CPSC 8100 -Deep Learning:

A Deep Learning Model to Detect Images with Cyberbully Actions using Tensorflow.

Submitted By:

Mihir Phatak (mphatak@clemson.edu)
Netra Inamdar (ninamda@clemson.edu)

CONTENTS

1. INTRODUCTION:

This sections describes deep learning and cyberbullying fundamentals.

Deep learning: It is a machine learning method that takes in an input X, and uses it to predict an output of Y.

Given a large dataset of input and output pairs, a deep learning algorithm will try to minimize the difference between its prediction and expected output. By doing this, it tries to learn the association/pattern between given inputs and outputs — this in turn allows a deep learning model to generalize to inputs that it hasn't seen before. The "deep" part of deep learning refers to creating deep neural networks. This refers a neural network with a large amount of layers — with the addition of more weights and biases, the neural network improves its ability to approximate more complex functions.

Cyberbullying is bullying that takes place over digital devices like cell phones, computers, and tablets. Cyberbullying can occur through SMS, Text, and apps, or online in social media, forums, or gaming where people can view, participate in, or share content. Cyberbullying includes sending, posting, or sharing negative, harmful, false, or mean content about someone else. It can include sharing personal or private information about someone else causing embarrassment or humiliation. Some cyberbullying crosses the line into unlawful or criminal behavior.

The most common places where cyberbullying occurs are:

- Social Media, such as Facebook, Instagram, Snapchat, and Twitter
- SMS (Short Message Service) also known as Text Message sent through devices
- Instant Message (via devices, email provider services, apps, and social media messaging features)

2. AIM:

To create a convolutional neural network based deep learning model for detecting and classifying various images with cyberbully actions. We plan to implement this with the help of cv2 and tensorflow packages in python 3.

3. PROCEDURE:

To build a convolutional neural network based image classifier, we have created a 6 layer neural network that identifies and classifies images having cyber bullying actions in it.

While training, images from all 9 classes are fed to a convolutional layer which is followed by 2 more convolutional layers. We have also added an extra class of non-bullying images to train the classifier. After convolutional layers, we flatten the output and add two fully connected layer in the end. The second fully connected layer has 10 outputs which represent the probabilities of an image being one among the given 10 classes. We have selected the class as predicted output for which the maximum value is obtained from the previous step.

The implemented model is trained using the below parameters:

Batch size: 4	Training images: 2179
Learning rate: 0.0001	Validation images: 544

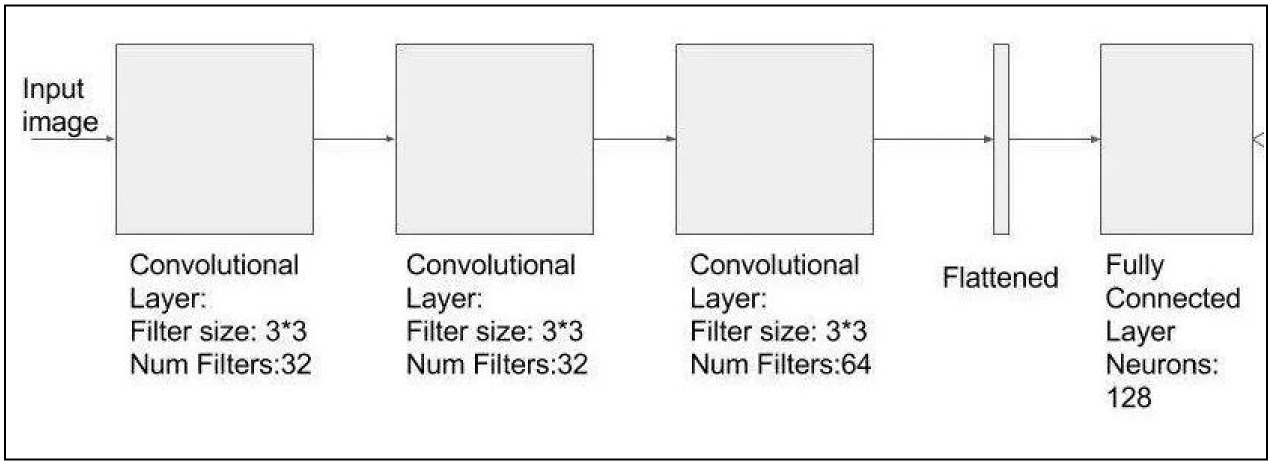

Input image
Convolutional Layer: Filter size: 3*3 Num Filters:32
Convolutional Layer: Filter size: 3*3 Num Filters:32
Convolutional Layer: Filter size: 3*3 Num Filters:64
Flattened
Fully Connected Layer Neurons: 128

Fig 1: Architecture of convolutional neural network based classifier

4. APPROACH:

4.1 :Understanding CNN, types of layers and training process:

i) CNN: Convolution is a mathematical operation that's used in single processing to filter signals, find patterns in signals. The most important parameter in a convolutional neuron is the filter size. In a convolutional layer, all neurons apply convolution operation to the inputs, hence they are called convolutional neurons. To calculate the dot product, it is required for the 3rd dimension of the filter to be same as the number of channels in the input. After each convolution, the output reduces in size. In a deep neural network with many layers, usually the output will become very small this way, which doesn't work so well. Hence, it's zeros are added on the boundary of the input layer to make output layer size same as input layer size.

ii) Max Pooling: The most used form of pooling is Max pooling. Pooling layer is used to reduce the spatial size (not the depth, just width and height) and most of the times used immediately after the convolutional layer. The computation is reduced because of reduction in number of parameters.

iii) Fully connected layer: It is a layer that receives input from all the neurons in the previous layer. The output of this layer is computed by matrix multiplication followed by bias offset.

iv) Weights and biases: After finalizing the architecture of the network; the important variables are the weights(w) and biases(b) or the parameters of the network. The objective of the training is to get the best possible values of the all these parameters which solve the problem reliably. Backward propagation process will help us find the best set of parameters, i.e. start with a random set of parameters and keep changing these weights such that for every training image we get the correct output.

v) Calculating cost: The learning rate variable determines how fast do we change the parameters of the network during training. The goal is to maximise the total correct classifications by the network i.e. for the whole training set, we want to make such changes such that the number of correct classifications by the network increases. So a single number called cost is defined for correct classification measure which indicates if the training is going in the right direction. Typically as the cost is reduced, the accuracy of the network increases. So, we keep an eye on the cost and we keep doing many iterations of forward and backward propagations till cost stops decreasing.

4.1 :Reading input images, and creating network layers for classifier model:

We have divided our input data in 2 parts:

1. Training data: we have use 80% of the training images
2. Validation data: 20% images will be used for validation. (To calculate accuracy during training process)

For detection and classification, we have following sections in our classifier:

tf.nn.conv2d: function that is used to build a convolutional layer which takes these inputs:
i) input= This should be a 4-D tensor. This is the output(activation) from the previous layer.

ii) filter= trainable variables defining the filter. We start with a random normal distribution and learn these weights. It’s a tensor whose specific shape is predefined as part of network design.

iii) strides= defines how much you move your filter when doing convolution. In this function, we have taken it as 1.

iv) padding=SAME means we have 0 padded the input such a way that output x,y dimensions are same as that of input.

After convolution, we add the biases of that neuron, which are also learnable/trainable. Again we start with random normal distribution and learn these values during training. Now, we apply max-pooling using tf.nn.max_pool function that has a very similar signature as that of conv2d function. Finally, we have used a RELU as our activation function which simply takes the output of max_pool and applies RELU using tf.nn.relu All these operations are done in a single convolution layer. The Output of a convolutional layer is a multi-dimensional Tensor. For converting this into a one-dimensional tensor, flattening layer is used. In fully connected layer, by taking all the inputs, the standard z=wx+b operation is done on it.

Here, we have completed defining the building blocks of the network.

4.3 Prediction:

Softmax: It is a function that converts K-dimensional vector 'x' containing real values to the same shaped vector of real values in the range of (0,1), whose sum is 1. We have applied the softmax function to the output of our convolutional neural network in order to, convert the output to the probability for each class. We have used function softmax_cross_entropy_with_logits which takes the output of last fully connected layer and actual labels to calculate cross_entropy whose average gives us the cost.

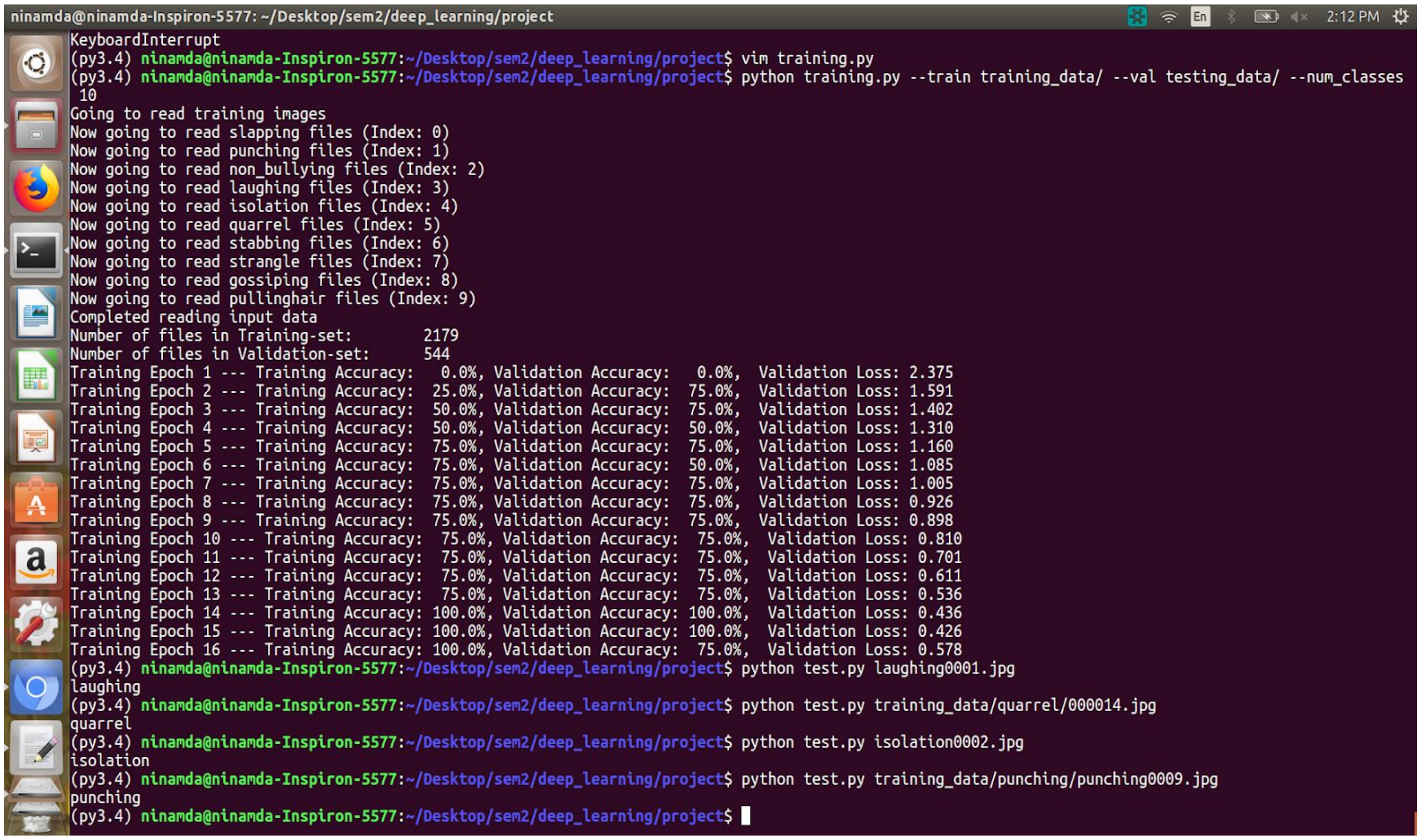
5. DATASET:

Classes	Size (Training set)
gossiping	415
isolation	253
laughing	158
pulling-hair	261
punching	366
quarrel	250
slapping	214
stabbing	142
strangle	365
Non-bullying	302

Table 1.1

6. OBSERVATIONS OF RESULTS:

For our training, we get upto 100% accuracy on training dataset, and more than 75% accuracy on validation dataset.


ninamda@ninamda-Inspiron-5577: ~/Desktop/sem2/deep_learning/project
2:12 PM

```
KeyboardInterrupt
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ vim training.py
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ python training.py --train training_data/ --val testing_data/ --num_classes
 10
Going to read training images
Now going to read slapping files (Index: 0)
Now going to read punching files (Index: 1)
Now going to read non_bullying files (Index: 2)
Now going to read laughing files (Index: 3)
Now going to read isolation files (Index: 4)
Now going to read quarrel files (Index: 5)
Now going to read stabbing files (Index: 6)
Now going to read strangle files (Index: 7)
Now going to read gossiping files (Index: 8)
Now going to read pullinghair files (Index: 9)
Completed reading input data
Number of files in Training-set:        2179
Number of files in Validation-set:      544
Training Epoch 1 --- Training Accuracy:   0.0%, Validation Accuracy:   0.0%,  Validation Loss: 2.375
Training Epoch 2 --- Training Accuracy:  25.0%, Validation Accuracy:  75.0%,  Validation Loss: 1.591
Training Epoch 3 --- Training Accuracy:  50.0%, Validation Accuracy:  75.0%,  Validation Loss: 1.402
Training Epoch 4 --- Training Accuracy:  50.0%, Validation Accuracy:  50.0%,  Validation Loss: 1.310
Training Epoch 5 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 1.160
Training Epoch 6 --- Training Accuracy:  75.0%, Validation Accuracy:  50.0%,  Validation Loss: 1.085
Training Epoch 7 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 1.005
Training Epoch 8 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.926
Training Epoch 9 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.898
Training Epoch 10 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.810
Training Epoch 11 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.701
Training Epoch 12 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.611
Training Epoch 13 --- Training Accuracy:  75.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.536
Training Epoch 14 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.436
Training Epoch 15 --- Training Accuracy: 100.0%, Validation Accuracy: 100.0%,  Validation Loss: 0.426
Training Epoch 16 --- Training Accuracy: 100.0%, Validation Accuracy:  75.0%,  Validation Loss: 0.578
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ python test.py laughing0001.jpg
laughing
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ python test.py training_data/quarrel/000014.jpg
quarrel
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ python test.py isolation0002.jpg
isolation
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ python test.py training_data/punching/punching0009.jpg
punching
(py3.4) ninamda@ninamda-Inspiron-5577:~/Desktop/sem2/deep_learning/project$ 
```

Fig 2: Training and validation accuracy of the classifier along with some test results

After we are done with training, there are below files in the folder:

1. trained-model.meta: contains the complete network graph and we can use this to recreate the graph later.
2. trained-model: contains the trained weights(values of variables) of the network.

As shown in the above figure, we tested the model with some of the images from training set, and it is able to detect and classify all the training set images correctly.

7. IMPROVEMENTS :

i) Image recognition: As part of the final submission, this model can be further improved by adding the algorithm for displaying bounding boxes around the victim and the one who is making bullying actions, in test images.

ii) Improved accuracy of the classifier: The accuracy can be enhanced by adding more hidden layers to the network and employing a different activation function. Also, currently, the model is not efficient for detection of images that do not fall under a single category, or fails for some of the non-bullying test images. In order to improve the accuracy, number of training and validation images in all the classes should be increased significantly.

iii) Pre-labeled datasets: The images used in the training can be labeled more accurately with bounding boxes that will help the network classify test images with even more accuracy.

8 .CONCLUSION

The report presents a trained model using deep learning and convolutional neural networks that detects images with cyberbully actions with training dataset of 9 bullying classes (gossiping, slapping, strangle, pullinghair, laughing, isolation, punching, quarrel, stabbing) and 1 non-bullying class, implemented in Tensorflow.

The parameters for training model were as follows:

Batch size: 4	Training images: 2179
Learning rate: 0.0001	Validation images: 544

The network architecture includes 6 layer neural network that identifies and classifies images. **The training accuracy obtained after 16 epoch was 100%**, **and the validation accuracy (20% images from the training data set) was observed to be more than 75%**. The validation loss was reduced from **2.375 to 0.578** by the final iteration. This model can be further enhanced to include the image recognition feature that will draw and display the bounding boxes around victim and the person making bullying actions.

For testing our model, we used some of the images from training dataset and the accuracy was found to be almost 100% on the training images.

For further testing, we used several images of each class that are not present in the training image dataset, and the model **was able to correctly classify 80% images** of them based on the features learned from training images.

9. REFERENCES:

i) https://towardsdatascience.com/an-introduction-to-deep-learning-af63448c122c
ii) https://www.tensorflow.org/tutorials
iii) https://pdfs.semanticscholar.org/b8e3/613d60d374b53ec5b54112dfb68d0b52d82c.pdf
iv)https://developers.google.com/machine-learning/practica/image-classification/convolutional-n
eural-networks
v) https://arxiv.org/abs/1807.05511
vi) https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012029