DEEP CONVOLUTIONAL NEURAL NETWORKS FOR TEXT CLASSIFICATION WITH DROPOUT AND RESIDUAL CONNECTION

Submitted by

Sabreena Job (Reg No: 31119026) **Shilpa S** (Reg No: 31119027) **Tarun Sharma** (Reg No: 31119028)
Thomas J Varghese (Reg No: 31119029)
Vishnu N (Reg No: 31119030)

In partial fulfillment of the requirements for the award of Master Of Science
in Computer Science with Specialization in Data Analytics Of

Cochin University of Science and Technology, Kochi

Conducted by

Indian Institute of Information Technology and Management-Kerala
Technopark Campus, Thiruvanathapuram-695 581

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to **Dr. Manoj Kumar T.K**, our project guide for providing us with the necessary facilities for the completion of this project. We are thankful for the valuable discussions we had at each phase of the project and for being a very supportive and encouraging project guide. We would also like to express our sincere and heartfelt gratitude to all people who were in one way or the other involved in my project. A lot of gratitude is reserved for the Director of the Institute for facilitating and nurturing an environment where I have been able to undertake this work. We would like to express my sincere thanks to all friends and classmates whose suggestions and creative criticism.

Names	**Registration Number**
Sabreena Job	**31119026**
Shilpa S	**31119027**
Tarun Sharma	**31119028**
Thomas J Varghese	**31119029**
Vishnu N	**31119030**

TABLE OF CONTENT

INTRODUCTION

For text processing we need to associate categories to parts of the text, and convert it to some other form which preserves all or part of the content. The level of granularity of this processing ranges from individual characters to sub word units.

Even though the use of (deep) neural networks in NLP has shown good results for many tasks, they are still not able to reach to the level of outperforming the state of art by a large margin.

CNN is very useful for computer vision. Introduction of continuous representations of words is considered as an important step in many NLP approaches where words were considered as basic units. But the best representation of the sequence of words is still unknown. This is mainly because same sentences may have local as well as long-range dependencies. Hence a change in the approach was made and now a sentence was considered as a sequence of tokens and to process them with RNN. The most successful RNN variant is LSTM. But LSTMs are also generic learning machines used for sequence processing which are lacking task specific structure.

We know that a fully connected one hidden layer neural network can learn any real-valued function. There is also the fact that better results can be obtained with a deep problem specific architecture which develops hierarchical representations. Hence the search space is heavily constrained with and efficient solutions can be learned with gradient descent.

In NLP, major challenge lies in developing deep architectures which are able to learn hierarchical representations of whole sentences. Deep architectures up to 29 layers were used for achieving this purpose. It was found that the proposed deep convolutional network shows significantly better results than previous convnets approach.

1. Natural Language Processing

Natural Language Processing is a process of communicating with an intelligent system using the natural language and extracting meaningful data from it to finally represent the data in another form. The two components of NLP are Natural Language Understanding and Natural Language Generation. Natural Language Processing is applied in Sentimental Analysis, language translation applications, Speech Recognition and Information extraction etc.

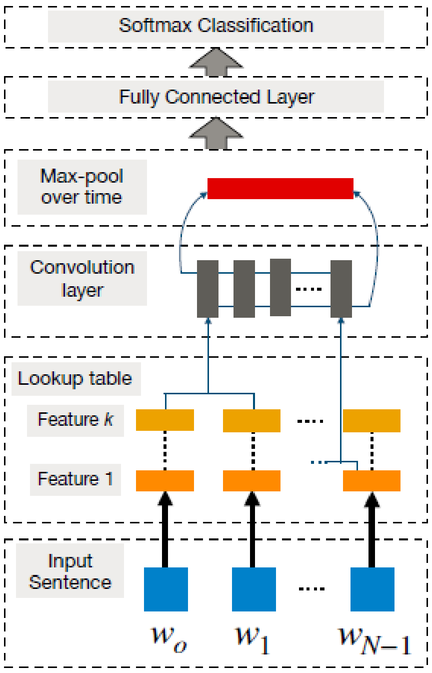
Neural networks have been used in NLP for many tasks which resulted in yielding good outputs. Here NLP has helped in developing deep architectures intelligent enough to learn the hierarchical representations of whole sentences. We have used NLP tasks like word embedding in this project.

2. Deep Learning

Deep Learning is a subset of machine learning that is concerned with algorithms inspired by structure and function of the brain called artificial neural networks .It includes statistics and predictive modeling. Deep learning algorithms attempt to learn multiple levels of representation of increasing complexity/abstraction. Named entity recognition (NER), part of speech (POS) tagging or sentiment analysis are some of the problems where neural network models have outperformed traditional approaches.

3. Deep Learning in Natural Language Processing

Deep Learning is an extension of neural networks and is widely used for vision based classification. . Deep learning tasks learn the features from natural language required by the model. We make the use of deep learning in NLP to tokenize sentences into words and further transforming them into a word embedding matrix followed by max-pooling in this project.



4. Recurrent Neural Network

RNNs are designed to take a series of input with no predetermined limit on size. A single input item from the series is related to others and likely has an influence on its neighbors.

They are a type of Neural Network where the output from the previous step is fed as input to the current step. Traditionally in neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.

RNNs are sketched for recognizing a data's sequential characteristics and use patterns to predict the next likely scenario.

While RNNs learn similarly while training, in addition, they remember things learnt from prior input while generating output.

- **Formula for calculating current state:**

$$h_t = f(h_{t-1}, x_t)$$

Where,

h_t->current state

h_{t-1}->previous state

x_t->input state

- **Formula for calculating output:**

$$y_t = W_{hy}h_t$$

Where,

Y_t ->output

W_{hy} ->weight at output layer

5. Convolutional Neural Network

Convolutional Neural Networks (**ConvNets** or **CNNs**) are a category of Neural Networks that is proven to be very effective in areas such as image recognition and classification. A Convolutional Neural Network is a deep learning algorithm which can take an input image, assign importance to various objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. It enables us to encode certain properties in the image architecture to recognize specific elements in the image. CNN's are widely used in image and video recognition.

Four main operations of ConvNets:

- Convolution
- Non Linearity(ReLu)
- Pooling or Sub sampling
- Classification(Fully connected layer)

CNN can also be used in text classification. An increase in the number of convolutional layers i.e., a new architecture VDCNN(Very Deep Convolutional Neural Network) which operates at character level directly using small convolutions and pooling increases the performance of the model with increase in depth.

6. Max Pooling and its advantages

One of the key aspects of CNN is Pooling Layers. This layers subsample the input. The most prevalent way is by applying a max operation to the result of each filter. A key aspect of Pooling is that it provides a fixed size output matrix, which is required by classification. In this way it helps us to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier. Also Pooling also reduces the output dimensionality but keeps the most salient information. Max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. Max Pooling reduces the number of parameters within the model this is called down-sampling or sub-sampling. Max pooling is done by applying a *max filter* to the non-overlapping sub regions of the initial representation. It generalizes the results from a convolutional filter - making the detection of features invariant to scale or orientation changes.

Advantages

1. Max pooling extracts the most relevant features like edges, points etc. and is better for extracting the extreme features. It selects the brightest pixels from the image. It is very useful when the background of an image is dark and we only want the lighter pixels of the image.

2. Max pooling is usually performs with overlapping regions as without overlaps, pooling operation may lose important information regarding the location of the object.

3.Max pooling is most widely used pooling method as it reduces the number of parameters within the model and also generalises the results from a convolutional filter, thus making the detection of features invariant to size or orientation changes.

7. LSTM Networks

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. In LSTM it also has this chain like structure, but the repeating module has a different structure. LSTM can be used to solve problems faced by the RNN model such as Long term dependency problem in RNNs and Vanishing Gradient & Exploding Gradient.

8. Shortcut Connections

Shortcut connections are the extra connections which are present between the nodes in different layers of the neural network. These are the connections which skips one or more layers. These connections also help in traversing the information at a faster rate in deep neural networks.

The most important problem regarding the deeper networks is the degradation problem. Due to this problem, there is a reduction in accuracy of the network when the depth of the network is increasing and when the depth reaches a maxima stage. We can see that if there is some accuracy in the shallow network then the deeper networks can do at least as well as the shallow networks.

For this purpose extra layers are added in the network. These extra layers serve as the identity mappings and give the same result as the shallow network.

Because of the degradation problem and the inability of the multiple non-linear layers to learn identity mappings deep residual learning framework was developed.

In deep residual learning framework the layers themselves develops a residual mapping. We also found that optimization is easier in residual mapping than the original mapping. Due to the shortcut connections developed in deep residual networks, degradation problem is solved to a great extent and helps in better performance than the plain networks. The main advantage regarding the shortcut connections is that it does not add any extra parameters and computational complexity to the network.

9. Temporal Convolutional Networks

Temporal Convolutional Networks (TCN) architecture is similar to RNN which can take a sequence of any length and map in to an output sequence of the same length. TCN uses a 1D fully-convolutional network architecture where each hidden layer is the same length as the input layer and zero padding of length (kernel size -1) is added to keep subsequent layers the same length as previous ones. In TCN there is no information leakage from future to past as they use casual convolutions, where an output at time t is convolved only with elements from time t and earlier in the previous layer.

Advantages

- Parallelism: In RNN predictions on future steps depends on the predictions from their predecessors where as in TCN since same filters are applied over all the layers , convolutions can be done in parallel i.e. long input sequence can be processed completely rather than processing it sequentially.
- Flexible Receptive field size: TCN can change its receptive field size. They can control memory size and they are easy to adapt in different domains.
- Low memory requirement for training: LSTM and GRU may use a lot of memory for storing their partial results whereas in TCN, since filters are shared across the layers with back propagation depending only on network depth, they require less memory for training.
- Capturing local information: along with temporal information we can also capture local information using convolutional operation.

10. Text Classification

Text classification is the process of assigning a set of pre- defined categories to documents according to its content and these documents can be a web page or an article etc. and has many applications like sentiment analysis, spam filtering, topic labeling, email routing etc. It is an example of supervised machine learning task. Text classification is very important as majority of the text data is unstructured and because of this messy nature, analyzing and sorting data is very difficult and time consuming and therefore by using text classifiers we can structure data and get effective information thereby saving time and helps to enhance business decisions in a cost effective way. It is effective, scalable, consistent and real time.

Text classification with machine learning based system

By using pre labeled training data, a machine learning algorithm can find associations between different pieces of text. One of the important steps in this system is the conversion of text into vectors so that we can perform operations on it. Most frequently used approaches for text classification are:

- Bag-of- words (BOW) which counts how many times a word appears in a document. If the dataset is small and context is domain specific then they work better than word embedding.

- N-grams model- They change the scope of vocabulary and allows the bag-of-words to capture a little bit more meaning from the document. It is a more sophisticated approach to create a vocabulary of grouped words.

- TF-IDF: It shows how important a word is to a document.

  Term Frequency: - The number of times a word appears in a document divided by the total number of words in a document.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

  Inverse Data Frequency: - The log of the number of documents divided by the number of documents that contain the word w

$$idf(w) = log(\frac{N}{df_t})$$

- Word2Vec- It is a two-layer neural network that converts text into vectors so that we can perform mathematical operations on it. This is typically done in the pre-processing stage and takes the semantic meaning of the word. The main purpose of this model is to group the vectors of similar words together in vector space i.e. it detects similarities mathematically.

Literature Review

The Project aims to puts forward a new approach to build deep networks for text processing. The Architecture presented here is a VDCNN which operates directly at character level and uses only small convolutions and pooling operations. The paper presents a new approach for document classification using deep neural networks.

It has been able to show the architectures improvement over the other state of the art by using up to 29 layered CNNs. This architecture is used to show that it can work very well on big datasets even for small depths. As the overall network depth is increased the architecture's performance is also improved. Another aspect is that the paper aims is to show that Max-pooling performs better than other pooling types. In this way the primary aim is to show that this model can outperform all other state of the art ConvNets. Also, we need to reduce the memory footprint of our network. In order to achieve the same, two design rules were implemented. These rules are:

(i) For the same temporal resolution, the layers have the same number of feature maps.

(ii) When the temporal resolution is halved, the feature maps are doubled.

These two design rules helped in reducing the memory footprint of the network.

ResNet shortcut connections of identity and 1*1 convolutions were also taken.

The idea of using Deep Convolutional Nets for Text Processing is to harness the potential of deep networks which have been extremely successful in Computer Vision and use them for Natural Language tasks.

ABSTRACT

VDCNN architecture was introduced for the implementation of text processing in neural networks. This was mainly because of the shallow nature of the previously used neural networks such as recurrent neural networks mainly Long Short Term Memory (LSTM). We followed word level CNN and character level CNN although our VDCNN architecture operates at the character level which uses only small convolutions and pooling operations. Further we increased the depth of the layers from 9 to 17 and we were able to show that the performance of the model increases with depth.

MODEL DESCRIPTION

Training deep neural networks with tens of layers is challenging as they can be sensitive to the initial random weights and configuration of the learning algorithm. One possible reason for this difficulty is the distribution of the inputs to layers deep in the network may change after each mini-batch when the weights are updated.

Batch normalization is a technique used for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

The goal of Batch Normalization is to achieve a stable distribution of activation values throughout training, and in our experiments we apply it before the nonlinearity since that is where matching the first and second moments is more likely to result in a stable distribution.

In a batch-normalized model, we have been able to achieve a training speedup from higher learning rates, with no ill side effects.

Deep neural networks can be quite sensitive to the technique used to initialize the weights prior to training.

The stability to training brought by batch normalization can make training deep networks less sensitive to the choice of weight initialization method.

Removing dropout from modified BN-Inception speeds up training, without increasing over fitting.

Batch normalization also sometimes reduces generalization error and allows dropout to be omitted, due to the noise in the estimate of the statistics used to normalize each variable.

Very deep models involve the composition of several functions or layers. The gradient tells how to update each parameter, under the assumption that the other layers do not change. In practice, we update all of the layers simultaneously. We refer to the change in the distributions of internal nodes of a deep network, in the course of training, as Internal Covariate Shift.

Batch normalization provides an elegant way of re parameterizing almost any deep network. The re parameterization significantly reduces the problem of coordinating updates across many layers.

By whitening the inputs to each layer, we would take a step towards achieving the fixed distributions of inputs that would remove the ill effects of the internal covariate shift.

Batch normalization not only acts to standardize only the mean and variance of each unit in order to stabilize learning, but also allows the relationships between units and the nonlinear statistics of a single unit to change.

Batch normalization can have a dramatic effect on optimization performance, especially for convolutional networks and networks with sigmoidal nonlinearities.

By only using Batch Normalization, we match the accuracy of Inception in less than half the number of training steps.

We find that when applied to very deep networks of RNNs on large data sets, the variant of Batch Norm we use substantially improves final generalization error in addition to accelerating training.

The complete training process of a neural network involves two steps.

1. Forward Propagation

Images are fed into the input layer in the form of numbers. These numerical values denote the intensity of pixels in the image.

2. Backward Propagation

Once the output is generated, the next step is to compare the output with the actual value. Based on the final output, and how close or far this is from the actual value (error), the values of the parameters are updated. The forward propagation process is repeated using the updated parameter values and new outputs are generated.

The hidden layers in CNN find features in the image. The convolutional neural network can be broken down into two parts:

The convolution layers: Extracts features from the input

The fully connected (dense) layers: Uses data from convolution layer to generate output.

Convolution is often represented mathematically with an asterisk * sign. If we have an input image represented as X and a filter represented with f, then the expression would be:

$Z = X * f$

If dimension of image = (n, n)

Dimension of filter = (f, f)

Dimension of output will be ((n-f+1) , (n-f+1))

Fully Connected Layer

Once the data is converted into a 1D array, it is sent to the fully connected layer. All of these individual values are treated as separate features that represent the image. The fully connected layer performs two operations on the incoming data – a linear transformation and a non-linear transformation.

We first perform a linear transformation on this data. The equation for linear transformation is:

Z = WT.X + b.

Here, X is the input, W is weight, and b (called bias) is a

constant. Non-Linear transformation

The linear transformation alone cannot capture complex relationships. Thus, we introduce an additional component in the network which adds non-linearity to the data. This new component in the architecture is called the activation function. Mathematical equation for activation function is given by:

f(x) = 1/(1+e^-x)

The range of a sigmoid function is between 0 and 1. This means that for any input value, the result would always be in the range (0, 1). A Sigmoid function is majorly used for binary classification problems and we will use this for both convolution and fully-connected layers.

Steps in forward propagation include:

Step 1: Load the input images in a variable (say X).

Step 2: Define (randomly initialize) a filter matrix. Images are convolved with the filter.

$Z1 = X * f$

Step 3: Apply the Sigmoid activation function on the result.

$A = \text{sigmoid}(Z1)$

Step 4: Define (randomly initialize) weight and bias matrix. Apply linear transformation on the values.

$Z2 = WT.A + b$

Step 5: Apply the Sigmoid function on the data. This will be the final output.

$O = \text{sigmoid}(Z2)$.

Backward propagation

During the forward propagation process, we randomly initialized the weights, biases and filters. These values are treated as parameters from the convolutional neural network algorithm. In the backward propagation process, the model tries to update the parameters such that the overall predictions are more accurate. For updating these parameters, we use the gradient descent technique.

Based on the value of the gradient, we can determine the updated parameter values. When the slope is negative, the value of the parameter will be increased, and when the slope is positive, the value of the parameter should be decreased by a small amount.

Equation for updating the parameter values is given by:

new parameter = old parameter - (learning rate * gradient of parameter)

The learning rate is a constant that controls the amount of change being made to the old value. The slope or the gradient determines the direction of the new values, that is, should the values be increased or decreased. So, we need to find the gradients, that is, change in error with respect to the parameters in order to update the parameter values.

Backward propagation in fully connected layer follows the equation

$\partial E/\partial W = \partial E/\partial O.\ \partial O/\ \partial Z2.\ \partial z/\partial W$. The value of derivatives can be found out by the following steps:

1. Change in error with respect to output

Suppose the actual values for the data are denoted as y’ and the predicted output is represented as O. Then the error would be given by this equation:

$E = (y' - O)2/2$

If we differentiate the error with respect to the output, we will get the following equation:

$\partial E/\partial O = -(y'-O)$

2. Change in output with respect to Z2 (linear transformation output)

To find the derivative of output O with respect to Z2, we must first define O in terms of Z2.

$f(x) = 1/(1+e\hat{}-x)$

The derivative of this function comes out to be:

$f'(x) = (1+e-x)-1[1-(1+e-x)-1]$

$f'(x) = \text{sigmoid}(x)(1-\text{sigmoid}(x))$

∂O/∂Z2 = (O)(1-O).

3. Change in Z2 with respect to W (Weights)

The value Z2 is the result of the linear transformation process. Here is the equation of Z2 in terms of weights:

Z2 = WT.A1 + b

On differentiating Z2 with respect to W, we will get the value A1 itself:

∂Z2/∂W = A1.

∂E/∂W = ∂E/∂O . ∂O/∂Z2. ∂Z2/∂W

∂E/∂W = -(y'-o) . sigmoid'. A1

The shape of ∂E/∂W will be the same as the weight matrix W. We can update the values in the weight matrix using the following equation:

W new = W old - lr*∂E/∂W

Change in Z2 with respect to A1

To find the value for ∂Z2/∂A1 , we need to have the equation for Z2 in terms of A1:

Z2 = WT.A1 + b

On differentiating the above equation with respect to A1, we get WT as the result:

∂Z2/∂A1 = WT.

Change in A1 with respect to Z1

The next value that we need to determine is ∂A1/∂Z1. Have a look at the equation of A1

A1 = sigmoid (Z1)

This is simply the sigmoid function. The derivative of Sigmoid would be:

∂A1/∂Z1 = (A1) (1-A1)

Change in Z1 with respect to filter f

Finally, we need the value for ∂Z1/∂f. Here’s the equation for Z1

Z1 = X * f

Differentiating Z with respect to X will simply give us X:

∂Z1/∂f = X

Architecture

Better performance can be achieved with very deep convolutional neural network (VDCNN) although standard and reusable architectures have not been adopted for classification tasks. There are benefits for hierarchical feature learning with VDCNN model. Key to this approach is embedding of individual characters rather than word embedding.

VDCNN worked well on small and large datasets. Max-pooling achieves better results than other sophisticated types of pooling.

The network begins with a lookup table, which generates the embeddings for the input text and stores them in a 2D tensor of size (f0,s). The number of input characters (s) is fixed to 1,024 while the embedding dimension (f0) is 16. The embedding dimension can be seen as the number of RGB channels of an image.

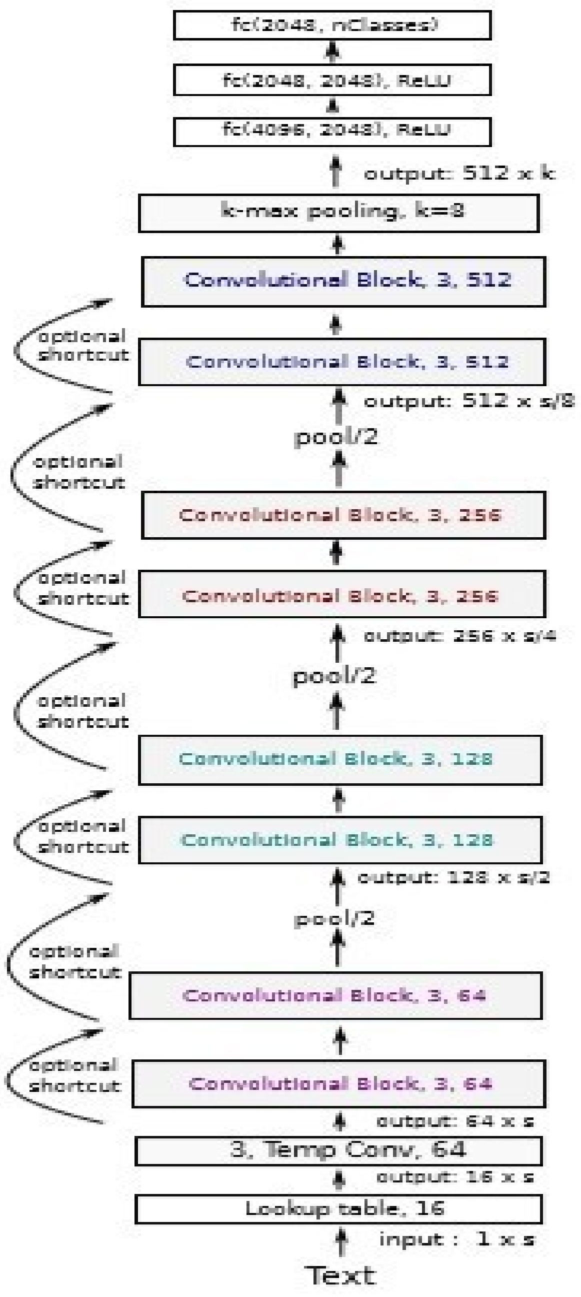
The following layer (3, Temp Convolution, 64) applies 64 temporal convolutions of kernel size 3, so the output tensor has size 64∗s. Its primary function is to fit the lookup table output with the modular network segment input composed by convolutional blocks. Each forenamed block is a sequence of two temporal convolutional layers, each one accompanied by a temporal batch normalization layer and a ReLU activation. Besides, the different network depths are obtained varying the number of convolutional blocks. As a convention, the depth of a network is given as its total number of convolutions. For instance, the architecture of depth 17 has two convolutional blocks of each level of feature maps, which results in 4 convolutional layers for each level.

Depth	9	17	29	49
Convolutional Block 512	2	4	4	6
Convolutional Block 256	2	4	4	10
Convolutional Block 128	2	4	10	16
Convolutional Block 64 2	4	10	16	
First Convolutional Layer	1	1	1	1

In this case we first apply one layer of 64 convolutions of size 3, followed by a stack of temporal convolutional blocks. There are 3 pooling operations resulting in 3 levels of 128, 56 and 512 feature maps.

The output is a tensor of size 512* Sd where Sd=S/2^p with p=3 number of down sampling operations. Padded input text of fixed size is used hence Sd is constant. But in case of variable size input, representation of the input text which depends on initial length s is provided by the convolutional encoder.

Also, in this work, we propose to create instead an architecture which uses many layers of small convolutions. This architecture is a temporal adaptation of the VGG network. For classification purpose temporal resolution of output of convolutional blocks is first down-sampled to fixed dimension. This was achieved by using k-max pooling. By this technique, network can extract the k most important features independently in position they occur in sentence. The number of neurons depends on the classification block.



Convolutional block

The convolutional block consists of sequence of 2 convolutional layers. Each one is followed by a temporal Batchnorm layer and ReLu activation.

BatchNorm impacts network training in a fundamental way: it makes the landscape of the corresponding optimization problem to be significantly smoother. This ensures, in particular, that the gradients are more predictive and thus allow for use of larger range of learning rates and faster network convergence.

Batch Renormalization extends batchnorm with a per-dimension correction to ensure that the activations match between the training and inference networks.

We add the BN transform immediately before the nonlinearity. We could have also normalized the layer inputs u, but since u is likely the output of nonlinearity, the shape of its distribution is likely to change during training and constraining its first and second moments would not eliminate the covariate shift.

Due to the very small parameters of very small convolutional filters in all layers we can steadily increase the depth of the network by adding more convolutional layer. For a mini batch of size m and feature maps of size s, sum and standard deviation for batchnorm algorithm are taken over m.s terms.

The pooling reduces temporal resolution by a factor of 2.The depths of our networks were taken as 9, 17, 29 and 49.It is obtained by summing the number of blocks with 64,128,256 and 512 filters.

Dataset Used

Dataset Name	**No Of Classes**	**Classification Task**
Ag_news	4	English news categorization
Amazon_review	2	Sentiment analysis
Amazon_review_polarity	2	Sentiment analysis
Dbpedia	14	Ontology classification
Sogou news	5	Chinese news categorization
Yahoo answers	10	Topic classification
Yelp review	2	Sentiment analysis
Yelp review polarity	2	Sentiment analysis

Libraries Used or popular in NLP

- nltk- It is an open source library which contains text processing libraries for tokenization, parsing, classification, stemming, tagging, and sematic reasoning.
- Spacy- It is used to pre-process text for deep learning or for embedding and helps to boost accuracy in text classification.
- argparse- To parse the arguments.
- pathlib- Provides with file system paths and makes it easy to access.
- h5py- It helps to store numerical data in an efficient way.
- datetime- It helps classes to work with date and time and finds duration of training.
- sys- It’s a system library used for system calls.
- custom_callbacks- A custom callback is a tool to customize the behavior of a keras model during training or evaluation.
- data_loader- Used for the bulk import or export of data.
- Keras- all are deep learning library
- tensorflow-deep learning library
- os- Basic system calls and os related issues are made easy using this.

Training

We basically did our testing on Tesla p100-pcie-12GB major: 6 minor: 0 memory clockrate (GHZ) 1.3285 Gpu. There are so many benchmark dataset used in the referred paper but due to unavailability of hardware we tested on the AGNEWS Data Set which has 120000 data Points and 4 classes.

For Experimental basis we also follow the previous work of this paper that is word level cnn paper and the second is the character level CNN. Although our vdcnn also follows the character level approach. Then after that we tested on the vdcnn on 9 layer deep architecture.

Experimental Evaluation

We get accuracy as follows on the test data:

1) Word Level CNN : 90.6%

2) Character Level CNN : 88.6%

3) VDCNN(9 layer deep) : 86.04%

As we can see both character level cnn and the word level cnn surpasses the vdcnn but it is because the vdcnn has its name suggest it works better as it becomes deeper and deeper.

Modified Model:

1. Average pooling with dropout before output layer replaces size 2048 fully connected layers.

2. Alphabet distinguishes between unknown, padded and space characters.

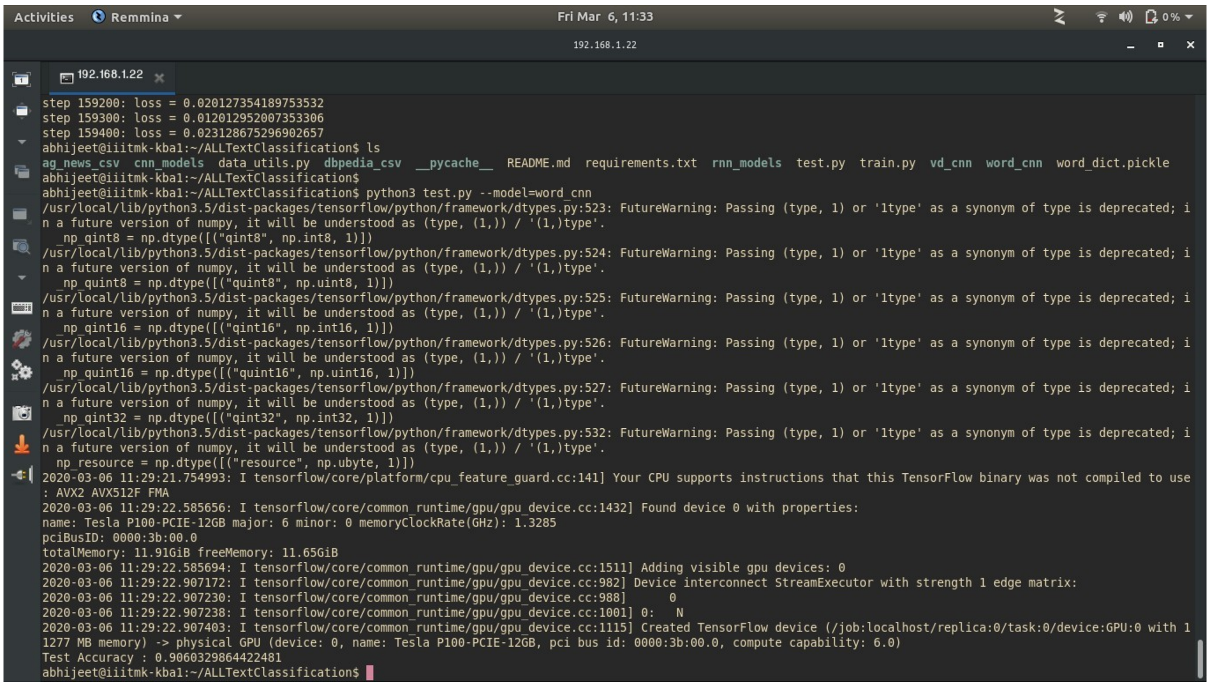
3. Depth increased(17 inspite of 9)

Accuracy achieved: **91.33% surpasses everyone.**

These are all test accuracy on the gpu.

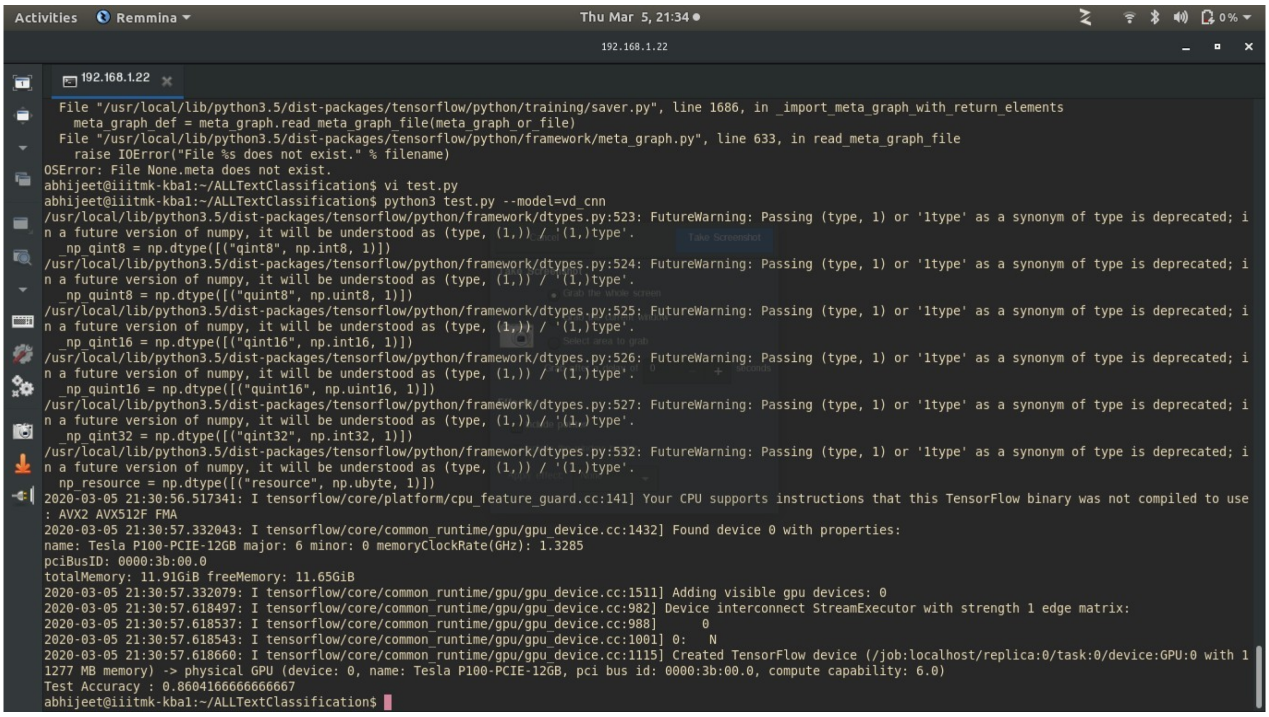
Screenshots of training and testing:

Word CNN



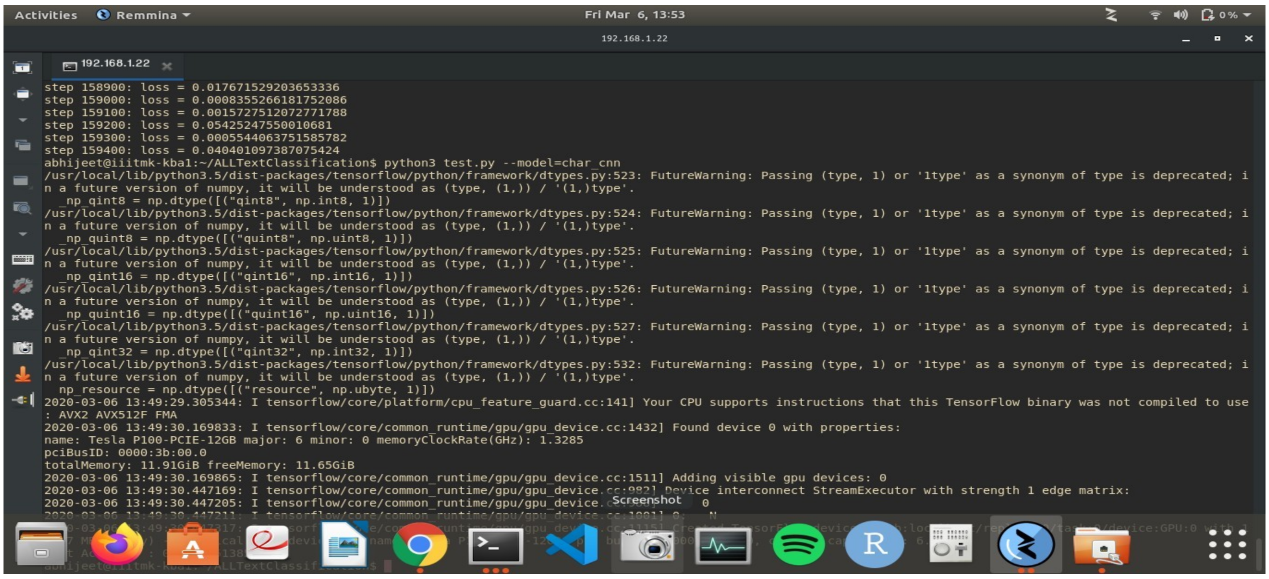
```
step 159200: loss = 0.020127354189753532
step 159300: loss = 0.012012952007353306
step 159400: loss = 0.023128675296902657
abhijeet@iiitmk-kba1:~/ALLTextClassification$ ls
ag_news_csv  cnn_models  data_utils.py  dbpedia_csv  __pycache__  README.md  requirements.txt  rnn_models  test.py  train.py  vd_cnn  word_cnn  word_dict.pickle
abhijeet@iiitmk-kba1:~/ALLTextClassification$
abhijeet@iiitmk-kba1:~/ALLTextClassification$ python3 test.py --model=word_cnn
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:524: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
2020-03-06 11:29:21.754993: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use
: AVX2 AVX512F FMA
2020-03-06 11:29:22.585656: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: Tesla P100-PCIE-12GB major: 6 minor: 0 memoryClockRate(GHz): 1.3285
pciBusID: 0000:3b:00.0
totalMemory: 11.91GiB freeMemory: 11.65GiB
2020-03-06 11:29:22.585694: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2020-03-06 11:29:22.907172: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-03-06 11:29:22.907230: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2020-03-06 11:29:22.907238: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2020-03-06 11:29:22.907403: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1
1277 MB memory) -> physical GPU (device: 0, name: Tesla P100-PCIE-12GB, pci bus id: 0000:3b:00.0, compute capability: 6.0)
Test Accuracy : 0.9060329864422481
abhijeet@iiitmk-kba1:~/ALLTextClassification$
```

VDCNN



```
  File "/usr/local/lib/python3.5/dist-packages/tensorflow/python/training/saver.py", line 1686, in _import_meta_graph_with_return_elements
    meta_graph_def = meta_graph.read_meta_graph_file(meta_graph_or_file)
  File "/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/meta_graph.py", line 633, in read_meta_graph_file
    raise IOError("File %s does not exist." % filename)
OSError: File None.meta does not exist.
abhijeet@iiitmk-kba1:~/ALLTextClassification$ vi test.py
abhijeet@iiitmk-kba1:~/ALLTextClassification$ python3 test.py --model=vd_cnn
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:524: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
2020-03-05 21:30:56.517341: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use
: AVX2 AVX512F FMA
2020-03-05 21:30:57.332043: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: Tesla P100-PCIE-12GB major: 6 minor: 0 memoryClockRate(GHz): 1.3285
pciBusID: 0000:3b:00.0
totalMemory: 11.91GiB freeMemory: 11.65GiB
2020-03-05 21:30:57.332079: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2020-03-05 21:30:57.618497: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-03-05 21:30:57.618537: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2020-03-05 21:30:57.618543: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2020-03-05 21:30:57.618660: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1
1277 MB memory) -> physical GPU (device: 0, name: Tesla P100-PCIE-12GB, pci bus id: 0000:3b:00.0, compute capability: 6.0)
Test Accuracy : 0.8604166666666667
abhijeet@iiitmk-kba1:~/ALLTextClassification$
```

CHAR CNN



```
step 158900: loss = 0.017671529203653336
step 159000: loss = 0.0008355266181752086
step 159100: loss = 0.0015727512072771788
step 159200: loss = 0.05425247550010681
step 159300: loss = 0.0005544063751585782
step 159400: loss = 0.040401097387075424
abhijeet@iiitmk-kba1:~/ALLTextClassification$ python3 test.py --model=char_cnn
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:524: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; i
n a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
2020-03-06 13:49:29.305344: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use
: AVX2 AVX512F FMA
2020-03-06 13:49:30.169833: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: Tesla P100-PCIE-12GB major: 6 minor: 0 memoryClockRate(GHz): 1.3285
pciBusID: 0000:3b:00.0
totalMemory: 11.91GiB freeMemory: 11.65GiB
2020-03-06 13:49:30.169865: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2020-03-06 13:49:30.447169: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-03-06 13:49:30.447205: I tensorflow/core/common_runtime/gpu/gpu_device.[illegible]      0
2020-03-06 13:49:30.447211: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
[illegible]
abhijeet@iiitmk-kba1:~/ALLTextClassification$
```

We present our results on AGNEWS Data Set which has 120000 data Points which cover datas representing news categorization.

The main goal of our project is to show that it is possible and beneficial to train very deep convolutional networks as text encoders. Data augmentation may improve the results even further.

As the depth of the VDCNN was increased to 17 Layers the accuracy of the model increased when compared with Word Level CNN, Character Level CNN and 9 Layered VDCNN .

<u>Common model settings:</u>

The settings that were found to be best in initial experiments have been used in all the experiments. All processing is done at the character level which is the atomic representation of a sentence.

The dictionary consists of characters such as”abcdefghijklmnopqrstuvwxyz0123456 789-,;.!?:’"/| #$%ˆ&*˜‘+=<>()[]{}” plus a special padding, space and unknown token which add up to a total of 69 tokens. The input text is padded to a fixed size of 1014, larger texts are truncated. The character embedding is of size 16. Training is performed with SGD, using a mini-batch of size 128, an initial learning rate of 0.01 and momentum of 0.9.

<u>Conclusion</u>

The Architecture presented follows two major principles:

1) operate at the lowest atomic representation of text, i.e. characters

2)use a deep stack of local operations, i.e. convolutions and max-pooling of size 3, to learn a high-level hierarchical representation of a sentence.

The overall accuracy increased as the depth was increased from 9 Layers to 17 Layers. It has been presented in the paper that increasing the depth up to 29 Convolutional Layers steadily improved performance. When compared to previous similar type models, this architecture outperforms them on the respective datasets.

Overall this Architecture has been able to prove that the proposed deep network is better or has comparable performance on large datasets. With increasing depth, accuracy saturates and starts degrading rapidly due to vanishing gradients**.** This issue is solved by using ResNet like shortcut connections.

Future Implementations

- We are Planning to do Named Entity Recognition using this deep cnn and we are planning to use bidirectional lstm at the end instead of others classifier like softmax by referring to the paper come named “Named entity recognition using bidirectional lstm and cnn”. We have never tried using vdcnn there so we are planning to do that in future.

References

1) An Introduction to Convolutional Neural Networks.(Main Paper published)

2) https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf

3) Character-level CNN for text Classification. https://papers.nips.cc/paper/5782-character- level-convolutional-networks-for-text-classification.pdf

4)Convolutional Neural Network for Sentence

Classification.https://arxiv.org/abs/1408.5882 5)Very Deep CNN for text

Classificationhttps://arxiv.org/abs/1606.01781

6)Comparative Study of CNN and RNN for Natural
Language Processinghttps://arxiv.org/abs/1702.01923

7) https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

8) https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks

9) https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network

10) https://www.researchgate.net/figure/Depth-9-VDCNN-architecture_fig1_330700625

11)Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In ICML, pages 160–167, Helsinki, Finland.

12)Ronan Collobert, Jason Weston Lon Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. JMLR, pages 2493–2537.

13)https://colah.github.io/posts/2015-08-Understanding-LSTMs/

14) https://www.kdnuggets.com/2015/11/understanding-convolutional-neural-networks-nlp.html/3