“TripleZero Detection by Using Convolutional Neural Network”

by

150116838 Necmettin Ilgın

BACHELOR OF SCIENCE

Supervised by: Dr. Öğr. Üyesi Mehmet Baran

Marmara University, Faculty of **Engineering**

Computer Engineering Department

2019

1. INTRODUCTION

My project is about a machine learning program which detects if there is a "TripleZero" (000) side by side in a given array of string which can have a max 250 length. I have used Python and:

1. numpy, tensorflow, keras libraries to train my program;
2. matplotlib to take the learning rate graphics from various trials.

I will show them later in my report. My sequential model contains below algorithms and functions:

1. Conv1d,
2. Maxpooling1D,
3. Flatten,
4. Dense blocks,
5. ReLU and sigmoid activation functions,
6. Binary cross entropy to calculate loss function and
7. Stochastic gradient descent algorithm for optimizing and updating the kernels and weights.

I have run my code on google colaboratory which is a free environment with many Python and deep learning libraries available, offering us a GPU (Nvidia Tesla K80 GPU) computing environment for developing deep learning applications.

2. IMPLEMENTATION DETAILS

2.1 Dataset Generator

I have created my dataset in range between four is minimum value and 250 is maximum value. Because to be able to detect if there is a TripleZero side by side, minimum length should be 4 for this dataset. Number of 0s and 1s in my dataset will be generated by randomly. Our program will be familiar with all possible length of input strings. After generating a random size of input string remaining part will be padded by 1s. Having TripleZero in my dataset will be also created randomly with a 0.5 probability. There is no possible for our program to be overfitted. In other words, the overall design of our model is as good as any other database that does not see the principle working well in this data set. So our model doesn’t needed a test case.

2.2 Labeling Model

2.2.1 Convolution1D

First I have created a LSTM (Long-short term memory) block to label my model but it didn’t work as well I wanted. Then I have used 1D convolutional model for detecting TripleZeros. 1D CNN is very effective when you expect to derive interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance, just like in my project.

- It is a sensitive network for frequencies,
- It provides learning according to the data frequency in the source,
- It can be used for audio files,
- This applies well to the analysis of time sequences of sensor data (such as gyroscope or accelerometer data).
- It also applies to the analysis of any kind of signal data over a fixed-length period (such as audio signals).
- Another application is NLP (although here LSTM networks are more promising since the proximity of words might not always be a good indicator for a trainable pattern).

2.2.2 Maxpooling

After applying convolutional layer to the given input string, I have chosen Max Pooling instead of Avg Pooling. Because it performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

2.2.3 Activation Functions

Activation functions are really important for an Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response/output variable.

Non-linear functions allow backpropagation because they have a derivative function which is related to the inputs.

They allow “stacking” of multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex data sets with high levels of accuracy.

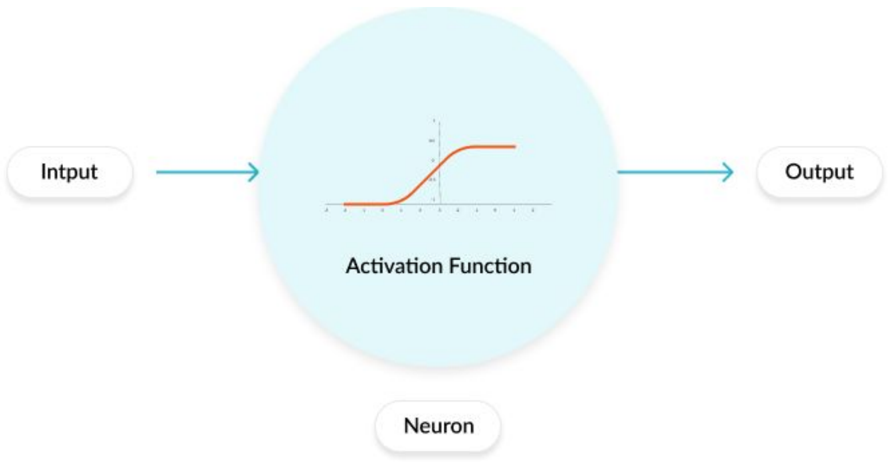


Figure 1. Activation Function

Increasingly, neural networks use non-linear activation functions, which can help the network learn complex data, compute and learn almost any function representing a question, and provide accurate predictions.

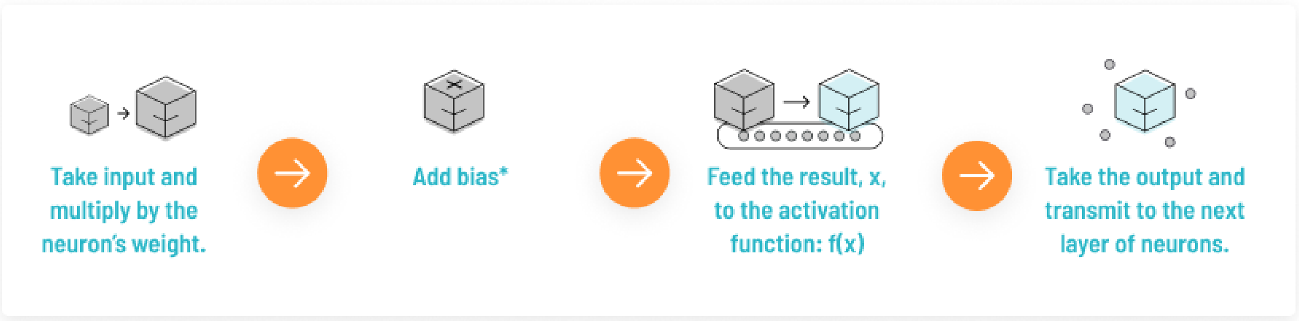

Take input and multiply by the neuron's weight.
Add bias*
Feed the result, x, to the activation function: f(x)
Take the output and transmit to the next layer of neurons.

Figure 2.The basic process carried out by a neuron in a neural network

2.2.3.1 ReLU(Rectified Linear Unit)

$$RELU(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x >= 0 \end{cases}$$

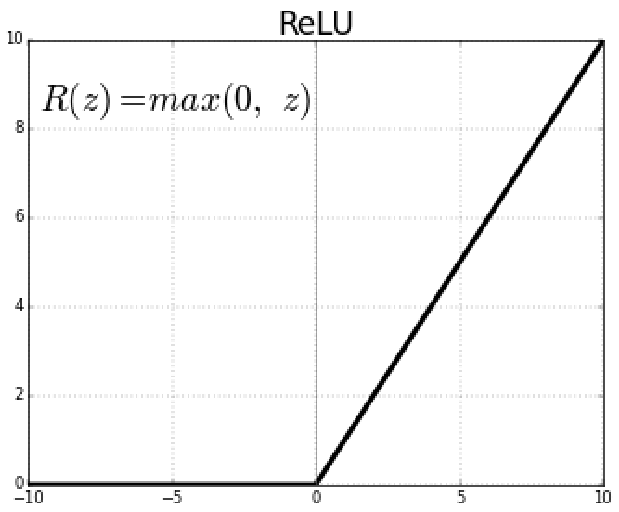


Figure 3.ReLU function:

ReLU function is non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.It was found to greatly accelerate the **convergence** of stochastic gradient descent compared to the sigmoid and tanh functions.It does not activate all the neurons at the same time. Since the output of some neurons is zero, only a few neurons are activated making the network sparse, efficient and easy for computation.

2.2.3.2 Sigmoid

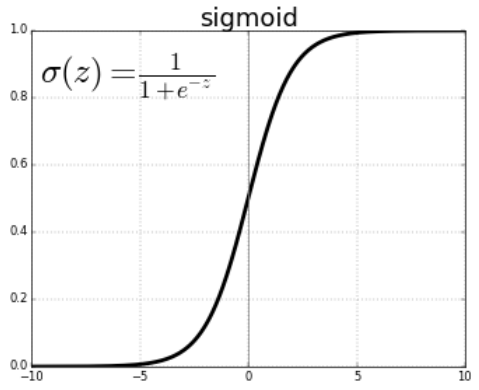


Figure 4. Sigmoid function

- This is a smooth function and is continuously differentiable.
- It is non-linear. Hence, the output is non-linear as well.
- It is easy to understand and apply.
- Easy to compute differential.

Generally, we use softmax activation instead of sigmoid with the cross-entropy loss because softmax activation distributes the probability throughout each output node. But, since my project is a binary classification, using sigmoid is same as softmax. For multi-class classification use softmax with cross-entropy

2.2.3.2 Binary Cross Entropy

We do binary classification.

So I used it because our output would be 1 or 0.
If we did the classification of 4 and more, we would use categorical cross entropy.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

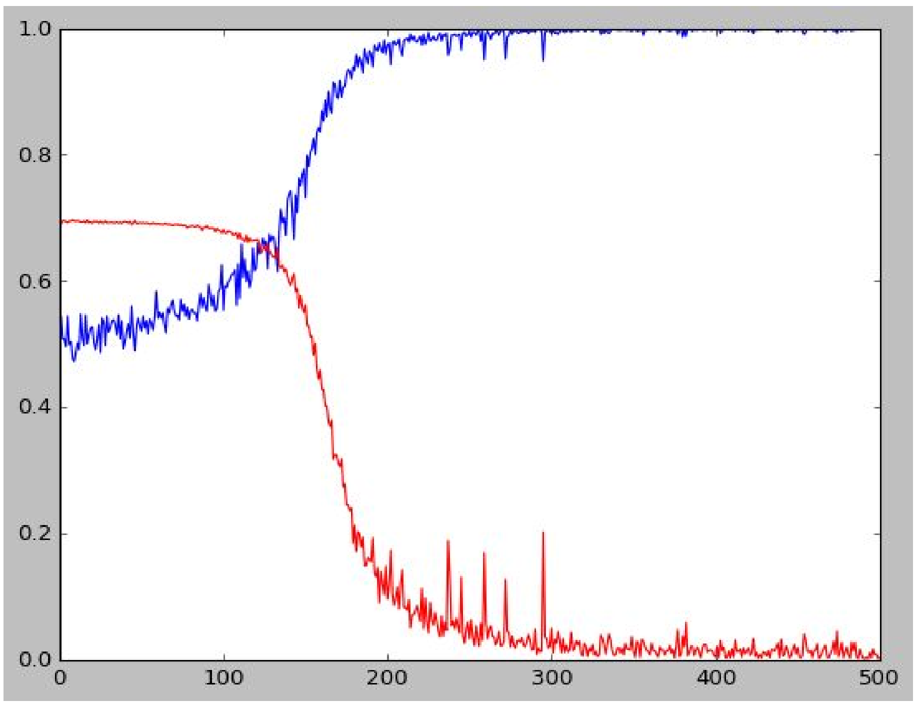
Figure 5. Binary Cross Entropy

2.2.3.3 Sgd (Stochastic Gradient Descent)

You update a set of parameters in an iterative manner to minimize an error function. While in GD, you have to run through ALL the samples in your training set to do a single update for a parameter in a particular iteration, in SGD, on the other hand, you use ONLY ONE or SUBSET of training sample from your training set to do the update for a parameter in a particular iteration. If you use SUBSET, it is called Minibatch Stochastic gradient Descent.

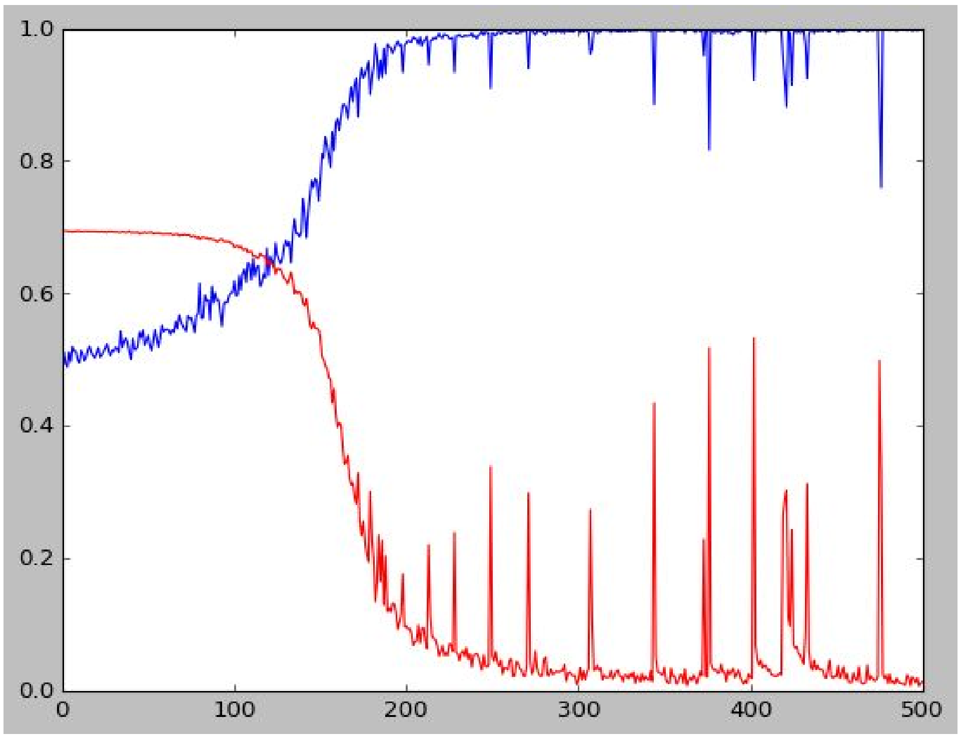
3. VARIOUS TRIALS AND COMPARISONS

3.1 Dataset number comparison



- **16 numbers of dataset ,**
- vector shape is (16,250,1);
- steps_per_epochs=40,
- epochs=500,
- 2 of Convolution blocks with 32 filters and kernel size=3 and
- 1 MaxPooling blocks (2),
- total parameters:81,853.

It took *4 minutes* to compile all the codes.



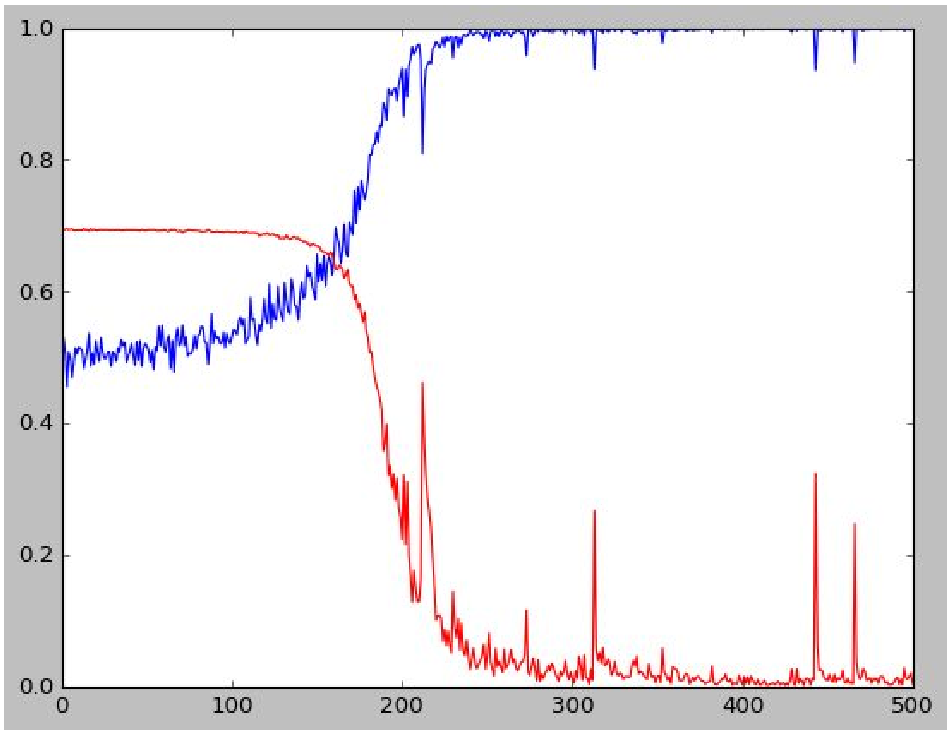
- **32 numbers of datasets**
- vector shape is (32,250,1);
- steps_per_epochs=40,
- epochs=500,
- 2 of Convolution block with 32 filters and kernel size=3 and
- 1 MaxPooling blocks (2),

- total parameters:81,853.

It took *4,2 minutes* to compile all the codes.

Although learning rates have a same approximations, the program has 16 generator is more accurate and have better learning than second one. In other graphic, there are much noise.

3.2 Additional Convolutional Block



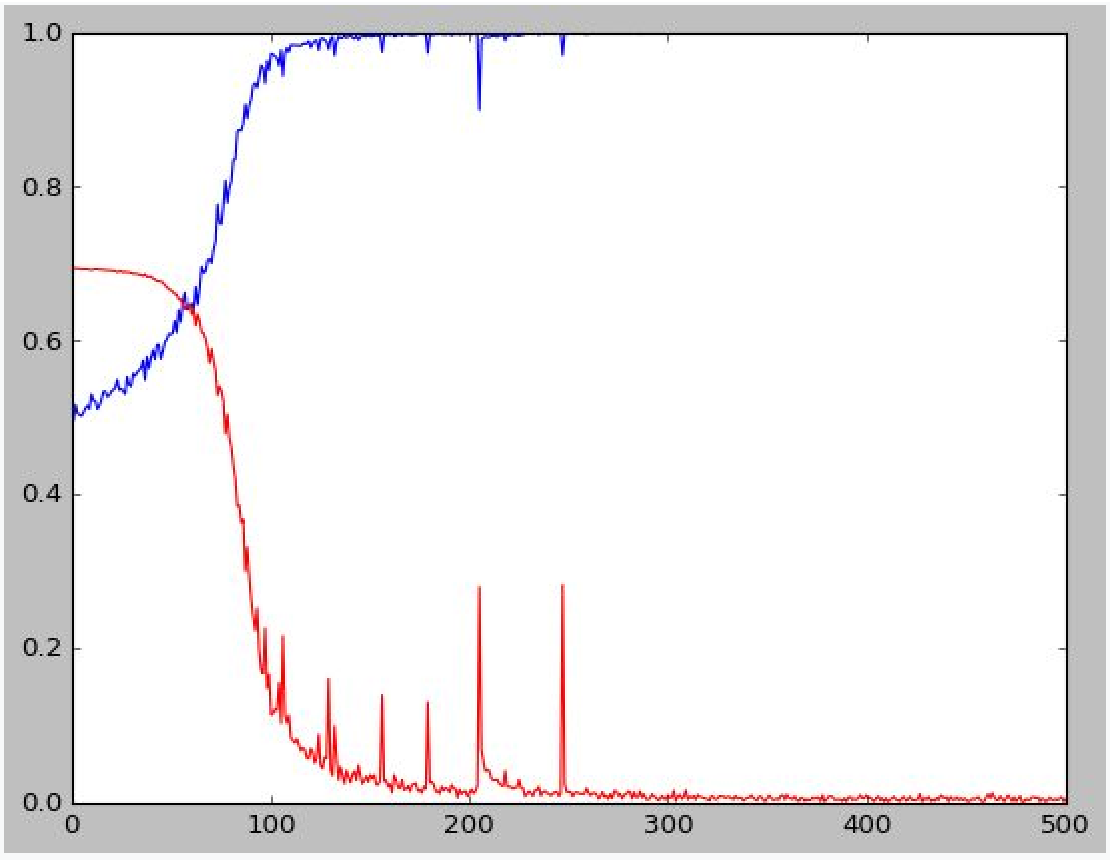
Another 1D Convolutional layer with 32 filters and kernel size=3 will allow the model to learn more complex feature. But in our example it slowed down the learning rate and increased the noises. So we don’t need a third convolutional block.

- 16 num.bers of dataset ,
- vector shape is (16,250,1);
- steps_per_epochs=40,
- epochs=500,
- **3 of Convolution blocks** with 32 filters and kernel size=3 and

- 1 MaxPooling blocks (2),
- total parameters 83,177

It took *4,9 minutes* to compile all the codes.

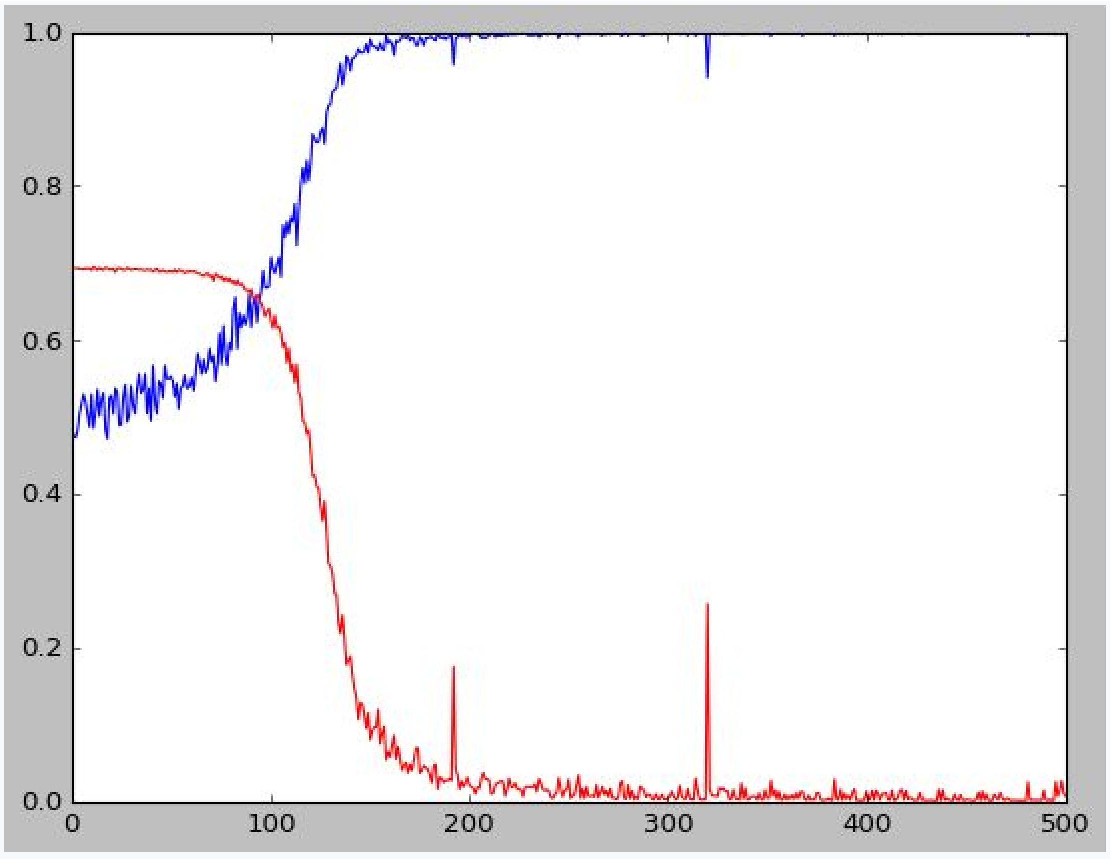
3.3 Increasing the steps per epoch



- 16 num.bers of dataset ,
- vector shape is (16,250,1);
- **steps_per_epochs=100,**
- epochs=500,
- 2 of Convolution block with 32 filters and kernel size=3 and
- 1 MaxPooling blocks (2),
- total parameters 83,177

It took *16 minutes* to compile all the codes.

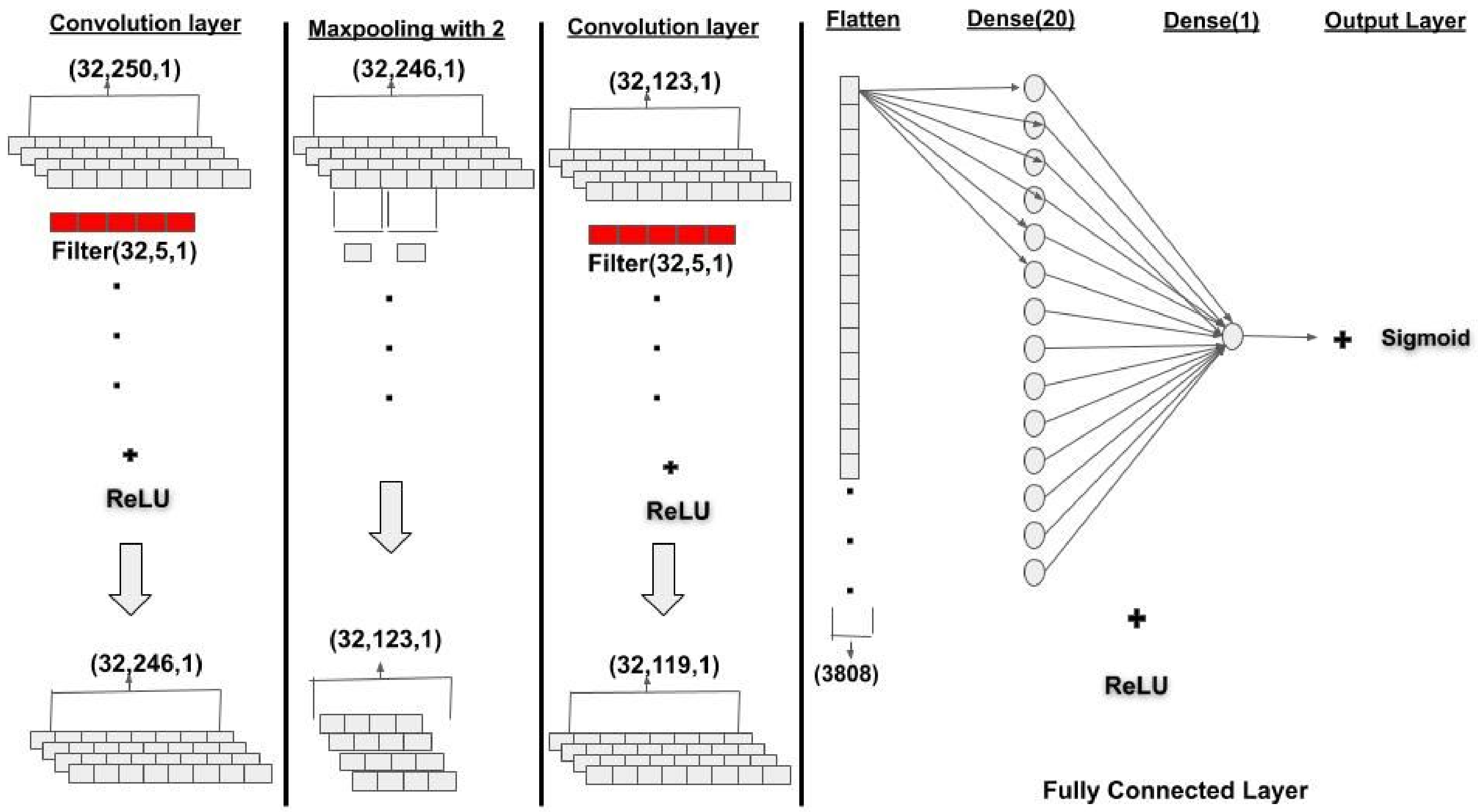
3.3 Changing kernel size from 3 to 5

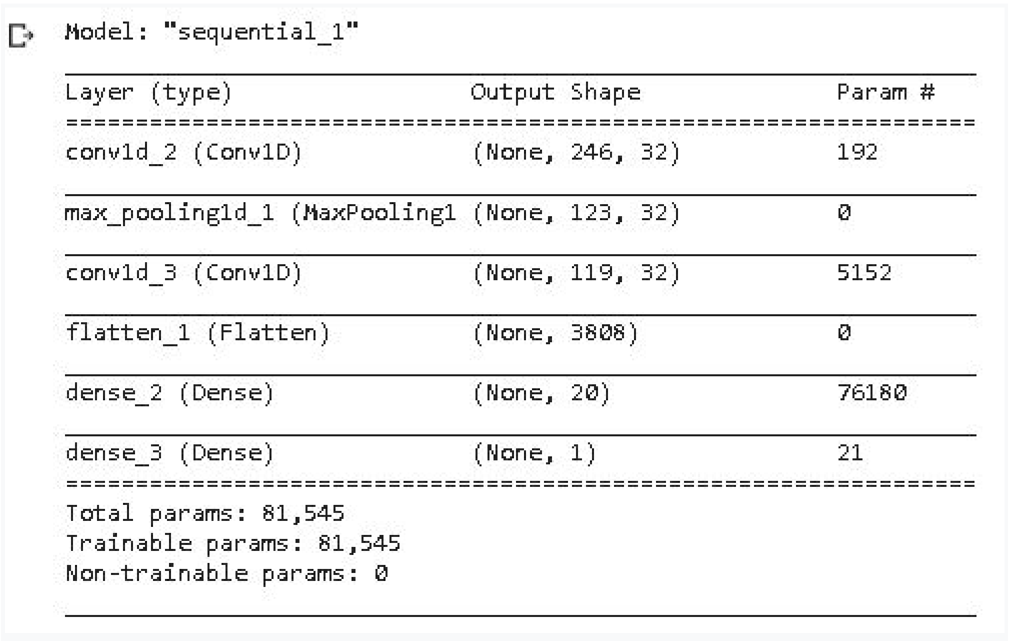


- *16 num.bers of dataset ,*
- *vector shape is (16,250,1);*
- *steps_per_epochs=40,*
- *epochs=500,*
- *2 of Convolution block with 32 filters and **kernel size=5** and*
- *1 MaxPooling blocks (2),*
- *total parameters 81,545*

It took *4.6 minutes* to compile the codes. Finally, my optimized values for my program are above. When I consider the time for teaching the computer and learning rate of machine, these values have more accurate results.

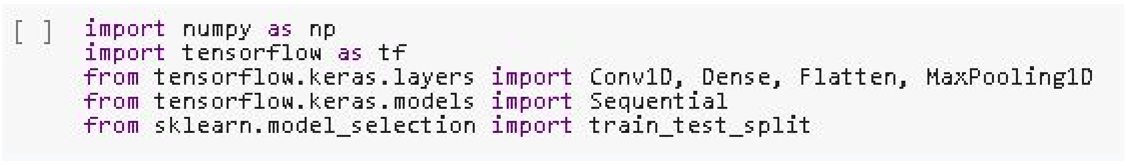
4. Architecture TripleZero Detection



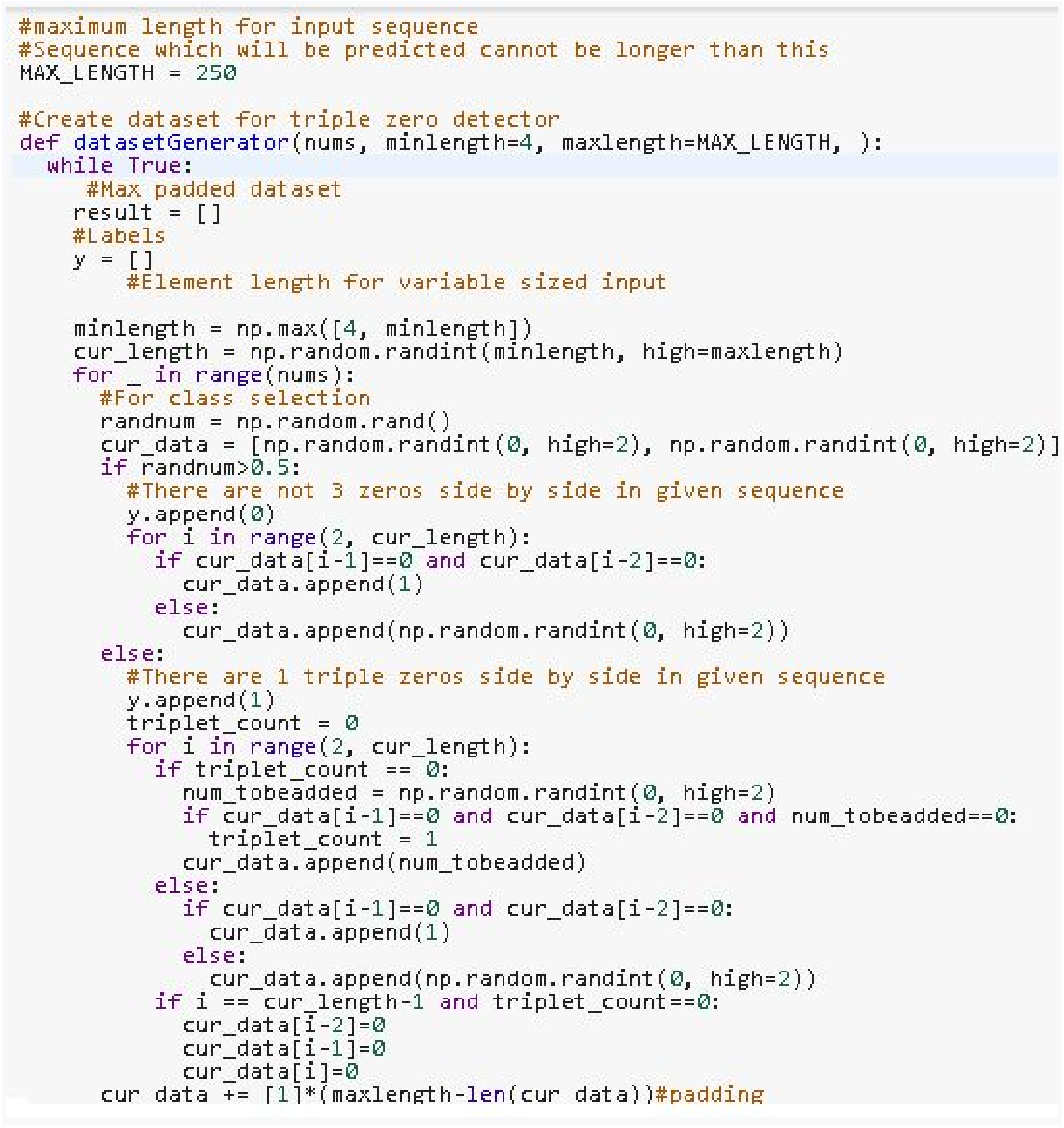


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 246, 32)           192
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 123, 32)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 119, 32)           5152
_________________________________________________________________
flatten_1 (Flatten)          (None, 3808)              0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                76180
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 21
=================================================================
Total params: 81,545
Trainable params: 81,545
Non-trainable params: 0
_________________________________________________________________
```

5.Source Codes

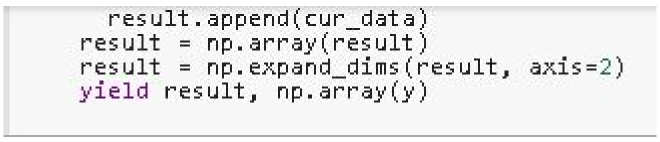

```
[ ]  import numpy as np
     import tensorflow as tf
     from tensorflow.keras.layers import Conv1D, Dense, Flatten, MaxPooling1D
     from tensorflow.keras.models import Sequential
     from sklearn.model_selection import train_test_split
```

DATASET GENERATOR

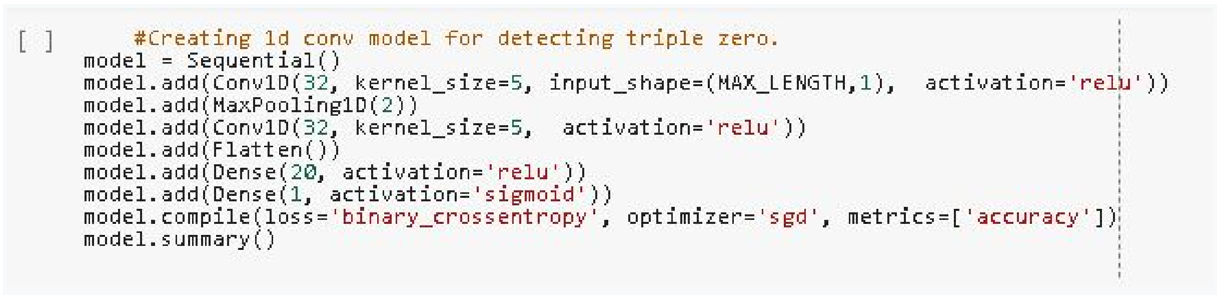

```
#maximum length for input sequence
#Sequence which will be predicted cannot be longer than this
MAX_LENGTH = 250

#Create dataset for triple zero detector
def datasetGenerator(nums, minlength=4, maxlength=MAX_LENGTH, ):
  while True:
     #Max padded dataset
    result = []
    #Labels
    y = []
        #Element length for variable sized input

    minlength = np.max([4, minlength])
    cur_length = np.random.randint(minlength, high=maxlength)
    for _ in range(nums):
      #For class selection
      randnum = np.random.rand()
      cur_data = [np.random.randint(0, high=2), np.random.randint(0, high=2)]
      if randnum>0.5:
        #There are not 3 zeros side by side in given sequence
        y.append(0)
        for i in range(2, cur_length):
          if cur_data[i-1]==0 and cur_data[i-2]==0:
            cur_data.append(1)
          else:
            cur_data.append(np.random.randint(0, high=2))
      else:
        #There are 1 triple zeros side by side in given sequence
        y.append(1)
        triplet_count = 0
        for i in range(2, cur_length):
          if triplet_count == 0:
            num_tobeadded = np.random.randint(0, high=2)
            if cur_data[i-1]==0 and cur_data[i-2]==0 and num_tobeadded==0:
              triplet_count = 1
            cur_data.append(num_tobeadded)
          else:
            if cur_data[i-1]==0 and cur_data[i-2]==0:
              cur_data.append(1)
            else:
              cur_data.append(np.random.randint(0, high=2))
          if i == cur_length-1 and triplet_count==0:
            cur_data[i-2]=0
            cur_data[i-1]=0
            cur_data[i]=0
      cur_data += [1]*(maxlength-len(cur_data))#padding
```

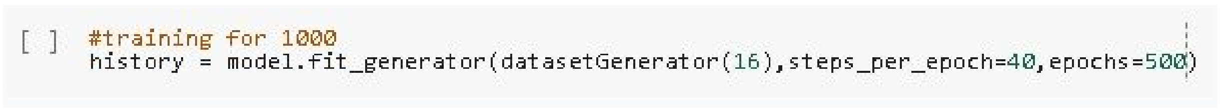

```
      result.append(cur_data)
    result = np.array(result)
    result = np.expand_dims(result, axis=2)
    yield result, np.array(y)
```

CONV1D TO LABEL MODEL



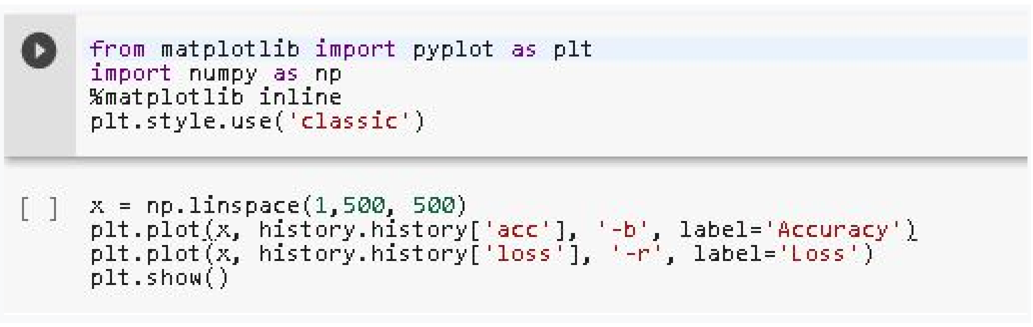
```
[ ]     #Creating 1d conv model for detecting triple zero.
    model = Sequential()
    model.add(Conv1D(32, kernel_size=5, input_shape=(MAX_LENGTH,1),  activation='relu'))
    model.add(MaxPooling1D(2))
    model.add(Conv1D(32, kernel_size=5,  activation='relu'))
    model.add(Flatten())
    model.add(Dense(20, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
    model.summary()
```

TRAINING PART



```
[ ]  #training for 1000
     history = model.fit_generator(datasetGenerator(16),steps_per_epoch=40,epochs=500)
```

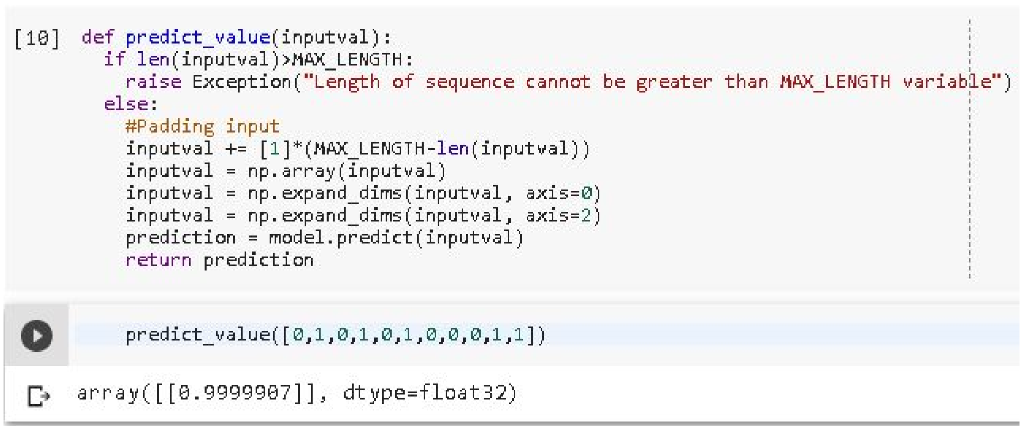
GRAPHIC



```
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline
plt.style.use('classic')
```

```
[ ]  x = np.linspace(1,500, 500)
     plt.plot(x, history.history['acc'], '-b', label='Accuracy')
     plt.plot(x, history.history['loss'], '-r', label='Loss')
     plt.show()
```

PREDICTION



```
[10]  def predict_value(inputval):
        if len(inputval)>MAX_LENGTH:
          raise Exception("Length of sequence cannot be greater than MAX_LENGTH variable")
        else:
          #Padding input
          inputval += [1]*(MAX_LENGTH-len(inputval))
          inputval = np.array(inputval)
          inputval = np.expand_dims(inputval, axis=0)
          inputval = np.expand_dims(inputval, axis=2)
          prediction = model.predict(inputval)
          return prediction
```

```
predict_value([0,1,0,1,0,1,0,0,0,1,1])
```

```
array([[0.9999907]], dtype=float32)
```

Given a sequence of strings which can be any length,11 above, our program according to the last values of parameters can predict if there is a TripleZero side by side with a probability of %9999 accuracy.