CPSC 8810 - Deep Learning: Midterm Project

Yadnyesh Y. Luktuke
C96091890

Eshaa D. Sood
C16170243

1 Introduction

The task for this project was to design and implement a deep neural network system for classifying different categories of images and also identify the bully and victim within each image. The most important part of a system that can classify images or identify objects for any data is the availability of good features. Good features are said to be representative of the given data, if they allow a model to correctly learn a function that can be used to predict the required output for a given input. Deep neural networks are a special category of models that learn a representative set of features from the given data. This ability to learn the best set of features for a given task allow deep neural networks to achieve better accuracy than most models that rely on hand - made features which may not be very representative of the data. For this project, we were given a set of bullying images which were separated into the following 9 categories; gossiping, isolation, laughing, pulling hair, punching, quarrelling, slapping, stabbing and strangling. The goal of the project was thus to study the design and implementation of deep neural networks, and build a system that could successfully identify an image as belonging to either of the categories mentioned above and also correctly identify the the bully and victims in each image . Additionally it was also important for the model to classify an external image that did not belong to any category mentioned above as a non - bullying image.

2 Methods

2.1 Pre - Processing

The first part of any image classification system is image pre - processing. This is because input images often originate from different sources, have different sizes and carry different information, such as greyscale and colour images. Hence it is important to standardize each input image to a neural network system. For this project we resized each input image to the size 224×224 as suggested in [2]. If an image was originally provided as a greyscale, the given information was copied twice and the image size appended to three channels, with each channel containing identical information. This was done to keep the input to the model consistent as a three channel input. Colour images contain three channels of information, with each channel containing information of the Red, Green and Blue components of each image pixel respectively. The images that contained an additional channel of information were treated as outliers and removed.

2.2 Data Augmentation

In order to learn a representative set of features, deep learning models require a lot of data. In cases where the data that is actually available is less, data augmentation is a handy tool. As the name suggests data augmentation creates copies of the existing data with sensible variations. The data is not simply copied many times over to avoid the possibility of the model learning a useless set of features. Some example of data augmentation include translation, cropping, scaling, rotating, changing brightness and contrast. The data augmentation techniques that we have used in our project are:

2.2.1 Flip

Flipping is the operation of rotating the image 180 deg about the column axis as shown in Figure 1. Flipping is one of the most easy ways to augment datasets containing images, and can be applied to almost all of these. It is also one of the most useful, as it forces the convolution layers in a Convolutional Neural Network (CNN) to look for matching spatial patterns all over an image. In cases where the specific pattern of interest appears in only one location in the original image, a CNN may get falsely trained to create a high response to that particular location in its receptor field. Flipping an image prevents this as the location of the pattern has now changed prompting the convolution layer to scan through the entire image to look for the representative pattern.

2.2.2 Gaussian Noise

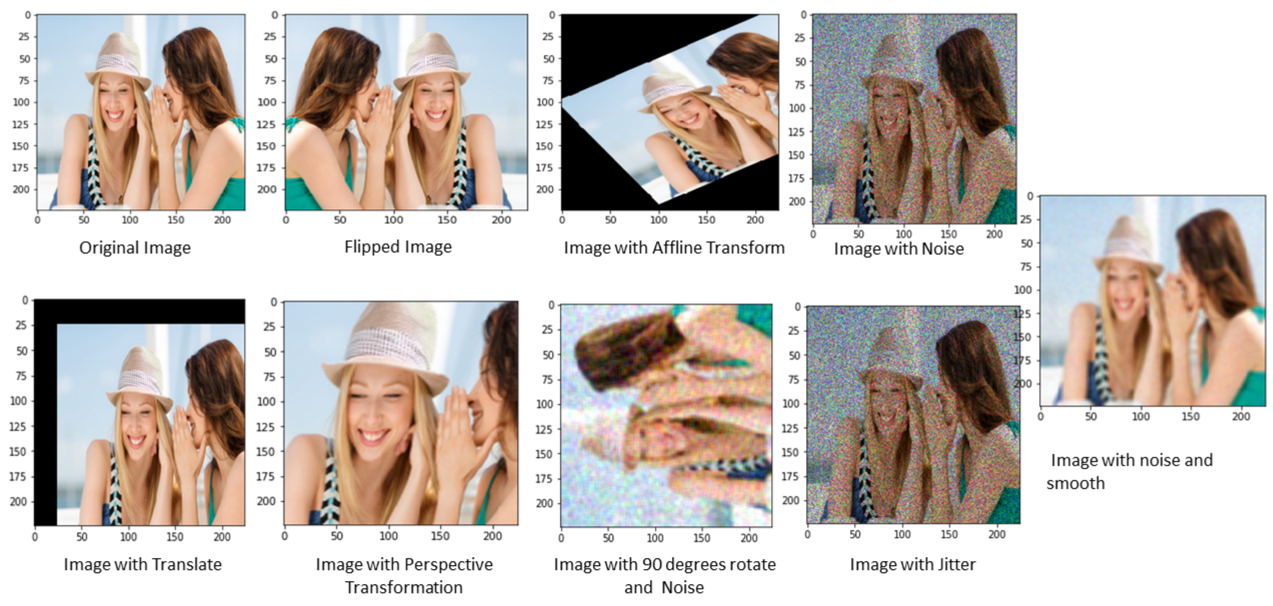
Another common technique to augment image data is by adding noise to each color channel. Gaussian noise is usually used as an additive noise in most models. The effect of applying Gaussian noise is that low frequency detail is often masked by the presence of the high frequency noise. This again means that the convolution layers need to scan the entire image to look for high frequency details such as edges and lines, as opposed to developing false responses tuned to the location of these high frequency features.

2.2.3 Jitter

Introducing color jitter can be thought of as multiplicative noise drawn from a unifrom distribution. Its effect is to distort the color information within each image. The authors of this report are unaware of the exact purpose of introducing jitter to augment the data. However it is thought that jitter works in the same way as Gaussian noise, by masking low frequency detail and forcing the convolution layer to scan the entire image for high frequency patterns. Adding noise is also thought to improve the network performance, by training a CNN to ignore the presence of noise in images.

2.3 Geometric Transformations

Geometric transformations such as affine transforms, perception transform, translation and rotation are also some of the popular techniques to increase the size of a dataset. These transforms are introduced to enable the convolution



Original Image
Flipped Image
Image with Affline Transform
Image with Noise
Image with noise and smooth
Image with Translate
Image with Perspective Transformation
Image with 90 degrees rotate and Noise
Image with Jitter

Figure 1: Data Augmentation Techniques

layers to learn translation and rotation invariant representations of the data.

2.4 Smoothing

Smoothing is an operation that is used to mask the effect of noise in digital images. However it has an added effect of introducing blur to an image that it is applied upon. As a result high frequency details are masked, and low frequency details are enhanced. This allows the CNN layers to develop good representations for the low frequency details in the image.

2.5 Network Architecture

Our network was inspired by the Convolutional Networks developed in [1] and [2]. It also borrowed from the concept of dimensionality reduction using Principal Component Analysis as explained in [6]. The concept was to continuously reduce the number of features at each stage, till the most representative features were retained by the network. This is done by using convolutional and max - pooling layers as explained in [1]. The convolutional layer consists of a stack of weights or filters that become sensitive to local features within the image, following this max - pooling retains only the strongest responses after the image has been filtered using each filter. Since the number of filters is often greater than the number of input channels, this operation transforms the input image into an image having greater number of channels (depth). The max - pooling layer reduces the size of output image by a factor of 2 along each dimension. It must be noted that this is because the value for the parameter 'stride' chosen was $[1, 2, 2, 1]$. The enthusiastic reader is encouraged to try different values of the middle parameters in order to get different filtered image sizes. The outer parameters need to be set to 1 in order to comply with the internal workings of TensorFlow. Another parameter which

Layer	Size	Input Image Size
Convolutional 1	[5, 5, 3, Depth 1]	[224, 224, 3]
Convolutional 2	[3, 3, Depth 1, Depth 2]	[112, 112, Depth 1]
Convolutional 3	[3, 3, Depth 2, Depth 3]	[56, 56, Depth 2]
Fully Connected Layer 1	[28 $\times$ 28$\times$ Depth 3, 200]	[28 $\times$ 28, Depth 3]
Fully Connected Layer 2	[200, 100]	[200]
Output Layer	[100, 10]	[100]

Table 1: Network Architecture

Layer	Size	Input Image Size
Convolutional 1	[3, 3, 3, Depth 1 = 64]	[224, 224, 3]
Convolutional 2	[3, 3, Depth 1, Depth 2 = 64]	[224, 224, Depth 1]
Convolutional 3	[3, 3, Depth 2, Depth 3 = 128]	[112,112, Depth 2]
Convolutional 4	[3, 3, Depth 3, Depth 4 = 128]	[112, 112, Depth 3]
Convolutional 5	[3, 3, Depth 4, Depth 5 = 256]	[56,56, Depth 4]
Convolutional 6	[3, 3, Depth 5, Depth 6 = 256]	[56, 56, Depth 5]
Convolutional 7	[3, 3, Depth 6, Depth 7 = 512]	[28,28, Depth 6]
Convolutional 8	[3, 3, Depth 7, Depth 8 = 512]	[28, 28, Depth 7]
Convolutional 9	[3, 3, Depth 8, Depth 9 = 512]	[7,7, Depth 8]
Convolutional 10	[3, 3, Depth 9, Depth 10 = 512]	[7, 7, Depth 9]
Fully Connected Layer 1	[28 $\times$ 28$\times$ Depth 10, 4096]	[28 $\times$ 28, Depth 10]
Fully Connected Layer 2	[4096, 4096]	[4096]
Output Layer	[4096, 10]	[4096]

Table 2: VGG Architecture

can be tuned for the convolutional network is the size of the receptive field or window, [1], [2].

For this project this value was set to 5 for the first convolutional layer and reduced to 3 for the successive convolutional layers. Each convolutional layer was followed by a max - pooling layer. Chaining three such convolutional and max - pooling stacks, results in a filtered image having size 28 $\times$ 28 and as many channels as specified by the depth of the final convolutional layer. This resulting image is flattened and used as input to the fully connected layers. Our model had two such layers having 200 and 100 neurons respectively. The final layer is known as the output layer, and it consists of neurons equal to the number of categories being predicted. This is due to the use of one - hot encoding [1], which converts the discrete label given for each image into a codeword containing a 1 in the respective index of the new label and 0 elsewhere. The advantage of this strategy is explained a little later in this report. The final network is shown in Table 2. Biases of size equal to the last dimension of each layer were added to each respective layer shown in Table 2. It must be noted that the size of the input to each subsequent convolutional layer reduces due to the max - pooling layer preceding it. Also, the number of neurons in the output layer depends on the classification type. For this project, we designed a network containing 10 neurons in its output layer, equal to the number of categories being predicted.

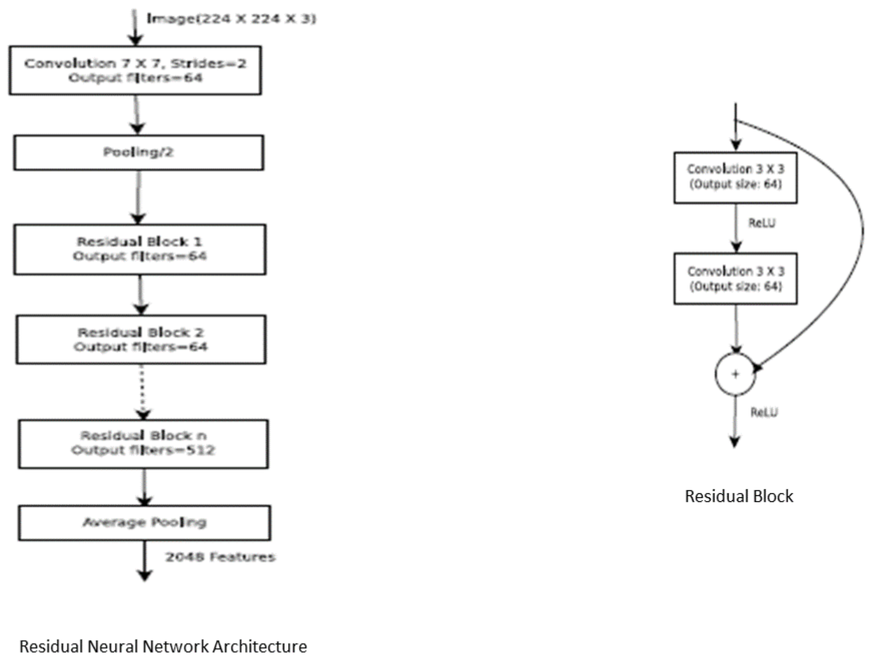


Figure 2: Residual Network Architecture

2.6 Training

2.6.1 Mini - Batch Size

Training a neural network consists of feeding it the input data in a fixed set of subsequent samples also known as batches or mini - batches [1]. Such a strategy allows the network to update its parameters in response to a small set of samples, which speeds up the learning process. Choosing batch size as a multiple of 2 also allows parallelization of the data on systems using GPU's [7].

2.6.2 Learning Rate

The learning rate (η), controls the actual weight updates in the weight update equations. Choosing a value close to 1.0 results in faster updates, but causes the weights to oscillate about their mean value. On the contrary choosing a very small value results in very slow weight updates and slower convergence of the network parameters in general. In addition learning rate decay is often used which slowly reduces the learning rate in successive epochs. The advantage of such a strategy is that the network is allowed to update its weights at a faster rate initially, prompting a quick move in the direction pointed by the gradient descent algorithm. However as the epochs progress, a slow learning rate ensures that oscillations are reduced, thereby allowing the network to move towards the global minimum of the error space in a finite number of epochs. Learning Rate Decay was implemented using the following formula:

$$\text{New Learning Rate} = \frac{1}{1 + \alpha \cdot \text{Current Epoch}} \cdot \text{Initial Learning Rate} \tag{1}$$

The term α in Equation 1 refers to the decay constant that governs how slowly, or how fast the learning rate reduces.

2.6.3 Dropout Probability

Dropout is the process of regularizing a neural network. During training only a random number of neurons, determined by the dropout probability (p_{keep}) are turned on to prevent network parameters from co - adapting. Since the number of neurons in the convolutional layer weights is small as compared to those in the fully connected layers, dropout is usually applied more strongly to the fully connected layers as compared to the convolution layers. Such a strategy prevents the useful weights in the convolutional layers from being discarded [1].

2.6.4 Batch Normalization

Batch normalization is a technique that is used to transform the inputs to successive layers within a neural network. In it the output of the previous layer is scaled to have zero mean and unit variance before being applied to the succesive layer right below it. This allows the network output to be compatible with a larger number of activation functions, allows the network to converge faster and acts as a form of input regularization in deep neural networks.

2.7 Identifying Bully And Victim

In addition to the regular approach of classifying images, another task for this project was to develop inference and label the bully and victim in each image. This is essentially an approach that involves image segmentation, and is explained further in this Subsection.

2.7.1 Labelling

For the second part of the project i.e identification of the bully and victim, we have labelled the humans within each image using a tool called Labelbox [8]. This tool enables one to draw bounding boxes around object of interest and creates a .json file of the coordinates of the bounding boxes of all labeled objects in each image. Storing a .json file for all images also allows convenient reading in a python script to load an image and its corresponding bounding box information.

2.7.2 Inspection Windows

In computer vision, inspection windows are used to detect objects within an image. While the overall position of objects is allowed to vary, the general position if often known. This allows us to scan a small region within the image. The position of this small region is fixed based on the pixel co-ordinates where we expect the object of interest to occur. The images provided for this project were mainly concerned with human - human interaction. Hence most images contain two operators, usually one being the bully and the other the victim. Hence a small portion of the left half of each image was chosen as first inspection window, while a small portion of the right half was chosen as the second inspection window. The exact co-ordinates in terms of the pixel (x, y) co -ordinates are given below:

$$\text{Inspection Window 1} = [(15, 15); (143, 15); (143, 143); (15, 143)]$$

$$\text{Inspection Window } 2 = [(15,75);(143,75);(143,203);(15,203)]$$

Note that some overlap was allowed, since it was observed that many images actually had an overlap with respect to the human operators. Also note that inspection windows were used only during testing. During training the cropped portions containing the human operators were stored separately as a dataset and used for training the model. The main reason behind using inspection windows was to allow the human operators to occur at different locations within the right and left halves, but detect them accurately during testing.

2.7.3 Component Images

Once the region within each inspection window was cropped out from the original image, these were labelled separately as 'Bully' or 'Victim' based on the Labelbox information. These were then used to train a separate CNN that could predict whether the person shown in each inspection window was acting as a bully or as a victim.

2.7.4 Choosing Network Architecture

Perhaps the most important part of the training process is choosing the parameters Depth 1, Depth 2, Depth 3 of the convolutional layers and the activation functions in each layer. For this project the depth of each successive convolutional layer were set as 10, 8 and 6. Hence the number of neurons in the fully connected layers were fixed to high values to allow the network to learn complex features that were produced as the output of the final convolutional layer. All neural network layers except the output layer had a Rectified Linear Unit activation function at their output, in order to allow us to design a network of sufficient depth [1]. The output of the output layer was fed through a softmax activation function to the further routines.

2.7.5 Number of non - bullying images

The Stanford 40 Actions dataset [3] was chosen for the non - bullying images. It contains 9532 images of people performing 40 different actions, none of which were associated with the bullying actions provided earlier. During each training session, a random number of images was selected from this dataset (after shuffling), and appended to the dataset that already contained the bullying images. The number of images chosen was kept as an open parameter, that could be selected uniquely for every model

2.8 Models Implemented

As our initial start point, we experimented with training a network similar to the VGG13 network as given in the [2]. However it did not result in good accuracy. Changing parameters such as the number of convolutional filters in each layer, adapting parameters such as the learning rate, p_{keep}, as well as using methods such as batch normalization [1] did not help to improve the accuracy of the model. Hence this approach was not considered any further.

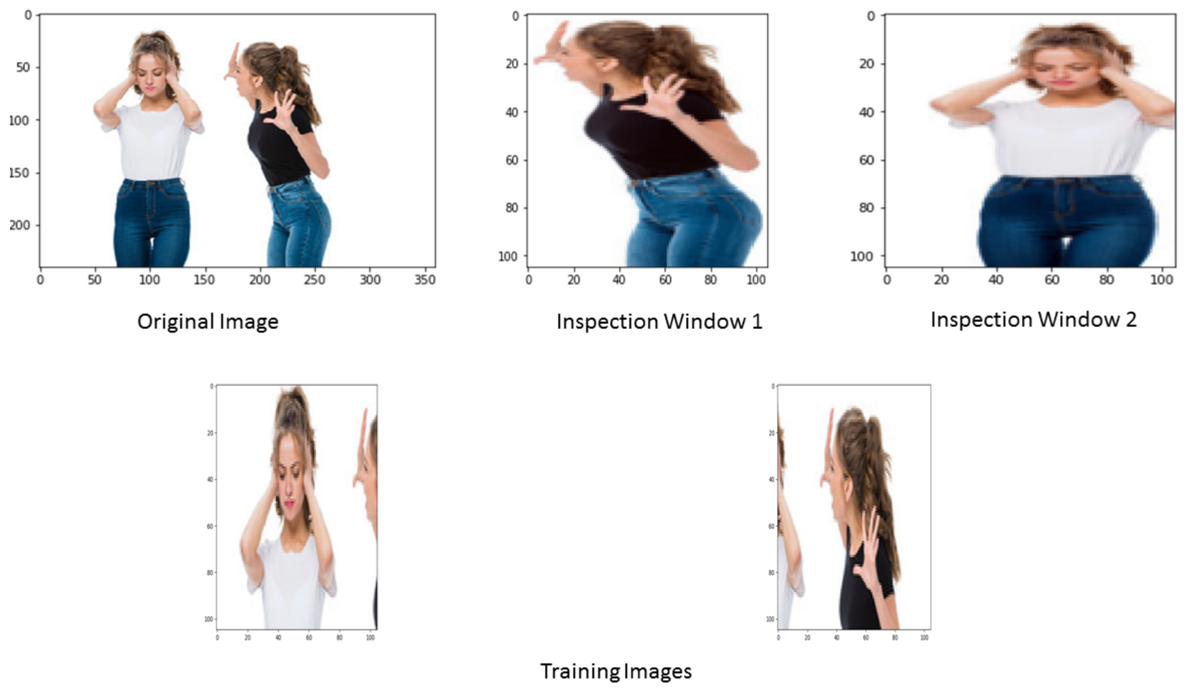


Figure 3: Model 2 Strategy

Another important model that we implemented is known as the ResNet Architecture shown in Figure 2. ResNet's have been popular in the computer vision community since their discovery because of the impressive results achieved by some groups on the ImageNet dataset [9]. The ResNet architecture is special because it employs a set of skip connections between layers. These skip connections allow the network output at these layers to skip its immediate successor, and apply to the output of the next layer as shown in [10]. This is the basic Residual Block as introduced by the authors, and it has been shown to achieve significant accuracy in classifying datasets of images containing 'action' scenes. Hence it was used as a reference when building a ResNet model for the classification task in this project. The exact architecture is given as follows:

1. Convolutional Layer: Kernel Size $\in [3, 5, 7, 9]$, Output Filter Depth $\in [4, 6, 8, 16]$ and ReLU Activation
2. Residual Block 1 as per [10]

   Convolutional Layer 1: Kernel Size $= [7, 7]$, ReLU Activation, Output Filter Depth $= 2 \times$ Previous Layer

   Convolutional Layer: Kernel Size $= [7, 7]$, ReLU Activation, Output Filter Depth = Same as above
3. Residual Block 2

   Convolutional Layer 1: Kernel Size $= [5, 5]$, ReLU Activation, Output Filter Depth $= 2 \times$ Previous Layer

   Convolutional Layer: Kernel Size $= [5, 5]$, ReLU Activation, Output Filter Depth = Same as above
4. Residual Block 3

   Convolutional Layer 1: Kernel Size $= [3, 3]$, ReLU Activation, Output Filter Depth $= 2 \times$ Previous Layer

   Convolutional Layer: Kernel Size $= [3, 3]$, ReLU Activation, Output Filter Depth = Same as above
5. Residual Block 4

   Convolutional Layer 1: Kernel Size $= [3, 3]$, ReLU Activation, Output Filter Depth $= 2 \times$ Previous Layer

Convolutional Layer: Kernel Size = $[3, 3]$, ReLU Activation, Output Filter Depth = Same as above

6. Residual Block 5

   Convolutional Layer 1: Kernel Size = $[3, 3]$, ReLU Activation, Output Filter Depth = $2 \times$ Previous Layer

   Convolutional Layer: Kernel Size = $[3, 3]$, ReLU Activation, Output Filter Depth = Same as above

Thus the depth of the convolution layers steadily increases in this architecture, while the size of the kernel window successively reduces. The former was implemented intentionally for the purpose of forming deep representations of the input images, while the latter was chosen since the image size goes on reducing at each stage due to the use of Max - Pooling layers at the end of the first convolution layer and each successive residual block.

This ResNet architecture was the deepest architecture that was implemented. However it was observed that both training and testing loss were too high with this model, without any gain in accuracy in successive epochs. Hence this model was discontinued.

2.9 Efforts To Remove Overfitting

Towards the middle of this project, it was observed that the original model chosen for this classification task was overfitting to the provided dataset. Hence several efforts were taken to reduce this behavior, these are described below:

1. Creating a seperate validation dataset: The dataset chosen for training was initially split into training and validation data at the start of the training. In subsequent epochs, the training set was reshuffled while the validation set was left untouched.

2. Convolutional dropout: In order to train deeper networks, with increasing filter depths, it is necessary to implement dropout in the convolution layer alongwith the fully connected layers. This provides a way to regularize the weights and keep subsequent filters from learning redundant representations.

3. Batch - Normalization: As discussed earlier, batch normalization provides a way of regularizing the input to successive convolution layers. It is also suggested that batch normalization may have an effect in preventing a deep network from overfitting.

4. Data Augmentation: Data augmentation prevents overfitting, by forcing the network to learn descriptive features present in images rather than developing strong localized responses unnecessarily.

5. Regularization: Adding $L2$ regularization to the weights has been reported as an effective strategy in dealing with deep neural networks that overfit to the training data. However when this was applied to the CNN models implemented in this project, it was observed that the regularization term completely dominated the loss term and hence weight updates were skewed towards reducing the weights as opposed to classifying images correctly. Hence this approach was not considered for deep neural networks such as the VGG and the ResNet described earlier.

6. Early Stopping: Continuously monitoring the loss and providing an exit condition when the loss does not change significantly over successive epochs is also a way of avoiding overfitting in neural networks, and was followed for a few models.

2.10 Model Evaluation

In order to measure the performance of the network, the chosen loss function was the cross entropy between the predicted labels and actual label of the image. Here we come back to the number of neurons in the output layer as mentioned in Sub - Section 2.2, choosing the activation function of the output layer as softmax enables the network to predict the probability of the image belonging to each class. When the one hot encoded true label of the image is also treated as a probability, these vectors can be compared through the use of cross entropy. This is consistent with the networks mentioned in [1]. In order to assess the ability of the network to identify non - bullying images, we calculated the True Negative Rate of the classification given by the following equation:

$$TrueNegativeRate = \frac{TrueNegative}{TrueNegative + FalsePositive} \tag{2}$$

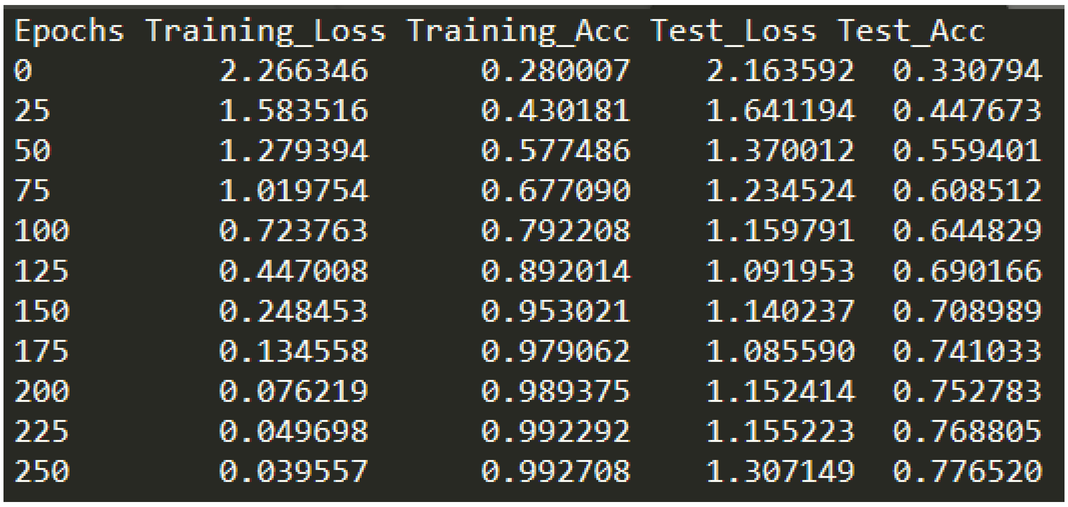
3 Results

In order to evaluate the network performance, and to ensure that the network was not overfitting to the training data provided, the dataset was randomly sampled into training and validation data. We experimented with different mini - batch sizes such as 32, 64 and 128. Choosing 32 made the training iterations run faster, but there was a noticeable decrease in the training and validation accuracy for each epoch. On the other hand, choosing 64 and 128 gave a network with higher accuracy for the training and validation data sets. Since there was no difference in the accuracy of the network trained with a batch size of 64 and 128, we chose the mini - batch size as 64 to speed up the training process. The values of η and p_{keep} were set to 0.001 and 0.7 respectively. The number of non - bullying images selected was varied from the following set of values [10, 20, 100, 200, 500, 1000, 2408] with the last number being equal to the number of bullying images in the training dataset. It was observed that choosing an equal number of non - bullying and bullying images resulted in the best network performance. Figure 4 displays the accuracy obtained during the training of the neural network for the categorical classification task. It can be seen that the network achieved good accuracy within a few epochs. This is a surprising result, since the complexity of the network was low as compared to the more complex networks in [1] and [2]. It is thought that the variability of the images in the dataset is small, which allowed the network to learn a good representative set of features for most images within the dataset.

Table 3 displays the cross confusion matrix obtained after evaluating the network on the training set images and all images selected from the Stanford 40 actions dataset [3]. It was hence observed that the tendency of the model to overfit was overcome by some measure, however it was not entirely successful. Moreover this negatively resulted in a decrease in network performance.

Non-Bullying	Gossiping	Isolation	Laughing	Pulling Hair	Punching	Quarrel	Slapping	Stabbing	Strangle
4501	54	40	46	44	114	16	52	84	113
0	1262	19	10	13	13	34	12	0	24
5	3	682	0	6	4	9	10	2	7
6	23	0	508	5	10	15	6	1	13
0	46	31	8	842	33	92	21	3	13
24	18	34	10	50	1150	38	44	17	40
0	22	14	4	14	5	727	20	2	5
0	16	55	5	11	17	23	595	13	17
6	2	40	3	1	14	7	10	386	6
21	190	89	38	58	104	39	70	56	1198

Table 3: Cross Confusion Matrix for All Categories



```
Epochs Training_Loss Training_Acc Test_Loss Test_Acc
0           2.266346      0.280007    2.163592  0.330794
25          1.583516      0.430181    1.641194  0.447673
50          1.279394      0.577486    1.370012  0.559401
75          1.019754      0.677090    1.234524  0.608512
100         0.723763      0.792208    1.159791  0.644829
125         0.447008      0.892014    1.091953  0.690166
150         0.248453      0.953021    1.140237  0.708989
175         0.134558      0.979062    1.085590  0.741033
200         0.076219      0.989375    1.152414  0.752783
225         0.049698      0.992292    1.155223  0.768805
250         0.039557      0.992708    1.307149  0.776520
```

Figure 4: Analysis of Model Performance with data augmentation

4 Conclusion

For this project we designed and built a deep neural network using TensorFlow for the purpose of identifying images as either bullying or non bullying. Further each image was classified as belonging to a specific category of bullying. The results obtained for this experiment suggest that the designed network needs to be made more complex either through the use of more layers or different layers, which could possibly improve the performance of the network. However an important part of training a more complex network is the availability of more data. Hence data augmentation will be needed in the next stage of this project. Additionally we also need better architectures in order to predict the victim and the culprit associated in each bullying action image. It is thought that a neural network using an Attention Model will be useful in this task.

References

[1] Google Cloud Platform, "Tensorflow and deep learning, without a PhD", available at `"https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0"`, Online; Accessed Jan 20, 2019

[2] K.Simoyan, A.Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", available at `"http://arxiv.org/abs/1409.1556"`, Online; Accessed Feb 28, 2019

[3] B. Yao, X. Jiang, A. Khosla, A.L. Lin, L.J. Guibas, and L. Fei-Fei, "Human Action Recognition by Learning Bases of Action Attributes and Parts", International Conference on Computer Vision (ICCV), Barcelona, Spain. November 6-13, 2011.

[4] S. van der Walt, S. Chris Colbert and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation", Computing in Science & Engineering, 13, 22-30 (2011).

[5] A.Clark and contributors, F.Lundh and contributors, "Pillow and Python Imaging Library (PIL)", available at `https://pillow.readthedocs.io/en/stable/index.html`, Online; Accessed Feb 28, 2019

[6] S.Theodoridis, K. Koutroumbas, "Pattern Recognition, Fourth Edition", Academic Press, Inc. 2008

[7] A. Ng, "Mini Batch Gradient Descent", available at `"https://www.youtube.com/watch?v=4qJaSmvhxi8"`, Online; Accessed Jan 20, 2019

[8] Labelbox, "The best way to create and manage visual training data", available at `"https://labelbox.com/"`, Online; Accessed Jan 20, 2019

[9] Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and Fei-Fei, L., "ImageNet: A Large-Scale Hierarchical Image Database", available at `"http://image-net.org/index"`, Online; Accessed Jan 20, 2019

[10] Sreela, SR and Idicula, Sumam Mary, "Action Recognition in Still Images using Residual Neural Network Features", Procedia computer science, Volume: 143, 563–569, 2018, Elsevier.