DL Assignment-2

Group Number: S21DL20
Korupolu Saideepthi S20180010087
Varakala Sowmya S20180010187
Gendeti Manjju Shree Devy S20180010055
Swathi Kedarasetty S20180010172

Part-1 Object Detection

YOLO

You only look once It's an object detector that uses features learned by a deep convolutional neural network to detect an object , over the time many versions of yolo has been improved yolo v1,...v5 (yolo v1 is implemented for the assignment)

Code

- Data
  - PascalVoc Dataset is used
  - Download from link

- Configuration (config.py)
  - Configuration for yolo on pascal_voc datasets is done in this file

- Data Annotation (voc_labe.py)
  - All the images files need to be annotated in this format **{object-class x y width height}** which is done in this file

- Data Augmentation (datset.py)
  - Reads Images from the path
  - Gets the bounding boxes , class predictions
  - Random shift
  - Random scale
  - Random crop
  - Random flip
  - Subtract mean
  - Encoder
    - function takes as input bounding boxes and corresponding labels for a particular image
      sample and outputs a target tensor of size SxSx(5xB+C)
      where S,S are the spatial dimensions, B is the number of bounding box predictions, C is the number of classes in the dataset and 5 represents the bounding box coordinates
  - getitem
    - All the above functions are called in this function to get the final augmented images

- Model (restnet_yolo.py)
    - Used resnet model
- Loss functions (yolo_loss.py)
    - Compute_iou (Intersection over union)
      Compute the intersection over union of two set of boxes, each box is [x1,y1,x2,y2].
      Args:
    - box1: (tensor) bounding boxes, sized [N,4].
    - box2: (tensor) bounding boxes, sized [M,4].
    - Return:
    - (tensor) iou, sized [N,M]
    - forward
      Args:
      pred_tensor: (tensor) size(batchsize,S,S,Bx5+20=30)
    - where B - number of bounding boxes this grid cell is a part of = 2
    - 5 - number of bounding box values corresponding to [x, y, w, h, c]
    - where x - x_coord, y - y_coord, w - width, h - height, c - confidence of having an object
    - 20 - number of classes
    - target_tensor: (tensor) size(batchsize,S,S,30)
        - Compute loss for the cells with no object bbox
        - Compute loss for the cells with objects
        - Choose the predicted bbox having the highest IoU for each target bbox
        - BBox location/size and objectness loss for the response boxes
        - Class probability loss for the cells which contain objects
        - Total loss
          Returns : Total Loss
- Hyperparameters
    - learning_rate = 0.001
    - num_epochs = 50
    - batch_size = 24

      # Yolo loss component coefficients (as given in Yolo v1 paper)

    - lambda_coord = 5
    - lambda_noobj = 0.5

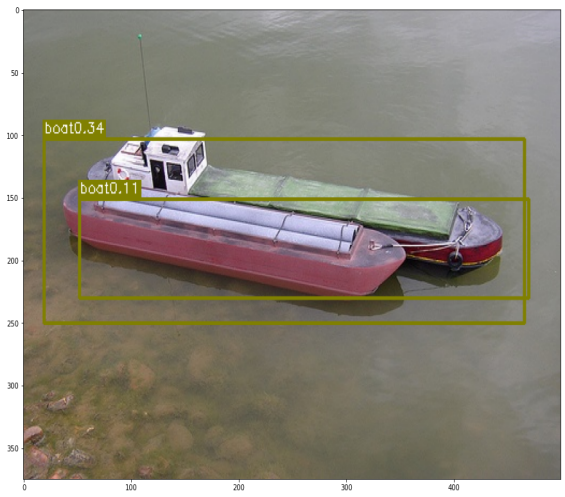
- Training (yolo.ipynb)
    - Model is trained with resnet model on the data augmented images with the above hyperparameters and the best detector is saved in **best_detector.pth**
    - **best_detector.pth** **link**

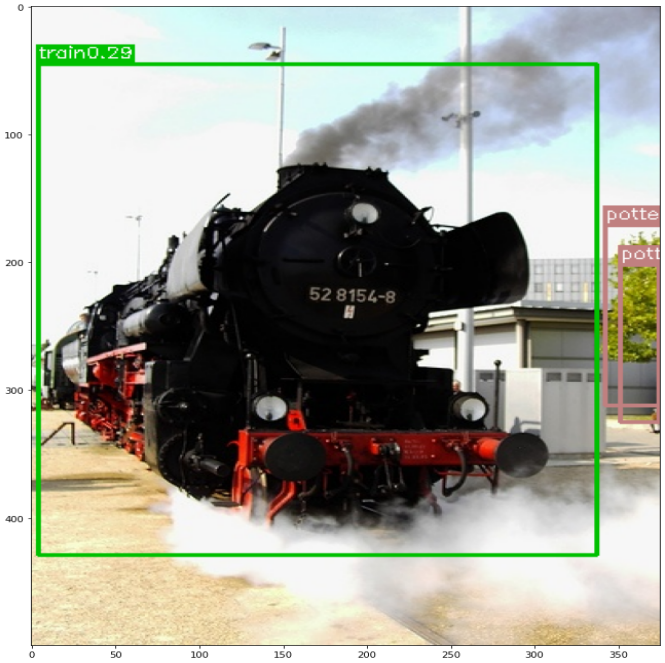
- Evaluating(eval_voc.py)
    - Using the best_detector.pth the test images are tested

- Results
    - Mean average precision over the test dataset : 0.49

```
---Evaluate model on test samples---
100%|██████████| 4950/4950 [02:23<00:00, 34.43it/s
---class aeroplane ap 0.461997978943023---
---class bicycle ap 0.6359974440609961---
---class bird ap 0.47355089012157403---
---class boat ap 0.2862735768253211---
---class bottle ap 0.2091828800787261---
---class bus ap 0.6336731764541319---
---class car ap 0.6458709332098127---
---class cat ap 0.6858524443643779---
---class chair ap 0.315842207049645---
---class cow ap 0.5132370276353126---
---class diningtable ap 0.3388517367570423---
---class dog ap 0.6307587953970472---
---class horse ap 0.6760313169372795---
---class motorbike ap 0.5649322219448399---
---class person ap 0.5295174355939133---
---class pottedplant ap 0.1868921931047106---
---class sheep ap 0.4647626789384095---
---class sofa ap 0.4620061549727634---
---class train ap 0.6708275376684594---
---class tvmonitor ap 0.5022007930301944---
---map 0.4944129711543791---
```

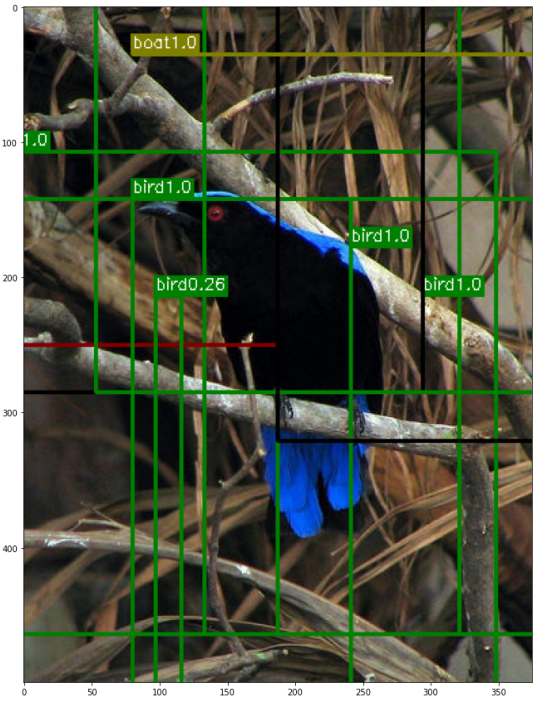
- Testing on random images(predict.py)





Faster R-CNN

Faster r-cnn is one more approach for object detection



Code

- Data
  - PascalVoc Dataset is used
  - Download from link

- Data augmentation (dataset/base.py)
  - In this file the data is augmented
    - Transforming all the images
    - Padding
    - Scaling ...etc

- Data Annotations (dataset/voc_2007.py)
  - In this file dataset is annotated
  - And also all the classes are labeled

- Configuration(config)
  - In this folder there will three .py files train_config.py, eval_config.py, config.py
  - In config.py basic configurations are given
    - Image sizes
    - Anchor sizes and ratios
  - In train_config.py all the necessary parameters needed for training are given
    - Batch_size
    - Learning_rate
    - Weight_decay
    - Steps(iterations)
    - RPN and NMS parameters
  - In eval_config.py all the necessary parameters needed for testing are given
    - RPN and NMS parameters

- Region Proposal Network (rpn/region_proposal_network.py)
  - Region proposal network (**RPN**) is used for generating region proposals and a network using these proposals to detect objects.

- Region of Interest (roi/pooler.py)
  - Region of Interest (**ROI**) pooling is used for utilising single feature map for all the proposals generated by RPN in a single pass
- bbox.py
  - Anchor boxes are defined in this file

- Non Maximum Suppression (support/layer/nms.py)
  - **Non-maximum suppression** (NMS) is used to get rid of overlapping boxes,

- Backbone (backbone)
  - resnet cnn is written in resnet101.py

- Model (model.py)
  - This file contains code for Resnet model , loss functions and for detections

- Training (train.py)
  - Dataset is loaded
  - All the above files mode,rpn,roi,hyperparameters are imported and used for training

- Evaluation (eval.py)
  - This file is used for evaluating

- Faster-rcnn.ipynb
  - Train data is trained and model is saved link
  - Using the model the test data is evaluated

- Results
  - Mean Average Precision : 0.43

```
2021-04-14 08:48:36 INFO     Start evaluating with 1 GPU (1 batch per GPU)
100% 2510/2510 [05:15<00:00,  7.97it/s]
2021-04-14 08:53:59 INFO     Done
2021-04-14 08:53:59 INFO     mean AP = :0.4316
1: aeroplane AP = 0.5056
2: bicycle AP = 0.0462
3: bird AP = 0.6708
4: boat AP = 0.4821
5: bottle AP = 0.2817
6: bus AP = 0.5478
7: car AP = 0.1867
8: cat AP = 0.4620
9: chair AP = 0.5142
10: cow AP = 0.6853
11: diningtable AP = 0.3388
12: dog AP = 0.5429
13: horse AP = 0.6760
14: motorbike AP = 0.3278
15: person AP = 0.5022
16: pottedplant AP = 0.6490
17: sheep AP = 0.4285
18: sofa AP = 0.5343
19: train AP = 0.3158
20: tvmonitor AP = 0.4523
```

Part 2 - Self Supervised learning

Dataset

CIFAR100 dataset

Model

CIFAR100Resnet → Built by using resnet34 as the base model

```
class CIFAR100Resnet(ImageClassificationBase):
        def __init__(self):
                super().__init__()
                self.network = torchvision.models.resnet34()
                num_ftrs = self.network.fc.in_features
                self.network.fc = nn.Linear(num_ftrs, 100)

        def forward(self, xb):
                return self.network(xb)

model = CIFAR100Resnet()
```

Training

For each epoch,

- Calculating the loss of the model (Note that, we used regularization also to prevent the weights from becoming too large) with training data
- Clipping the gradients (to limit the gradients to a small range, preventing undesirable changes in the parameters).
- Optimizing the model
- Repeating the above steps

Optimizer function chosen : Adam

Weight initialization :

- He's initialization is used
- The weights are initialized keeping in mind the size of the previous layer which helps in attaining a global minimum of the cost function faster and more efficiently.
- The weights are still random but differ in range depending on the size of the previous layer of neurons.

Hypterparameters:

- For weight decay : 5e-4
- For gradient clipping : 0.1

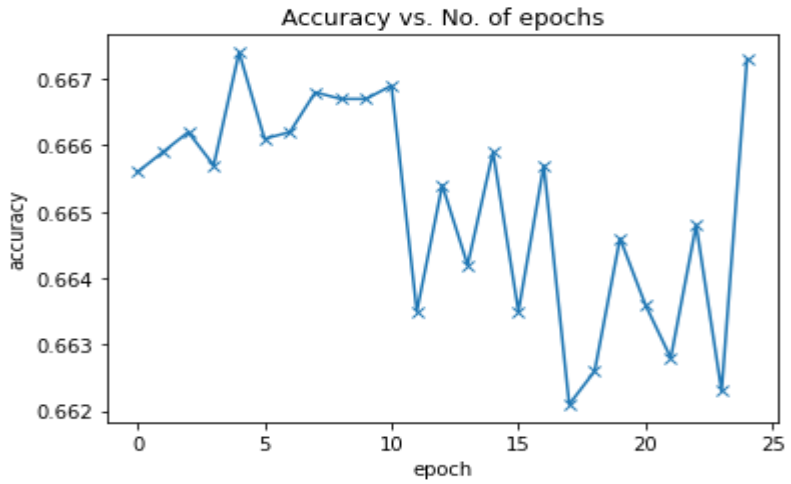
Number of epochs	Learning rate used

8	1e-3
10	1e-5
5	1e-4
15	1e-4
25	1e-5

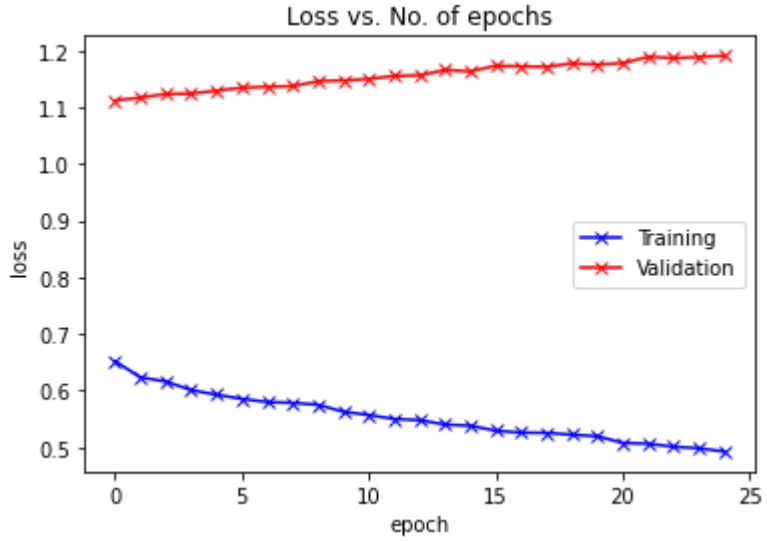
Results obtained

Accuracy : 66%
Training loss : 0.4921
Validation loss : 1.1913

Plot between Accuracy vs number of epochs



Plot between Loss vs number of epochs



Self supervised Learning

- Creating an augmented version of the cifar 100 dataset by rotating all the images by 0,90,180 or 270 degrees.
- Training the Resnet based neural network with this rotated dataset to classify images into their appropriate levels of rotation.
- Then, training the ResNet model with and without the self-supervised feature extractor on different portions of the data, to see how potential performance improvements depend on the amount of labeled data available.

Conclusion

Self supervised learning learns general features from a large distribution of data that cannot be obtained from a small labeled but non-representative subset of the data.

As that subset gets larger and larger, it becomes more and more representative and thus more conducive to learn general features that boost the performance on unseen data.

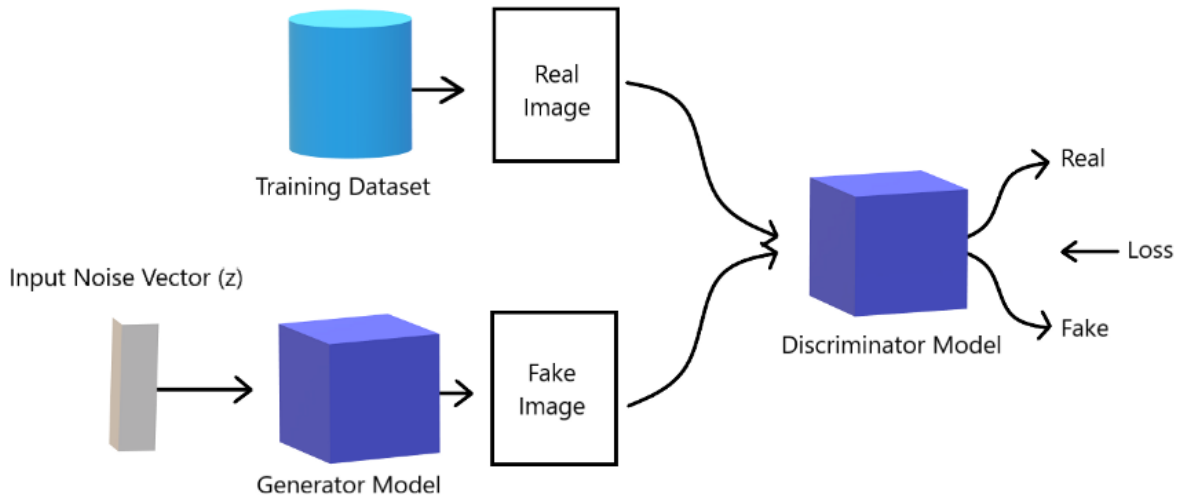
Part - 3 Image-to-Image Translation using Conditional GAN

Conditional GAN :

Conditional generative adversarial network is a type of GAN that involves the conditional generation of images by a generator model. GANs rely on a generator that learns to generate new images, and a discriminator that learns to distinguish synthetic images from real images.

Gans consists of two models, namely:

- Generator : It is a function that takes an input noise vector(z) and maps to an image that hopefully resembles the images in the training dataset.
- Discriminator : The primary purpose of the discriminator model is to find out which image is from the actual training dataset and which is an output from the generator model.

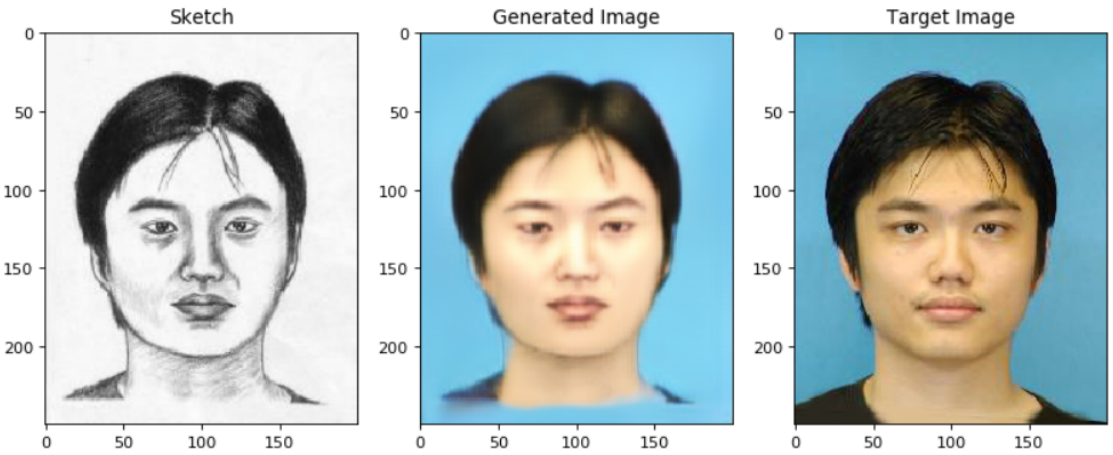


Source: https://towardsdatascience.com/generative-adversarial-networks-gans-8fc303ad5fa1

Code

- Dataset :
  - CUHK student dataset is used.
  - 88 training faces.
  - 100 testing faces.
- Data augmentation(dataPreprocessing.ipynb)
  - Reads Images from the path
  - All the training images and sketches are rotated into different angles to generate different parameters for transformation.
  - Random shift
  - Random scale

    - Random crop
    - Random flip.
- Conditional Gan (model) notebook : ConditionalGAN.ipynb
    - Need to install keras as pip install git+https://www.github.com/keras-team/keras-contrib.git
    - Functions ‘load_filename’ and ‘load_images’ reads the input images.
    - Function ‘generate_real_samples’ selects a batch of random samples, returns real images and target.
    - Function ‘generate_fake_samples’ generates a batch of images, returns images and targets.
    - In the function generator we have 2 functions ‘conv2d’ and ‘deconv2d’.
    - Function conv2d is used to downsample the image.
    - Function deconv2d is used to upsample the image.
    - Function ‘discriminator’ used to differentiate target and predicted images.
    - Function ‘train’ is used train the model
    - Function ‘summarize_performance’ is used to save the models at target location.
    - Both models are too large in size so the models are uploaded in google drive.
    - Link for d_model - link
    - Link for g_model - link
- Testing and predicting(testingImage.ipynb)
    - Loads the generator model from the Models folder.
    - Load the sketch and target images from the testing folder.
    - The model ‘g_model’ generates the photo.
    - Results :



    - Functions ‘load_filename’ and ‘load_images’ reads the input images.
    - Function ‘pred_images’ is used to generate the images.
    - The generated images are saved in GeneratedImages/Generated_Pixel[02]_Context[08] folder.

PART - 4 - Sentiment analysis on IMDB reviews

LSTM(Long Short-term Memory)

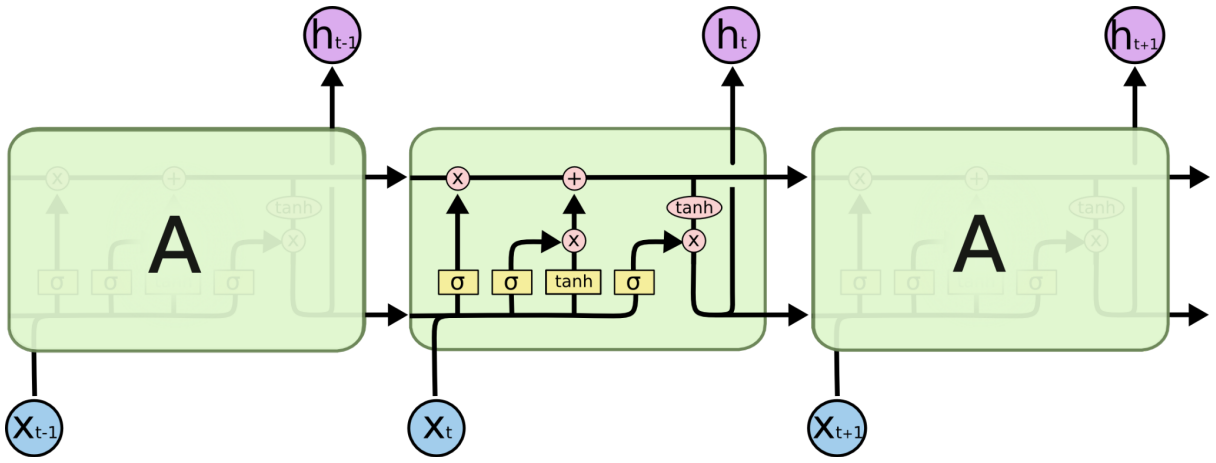
Dataset - IMDB Dataset of 50K Movie Reviews
https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

About Dataset

- Samples - 50K rows
- 25K positive and 25K negative examples

LSTM

- Long short-term memory is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate.

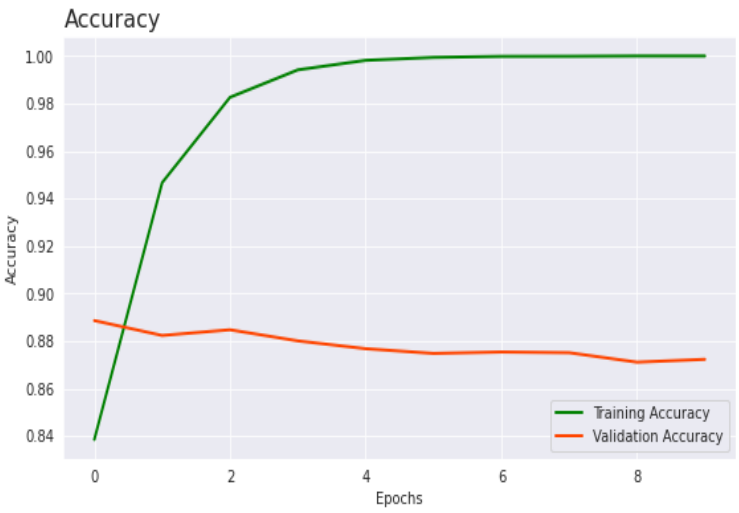


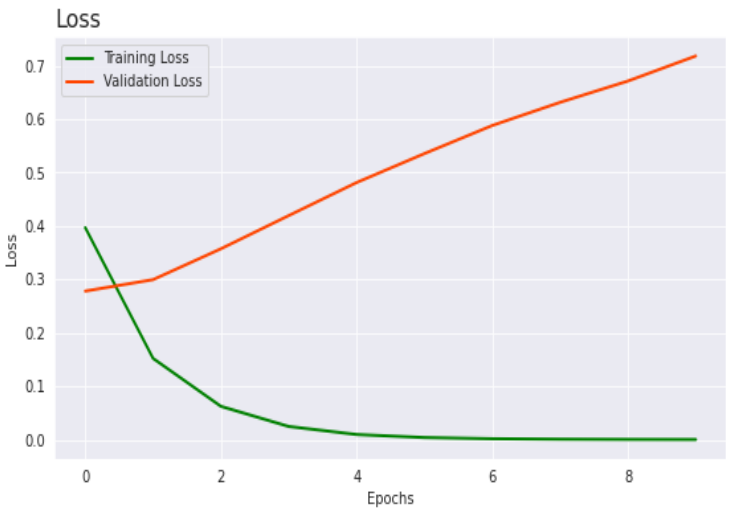
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Code

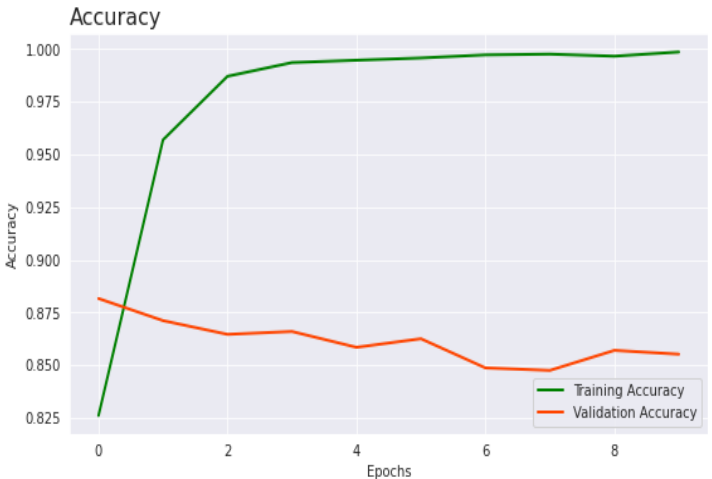
- Preprocessing
  - Removed stop words
  - Removed non alphanumeric characters
  - Removed URLs
  - Removed Html tags
  - Using Labelencoder encoded target labels with values between 0 and 1.
  - Splitted the data into train and test where 33500 training set and 16500 test set.

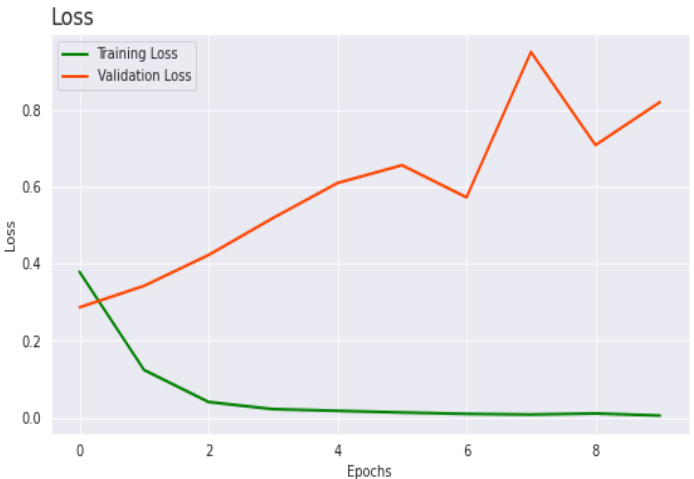
- Word Embedding
    - Embedding Layer
    - Pooling Layer - used GlobalAveragePooling1D
    - Dense Layer - with activation function as relu
    - Dense Layer - with activation function as sigmoid
    - Used binary cross entropy loss function and adam optimizer.
    - Used 10 epochs.
- Results
    - Accuracy - train 100 and test 87.23
- Plots



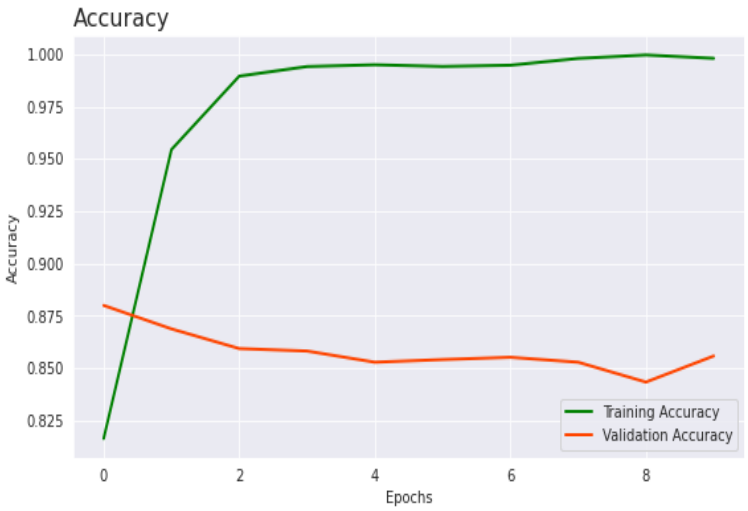


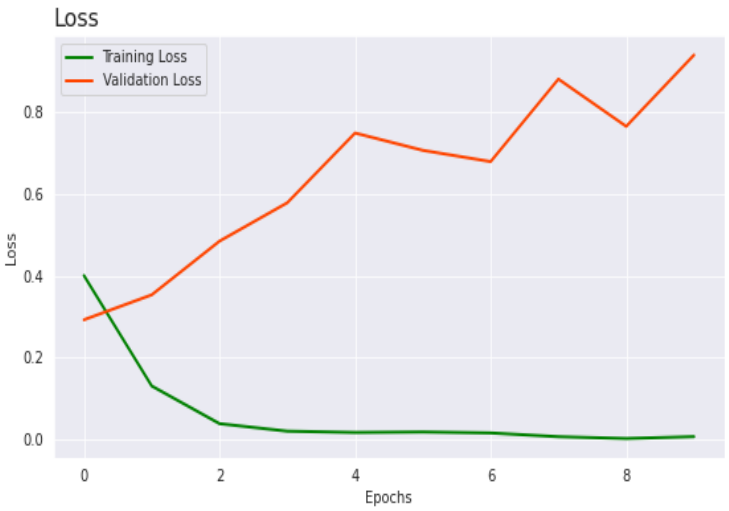
- LSTM
    - Embedding Layer
    - Bidirectional lstm - used lstm(64)
    - Dense Layer - with activation function as relu
    - Dense Layer - with activation function as sigmoid
    - Used binary cross entropy loss function and adam optimizer.
    - Used 10 epochs.
- Results
    - accuracy - train 99.8 and test 85.53
- Plots



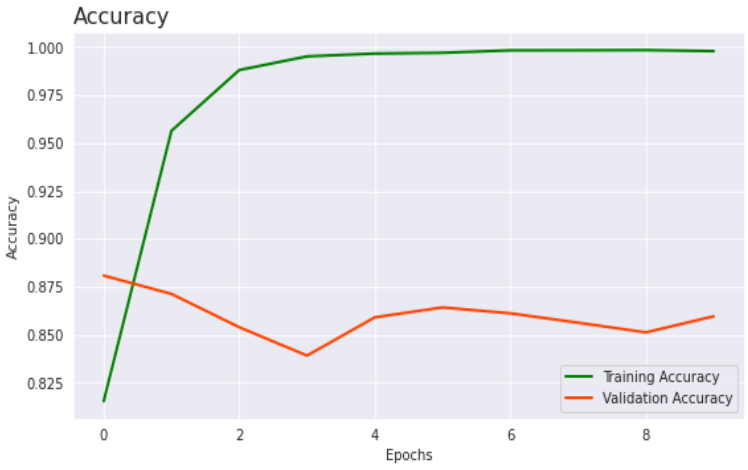


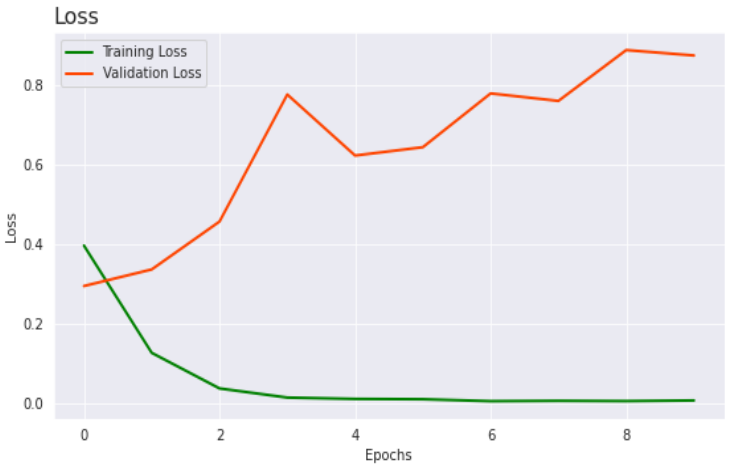
- LSTM 2 layer
    - Embedding Layer
    - Bidirectional lstm - used LSTM(64)
    - Bidirectional lstm - used LSTM(32)
    - Dense Layer - with activation function as relu
    - Dense Layer - with activation function as sigmoid
    - Used binary cross entropy loss function and adam optimizer.
    - Used 10 epochs.
- Results
    - accuracy - train 99.8 and test 85.58
- Plots



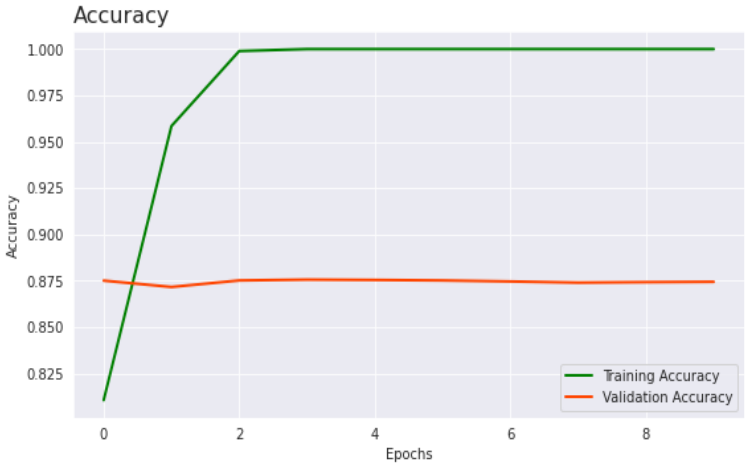


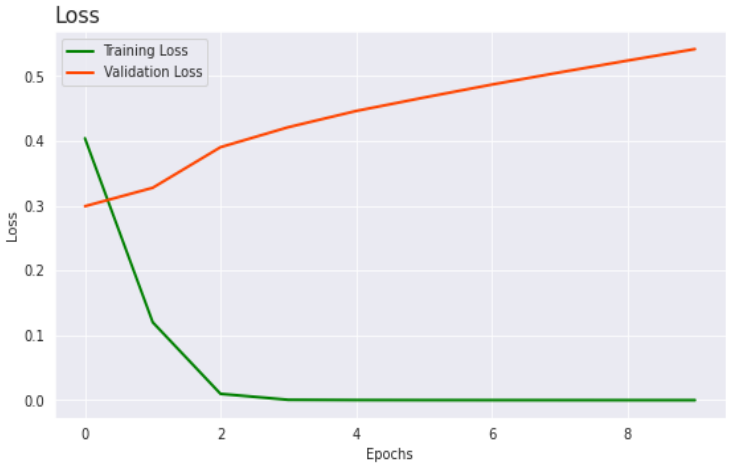
- GRU
    - Embedding Layer
    - Bidirectional GRU - used GRU(64)
    - Dense Layer - with activation function as relu
    - Dense Layer - with activation function as sigmoid
    - Used binary cross entropy loss function and adam optimizer.
    - Used 10 epochs.
- Results
    - accuracy - train 99.8 and test 85.96
- Plots





- Convolution
    - Embedding Layer
    - Conv Layer - Conv1D, activation is relu
    - Pooling Layer - GlobalMaxPooling1D
    - Dense Layer - with activation function as relu
    - Dense Layer - with activation function as sigmoid
    - Used binary cross entropy loss function and adam optimizer.
    - Used 10 epochs.
- Results
    - accuracy - train 100 and test 87.45
- Plots





Contributions

- Korupoulu Saideepthi S20180010087
    - Part-1 Object Detection Yolo ,Faster R-CNN

- Swathi Kedarasetty S20180010172
    - Part-2 Self- Supervised Learning

- Varakala Sowmya S20180010187
    - Part-3 Image-to-Image Translation using Conditional GAN

- Gendeti Manjju Shree Devy S20180010055
    - Part-4 Sentiment analysis on IMDB reviews