Employing CNN on the text classification

CSE4022

NLP DA

Slot: - G2 + TG2

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Team Details:

Team mate 1: Mihir Agarwal
Reg No: 18BCE2526
Team Mate 2: Shashank Shukla
Reg No. 18BCE2522

Link to code:
https://colab.research.google.com/drive/1zAdxm7tPWCqEsv9HgemzeQz1uRcuZ1CR?usp=sharing

Prof. RAJESHKANNAN R

Problem Statement

In today's world we are completely surrounded with different types of news articles, and it becomes a time-taking task if someone wants to read news of a particular topic. One of the research by Oxford University shows that out of 11000 people studied, Men are much more interested in sports news, political news, and business and economic news. Women place more importance on health and education news, local news, and especially entertainment and celebrity news. In terms of age, interest in political and economic news grows as people get older. In contrast, entertainment and science and technology news are of greater interest for younger groups. From these figures it's clearly visible that different sections of people are interested in different types of news. In current times, people are highly busy with their work and have less time to spend on news and no one wants to read an article which is not of their interest. So, it's highly important to decide which article is of our interest without reading it completely. In this assignment we have tried to classify different news articles into different classes.

Dataset

Link to the dataset - http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/news20.html

The dataset is a collection of 20,000 messages collected from 20 different newsgroups, thousand messages each from the twenty newsgroups were chosen at random and partitioned by newsgroup name. So each message has a newsgroup allocated to it. The list of newsgroups from which the messages were chose is as follows:

- alt.atheism
- talk.politics.guns
- talk.politics.mideast
- talk.politics.misc
- talk.religion.misc
- soc.religion.christian
- comp.sys.ibm.pc.hardware

- comp.graphics
- comp.os.ms-windows.misc
- comp.sys.mac.hardware
- comp.windows.x
- rec.autos
- rec.motorcycles
- rec.sport.baseball
- rec.sport.hockey
- sci.crypt
- sci.electronics
- sci.space
- sci.med
- misc.forsale

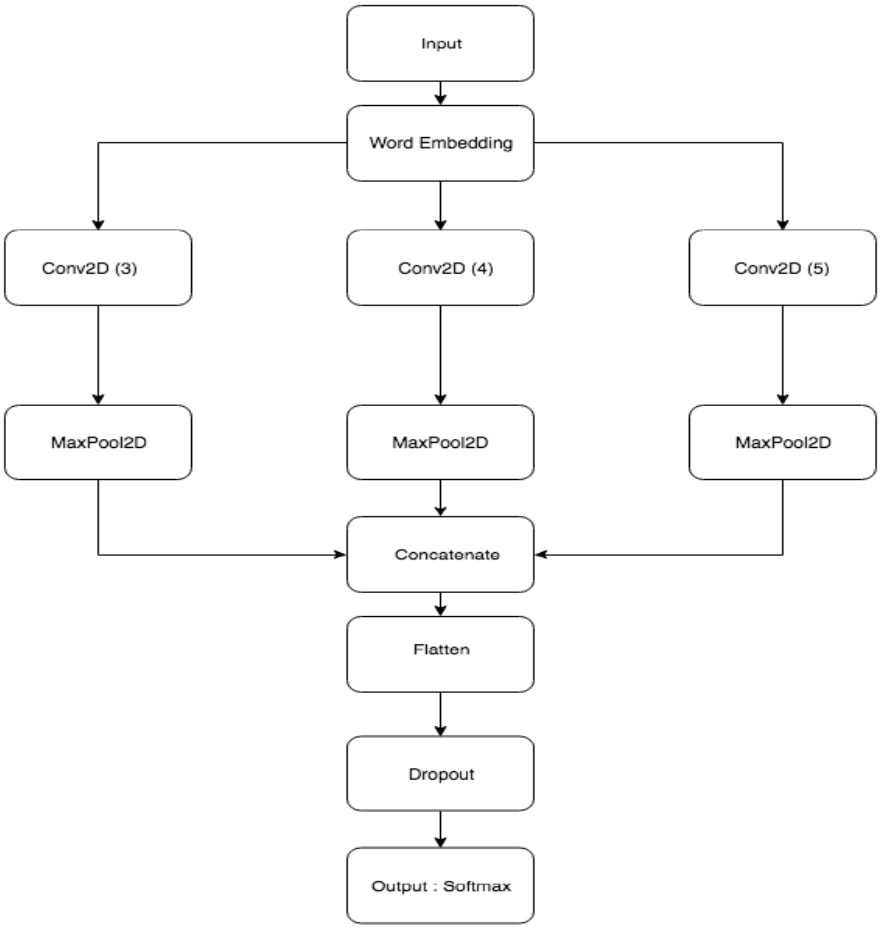
One sample of such message of class comp.graphics from dataset is show below-

```
                        CALL FOR PRESENTATIONS

      NAVY SCIENTIFIC VISUALIZATION AND VIRTUAL REALITY SEMINAR

                        Tuesday, June 22, 1993

             Carderock Division, Naval Surface Warfare Center
               (formerly the David Taylor Research Center)
                           Bethesda, Maryland

SPONSOR: NESS (Navy Engineering Software System) is sponsoring a
one-day Navy Scientific Visualization and Virtual Reality Seminar.
The purpose of the seminar is to present and exchange information for
Navy-related scientific visualization and virtual reality programs,
research, developments, and applications.

PRESENTATIONS: Presentations are solicited on all aspects of
Navy-related scientific visualization and virtual reality.  All
current work, works-in-progress, and proposed work by Navy
organizations will be considered.  Four types of presentations are
available.

     1. Regular presentation: 20-30 minutes in length
     2. Short presentation: 10 minutes in length
     3. Video presentation: a stand-alone videotape (author need not
        attend the seminar)
     4. Scientific visualization or virtual reality demonstration (BYOH)

Accepted presentations will not be published in any proceedings,
however, viewgraphs and other materials will be reproduced for
seminar attendees.

ABSTRACTS: Authors should submit a one page abstract and/or videotape to:

     Robert Lipman
     Naval Surface Warfare Center, Carderock Division
     Code 2042
     Bethesda, Maryland  20084-5000
```

Approach-

Approach used is famous text/document classification. Around 2014 yoon kim et al. started to experiment with the relevance of CNN in the field of NLP. In the paper "Convolutional Neural Networks for Sentence Classification" yoon kim et al. experiments with multiple CNN models (single channel, multiple channel) on top of word embeddings for text classification.
Link of paper- https://arxiv.org/pdf/1408.5882.pdf
We used a single channel model which is easy with pretrasined Glove embeddings. The data set used is the famous 20_newsgroup dataset. In this code, we will do the processing of the dataset, followed by a keras implementation of text classification using the preexisting Glove embeddings.

Architecture Diagram-



Code-

```
from google.colab import drive
drive.mount('/content/drive')

"""#Unzipping zip file containing dataset"""

!unzip "/content/drive/MyDrive/NLP_Da/20_newsgroup.zip" -d
"/content/drive/MyDrive/NLP_Da/Dataset"

"""#Import necessary packages"""

import os
import sys
import numpy as np
import keras
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Activation, Conv2D, Input, Embedding,
Reshape, MaxPool2D, Concatenate, Flatten, Dropout, Dense, Conv1D
from keras.layers import MaxPool1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adam

"""#View content of dataset"""

# just to make sure the dataset is added properly
!ls '/content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup'

"""#Initializing required parameters """

# the dataset path
TEXT_DATA_DIR = r'../input/20-newsgroup-
original/20_newsgroup/20_newsgroup/'
#the path for Glove embeddings
GLOVE_DIR = r'../input/glove6b/'
# make the max word length to be constant
MAX_WORDS = 10000
MAX_SEQUENCE_LENGTH = 1000
# the percentage of train test split to be applied
VALIDATION_SPLIT = 0.20
# the dimension of vectors to be used
EMBEDDING_DIM = 100
```

```
# filter sizes of the different conv layers
filter_sizes = [3,4,5]
num_filters = 512
embedding_dim = 100
# dropout probability
drop = 0.5
batch_size = 30
epochs = 2

"""#DATASET STRUCTURE

The dataset is present in a hierarchical structure, i.e. all
files of a given class are located in their respective folders
and each datapoint has its own '.txt' file.

* First we go through the entire dataset to build our text list
and label list.
* Followed by this we tokenize the entire data using Tokenizer,
which is a part of keras.preprocessing.text.
* We then add padding to the sequences to make them of a uniform
length.
"""

## preparing dataset


texts = []  # list of text samples
labels_index = {}  # dictionary mapping label name to numeric id
labels = []  # list of label ids
for name in sorted(os.listdir(TEXT_DATA_DIR)):
   path = os.path.join(TEXT_DATA_DIR, name)
   if os.path.isdir(path):
       label_id = len(labels_index)
       labels_index[name] = label_id
       for fname in sorted(os.listdir(path)):
           if fname.isdigit():
               fpath = os.path.join(path, fname)
               if sys.version_info < (3,):
                   f = open(fpath)
               else:
                   f = open(fpath, encoding='latin-1')
               t = f.read()
               i = t.find('\n\n')  # skip header
               if 0 < i:
                   t = t[i:]
               texts.append(t)
```

```
                f.close()
                labels.append(label_id)
print(labels_index)

print('Found %s texts.' % len(texts))

"""#Printing one row of text data"""

print(texts[1000])

"""#Using tokenizer to get tokens"""

tokenizer  = Tokenizer(num_words = MAX_WORDS)
tokenizer.fit_on_texts(texts)
sequences =  tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print("unique words : {}".format(len(word_index)))

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
print(labels)

"""#Getting one row for testing purpose"""

x_test=data[1000]
y_test=labels[1000]
print(x_test.shape, y_test.shape)

labels[1000]

"""#Split the data into a training set and a validation set"""

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

```
"""#Glove Embedding

Since we have our train-validation split ready, our next step is
to create an embedding matrix from the precomputed Glove
embeddings.
For convenience we are freezing the embedding layer i.e we will
not be fine tuning the word embeddings. From what can be seen,
the Glove embeddings are universal features and tend to perform
great in general.
"""

embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

from keras.layers import Embedding

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)

"""#Initializing the model"""

inputs = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedding = embedding_layer(inputs)

print(embedding.shape)
reshape =
Reshape((MAX_SEQUENCE_LENGTH,EMBEDDING_DIM,1))(embedding)
```

```
print(reshape.shape)

conv_0 = Conv2D(num_filters, kernel_size=(filter_sizes[0],
embedding_dim), padding='valid', kernel_initializer='normal',
activation='relu')(reshape)
conv_1 = Conv2D(num_filters, kernel_size=(filter_sizes[1],
embedding_dim), padding='valid', kernel_initializer='normal',
activation='relu')(reshape)
conv_2 = Conv2D(num_filters, kernel_size=(filter_sizes[2],
embedding_dim), padding='valid', kernel_initializer='normal',
activation='relu')(reshape)

maxpool_0 = MaxPool2D(pool_size=(MAX_SEQUENCE_LENGTH -
filter_sizes[0] + 1, 1), strides=(1,1), padding='valid')(conv_0)
maxpool_1 = MaxPool2D(pool_size=(MAX_SEQUENCE_LENGTH -
filter_sizes[1] + 1, 1), strides=(1,1), padding='valid')(conv_1)
maxpool_2 = MaxPool2D(pool_size=(MAX_SEQUENCE_LENGTH -
filter_sizes[2] + 1, 1), strides=(1,1), padding='valid')(conv_2)

concatenated_tensor = Concatenate(axis=1)([maxpool_0, maxpool_1,
maxpool_2])
flatten = Flatten()(concatenated_tensor)
dropout = Dropout(drop)(flatten)
output = Dense(units=20, activation='softmax')(dropout)

# this creates a model that includes
model = Model(inputs=inputs, outputs=output)

checkpoint = ModelCheckpoint('weights_cnn_sentece.hdf5',
monitor='val_acc', verbose=1, save_best_only=True, mode='auto')
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
decay=0.0)

model.compile(optimizer=adam, loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()

"""#Training the model"""

print("Traning Model...")
model.fit(x_train, y_train, batch_size=batch_size, epochs=20,
verbose=1, callbacks=[checkpoint], validation_data=(x_val,
y_val))

"""#Loss and Accuracy of the trained model"""
```

```
score, acc = model.evaluate(x_val, y_val)
print("Loss: ", score)
print("Accuracy: ", acc*100)

"""#Prwedicting on a single text"""

x_test=x_test.reshape(1, 1000)
pred=model.predict(x_test).argmax()

print("Actual label: ", y_test.argmax())
print("Predicted label: ", pred)

print(labels[1000].argmax())

"""#Label of displayed text is predicted which is accurate"""

labels_index

print(texts[1000])
```

Output-

Training Result after 20 epochs-

```
534/534 [==============================] - 142s 266ms/step - loss: 0.1582 - accuracy: 0.9571 - val_loss: 0.6311 - val_accuracy: 0.8070
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 42/50
534/534 [==============================] - 142s 267ms/step - loss: 0.1568 - accuracy: 0.9578 - val_loss: 0.6163 - val_accuracy: 0.8117
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 43/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1457 - accuracy: 0.9605 - val_loss: 0.6242 - val_accuracy: 0.8122
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 44/50
534/534 [==============================] - 141s 265ms/step - loss: 0.1436 - accuracy: 0.9609 - val_loss: 0.6248 - val_accuracy: 0.8137
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 45/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1420 - accuracy: 0.9602 - val_loss: 0.6246 - val_accuracy: 0.8112
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 46/50
534/534 [==============================] - 142s 267ms/step - loss: 0.1401 - accuracy: 0.9638 - val_loss: 0.6250 - val_accuracy: 0.8097
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 47/50
534/534 [==============================] - 141s 265ms/step - loss: 0.1425 - accuracy: 0.9610 - val_loss: 0.6284 - val_accuracy: 0.8137
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 48/50
534/534 [==============================] - 141s 265ms/step - loss: 0.1348 - accuracy: 0.9625 - val_loss: 0.6219 - val_accuracy: 0.8137
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 49/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1297 - accuracy: 0.9646 - val_loss: 0.6234 - val_accuracy: 0.8122
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 50/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1362 - accuracy: 0.9603 - val_loss: 0.6294 - val_accuracy: 0.8125
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
<tensorflow.python.keras.callbacks.History at 0x7fc166195310>
```

Loss and Accuracy after 20 epochs-

```
[52] score, acc = model.evaluate(x_val, y_val)
     print("Loss: ", score)
     print("Accuracy: ", acc*100)
```

```
125/125 [==============================] - 8s 66ms/step - loss: 0.6294 - accuracy: 0.8125
Loss:  0.6294453740119934
Accuracy:  81.24530911445618
```

Testing for a single text-

```
                        CALL FOR PRESENTATIONS

      NAVY SCIENTIFIC VISUALIZATION AND VIRTUAL REALITY SEMINAR

                        Tuesday, June 22, 1993

             Carderock Division, Naval Surface Warfare Center
               (formerly the David Taylor Research Center)
                           Bethesda, Maryland

SPONSOR: NESS (Navy Engineering Software System) is sponsoring a
one-day Navy Scientific Visualization and Virtual Reality Seminar.
The purpose of the seminar is to present and exchange information for
Navy-related scientific visualization and virtual reality programs,
research, developments, and applications.

PRESENTATIONS: Presentations are solicited on all aspects of
Navy-related scientific visualization and virtual reality.  All
current work, works-in-progress, and proposed work by Navy
organizations will be considered.  Four types of presentations are
available.

     1. Regular presentation: 20-30 minutes in length
     2. Short presentation: 10 minutes in length
     3. Video presentation: a stand-alone videotape (author need not
        attend the seminar)
     4. Scientific visualization or virtual reality demonstration (BYOH)

Accepted presentations will not be published in any proceedings,
however, viewgraphs and other materials will be reproduced for
seminar attendees.

ABSTRACTS: Authors should submit a one page abstract and/or videotape to:

     Robert Lipman
     Naval Surface Warfare Center, Carderock Division
     Code 2042
     Bethesda, Maryland  20084-5000
```

Actual label and Predicted label for the given text-

```
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
Actual label:  1
Predicted label:  1
```

Label Encoding for the given text-

```
{'alt.atheism': 0,
 'comp.graphics': 1,
 'comp.os.ms-windows.misc': 2,
 'comp.sys.ibm.pc.hardware': 3,
 'comp.sys.mac.hardware': 4,
 'comp.windows.x': 5,
 'misc.forsale': 6,
 'rec.autos': 7,
 'rec.motorcycles': 8,
 'rec.sport.baseball': 9,
 'rec.sport.hockey': 10,
 'sci.crypt': 11,
 'sci.electronics': 12,
 'sci.med': 13,
 'sci.space': 14,
 'soc.religion.christian': 15,
 'talk.politics.guns': 16,
 'talk.politics.mideast': 17,
 'talk.politics.misc': 18,
 'talk.religion.misc': 19}
```

Screenshot of Entire Code -

Double-click (or enter) to edit

```
[20] #Mounting google drive to get access to data uploaded in drive having dataset
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!unzip "/content/drive/MyDrive/NLP_Da/20_newsgroup.zip" -d "/content/drive/MyDrive/NLP_Da/Dataset"
```

```
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54905
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54906
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54907
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54908
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54909
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54910
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54911
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54912
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54913
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54914
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54915
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54916
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54917
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54918
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54919
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54920
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54921
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54942
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54943
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54944
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54947
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54948
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54949
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54950
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54951
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54952
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54953
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54954
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54955
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54956
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54957
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54958
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54959
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/54960
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55034
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55036
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55037
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55038
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55039
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55040
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55041
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55048
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55049
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55050
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55051
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55052
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55053
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55054
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55055
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55056
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55057
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55058
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55059
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55060
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55061
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55062
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55063
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55064
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55065
  inflating: /content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup/talk.politics.guns/55066
```

```
[38] import os
     import sys
     import numpy as np
     import keras
     from keras.preprocessing.text import Tokenizer
     from keras.utils import to_categorical
     from keras.preprocessing.sequence import pad_sequences
     from keras.layers import Activation, Conv2D, Input, Embedding, Reshape, MaxPool2D, Concatenate, Flatten, Dropout, Dense, Conv1D
     from keras.layers import MaxPool1D
     from keras.models import Model
     from keras.callbacks import ModelCheckpoint
     from keras.optimizers import Adam
```

```
[39] # just to make sure the dataset is added properly
     !ls '/content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup'
```

```
alt.atheism               rec.autos           sci.space
comp.graphics             rec.motorcycles     soc.religion.christian
comp.os.ms-windows.misc   rec.sport.baseball  talk.politics.guns
comp.sys.ibm.pc.hardware  rec.sport.hockey    talk.politics.mideast
comp.sys.mac.hardware     sci.crypt           talk.politics.misc
comp.windows.x            sci.electronics     talk.religion.misc
misc.forsale              sci.med
```

```
[40] # the dataset path
     TEXT_DATA_DIR = r'/content/drive/MyDrive/NLP_Da/Dataset/20_newsgroup'
     #the path for Glove embeddings
     GLOVE_DIR = r'/content/drive/MyDrive/NLP_Da/Dataset/Glove'
     # make the max word length to be constant
     MAX_WORDS = 10000
     MAX_SEQUENCE_LENGTH = 1000
     # the percentage of train test split to be applied
```

```
VALIDATION_SPLIT = 0.20
# the dimension of vectors to be used
EMBEDDING_DIM = 100
# filter sizes of the different conv layers
filter_sizes = [3,4,5]
num_filters = 512
embedding_dim = 100
# dropout probability
drop = 0.5
batch_size = 30
epochs = 2
```

DATASET STRUCTURE

The dataset is present in a hierarchical structure, i.e. all files of a given class are located in their respective folders and each datapoint has its own '.txt' file.

- First we go through the entire dataset to build our text list and label list.
- Followed by this we tokenize the entire data using Tokenizer, which is a part of keras.preprocessing.text.
- We then add padding to the sequences to make them of a uniform length.

```
[41] ## preparing dataset
     texts = []  # list of text samples
     labels_index = {}  # dictionary mapping label name to numeric id
     labels = []  # list of label ids
     for name in sorted(os.listdir(TEXT_DATA_DIR)):
         path = os.path.join(TEXT_DATA_DIR, name)
         if os.path.isdir(path):
             label_id = len(labels_index)
             labels_index[name] = label_id
             for fname in sorted(os.listdir(path)):
                 if fname.isdigit():
                     fpath = os.path.join(path, fname)
                     if sys.version_info < (3,):
                         f = open(fpath)
                     else:
                         f = open(fpath, encoding='latin-1')
                     t = f.read()
                     i = t.find('\n\n')  # skip header
                     if 0 < i:
                         t = t[i:]
                     texts.append(t)
                     f.close()
                     labels.append(label_id)
     print(labels_index)

     print('Found %s texts.' % len(texts))
```

```
{'alt.atheism': 0, 'comp.graphics': 1, 'comp.os.ms-windows.misc': 2, 'comp.sys.ibm.pc.hardware': 3, 'comp.sys.mac.hardware': 4, 'comp.windows.x': 5, 'misc.forsale': 6, 'rec.autos': 7, '
Found 19997 texts.
```

```
[42] print(texts[1000])
```

```
                    CALL FOR PRESENTATIONS

      NAVY SCIENTIFIC VISUALIZATION AND VIRTUAL REALITY SEMINAR

                    Tuesday, June 22, 1993

          Carderock Division, Naval Surface Warfare Center
            (formerly the David Taylor Research Center)
                      Bethesda, Maryland

SPONSOR: NESS (Navy Engineering Software System) is sponsoring a
one-day Navy Scientific Visualization and Virtual Reality Seminar.
The purpose of the seminar is to present and exchange information for
Navy-related scientific visualization and virtual reality programs,
research, developments, and applications.

PRESENTATIONS: Presentations are solicited on all aspects of
Navy-related scientific visualization and virtual reality.  All
current work, works-in-progress, and proposed work by Navy
organizations will be considered.  Four types of presentations are
available.

     1. Regular presentation: 20-30 minutes in length
     2. Short presentation: 10 minutes in length
     3. Video presentation: a stand-alone videotape (author need not
        attend the seminar)
     4. Scientific visualization or virtual reality demonstration (BYOH)

Accepted presentations will not be published in any proceedings,
however, viewgraphs and other materials will be reproduced for
seminar attendees.

ABSTRACTS: Authors should submit a one page abstract and/or videotape to:

     Robert Lipman
     Naval Surface Warfare Center, Carderock Division
     Code 2042
     Bethesda, Maryland  20084-5000

     VOICE (301) 227-3618;  FAX (301) 227-5753
     E-MAIL  lipman@oasys.dt.navy.mil

Authors should include the type of presentation, their affiliations,
addresses, telephone and FAX numbers, and addresses.  Multi-author
papers should designate one point of contact.

DEADLINES: The abstact submission deadline is April 30, 1993.
Notification of acceptance will be sent by May 14, 1993.
Materials for reproduction must be received by June 1, 1993.

For further information, contact Robert Lipman at the above address.

          PLEASE DISTRIBUTE AS WIDELY AS POSSIBLE, THANKS.
```

```
[43] tokenizer  = Tokenizer(num_words = MAX_WORDS)
     tokenizer.fit_on_texts(texts)
     sequences =  tokenizer.texts_to_sequences(texts)

     word_index = tokenizer.word_index
     print("unique words : {}".format(len(word_index)))

     data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

     labels = to_categorical(np.asarray(labels))
     print('Shape of data tensor:', data.shape)
     print('Shape of label tensor:', labels.shape)
     print(labels)
```

```
unique words : 174074
Shape of data tensor: (19997, 1000)
Shape of label tensor: (19997, 20)
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
```

```
[44] x_test=data[1000]
     y_test=labels[1000]
     print(x_test.shape, y_test.shape)
```

```
(1000,) (20,)
```

```
[45] labels[1000]
```

```
array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0.], dtype=float32)
```

```
[46] # split the data into a training set and a validation set
     indices = np.arange(data.shape[0])
     np.random.shuffle(indices)
     data = data[indices]
     labels = labels[indices]
     nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

     x_train = data[:-nb_validation_samples]
     y_train = labels[:-nb_validation_samples]
     x_val = data[-nb_validation_samples:]
     y_val = labels[-nb_validation_samples:]
```

Since we have our train-validation split ready, our next step is to create an embedding matrix from the precomputed Glove embeddings. For convenience we are freezing the embedding layer i.e we will not be fine tuning the word embeddings. Feel free to test it out for better accuracy on very specific examples. From what can be seen, the Glove embeddings are universal features and tend to perform great in general.

```
[47] embeddings_index = {}
     f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'))
     for line in f:
         values = line.split()
         word = values[0]
         coefs = np.asarray(values[1:], dtype='float32')
         embeddings_index[word] = coefs
     f.close()

     print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

```
[48] embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
     for word, i in word_index.items():
         embedding_vector = embeddings_index.get(word)
         if embedding_vector is not None:
             # words not found in embedding index will be all-zeros.
             embedding_matrix[i] = embedding_vector
```

```
[49] from keras.layers import Embedding

     embedding_layer = Embedding(len(word_index) + 1,
                                 EMBEDDING_DIM,
                                 weights=[embedding_matrix],
                                 input_length=MAX_SEQUENCE_LENGTH,
                                 trainable=False)
```

```
[50] inputs = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
     embedding = embedding_layer(inputs)

     print(embedding.shape)
     reshape = Reshape((MAX_SEQUENCE_LENGTH,EMBEDDING_DIM,1))(embedding)
     print(reshape.shape)

     conv_0 = Conv2D(num_filters, kernel_size=(filter_sizes[0], embedding_dim), padding='valid', kernel_initializer='normal', activation='relu')(reshape)
     conv_1 = Conv2D(num_filters, kernel_size=(filter_sizes[1], embedding_dim), padding='valid', kernel_initializer='normal', activation='relu')(reshape)
     conv_2 = Conv2D(num_filters, kernel_size=(filter_sizes[2], embedding_dim), padding='valid', kernel_initializer='normal', activation='relu')(reshape)

     maxpool_0 = MaxPool2D(pool_size=(MAX_SEQUENCE_LENGTH - filter_sizes[0] + 1, 1), strides=(1,1), padding='valid')(conv_0)
     maxpool_1 = MaxPool2D(pool_size=(MAX_SEQUENCE_LENGTH - filter_sizes[1] + 1, 1), strides=(1,1), padding='valid')(conv_1)
     maxpool_2 = MaxPool2D(pool_size=(MAX_SEQUENCE_LENGTH - filter_sizes[2] + 1, 1), strides=(1,1), padding='valid')(conv_2)

     concatenated_tensor = Concatenate(axis=1)([maxpool_0, maxpool_1, maxpool_2])
     flatten = Flatten()(concatenated_tensor)
```

```
dropout = Dropout(drop)(flatten)
output = Dense(units=20, activation='softmax')(dropout)

# this creates a model that includes
model = Model(inputs=inputs, outputs=output)

checkpoint = ModelCheckpoint('weights_cnn_sentece.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='auto')
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
(None, 1000, 100)
(None, 1000, 100, 1)
Model: "model_2"
```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 1000)]	0	
embedding_2 (Embedding)	(None, 1000, 100)	17407500	input_3[0][0]
reshape_2 (Reshape)	(None, 1000, 100, 1)	0	embedding_2[0][0]
conv2d_6 (Conv2D)	(None, 998, 1, 512)	154112	reshape_2[0][0]
conv2d_7 (Conv2D)	(None, 997, 1, 512)	205312	reshape_2[0][0]
conv2d_8 (Conv2D)	(None, 996, 1, 512)	256512	reshape_2[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 1, 1, 512)	0	conv2d_6[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 1, 1, 512)	0	conv2d_7[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 1, 1, 512)	0	conv2d_8[0][0]
concatenate_2 (Concatenate)	(None, 3, 1, 512)	0	max_pooling2d_6[0][0] max_pooling2d_7[0][0] max_pooling2d_8[0][0]
flatten_2 (Flatten)	(None, 1536)	0	concatenate_2[0][0]
dropout_2 (Dropout)	(None, 1536)	0	flatten_2[0][0]
dense_2 (Dense)	(None, 20)	30740	dropout_2[0][0]

```
Total params: 18,054,176
Trainable params: 646,676
Non-trainable params: 17,407,500
```

```
[51] print("Traning Model...")
     model.fit(x_train, y_train, batch_size=batch_size, epochs=50, verbose=1, callbacks=[checkpoint], validation_data=(x_val, y_val))
```

```
534/534 [==============================] - 142s 266ms/step - loss: 0.2188 - accuracy: 0.9426 - val_loss: 0.6195 - val_accuracy: 0.8040
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 32/50
534/534 [==============================] - 142s 266ms/step - loss: 0.2077 - accuracy: 0.9476 - val_loss: 0.6123 - val_accuracy: 0.8075
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 33/50
534/534 [==============================] - 142s 267ms/step - loss: 0.1935 - accuracy: 0.9503 - val_loss: 0.6173 - val_accuracy: 0.8065
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 34/50
534/534 [==============================] - 143s 267ms/step - loss: 0.1903 - accuracy: 0.9525 - val_loss: 0.6149 - val_accuracy: 0.8050
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 35/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1828 - accuracy: 0.9557 - val_loss: 0.6104 - val_accuracy: 0.8105
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 36/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1855 - accuracy: 0.9547 - val_loss: 0.6228 - val_accuracy: 0.8055
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 37/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1686 - accuracy: 0.9592 - val_loss: 0.6152 - val_accuracy: 0.8077
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 38/50
534/534 [==============================] - 143s 267ms/step - loss: 0.1706 - accuracy: 0.9554 - val_loss: 0.6107 - val_accuracy: 0.8142
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 39/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1634 - accuracy: 0.9615 - val_loss: 0.6265 - val_accuracy: 0.8090
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 40/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1627 - accuracy: 0.9580 - val_loss: 0.6135 - val_accuracy: 0.8125
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 41/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1582 - accuracy: 0.9571 - val_loss: 0.6311 - val_accuracy: 0.8070
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 42/50
534/534 [==============================] - 142s 267ms/step - loss: 0.1568 - accuracy: 0.9578 - val_loss: 0.6163 - val_accuracy: 0.8117
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 43/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1457 - accuracy: 0.9605 - val_loss: 0.6242 - val_accuracy: 0.8122
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 44/50
534/534 [==============================] - 141s 265ms/step - loss: 0.1436 - accuracy: 0.9609 - val_loss: 0.6248 - val_accuracy: 0.8137
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 45/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1420 - accuracy: 0.9602 - val_loss: 0.6246 - val_accuracy: 0.8112
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 46/50
534/534 [==============================] - 142s 267ms/step - loss: 0.1401 - accuracy: 0.9638 - val_loss: 0.6250 - val_accuracy: 0.8097
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 47/50
534/534 [==============================] - 141s 265ms/step - loss: 0.1425 - accuracy: 0.9610 - val_loss: 0.6284 - val_accuracy: 0.8137
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 48/50
534/534 [==============================] - 141s 265ms/step - loss: 0.1348 - accuracy: 0.9625 - val_loss: 0.6219 - val_accuracy: 0.8137
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 49/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1297 - accuracy: 0.9646 - val_loss: 0.6234 - val_accuracy: 0.8122
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 50/50
534/534 [==============================] - 142s 266ms/step - loss: 0.1362 - accuracy: 0.9603 - val_loss: 0.6294 - val_accuracy: 0.8125
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
<tensorflow.python.keras.callbacks.History at 0x7fc166195310>
```

```
[52] score, acc = model.evaluate(x_val, y_val)
     print("Loss: ", score)
     print("Accuracy: ", acc*100)
```

```
125/125 [==============================] - 8s 66ms/step - loss: 0.6294 - accuracy: 0.8125
Loss:  0.6294453740119934
Accuracy:  81.24530911445618
```

```
[53] x_test=x_test.reshape(1, 1000)
     pred=model.predict(x_test).argmax()
```

```
[54] print("Actual label: ", y_test.argmax())
     print("Predicted label: ", pred)
```

```
Actual label:  1
Predicted label:  1
```

```
[55] print(labels[1000].argmax())
```

```
15
```

```
[56] labels_index
```

```
{'alt.atheism': 0,
 'comp.graphics': 1,
 'comp.os.ms-windows.misc': 2,
 'comp.sys.ibm.pc.hardware': 3,
 'comp.sys.mac.hardware': 4,
 'comp.windows.x': 5,
 'misc.forsale': 6,
 'rec.autos': 7,
 'rec.motorcycles': 8,
 'rec.sport.baseball': 9,
 'rec.sport.hockey': 10,
 'sci.crypt': 11,
 'sci.electronics': 12,
 'sci.med': 13,
 'sci.space': 14,
 'soc.religion.christian': 15,
 'talk.politics.guns': 16,
 'talk.politics.mideast': 17,
 'talk.politics.misc': 18,
 'talk.religion.misc': 19}
```

```
[57] print(texts[1000])
```

```
                Carderock Division, Naval Surface Warfare Center
                  (formerly the David Taylor Research Center)
                              Bethesda, Maryland

SPONSOR: NESS (Navy Engineering Software System) is sponsoring a
one-day Navy Scientific Visualization and Virtual Reality Seminar.
The purpose of the seminar is to present and exchange information for
Navy-related scientific visualization and virtual reality programs,
research, developments, and applications.

PRESENTATIONS: Presentations are solicited on all aspects of
Navy-related scientific visualization and virtual reality.  All
current work, works-in-progress, and proposed work by Navy
organizations will be considered.  Four types of presentations are
available.

     1. Regular presentation: 20-30 minutes in length
     2. Short presentation: 10 minutes in length
     3. Video presentation: a stand-alone videotape (author need not
        attend the seminar)
     4. Scientific visualization or virtual reality demonstration (BYOH)

Accepted presentations will not be published in any proceedings,
however, viewgraphs and other materials will be reproduced for
seminar attendees.

ABSTRACTS: Authors should submit a one page abstract and/or videotape to:

     Robert Lipman
     Naval Surface Warfare Center, Carderock Division
     Code 2042
     Bethesda, Maryland  20084-5000

     VOICE (301) 227-3618;  FAX (301) 227-5753
     E-MAIL  lipman@oasys.dt.navy.mil

Authors should include the type of presentation, their affiliations,
addresses, telephone and FAX numbers, and addresses.  Multi-author
papers should designate one point of contact.

DEADLINES: The abstact submission deadline is April 30, 1993.
Notification of acceptance will be sent by May 14, 1993.
Materials for reproduction must be received by June 1, 1993.

For further information, contact Robert Lipman at the above address.

          PLEASE DISTRIBUTE AS WIDELY AS POSSIBLE, THANKS.



Robert Lipman                    | Internet: lipman@oasys.dt.navy.mil
David Taylor Model Basin - CDNSWC |       or: lip@ocean.dt.navy.mil
Computational Signatures and     | Voicenet: (301) 227-3618
   Structures Group, Code 2042   | Factsnet: (301) 227-5753
Bethesda, Maryland  20084-5000   | Phishnet: stockings@long.legs

The sixth sick shiek's sixth sheep's sick.
```

```
[ ]
```

[ ]

✓ 8s completed at 9:57 PM