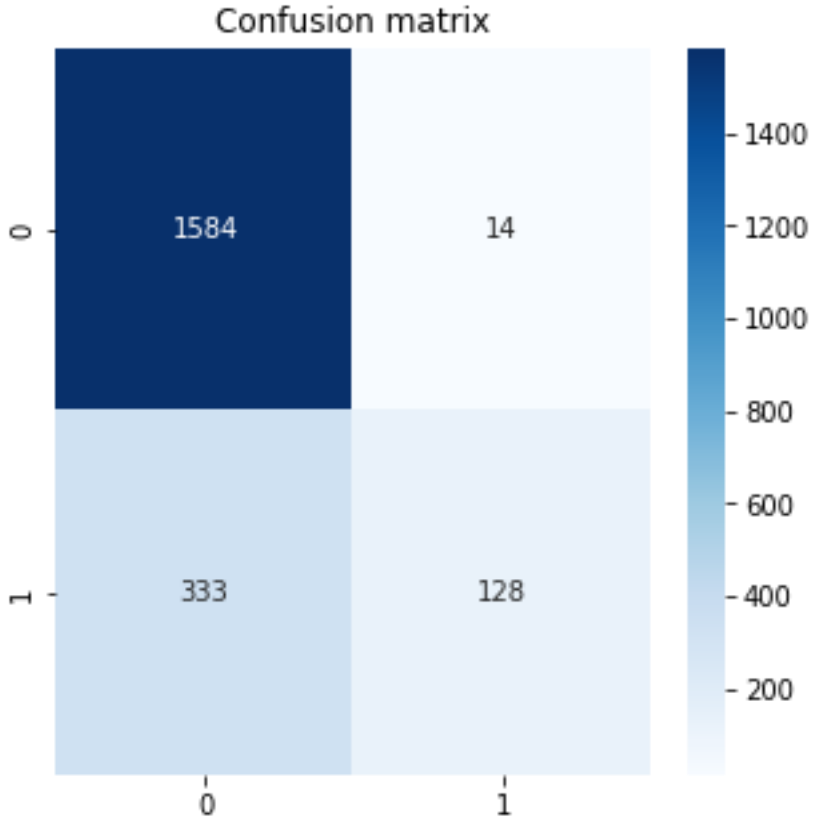
3.1.2. Loss graph plotting



3.1.3. Accuracy graph plotting



3.1.4. Confusion matrices



3.1.5. Other's performance matrices

	precision	recall	f1-score	support
0	0.83	0.99	0.90	1598
1	0.90	0.28	0.42	461
accuracy			0.83	2059
macro avg	0.86	0.63	0.66	2059
weighted avg	0.84	0.83	0.79	2059

6. References

Source code:

https://colab.research.google.com/drive/162ys8eXTEQi6Lj6CH6EKT_C88F9RnYlw

Data info:

https://drive.google.com/drive/folders/1-PImfvx9IGyMHo7ryZoFXgCrfRLlKu9K?usp=sharing