HDFS Logs Anomaly Detection using Convolutional Neural Networks (CNN)

By Andrew Nguyen

Abstract:

The systems we work with on a daily basis are becoming increasingly sophisticated. It has now reached a point where any manual intervention is highly prone to errors. These systems generate a large number of logs, making it very hard for human eyes to correctly identify each one. As a result, machines must intervene to read the logs and translate them into understandable language for better and faster analysis. Multiple options exist to meet the expanding need, but many of them still require manual intervention. A Convolutional Neural Networks (CNN) model is proposed. There is a need for data pre-processing and certain well-crafted features. The model was trained and tested using a dataset of HDFS logs with 10k raw lines, 75 percent of which was used for training and the remainder for testing.

In reliable distributed software systems, anomaly detection is critical. Anomaly detection can improve system behaviour communication, enhance your root cause analysis, threats to the software ecosystem could be reduced. Anomaly detection is traditionally done by hand. Machine learning approaches, on the other hand, are enhancing the accuracy of anomaly detectors. Any procedure that detects the outliers in a dataset; those objects that don't belong, is known as anomaly detection. These anomalies might indicate anomalous network activity, reveal a malfunctioning sensor, or simply highlight data that must be cleaned before analysis. Managing and monitoring the functioning of distributed systems is a daily necessary thing today. With hundreds or thousands of things to monitor, anomaly detection can assist in identifying where an error is occurring, improving root cause investigation and allowing for faster tech assistance. Anomaly detection support the chaotic engineering monitoring purpose by finding outliers and alerting the appropriate parties to take action.

Anomaly detection is a popular topic in research, with many solutions emerging from domains as diverse as statistics, process control, signal processing, and machine learning. The purpose is to recognize data that deviates from or does not correspond with what is regarded normal, anticipated, or likely in terms of the data probability distribution, or the presence and intensity of a flag in a time series. An outlier is another synonym for an anomaly, and the two terms are sometimes interchanged.

Introduction

A person cannot analyse or work with a massive volume of data due to the increasing amount of data every day. We expect machines to handle things for us in the business world. Process abnormalities can arise in any element of such a system, including networking, software execution, machine performance, and so on, and most of these systems create and store logs that can be analysed to find faults. It's critical to spot abnormalities as soon as possible since they might cost the company money. Previously, abnormalities were discovered manually. However, we now have a new method to address this issue.

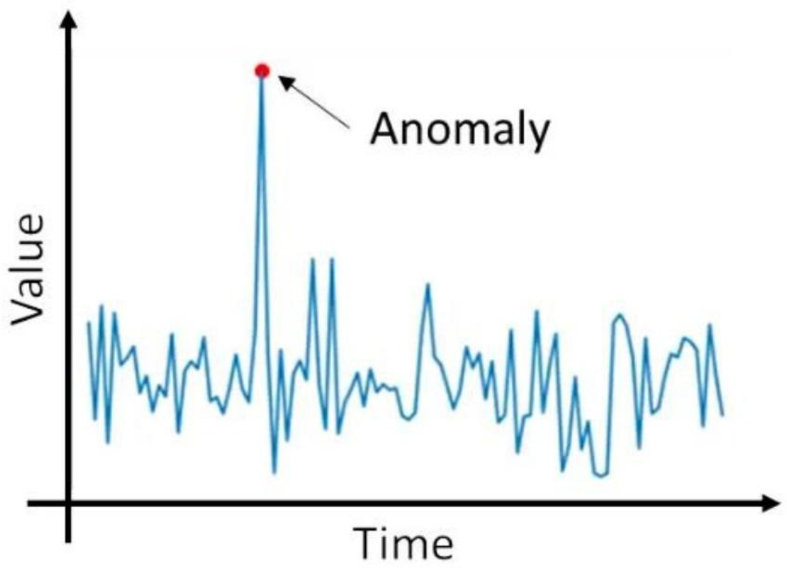
The first iterations of the contemporary system relied on the human touch, and while the machines aided, they lacked the capacity, storage, and capability required for great performance. Modern systems have advanced significantly. The approach has been established to work on a distribution system where hundreds of workstations would work on massive data logs using Hadoop or Spark. Supercomputers with a large number of processors may be used. These systems are establishing themselves as a critical component of the IT sector, enabling a wide range of online services (such as search engines, social networks, and e-commerce) as well as intelligent applications (such as weather forecasting, business intelligence, and biomedical engineering).

Because the majority of these systems are meant to operate 24 hours a day, seven days a week, and serve millions of online users throughout the world, high availability and authenticity are essential. Any failure of these systems, such as service blackouts or deterioration in service quality, can cause apps to crash and result in significant revenue loss. In this project, we present a CNN-based model with an embedding layer that detects abnormal log content while requiring little raw log file pre-processing. Anomaly detection's main goal is to identify anomalous behaviour in a systematic way. It's essential for large-scale systems with millions of rows of logs to manage incidents.

Seasonable anomaly detection allows system developers (or operators) to quickly identify and rectify issues, resulting in less system downtime. The generated logs include complete runtime information. For system anomaly detection, such extensively produced logs are employed as the principal data source. Both in academia and in industry, log-based anomaly detection has become a research area of functional relevance. Developers of traditional standalone systems manually check system logs or create laws to identify abnormalities based on their domain expertise, supplemented by keyword search or conventional expression equivalent. However, such anomaly detection is heavily reliant on manual log examination, which has proven ineffective for large-scale operations due to its high error rate.

As a result, automated log analysis approaches for detecting anomalies are in great demand. Anomaly detection using logs has been extensively researched in recent decades.

Background



Anomaly research is critical in a variety of domains, including data mining and machine learning. Its goal is to identify fields in data whose replies or patterns do not match the necessary values. Anomalies or exceptions are terms used to describe unexpected responses that are notably different from the rest of the data. However, there is no universally accepted understanding of this concept. Depending on the application scenario, an anomaly is also referred to as an outlier, a dissonant item, an exception, a distortion, or a strangeness in the research.

Anomaly detection has a wide range of applications as a result of this occurrence, including public health, credit card fraud and network penetration, and data wrangling. Many specialties, such as decision making, business intelligence, and data mining, rely heavily on classifying unusual or unexpected patterns. An unusual network transfer, for example, might indicate that a computer system is under assault by hackers or viruses, an unusual credit card transaction could indicate illegal use, and unexpected geological activity in nature could signal an earthquake or tsunami.

Data collected from real-world settings is growing more extensive and bigger, not just in terms of volume but also in terms of dimensionality, as new technologies emerge. The data items are roughly equidistant from each other due to the high-dimensional property. This means that as the dimensionality of data increases, any data items grow increasingly near together, resulting in ludicrous distances between them. The early versions of anomaly detection systems in this scenario are incapable of handling high-dimensional data properly. Furthermore, most traditional detection approaches presume that the data all have the same features. However, different attribute kinds, such as numerical, binary, categorical, or nominal, are frequently included in the data. This leads to increased difficulty in detecting anomalies. A large number of detection methods have been seen in the past due to the wide variety of conceivable applications for anomaly detection. Depending on the approaches utilised, the real anomaly detection strategies may be divided into three categories: neighbor-based, subspace-based, and ensemble-based detection methods.

Currently, there are quite a large amount of anomaly detection research papers that has been published. However, they went to different directions and used different methods such that old-fashion. For instance, 2 of the articles I found are Anomaly Detection by Combining Decision Trees and Parametric Densities (Matthias Reif, Markus Goldstein, Armin Stahl, Thomas M. Breuel, 2009) using Decision Tree model; and Machine Learning Techniques for Anomaly Detection (Salima Omar, Asri Ngadi, Hamid H. Jebur, 2013). The advancements for this project to detect anomalies for the data with high dimensionality and mixed types, which the classical detection methods cannot handle well. CNN would assist us to implement machine learning faster way and the capability of handling bigger data.

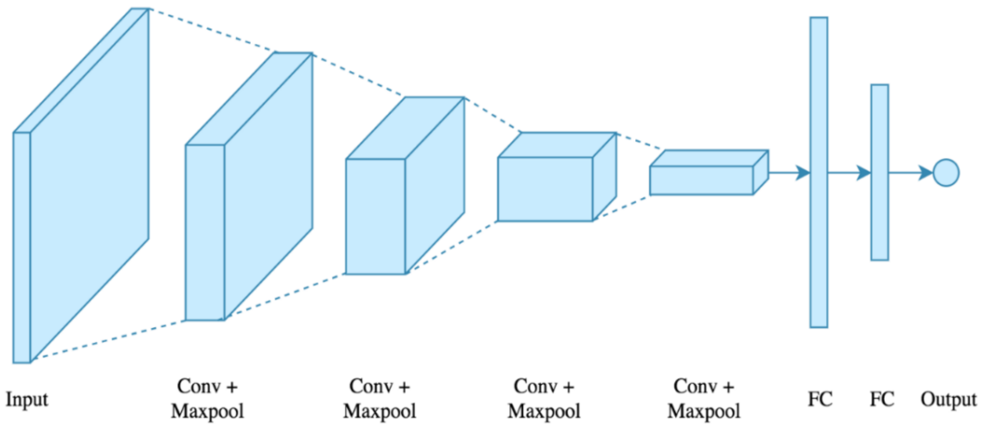
Methodology:

The technique we are using here is a classification method - CNN Model using Pytorch. Classification is simply defined as a task of assigning objects to one of the several predefined categories. Convolution Neural Network is a class of Neural network that has proven very effective in areas of image recognition, processing, and classification.

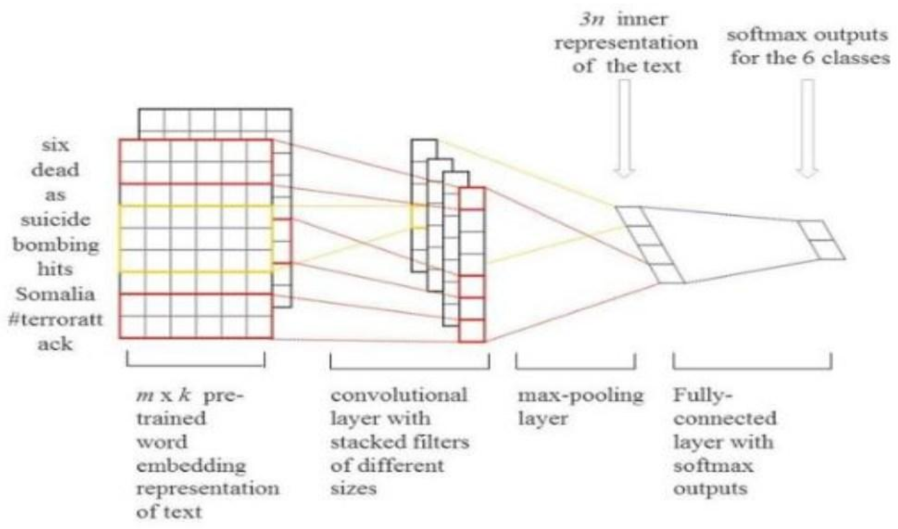
CNN can efficiently handle high-dimensional embeddings and automatically identify important local patterns in the log sequences. Convolution and pooling operations further reduce dimensionality while preserving essential features. As well, it also helps to reduce the vector by applying the Kernel and Pooling method. Additionally, it reduces the vector in efficient space with storing all the important feature. In this case, we don’t use RNN here because the sequence length can be large. Looking further, we don’t choose LSTM here because the text content is not greater than 5000 words, therefore, we don’t need to use forgetting and updating technique.

CNN model requires training data for training weights and validation for checking its performance. Each input images passes through a series of convolution layers with filters (Kernels) : Convolution layers, Pooling layers and Fully-connected layer (FC) which is applying the sigmoid function. The sigmoid function is used to classify an object with a probabilistic value which turns out as 0 or 1 for binary classification.

Here we can see a simple CNN model used for binary classification.



The Convolution + Max Pooling layers act as feature extractors from the input image while a fully connected layer acts as a classifier. In the above image figure, on receiving an image as input, the network will assign assigns the highest probability for it and predict which class the input image belongs to. The operations listed above are the basic building blocks of every Convolutional Neural Network.

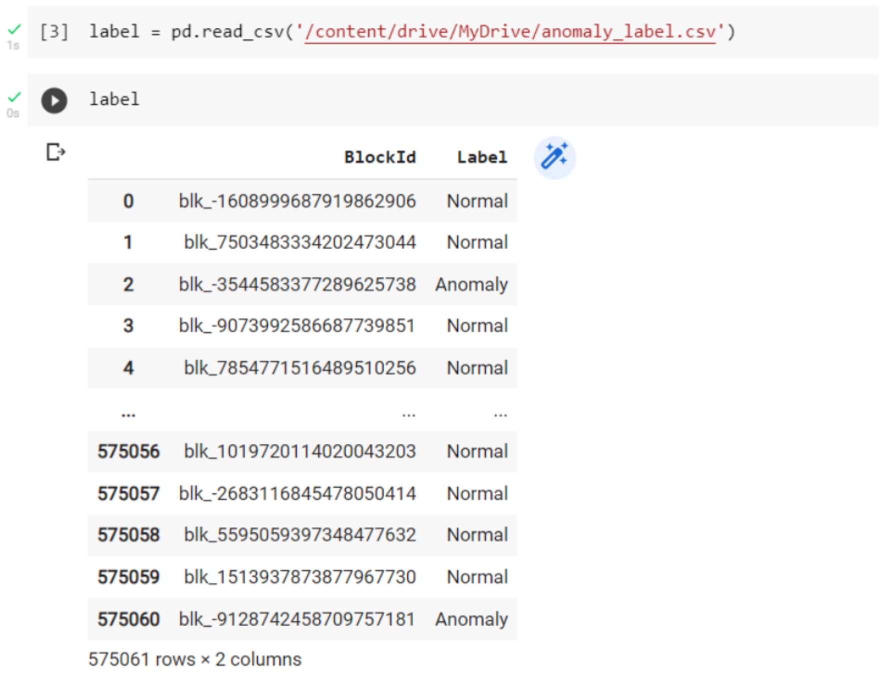

3n inner representation of the text
softmax outputs for the 6 classes
six
dead
as
suicide
bombing
hits
Somalia
#terroratt
ack
m x k pre-trained word embedding representation of text
convolutional layer with stacked filters of different sizes
max-pooling layer
Fully-connected layer with softmax outputs

We use PyTorch's built-in embedding layer, which learns embeddings directly during model training. When we do dot product of vectors representing text, they might turn out zero even when they belong to same class but if you do dot product of those embedded word vectors to find similarity between them then you will be able to find the interrelation of words for a specific class. Then, we slide the filter/ kernel over these embeddings to find convolutions and these are further dimensionally reduced in order to reduce complexity and computation by the Max Pooling layer. Lastly, we have the fully connected layers and the activation function on the outputs that will give values for each class.

Padding: Padding is a term relevant to convolutional neural networks as it refers to the amount of 0 added in the last to make vector equal. when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero.

Experiment:

1. Read label data

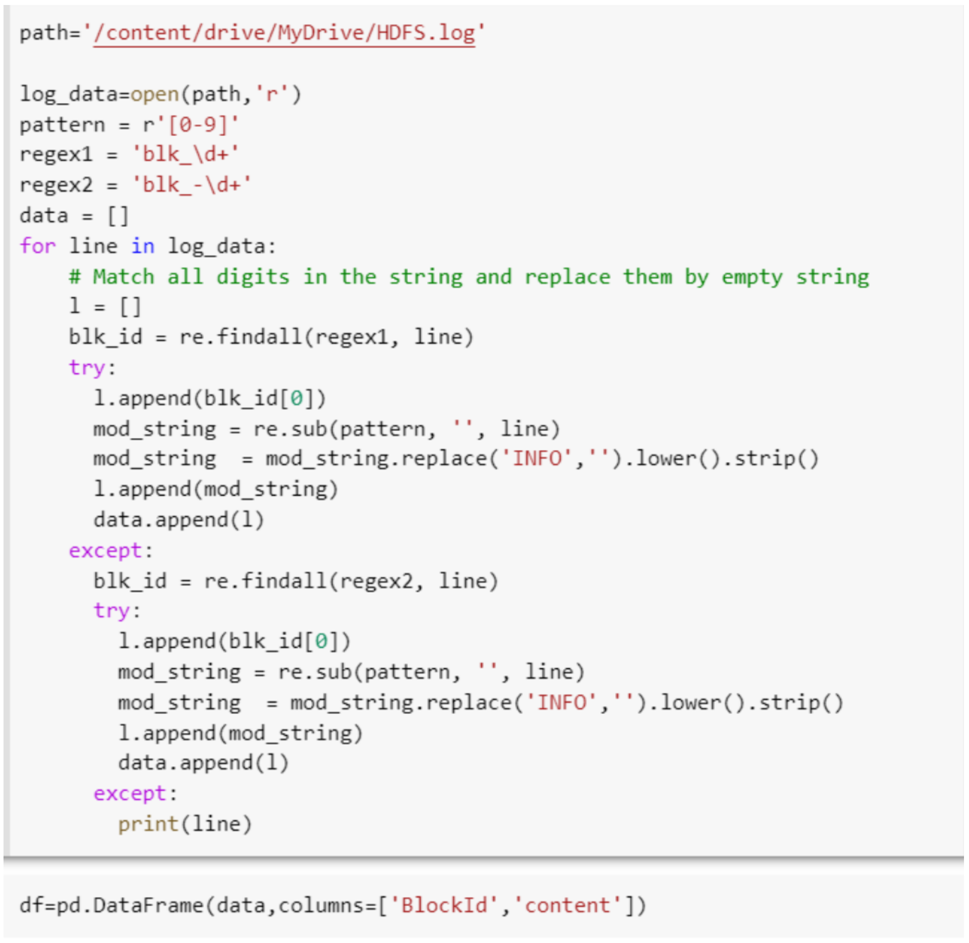


```
[3] label = pd.read_csv('/content/drive/MyDrive/anomaly_label.csv')
```

```
label
```

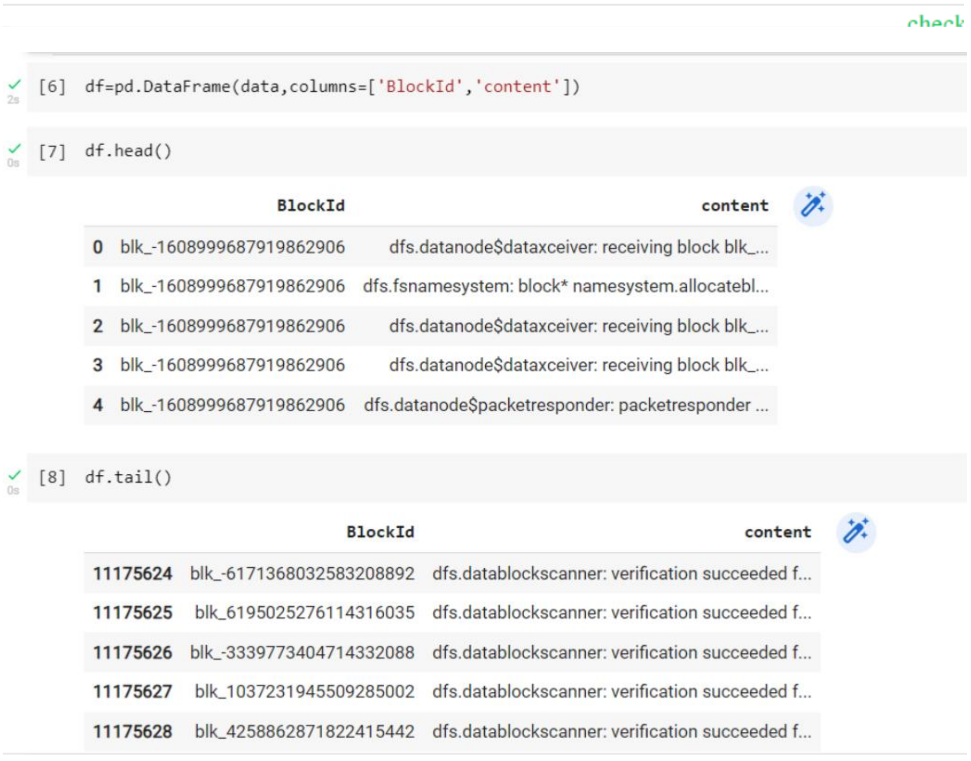
```
                         BlockId    Label
0       blk_-1608999687919862906   Normal
1        blk_7503483334202473044   Normal
2       blk_-3544583377289625738  Anomaly
3       blk_-9073992586687739851   Normal
4        blk_7854771516489510256   Normal
...                          ...      ...
575056   blk_1019720114020043203   Normal
575057  blk_-2683116845478050414   Normal
575058   blk_5595059397348477632   Normal
575059   blk_1513937873877967730   Normal
575060  blk_-9128742458709757181  Anomaly
575061 rows × 2 columns
```

2. **HDFS log data parsing:**
   Integer have no meaning in the text data, that’s why I am removing that.
   We are fetching the content by removing the INFO because it’s came in each sentence**.**



```python
path='/content/drive/MyDrive/HDFS.log'

log_data=open(path,'r')
pattern = r'[0-9]'
regex1 = 'blk_\d+'
regex2 = 'blk_-\d+'
data = []
for line in log_data:
    # Match all digits in the string and replace them by empty string
    l = []
    blk_id = re.findall(regex1, line)
    try:
      l.append(blk_id[0])
      mod_string = re.sub(pattern, '', line)
      mod_string  = mod_string.replace('INFO','').lower().strip()
      l.append(mod_string)
      data.append(l)
    except:
      blk_id = re.findall(regex2, line)
      try:
        l.append(blk_id[0])
        mod_string = re.sub(pattern, '', line)
        mod_string  = mod_string.replace('INFO','').lower().strip()
        l.append(mod_string)
        data.append(l)
      except:
        print(line)
```

```python
df=pd.DataFrame(data,columns=['BlockId','content'])
```



check

```
[6] df=pd.DataFrame(data,columns=['BlockId','content'])
```

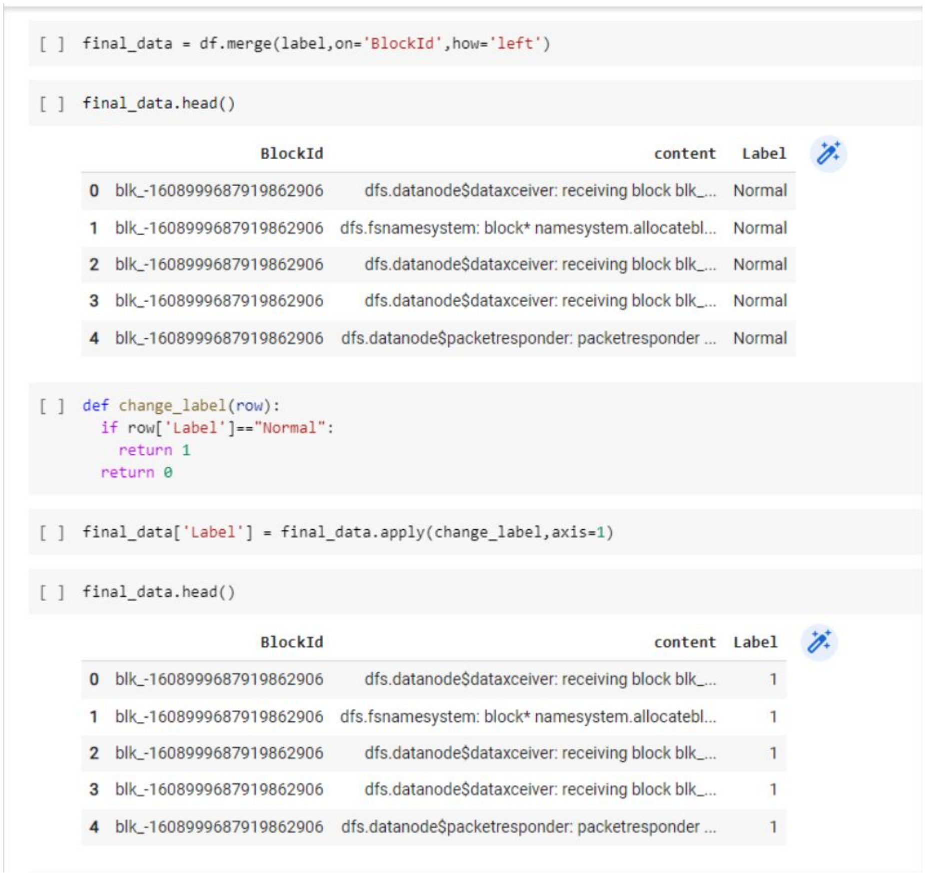
```
[7] df.head()
```

```
                  BlockId                                           content
0  blk_-1608999687919862906      dfs.datanode$dataxceiver: receiving block blk_...
1  blk_-1608999687919862906  dfs.fsnamesystem: block* namesystem.allocatebl...
2  blk_-1608999687919862906      dfs.datanode$dataxceiver: receiving block blk_...
3  blk_-1608999687919862906      dfs.datanode$dataxceiver: receiving block blk_...
4  blk_-1608999687919862906  dfs.datanode$packetresponder: packetresponder ...
```

```
[8] df.tail()
```

```
                          BlockId                                      content
11175624  blk_-6171368032583208892  dfs.datablockscanner: verification succeeded f...
11175625   blk_6195025276114316035  dfs.datablockscanner: verification succeeded f...
11175626  blk_-3339773404714332088  dfs.datablockscanner: verification succeeded f...
11175627   blk_1037231945509285002  dfs.datablockscanner: verification succeeded f...
11175628   blk_4258862871822415442  dfs.datablockscanner: verification succeeded f...
```

3. **Merging Label with log datasets, convert label to numeric binary.**
   Merging the data with labelled data by using block ID.
   Converting the Label in integer value by Normal reply by 1 and Anomy by 0



```
[ ] final_data = df.merge(label,on='BlockId',how='left')
```

```
[ ] final_data.head()
```

	BlockId	content	Label
0	blk_-1608999687919862906	dfs.datanode$dataxceiver: receiving block blk_...	Normal
1	blk_-1608999687919862906	dfs.fsnamesystem: block* namesystem.allocatebl...	Normal
2	blk_-1608999687919862906	dfs.datanode$dataxceiver: receiving block blk_...	Normal
3	blk_-1608999687919862906	dfs.datanode$dataxceiver: receiving block blk_...	Normal
4	blk_-1608999687919862906	dfs.datanode$packetresponder: packetresponder ...	Normal

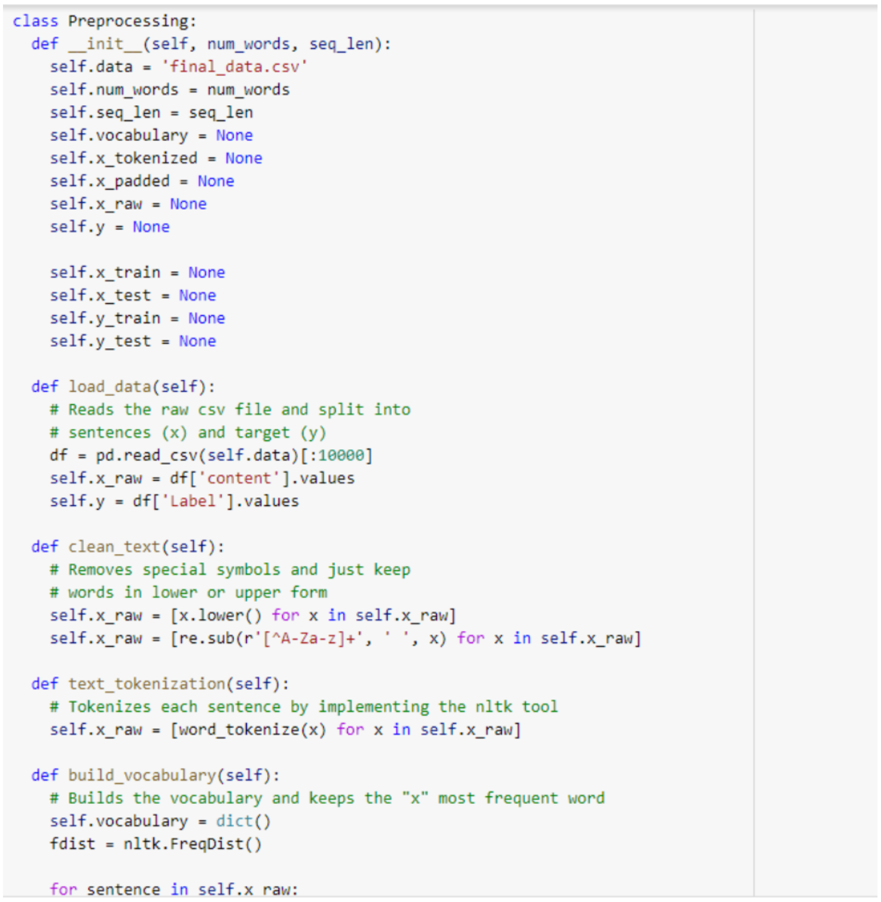
```
[ ] def change_label(row):
      if row['Label']=="Normal":
        return 1
      return 0
```

```
[ ] final_data['Label'] = final_data.apply(change_label,axis=1)
```

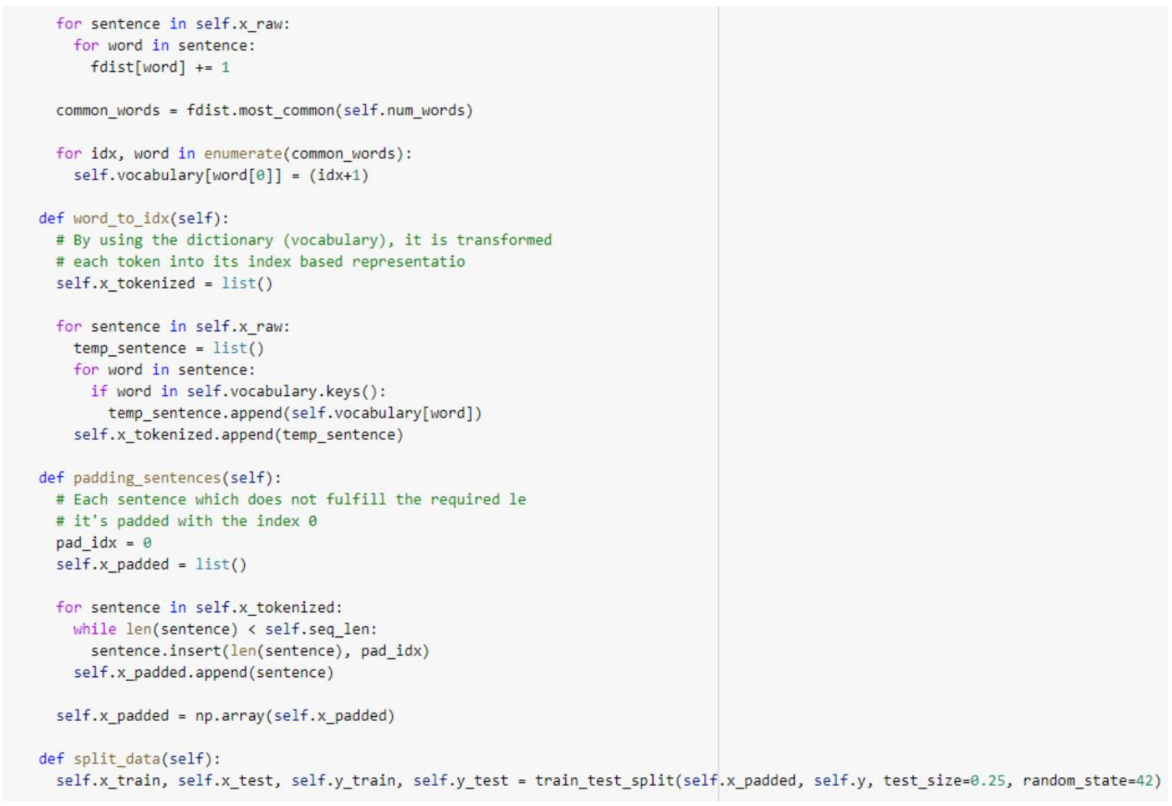
```
[ ] final_data.head()
```

	BlockId	content	Label
0	blk_-1608999687919862906	dfs.datanode$dataxceiver: receiving block blk_...	1
1	blk_-1608999687919862906	dfs.fsnamesystem: block* namesystem.allocatebl...	1
2	blk_-1608999687919862906	dfs.datanode$dataxceiver: receiving block blk_...	1
3	blk_-1608999687919862906	dfs.datanode$dataxceiver: receiving block blk_...	1
4	blk_-1608999687919862906	dfs.datanode$packetresponder: packetresponder ...	1

4. **Pre-processing for CNN**: Pre-processing for CNN: The data is loaded and cleaned, then tokenized into integer indices. Padding is applied to ensure all sequences have equal length. Finally, the dataset is split into training and testing sets.

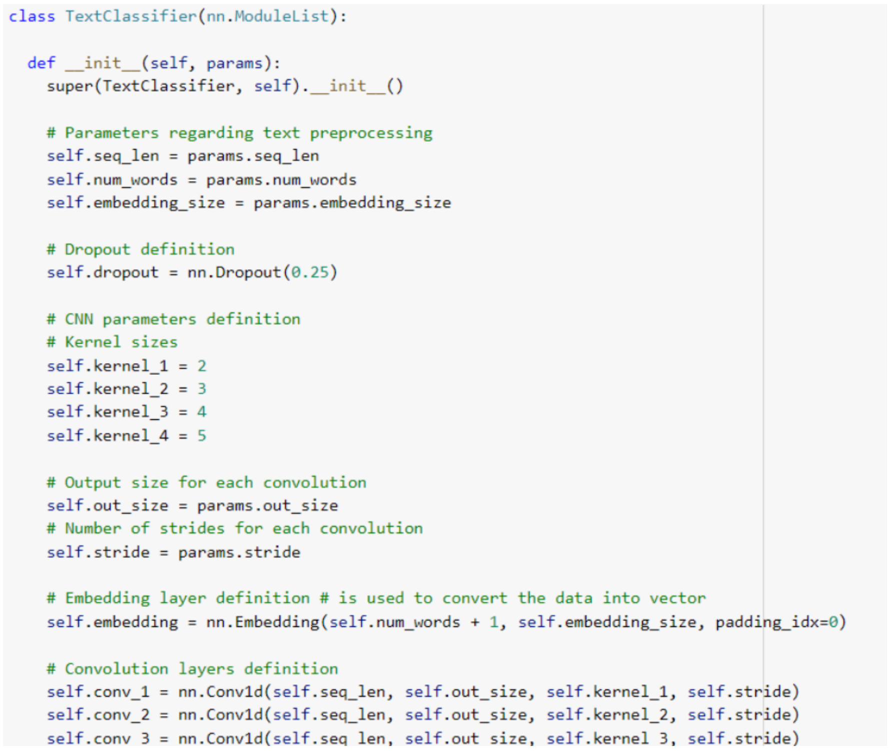


```python
class Preprocessing:
  def __init__(self, num_words, seq_len):
    self.data = 'final_data.csv'
    self.num_words = num_words
    self.seq_len = seq_len
    self.vocabulary = None
    self.x_tokenized = None
    self.x_padded = None
    self.x_raw = None
    self.y = None

    self.x_train = None
    self.x_test = None
    self.y_train = None
    self.y_test = None

  def load_data(self):
    # Reads the raw csv file and split into
    # sentences (x) and target (y)
    df = pd.read_csv(self.data)[:10000]
    self.x_raw = df['content'].values
    self.y = df['Label'].values

  def clean_text(self):
    # Removes special symbols and just keep
    # words in lower or upper form
    self.x_raw = [x.lower() for x in self.x_raw]
    self.x_raw = [re.sub(r'[^A-Za-z]+', ' ', x) for x in self.x_raw]

  def text_tokenization(self):
    # Tokenizes each sentence by implementing the nltk tool
    self.x_raw = [word_tokenize(x) for x in self.x_raw]

  def build_vocabulary(self):
    # Builds the vocabulary and keeps the "x" most frequent word
    self.vocabulary = dict()
    fdist = nltk.FreqDist()

    for sentence in self.x_raw:
```



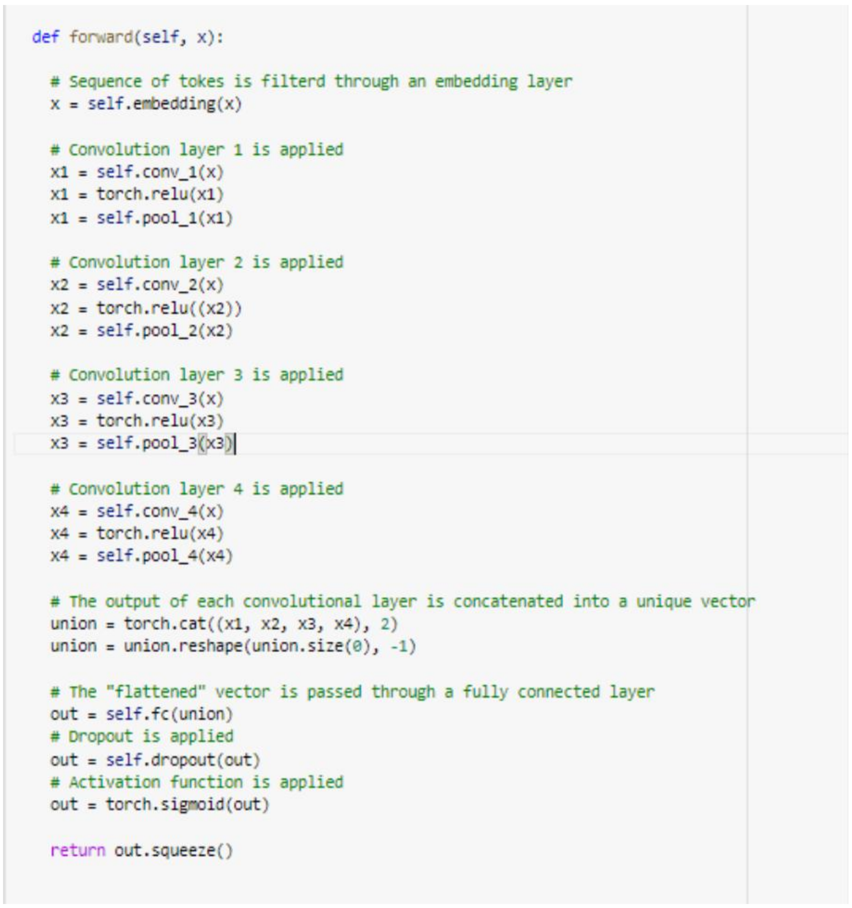
```python
    for sentence in self.x_raw:
      for word in sentence:
        fdist[word] += 1

    common_words = fdist.most_common(self.num_words)

    for idx, word in enumerate(common_words):
      self.vocabulary[word[0]] = (idx+1)

  def word_to_idx(self):
    # By using the dictionary (vocabulary), it is transformed
    # each token into its index based representatio
    self.x_tokenized = list()

    for sentence in self.x_raw:
      temp_sentence = list()
      for word in sentence:
        if word in self.vocabulary.keys():
          temp_sentence.append(self.vocabulary[word])
      self.x_tokenized.append(temp_sentence)

  def padding_sentences(self):
    # Each sentence which does not fulfill the required le
    # it's padded with the index 0
    pad_idx = 0
    self.x_padded = list()

    for sentence in self.x_tokenized:
      while len(sentence) < self.seq_len:
        sentence.insert(len(sentence), pad_idx)
      self.x_padded.append(sentence)

    self.x_padded = np.array(self.x_padded)

  def split_data(self):
    self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(self.x_padded, self.y, test_size=0.25, random_state=42)
```

5. **Increasing the size of kernel** as we are moving in the layer we decrease the vector as soon as possible, then we can use fully connected layer to calculate the final output.



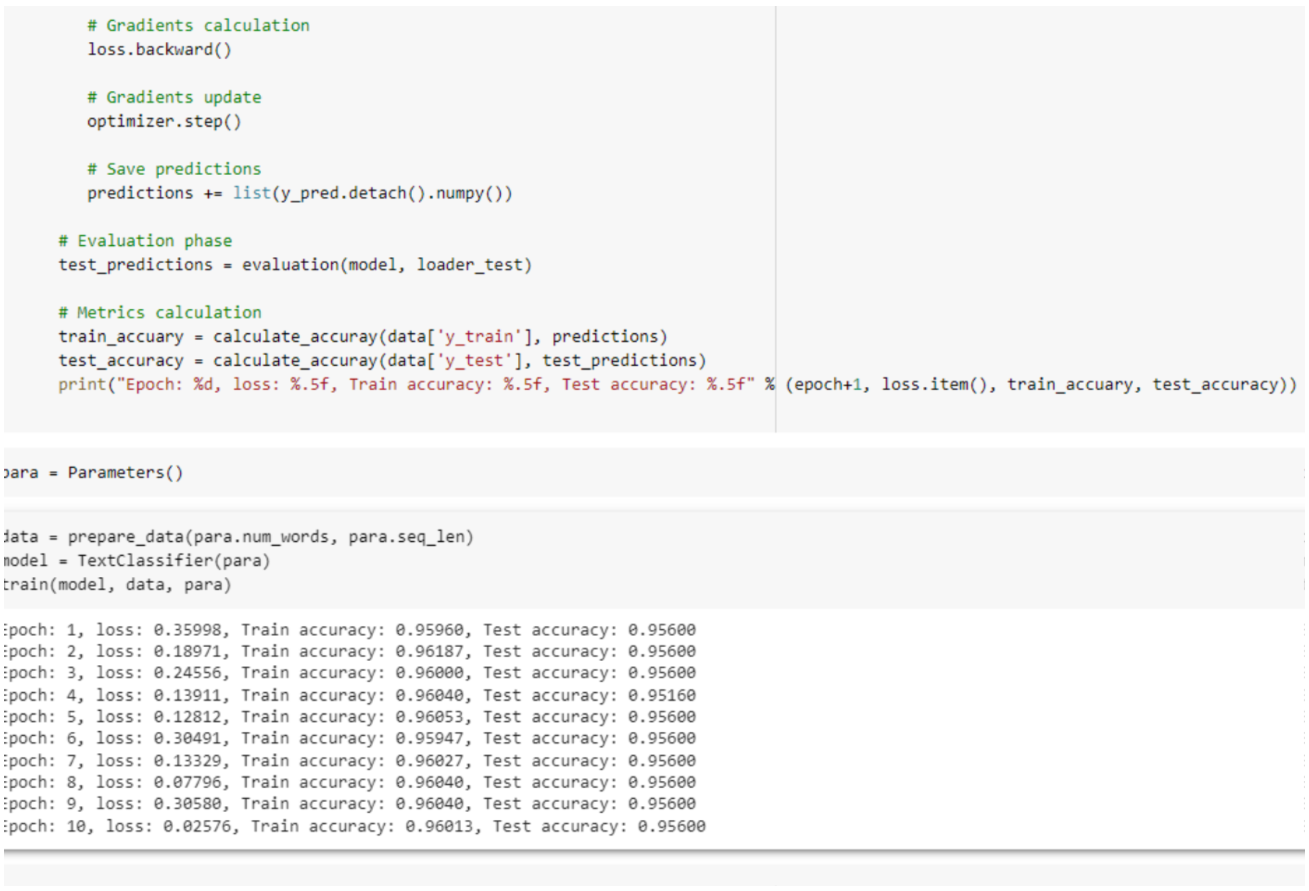
```
class TextClassifier(nn.ModuleList):

  def __init__(self, params):
    super(TextClassifier, self).__init__()

    # Parameters regarding text preprocessing
    self.seq_len = params.seq_len
    self.num_words = params.num_words
    self.embedding_size = params.embedding_size

    # Dropout definition
    self.dropout = nn.Dropout(0.25)

    # CNN parameters definition
    # Kernel sizes
    self.kernel_1 = 2
    self.kernel_2 = 3
    self.kernel_3 = 4
    self.kernel_4 = 5

    # Output size for each convolution
    self.out_size = params.out_size
    # Number of strides for each convolution
    self.stride = params.stride

    # Embedding layer definition # is used to convert the data into vector
    self.embedding = nn.Embedding(self.num_words + 1, self.embedding_size, padding_idx=0)

    # Convolution layers definition
    self.conv_1 = nn.Conv1d(self.seq_len, self.out_size, self.kernel_1, self.stride)
    self.conv_2 = nn.Conv1d(self.seq_len, self.out_size, self.kernel_2, self.stride)
    self.conv_3 = nn.Conv1d(self.seq_len, self.out_size, self.kernel_3, self.stride)
```

6. **Supply CNN on data**



```
  def forward(self, x):

    # Sequence of tokes is filterd through an embedding layer
    x = self.embedding(x)

    # Convolution layer 1 is applied
    x1 = self.conv_1(x)
    x1 = torch.relu(x1)
    x1 = self.pool_1(x1)

    # Convolution layer 2 is applied
    x2 = self.conv_2(x)
    x2 = torch.relu((x2))
    x2 = self.pool_2(x2)

    # Convolution layer 3 is applied
    x3 = self.conv_3(x)
    x3 = torch.relu(x3)
    x3 = self.pool_3(x3)

    # Convolution layer 4 is applied
    x4 = self.conv_4(x)
    x4 = torch.relu(x4)
    x4 = self.pool_4(x4)

    # The output of each convolutional layer is concatenated into a unique vector
    union = torch.cat((x1, x2, x3, x4), 2)
    union = union.reshape(union.size(0), -1)

    # The "flattened" vector is passed through a fully connected layer
    out = self.fc(union)
    # Dropout is applied
    out = self.dropout(out)
    # Activation function is applied
    out = torch.sigmoid(out)

    return out.squeeze()
```

7. **Train and complete train CNN**



```
        # Gradients calculation
        loss.backward()

        # Gradients update
        optimizer.step()

        # Save predictions
        predictions += list(y_pred.detach().numpy())

    # Evaluation phase
    test_predictions = evaluation(model, loader_test)

    # Metrics calculation
    train_accuary = calculate_accuray(data['y_train'], predictions)
    test_accuracy = calculate_accuray(data['y_test'], test_predictions)
    print("Epoch: %d, loss: %.5f, Train accuracy: %.5f, Test accuracy: %.5f" % (epoch+1, loss.item(), train_accuary, test_accuracy))
```

```
ara = Parameters()
```

```
ata = prepare_data(para.num_words, para.seq_len)
odel = TextClassifier(para)
rain(model, data, para)
```

```
poch: 1, loss: 0.35998, Train accuracy: 0.95960, Test accuracy: 0.95600
poch: 2, loss: 0.18971, Train accuracy: 0.96187, Test accuracy: 0.95600
poch: 3, loss: 0.24556, Train accuracy: 0.96000, Test accuracy: 0.95600
poch: 4, loss: 0.13911, Train accuracy: 0.96040, Test accuracy: 0.95160
poch: 5, loss: 0.12812, Train accuracy: 0.96053, Test accuracy: 0.95600
poch: 6, loss: 0.30491, Train accuracy: 0.95947, Test accuracy: 0.95600
poch: 7, loss: 0.13329, Train accuracy: 0.96027, Test accuracy: 0.95600
poch: 8, loss: 0.07796, Train accuracy: 0.96040, Test accuracy: 0.95600
poch: 9, loss: 0.30580, Train accuracy: 0.96040, Test accuracy: 0.95600
poch: 10, loss: 0.02576, Train accuracy: 0.96013, Test accuracy: 0.95600
```

Conclusion:

Working with HDFS dataset using CNN model above, we are trying to decrease our loss, by increasing the epoch number, but we see that after 1 epoch, the model converges. As we move forward, the train and test accuracy remain same, so we don't need run for 10 epochs, we can only run with 1 epoch.

References:

Johnathan Johnson, 2020. Anomaly Detection with Machine Learning: An Introduction.

Matthias Reif, Markus Goldstein, Armin Stahl, Thomas M. Breuel, 2009. Anomaly Detection by Combining Decision Trees and Parametric Densities.

Salima Omar, Asri Ngadi, Hamid H. Jebur, 2013. Machine Learning Techniques for Anomaly Detection.

Vijay Choubey, 2020. Text classification using CNN.