IIT Bhubaneswar

Indian Institute of Technology Bhubaneswar

A project report on

VIDEO ANOMALY DETECTION using AlexNet,MxNet,VGG16

Submitted for Internship
in
B. Tech in Computer Science Engineering

by

Abhishek Nanda (1805185)

under the guidance of

Prof. Debi Prosad Dogra

School of Electrical Engineering
Indian Institutes of Technology
Bhubaneswar

May 2021

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to everyone who made it possible for me to complete the report. A special thanks to Prof. Debi Prosad Dogra, for their assistance, encouraging feedback, and motivation in coordinating our project.

Prof Debi Prosad Dogra has put in a lot of effort in guiding me for achieving the goal and as well as encouraging me to keep moving forward.

ABHISHEK NANDA

ABSTRACT

In the recent decades, Video anomaly detection has become one of the most promising issue in the field of road security . In this paper, I proposed few models that include AlexNet, VGG16 & MxNet that helps in detecting anomaly in frames. Here I have used various convolution neural network (CNNs) to train the different models. The use of all the three models are done separately so that I have a proper knowledge which among the CNNs are better at different levels. All the models are trained on a single dataset which is UCSD dataset. This dataset is widely popular for its well defined anomalies and organized in folder.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The objective of the project is to determine which among the following models used is efficient and reliable. According tot he project it will help us in various instances like for maintaining peace in the road and having a secure traffic regulations.

The models are trained on various datasets before also. These can also be called as pre-trained models with high efficiency rate and accuracy rate. Here we have also used a auto-encoder to show that even the model is self sufficient for plotting anomaly.

Image processing has found multiple uses. In the field of medical and also for various types of security fields. It has a bigger role in analyzing such type of anomaly in photo frames / video.

The future of image processing with bringing an impact on road safety and security can even bring impact on the lives of the people with getting results easier.

CHAPTER 2

BACKGROUND

As the project suggest we are detecting anomaly in a video using the frames of those video. Here I have used three different CNNs for checking their efficiency on the UCSD dataset.

So the theory behind it is that we are giving a set of training data to the model and it detects the anomaly in the frames. The model is trained by setting various parameters and hyper-parameters for making it suitable to detect. After setting the parameter then we are setting the path directory to the train folder and test folder.

The path are set to the code and then all the frames are extracted from the folder specified to label it . Labeling is a process where the frames are marked if they have anomaly or not. There are two key words for labeling the frame as 1(for anomaly) & 0(for no anomaly). I have also used other ways for labeling too like using a keyword instead.

There are different pre-processing methods used in these three models to pre-train the model for the original training dataset. The frames are first converted to 2d formats and then stores in array format using numpy . It stores the image in array format so that it will be easy for the model to detect anomaly later on.

The layers used in every model is different but there are few things common in all the layers , that is max-pooling, dense layer & etc. These layers are the neural network with are connected to each other and the connectivity is assured by the dense layer.

After training the model in the training dataset , we plot the accuracy rate and loss rate from the epochs we had done earlier. Using the matplot library we plot the graph to show the efficiency of each model we have used here.

CHAPTER 3

3.1 Block Diagram:

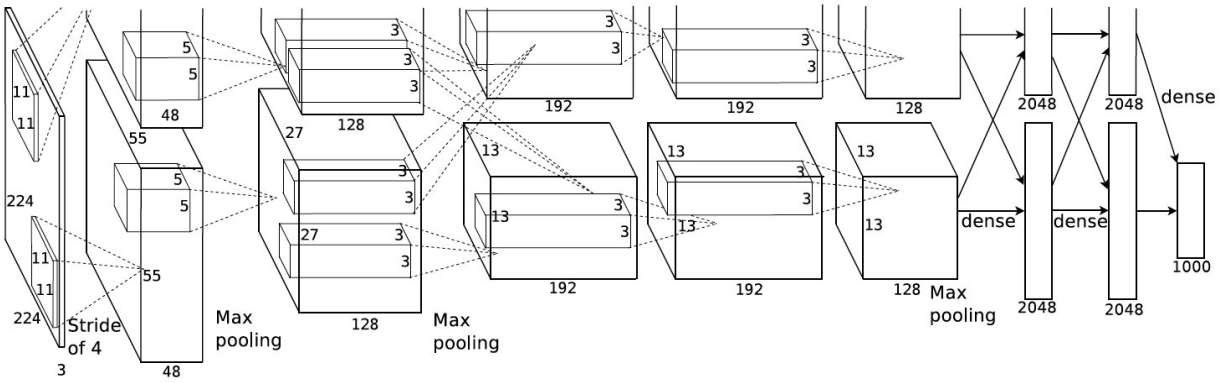
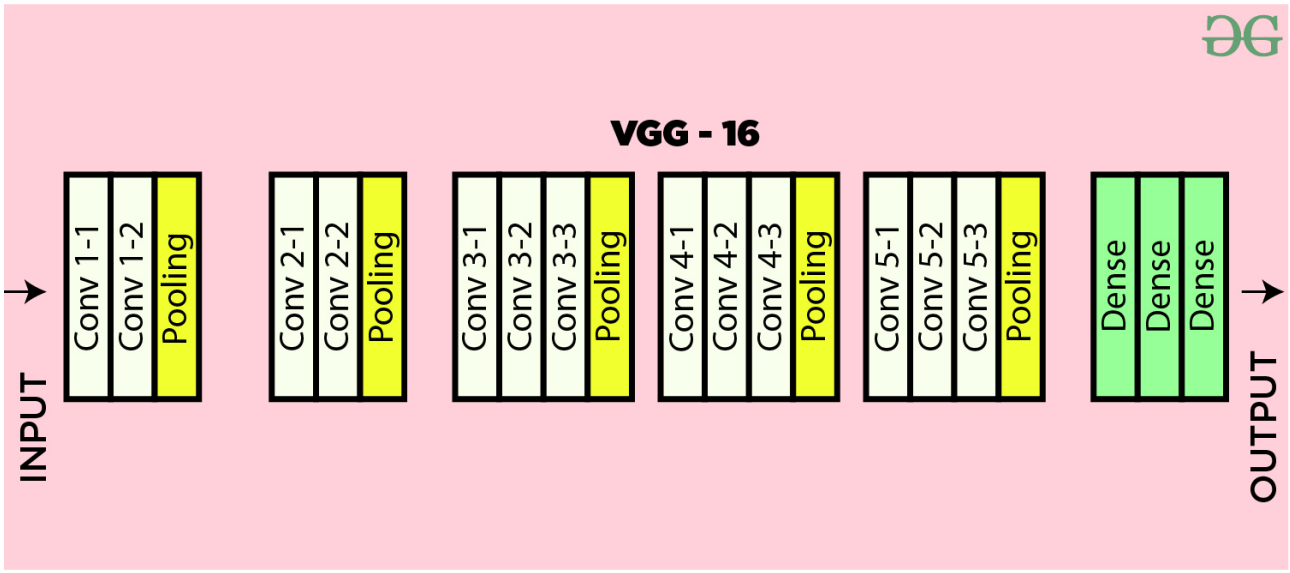


Fig.1

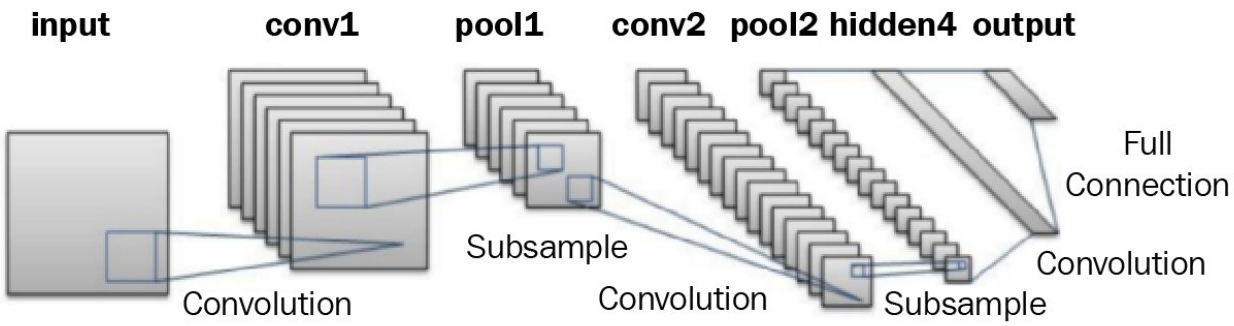
A block diagram is a diagram of a system in which the principal part or functions are represented by block connected by lines that show the relationship between the blocks. Here I have shown the architecture of AlexNet in detail. These blocks you see are the layers or you can say them as filters through which a photo/frame is passed. I have also set the various parameters for each layers so that it will act according to my desired output. There are around 11 layers in AlexNet but we don't need them all so we set our parameters according to that.

Similarly I have shown the architecture of VGG16 and MxNet below. It follows the same concept as the AlexNet but with few modifications and variations. Although the architecture is same but all the other process and optimization is different for different models.

The feture extraction process in all the three models are different but the function is same , max-pooling. Max-pooling is a process of feature extraction, in it the image with high frequency is extracted and only those pixels are used later in the model training.



Above is the architecture of VGG16 , which includes conv2D, max-pooling, dense layers. All the layers are inter-connected to each other with the help of dense layer.



Above is the architecture of MxNet, which includes conv2D, few hidden layers, dense layer and max-pooling. These layers convert the input image to sample and sub-sample images while training.

3.2 Software Stimulation:

Keras is a neural network library while TensorFlow is the open-source library for a number of various tasks in machine learning. TensorFlow provides both high-level and low-level APIs while Keras provides only high-level APIs.

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. **TensorFlow** was originally developed for large numerical computations without keeping deep learning in mind.

NumPy is a **Python** library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. **NumPy** was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

Pandas is mainly used for data analysis. **Pandas** allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. **Pandas** allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

OpenCV-Python is a library of **Python** bindings designed to solve computer vision problems. **cv2**. imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

Colaboratory, or "**Colab**" for short, is a product from **Google** Research. **Colab** allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

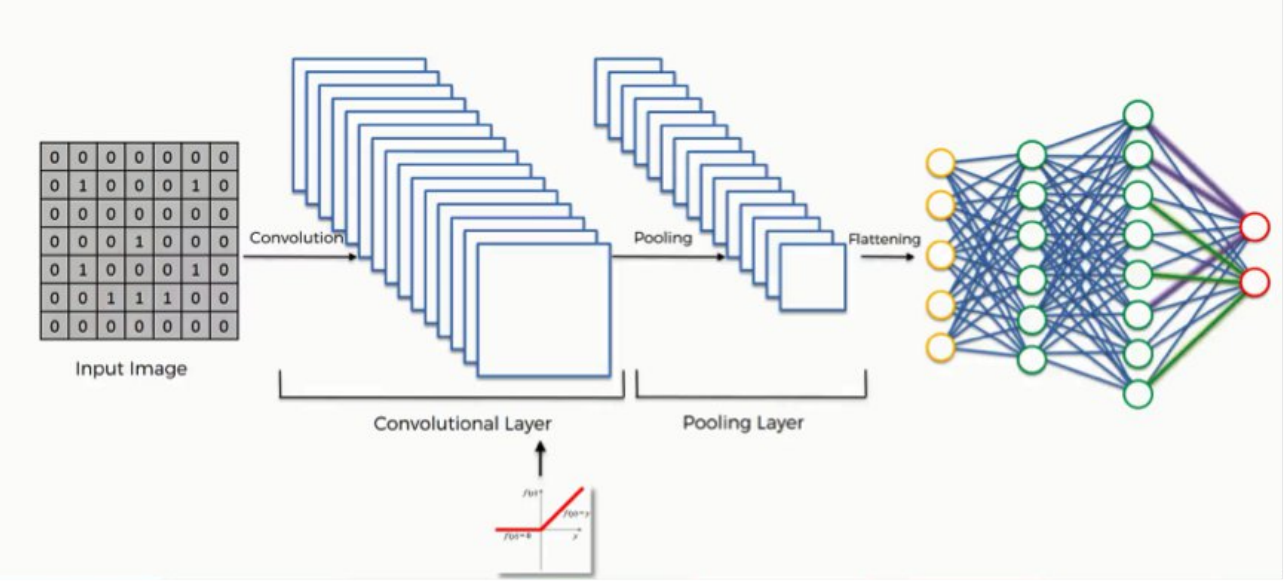
Python is a general-purpose coding language—which means that, unlike HTML, CSS, and JavaScript, it can be **used for** other types of programming and software development besides web development. That includes back end development, software development, data science and writing system scripts among other things.

Jupyter is a free, open-source, interactive web tool known as a computational **notebook**, which researchers can **use** to combine software code, computational output, explanatory text and multimedia resources in a single document.

Anaconda Navigator is a desktop graphical user interface (GUI) included in **Anaconda**® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on **Anaconda**.org or in a local **Anaconda** Repository.

3.3 Algorithm:

The algorithm used is simple as used in any other ML projects. Here we send the frames of the video to the model which converts the frames to its desirable size and array . Then the frames are converted to batch using batch normalization. Now these batches are send to the model for training . The model is pre-defined with the parameters which we have set . We have used various activation function to make our model know which type of training it has to do if the image is of something different than usually send.



Above is the simple explanation of the algorithm used in CNN. There are various activation function like “reLU” , “softmax” & etc. As I have worked on three different models so the I have used these two activation function in my model.

There is also a concept known as feature extraction, here the model tries to gain knowledge from the training dataset which later on it used during the testing dataset. It can also mean collecting data from the training phase.

3.4 Social and Environmental Impact:

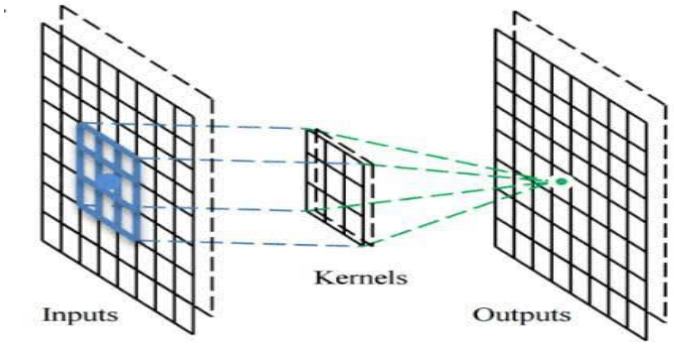
Keeping in mind the current situation of the machine learning world, this project might add a few good deeds to it. This project is indeed a eco-friendly project and also it will help in maintaining the traffic peace. Using this project we can detect if someone is violating any traffic rules or not. Social impact being that the citizens will be aware of it and they will follow the rules properly.

CHAPTER 4

4.1 Result and Discussions:

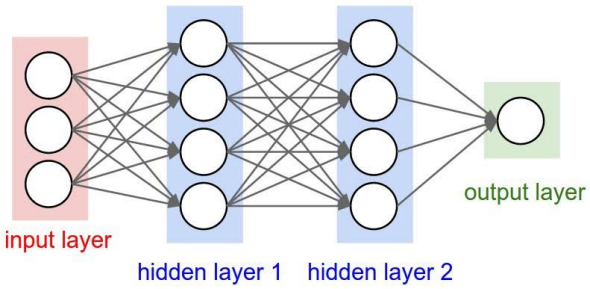
Layers used:

CONV2D: This layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs. They are generally smaller than the input image and so we move them across the whole image.As,it is easy to process in a faster manner.



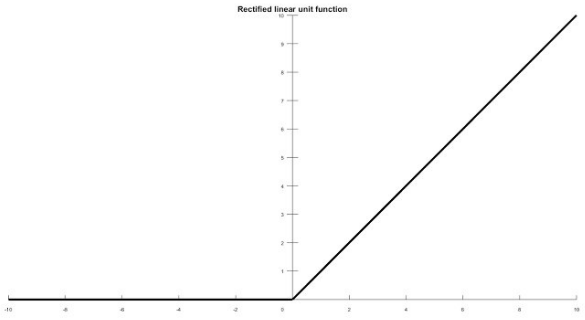
{Fig 4.1}

BATCH NORMALISATION:This allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers.This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep network. Regularizes the model and reduces the need for dropout.



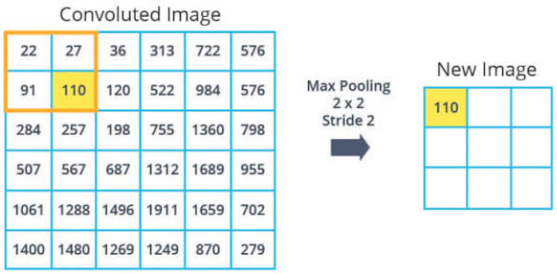
{Fig 4.2}

reLU:The rectified linear activation function or reLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It overcomes the vanishing gradient problem, allowing models to learn faster and perform better.



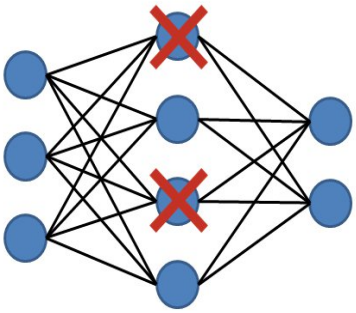
{Fig 4.3}

Max-pool:It is a down sampling layer .This operation selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map. Background in these images is made black to reduce the computation cost and reducing the noise.

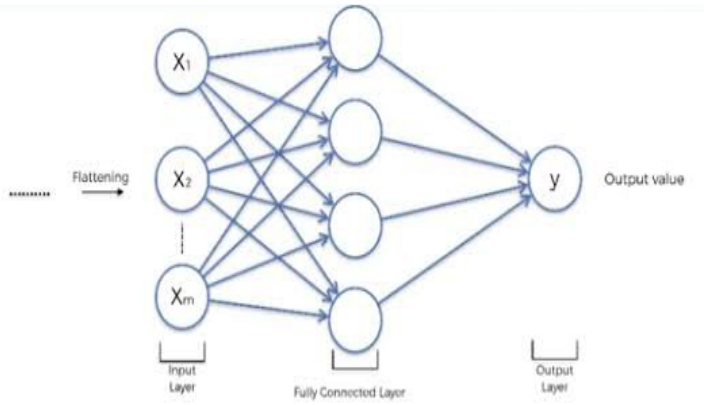


{Fig 4.4}

DROPOUT: This prevents our sequential model from over fitting.It deactivates some of the neurons randomly.This effects the network to be less sensitive.



DENSE: Adds the fully connected layer to the neural network.A Dense layer feeds all outputs from the previous layer to all its neurons, each neuron providing one output to the next layer. It's the most basic layer in neural networks.It classifies the class.

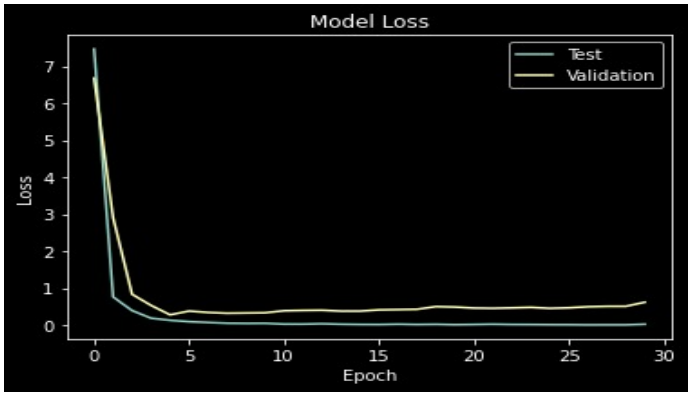


{Fig 4.7}

EPOCH: An epoch refers to one cycle through the full training data-set. Since one epoch is too big to feed to the computer at once we divide it in several smaller batches.One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

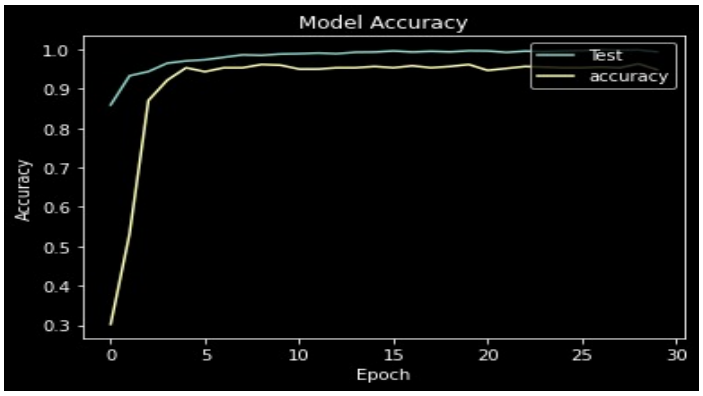
4.2 ANALYSIS:

Model Loss:



Loss & Validation Loss: A loss function quantifies how "good" or "bad" a given predictor is at classifying the input data points in a data set. The smaller the loss, the better a job the classifier is at modelling the relationship between the input data and the output targets. Training loss is measured during each epoch while validation loss is measured after each epoch.

Model Accuracy:



Accuracy & Validation Accuracy: A accuracy function determines the efficiency of the model after the model is trained. The more the accuracy the better the job the classifier is at modelling the relationship between the input data and the output target. Training accuracy is measured during each epoch while validation accuracy is measured after each epoch.

Model Building:

Table 4.1 showing details related to Model Building of VGG16 -

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 128)     49280
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              102764544
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2002
=================================================================
Total params: 138,334,970
Trainable params: 138,334,970 ,Non-trainable params: 0
```

Table 4.2 showing details related to Model Building of AlexNet -

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
batch_normalization (BatchNo (None, 54, 54, 96)        384
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 26, 26, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 256)       614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 256)       1024
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 384)       885120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 384)       1327488
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 256)       884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              26218496
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2002
_________________________________________________________________
activation (Activation)      (None, 2)                 0
=================================================================
Total params: 50,847,418
Trainable params: 50,846,714
Non-trainable params: 704
```

Epoch:

Table 4.2 showing details related to Epoch

Epoch 2/30
61/61 [==============================] - 88s 1s/step - loss: 13.0169 - accuracy: 0.8059 - val_loss: 6.6748 - val_accuracy: 0.3023
Epoch 2/30
61/61 [==============================] - 84s 1s/step - loss: 0.9563 - accuracy: 0.9258 - val_loss: 2.9055 - val_accuracy: 0.5316
Epoch 3/30
61/61 [==============================] - 81s 1s/step - loss: 0.4489 - accuracy: 0.9350 - val_loss: 0.8319 - val_accuracy: 0.8704
Epoch 4/30
61/61 [==============================] - 81s 1s/step - loss: 0.1590 - accuracy: 0.9652 - val_loss: 0.5311 - val_accuracy: 0.9219
Epoch 5/30
61/61 [==============================] - 88s 1s/step - loss: 0.1504 - accuracy: 0.9689 - val_loss: 0.2765 - val_accuracy: 0.9535
Epoch 6/30
61/61 [==============================] - 85s 1s/step - loss: 0.0961 - accuracy: 0.9757 - val_loss: 0.3779 - val_accuracy: 0.9435
Epoch 7/30
61/61 [==============================] - 82s 1s/step - loss: 0.0621 - accuracy: 0.9793 - val_loss: 0.3399 - val_accuracy: 0.9535
Epoch 8/30
61/61 [==============================] - 81s 1s/step - loss: 0.0529 - accuracy: 0.9826 - val_loss: 0.3199 - val_accuracy: 0.9535
Epoch 9/30
61/61 [==============================] - 83s 1s/step - loss: 0.0469 - accuracy: 0.9800 - val_loss: 0.3257 - val_accuracy: 0.9618
Epoch 10/30
61/61 [==============================] - 85s 1s/step - loss: 0.0425 - accuracy: 0.9882 - val_loss: 0.3316 - val_accuracy: 0.9601
Epoch 11/30
61/61 [==============================] - 85s 1s/step - loss: 0.0240 - accuracy: 0.9898 - val_loss: 0.3863 - val_accuracy: 0.9502
Epoch 12/30
61/61 [==============================] - 85s 1s/step - loss: 0.0292 - accuracy: 0.9901 - val_loss: 0.3957 - val_accuracy: 0.9502
Epoch 13/30
61/61 [==============================] - 85s 1s/step - loss: 0.0259 - accuracy: 0.9910 - val_loss: 0.4007 - val_accuracy: 0.9535
Epoch 14/30
61/61 [==============================] - 85s 1s/step - loss: 0.0230 - accuracy: 0.9931 - val_loss: 0.3767 - val_accuracy: 0.9535
Epoch 15/30
61/61 [==============================] - 87s 1s/step - loss: 0.0193 - accuracy: 0.9915 - val_loss: 0.3768 - val_accuracy: 0.9568
Epoch 16/30
61/61 [==============================] - 86s 1s/step - loss: 0.0125 - accuracy: 0.9971 - val_loss: 0.4102 - val_accuracy: 0.9535
Epoch 17/30
61/61 [==============================] - 88s 1s/step - loss: 0.0259 - accuracy: 0.9899 - val_loss: 0.4178 - val_accuracy: 0.9585
Epoch 18/30
61/61 [==============================] - 86s 1s/step - loss: 0.0161 - accuracy: 0.9966 - val_loss: 0.4263 - val_accuracy: 0.9535
Epoch 19/30
61/61 [==============================] - 85s 1s/step - loss: 0.0204 - accuracy: 0.9923 - val_loss: 0.4991 - val_accuracy: 0.9568
Epoch 20/30
61/61 [==============================] - 85s 1s/step - loss: 0.0105 - accuracy: 0.9945 - val_loss: 0.4872 - val_accuracy: 0.9618
Epoch 21/30
61/61 [==============================] - 85s 1s/step - loss: 0.0142 - accuracy: 0.9964 - val_loss: 0.4587 - val_accuracy: 0.9468
Epoch 22/30
61/61 [==============================] - 85s 1s/step - loss: 0.0193 - accuracy: 0.9936 - val_loss: 0.4521 - val_accuracy: 0.9518
Epoch 23/30
61/61 [==============================] - 84s 1s/step - loss: 0.0252 - accuracy: 0.9924 - val_loss: 0.4642 - val_accuracy: 0.9568
Epoch 24/30
61/61 [==============================] - 84s 1s/step - loss: 0.0119 - accuracy: 0.9959 - val_loss: 0.4779 - val_accuracy: 0.9551
Epoch 25/30
61/61 [==============================] - 85s 1s/step - loss: 0.0090 - accuracy: 0.9962 - val_loss: 0.4512 - val_accuracy: 0.9535
Epoch 26/30
61/61 [==============================] - 84s 1s/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.4657 - val_accuracy: 0.9535
Epoch 27/30
61/61 [==============================] - 84s 1s/step - loss: 0.0034 - accuracy: 0.9996 - val_loss: 0.4950 - val_accuracy: 0.9551
Epoch 28/30
61/61 [==============================] - 87s 1s/step - loss: 0.0063 - accuracy: 0.9981 - val_loss: 0.5064 - val_accuracy: 0.9535
Epoch 29/30
61/61 [==============================] - 90s 1s/step - loss: 0.0024 - accuracy: 0.9997 - val_loss: 0.5071 - val_accuracy: 0.9635
Epoch 30/30
61/61 [==============================] - 88s 1s/step - loss: 0.0179 - accuracy: 0.9953 - val_loss: 0.6171 - val_accuracy: 0.9485

This list shows all the epochs we have run in the model for training it. Here we can see that the loss rate is decreasing and the accuracy rate is increasing and this is a good sign. The less the loss rate and high the accuracy rate, it is better for saying that our model is efficient. At the end the accuracy is 0.9979 and the loss rate is 0.0063.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this daily developing world , there are a lot of scopes for Machine Learning as their need in this world will never be less. Looking into the future i can say that developers have a lot of things to achieve . I as a developers feel the same way. This project of mine is my first step towards making the virtual world a better and safe place for every generations to come. I have learned many new techniques during this project and tried to understand each and every problem with a great spirit . I hope this project makes a bit difference in your experience of learning machine and their use.

5.2 Planning And Project Management:

Table 5.1 showing details about project planning and management

Activity	**Starting week**	**Number of weeks**
Planning	1st week of DEC	4
Research	1st week of JAN	3
Software Stimulation	4st week of JAN	1
Training & Testing	1nd week of FEB	8
Implementation	1st week of APR	4
Deployment	1rd week of MAY	2
Project Report	3nd week of MAY	2

The Gantt chart is shown below:

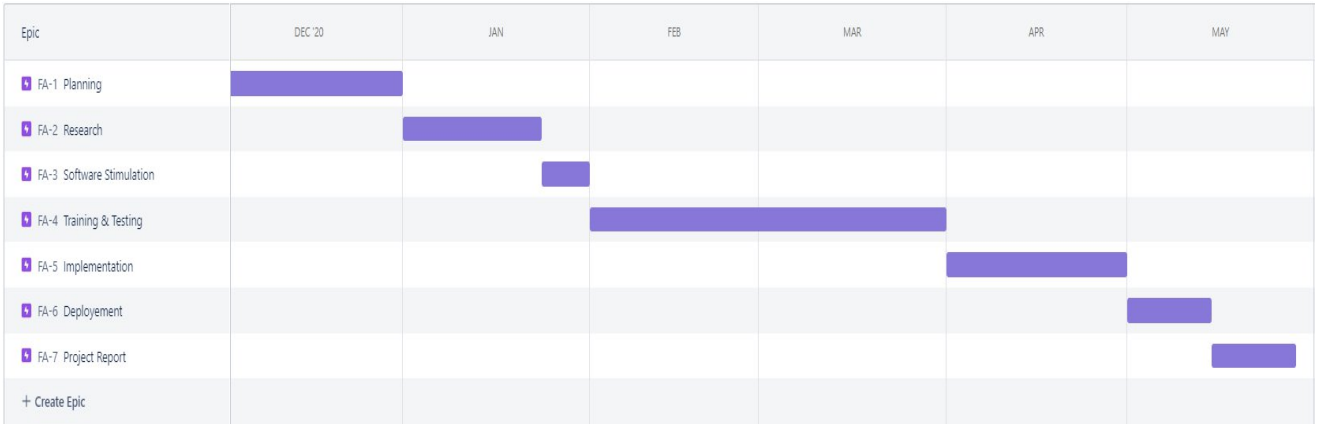


Fig.5

REFERENCES

1. https://github.com/AbhishekNanda7429/Video-Anomaly-Detection.git
2. http://www.svcl.ucsd.edu/projects/anomaly/dataset.html
3. https://keras.io/guides/
4. https://www.tensorflow.org/guide
5. https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook
6. https://docs.anaconda.com/anaconda/user-guide/getting-started/
7. https://youtube.com/playlist?list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU