A Project Report
on

Violence Detection in Real Life Videos using Convolutional Neural Networks

Submitted in partial fulfillment of the
requirement for the award of the degree of

B.Tech Computer Science and Engineering with Specialization in Artificial Intelligence & Machine Learning

GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Dr. B. Balamururgan
Professor
Department of Computer Science and Engineering

Submitted By

NANDINI BAGGA
18021180008

GAJANAND SINGH
18021180028

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"Violence Detection in Real Life Videos using Convolutional Neural Networks"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY-2021 to DECEMBER-2021**, under the supervision of **Dr. B. Balamurugan, Professor, Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other place.

18SCSE1180008 - NANDINI BAGGA
18SCSE1180029 - GAJANAND SINGH

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor

(Dr. B. Balamurugan, Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of 18SCSE1180008 - NANDINI BAGGA, 18SCSE1180029 - GAJANAND SINGH has been held on __________________ and his/her work is recommended for the award of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.

Signature of Examiner(s) Signature of Supervisor(s)

Signature of Project Coordinator Signature of Dean

Date:

Place:

Abstract

Detecting violence in video recordings through artificial intelligence is critical for law requirements and also keeping track of public security with the help of surveillance cameras. Additionally, it very well might be an incredible tool for protecting children from accessing inappropriate content and assist guardians with settling on a better choice with regards to what their children should watch. This is a difficult issue since the actual meaning of violence is expansive and exceptionally abstract. Subsequently, detecting such subtleties from recordings with no human management isn't just technical, yet in addition a theoretical issue. In light of this, in this work we will investigate how to more readily depict the idea of violence using a convolutional neural network.

At first by breaking it into more even handed and concrete related ideas, like fights, explosions, blood, and so on, for later combining them in a meta-order to depict violence. We will likewise investigate approaches to address time-based events for the network, since numerous violent demonstrations are portrayed in terms of development. And at long last we will investigate how to localize violent events, since numerous video transfers don't contain just violence, yet are a combination of violent and non-violent scenes.

There are a lot of pretrained convolutional neural networks which can be used for image classification. These models have learned to extract powerful and informative features from natural images and use it as a starting point to learn a new task. These networks have been trained on more than a million images and can classify images into 1000 object categories. We will be using a pretrained network with transfer learning because it's typically much faster and easier than training a network from scratch. For this project we will be exploring which model will give the best accuracy for detecting violence in videos using various deep learning techniques.

We have taken a dataset of 500 videos of violence and non-violence each. We have tried video classification using two pre-trained CNNs InceptionV3 and MobilenetV2 models. Out of these two models MobileNetV2 gave a better result. InceptionV3 gave an accuracy of 60% while MobilenetV2 gave a validation accuracy of 90%. We were able to reduce the loss using the MobileNetV2 model.

In future we want to make better use of PCA and validation techniques in order to improve the accuracy and reduce loss. We can also extend this project to violence recognition in crowds. The model can give better results when it's trained with more data.

Keywords: Violence detection; violence recognition; deep learning; convolutional neural network; inception v3; MobileNetV2

List of Figures

List of Tables

Table of Contents

CHAPTER-1

Introduction

The acknowledgment of human activities from surveillance videos has turned into a functioning and reformist research area in machine learning. The classification of media content as recordings depends on human activity, which portrays general human conduct. Human conduct and activities are perceived dependent on various video includes that arrange activities as typical i.e. Normal or strange i.e. abnormal. Movements of every kind of regular daily existence, including strolling, running on the ground, eating food, plunking down and ascending from a seat, lying in bed, picking a thing from a table or floor, and diving steps, are called ordinary activities. Abnormal activities, additionally called dubious activities, veer off from normal human activities [1]. The actions are abnormal for one situation but may be considered normal for another situation. For example, running on a playground is normal, yet running in a bank or a commercial center is considered unusual. The most pivotal and critical unusual exercises are brutal exercises that truly portray activities to cause mischief or harm with forceful practices. Fighting, killing, and beating somebody are the most well-known instances of savagery in public places.

Violent behavior in public settings poses a real threat to individual safety and society stability. Currently, a great amount of equipment is deployed in public locations, putting a huge pressure on security personnel. As a result, identifying violent occurrences from vast amounts of surveillance video data [2] is critical. Violent activities are physically examined through the screen of a surveillance camera in a semi-computerized system. This isn't useful because regular observation is required, yet it is inconvenient to constantly watch screens to see fierce exercises. Because such workouts can occur at any time, there is no space for error when performing them. It is important to convert such a semi-robotized system into a completely automated system capable of identifying and detecting aggressive behaviors without human intervention. Fully mechanized systems, as opposed to semi-robotized systems, are more capable and capable of detecting object developments and perceiving human action thanks to computer vision and AI. Due to a variety of circumstances such as real-time classification, low video quality of security cameras, and fluctuating light intensity during monitoring, detecting human action is a difficult process.

There is a demand for an effective and easy to use violence detection tool which can be integrated with live surveillance cameras so that detection of violence becomes automated and in this way we can take a step forward in preventing violence. A violence detection tool can help reduce violence in various situations like, domestic violence, public violence, violence in crowds, violence at different institutions like schools, colleges, law and enforcement departments, etc.

Violent crimes shouldn't go unnoticed, there should be a notification system for when violent behavior is identified, it should be notified to the nearest station. It will be really helpful for people who cannot speak for themselves due to fear and other situational reasons, like differently abled people, youngsters and old-aged individuals. Studies have identified the fact that violent activities take a toll on mental health. We should make sure that we protect the mental health of everyone, especially when at a young age, traumatic experiences can haunt you for life. AI has provided ways, many in number, to automate the strenuous activities that would otherwise take human presence and hours of work to accomplish a simple but

important task at hand. Computer vision is a branch of deep learning and artificial intelligence which allows computers to see the world through the lens of cameras, for recognition, classification and detection of objects and classes.
We propose in this paper to use a pre-trained 3D Convolutional Neural Network called MobileNetV2 in conjunction with InceptionV3. This work makes the following unique contributions to the best in class in violence detection:

● It demonstrates how a deep neural network, pre-trained on datasets that are not expected to be used for violence detection, can be used to calculate video feature descriptors, which can subsequently be used to train a classifier to distinguish between videos that are violent and non-violent.

● It provides a system capable of detecting both individual-to-individual and group-to-individual violence with great precision.

● Through correlation tests on three common datasets, it depicts the improvement in exactness as for existing algorithms.

Our proposed framework for detecting violence using a deep learning-based technique: - In this Proposed System the framework takes the video as input and generates it into frames. Afterwards the keyframe extraction technique will be used to eliminate or delete consecutive duplicate multiple frames. These extracted frames are then used for training the DL models. Convolutional neural networks and Inceptionv3 deep learning architectures were used for feature selection and classification.

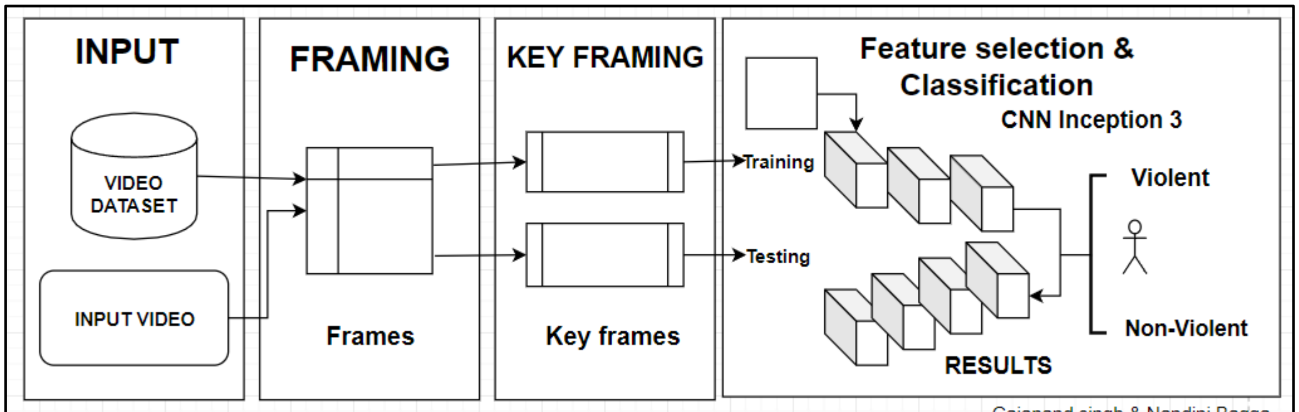


Figure 1: Proposed framework for violence detection using DL

The proposed violence detection method's framework is taking the input video, extracting frames from the video, performing image augmentations on the frames, separating the labelled image data into a training and a test set,Finally, using the training and testing data, the CNN model is trained to attain the desired loss and accuracy.

A video stream is acquired in the first phase, and frames are extracted. In the next phase, a 3D CNN model is feeded a selected series of frames and deep features are extracted that recognises violent behavior. Detection in Videos is different from detection in images as we have to store the predictions of a few images in a queue and then calculate the probability of violence or non-violence, based on that probability value we can predict the category of the frames.

This method of storing the prediction values of a few frames in a queue in order to predict the category of the frames is done to avoid fluctuations in prediction. If we consider the prediction value of each frame then there is a high change of fluctuations in prediction in the output video.

Our output video clearly shows whether the incidents in the video are violent or non-violent which can be noticed by the label on the top left corner. Finally, if a violent behavior is identified, we may notify the nearest station so that prompt action can be taken before any injury or disaster occurs.

CHAPTER-2

Literature Survey

Traditional machine learning algorithms such as Adaboost, KNN, and others have been used as a classifier in violence detection using machine learning techniques. Motion blobs are considered to have specified places and forms in "Fast Violence Detection" [2]. It outperformed state-of-the-art approaches that use a classifier like KNN, SVM, or Adaboost. RIMOC [3] (Rotation-Invariant Feature Modeling Motion Coherence) is a novel and problem-specific approach. Two sites were used to construct a dataset: in-lab simulated trains and real-world train stops. The proposed strategy beat current state-of-the-art methods in terms of false-positive rate. Motion-based analysis was given to detect conflicts in a natural manner [4].Using the Bag of words approach, a collection of visual words is formed.. The histogram is then utilised as a vector to find aggression among the visual words.

For crowd aggression detection, a spatio-temporal model based Lagrangian direction field has been developed [5]. Violent crowds, hockey fights, and movie violence have all been employed as data sources. The proposed method for detecting breaking violence in crowded scenes [6] investigates statistics for how vector flow varies over time. When it comes to 2D CNNs, they solely supply spatial data. A 3D CNN model outperforms the competition, with a 91 percent accuracy rate [7]. Spatial-temporal interest points is a widely used method for recognising and classifying actions in films. It also has a feature that incorporates audio from the videos. It considers characteristics from both the geographical and temporal dimensions [8]. It has been claimed that, pretrained imagenet models like ResNet50, VGG19, VGG16 when used for extraction of features, only one of the models, that is ResNet50 is proved to be effective in detecting violence [10]. Action movies video dataset has been really useful in detecting fight actions and non-fight actions with 90 percent accuracy using the traditional STIP and MoSIFT methods along with bag-of-words model network [11]. 3D ConvNets have been used on the hockey dataset and showed superior performance with automatic feature learning [12]. A few researchers used LSTM as a temporal features extraction method [10, 13]. LSTM - Long Short Term Memory networks are well suited for time series data prediction. That’s why in the case of video datasets, by taking temporal features from the data LSTM can be put to use for prediction and for the purpose of classification. Handcrafted networks are also of interest in this research field, FighNet is trained with different kinds of formalities and showed better robustness and accuracy score [14].

CHAPTER-3

Functionality/ Working of Project

Detection of violence in real-life videos has taken a global lead. In this work, we intend to utilize MobileNetV2 CNN on data from 1000 violent videos for efficient detection and prediction accuracy of 90.26 percent.

3.1. Dataset

Our goal is to work on a novel video dataset built exclusively for testing violence detection algorithms, in which both routine and violent actions occur under similar, dynamic conditions. We have used a video dataset from Kaggle [9]. This dataset is a real life video dataset of violence and non-violence containing 1000 videos of each category. Many real street combat situations in various environments are depicted in the violence videos and conditions and Nonviolence videos are compiled from a variety of human activities such as sports, eating, strolling, and so on.

3.2. Extracting Frames

For recognition or detection in videos we need to extract image frames from the video first so that we can use those frames as a training dataset. You can either extract frames from the videos as images and store them in a file folder and later use computer vision library OpenCV for reading those images or you can directly read the images from the video using OpenCV library features and store the image data in a list. We have tried both the methods and the latter is more efficient, where we directly store the image data in a list.

3.3. MobileNetV2

Depth-wise separable convolution was implemented in MobileNetV1, to reduce the number of parameters and computations. Depth wise separable convolution is made up of two types of convolutions: pointwise and depth wise. Depth wise convolution spatially filters each input channel with a convolution filter, whereas pointwise convolution mingles the channels by performing 1 x 1 convolution to the depth wise convolution output.

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity. As a whole, the architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers. The architecture delivers high accuracy results while keeping the parameters and mathematical operations as low as possible to bring deep neural networks to mobile devices. Keras includes a number of pretrained networks ('applications') that you can download and use straight away. One of these is MobileNetV2, which has been trained to classify images. Neural networks are picky when it comes to the kind of input that they expect. If you provide input in a format that the network doesn't expect, then the predictions won't make sense (if the structure of the input is superficially compatible with the network) or the code will simply crash (if the structure of the input is incompatible).

MobileNetV2 expects images of 224 × 224 pixels with three color channels. In other words, it expects input of shape (224, 224, 3). You can pass multiple images at once to the model.

MobileNetV2 is very similar to the original MobileNet, except that it uses inverted residual blocks with bottlenecking features. It has a drastically lower parameter count than the original MobileNet. MobileNets support any input size greater than 32 x 32, with larger image sizes offering better performance.

You can simply import MobileNetV2 from keras.applications and create an instance of it. The first time that you do this, the network will be downloaded from the internet.

If you don't pass any keywords to MobileNetV2(), then the network will have random weights; that is, you will get the architecture of the network, but not the weights, and you will therefore have to train it yourself. By specifying weights='imagenet', you indicate that you want the network to be pretrained.

MobileNetV2 introduced two main methods: 1. linear bottleneck, 2. inverted residual blocks. The input channel dimension is enlarged in the linear bottleneck layer to lessen the possibility of information loss due to nonlinear functions like ReLU.

We have used sigmoid activation function as there are only two classes, violence and non-violence, and we have used MobileNetV2 model with Adam optimizer.

3.4. InceptionV3

The Inception network, on the opposite hand, is complex. It defines various things in its architecture to improve performance, in terms of speed and accuracy both. Deep neural networks are computationally expensive. To make it computationally cheaper, the authors add an extra 1x1 convolution, in order to try to limit the number of input channels. To improve on the InceptionV2, possibilities were to be investigated without drastically changing the modules. InceptionV3 utilises a Root Mean Squared Propagation optimizer in the auxiliary classifiers, as well as factors such as access 7x7 convolutions and BatchNorm.

Inception v3 is a convolutional neural network for assisting in image analysis and object detection, and got its start as a module for Googlenet. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge. Inception v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years.

Inception v3 was trained on ImageNet and compared with other contemporary models, as shown below. As shown in the table, when augmented with an auxiliary classifier, factorization of convolutions, RMSProp, and Label Smoothing, Inception v3 can achieve the lowest error rates compared to its contemporaries.

Since InceptionV3 is usually preferred for huge datasets, it didn’t perform well in our case. We observed overfitting in the model while using the Inception Network model.

The architecture of an Inception v3 network is progressively built, step-by-step, as explained below:

1. Factorized Convolutions: this helps to reduce the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.

2. Smaller convolutions: replacing bigger convolutions with smaller convolutions definitely leads to faster training. Say a 5 × 5 filter has 25 parameters; two 3 × 3 filters replacing a 5 × 5 convolution has only 18 (3*3 + 3*3) parameters instead.

3. Asymmetric convolutions: A 3 × 3 convolution could be replaced by a 1 × 3 convolution followed by a 3 × 1 convolution. If a 3 × 3 convolution is replaced by a 2 × 2 convolution, the number of parameters would be slightly higher than the asymmetric convolution proposed.

4. Auxiliary classifier: an auxiliary classifier is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In GoogLeNet auxiliary classifiers were used for a deeper network, whereas in Inception v3 an auxiliary classifier acts as a regularizer.

5. Grid size reduction: Grid size reduction is usually done by pooling operations. However, to combat the bottlenecks of computational cost, a more efficient technique is proposed:

3.5. Implementation Details

Our networks were built using the Jupyter Notebook. The photographs in the input were resized to a fixed size of 128 x 128 pixels. To avoid duplication of frames it was set for each video to guarantee that The input frame count was a multiple of seven., and frames were placed at regular intervals. Due to the differing lengths of films in various databases, the time intervals between frames vary. InceptionV3, MobileNetV2, and CNN were chosen as backbones. We also employed image augmentations like brightness jittering, random horizontal flips, zooming and rotations, to avoid model overfitting and to make the model more robust. The labelled picture data is separated into two sets: training and test.

The proposed violence detection method's framework is taking the input video, extracting frames from the video, performing image augmentations on the frames, separating the labelled image data into a training and a test set. Finally, using the training and testing data, the CNN model is trained to attain the desired loss and accuracy. A video stream is acquired in the first phase, and frames are extracted. In the next phase, a 3D CNN model is fed a selected series of frames and deep features are extracted that recognises violent behaviour. Detection in Videos is different from detection in images as we have to store the predictions of a few images in a queue and then calculate the probability of violence or non-violence, based on that probability value we can predict the category of the frames. This method of storing the prediction values of a few frames in a queue in order to predict the category of the frames is done to avoid fluctuations in prediction. If we consider the prediction value of each frame then there is a high change of fluctuations in prediction in the output video. Our output video clearly shows whether the incidents in the video are violent or non-violent which can be noticed by the label on the top left corner.

Finally, if a violent behaviour is identified, we may notify the nearest station so that prompt action can be taken before any injury or disaster occurs.

3.6 Code:

```
import os
import platform
from IPython.display import clear_output
print(platform.platform())

def resolve_dir(Dir):
    if not os.path.exists(Dir):
        os.mkdir(Dir)

def reset_path(Dir):
    if not os.path.exists(Dir):
        os.mkdir(Dir)
    else:
        os.system('rm -f {}/*'.format( Dir))
```

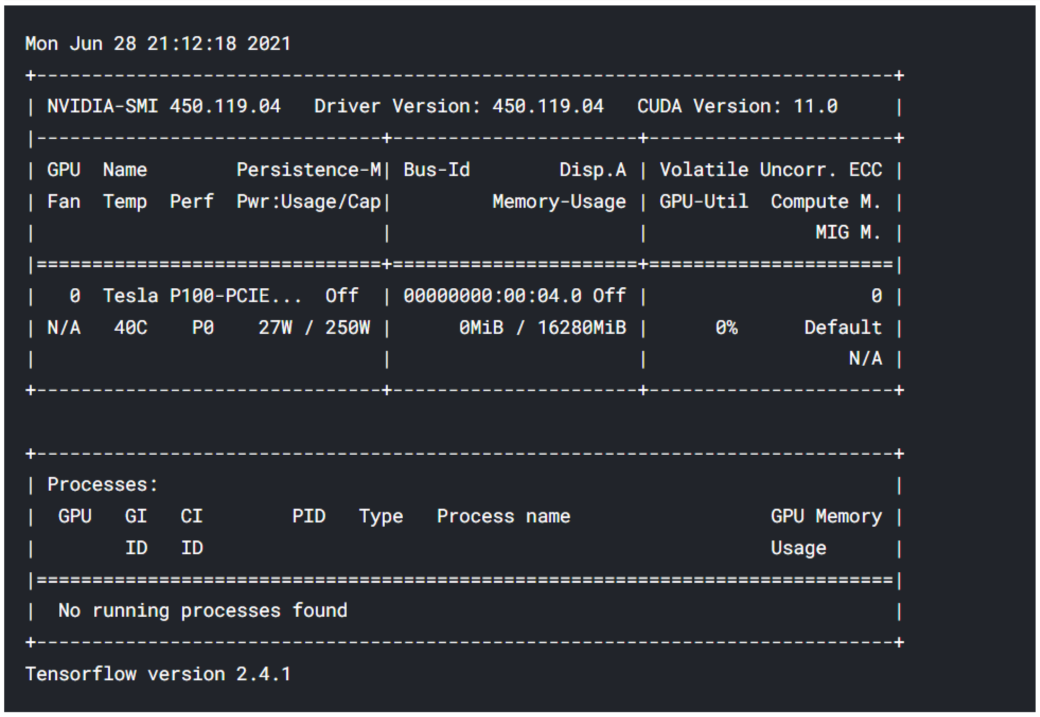
Output Window:

```
Linux-5.4.120+-x86_64-with-debian-buster-sid
```

```
import tensorflow as tf
tf.random.set_seed(73)
TPU_INIT = False
```

```
if TPU_INIT:
    try:
        tpu = tf.distribute.cluster_resolver.TPUClusterResolver.connect()
        tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
    except ValueError:
        raise BaseException('ERROR: Not connected to a TPU runtime!')
else:
    !nvidia-smi;
print("Tensorflow version " + tf.__version__)
```

Output Window:



```
Mon Jun 28 21:12:18 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.119.04   Driver Version: 450.119.04   CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P0    27W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Tensorflow version 2.4.1
```

```
MyDrive = '/kaggle/working'
PROJECT_DIR = '../input/real-life-violence-situations-dataset'
```

```
!ls {PROJECT_DIR}
```

Output Window:

```
'Real Life Violence Dataset'  'real life violence situations'
```

Preprocessing

- **Getting frames form video**
- **Some image augmentations**

```
import cv2

import os

import imageio

import imgaug.augmenters as iaa

import imgaug as ia

IMG_SIZE = 128

ColorChannels = 3

def video_to_frames(video):

    vidcap = cv2.VideoCapture(video)

    import math

    rate = math.floor(vidcap.get(3))
```

```
count = 0


ImageFrames = []

while vidcap.isOpened():

    ID = vidcap.get(1)

    success, image = vidcap.read()



    if success:

        # skipping frames to avoid duplications

        if (ID % 7 == 0):

            flip = iaa.Fliplr(1.0)

            zoom = iaa.Affine(scale=1.3)

            random_brightness = iaa.Multiply((1, 1.3))

            rotate = iaa.Affine(rotate=(-25, 25))



            image_aug = flip(image = image)

            image_aug = random_brightness(image = image_aug)

            image_aug = zoom(image = image_aug)
```

```
                image_aug = rotate(image = image_aug)



                rgb_img = cv2.cvtColor(image_aug, cv2.COLOR_BGR2RGB)

                resized = cv2.resize(rgb_img, (IMG_SIZE, IMG_SIZE))

                ImageFrames.append(resized)

            count += 1

        else:

            break

    vidcap.release()

    return ImageFrames

%%time

from tqdm import tqdm

VideoDataDir = PROJECT_DIR + '/Real Life Violence Dataset'

print('we have \n{} Violence videos \n{} NonViolence videos'.format(

              len(os.listdir(VideoDataDir + '/Violence')),

              len(os.listdir(VideoDataDir + '/NonViolence'))))

X_original = []

y_original = []
```

```
print('i choose 700 videos out of 2000, cuz of memory issue')

CLASSES = ["NonViolence", "Violence"]

#700 <- 350 + 350

for category in os.listdir(VideoDataDir):

    path = os.path.join(VideoDataDir, category)

    class_num = CLASSES.index(category)

    for i, video in enumerate(tqdm(os.listdir(path)[0:350])):

        frames = video_to_frames(path + '/' + video)

        for j, frame in enumerate(frames):

            X_original.append(frame)

            y_original.append(class_num)
```

Output Window:

```
 0%|          | 1/350 [00:00<01:07,  5.19it/s]
we have
1000 Violence videos
1000 NonViolence videos
i choose 700 videos out of 2000, cuz of memory issue
100%|██████████| 350/350 [01:17<00:00,  4.52it/s]
100%|██████████| 350/350 [02:29<00:00,  2.33it/s]

CPU times: user 5min 53s, sys: 14.7 s, total: 6min 8s
```

```
Wall time: 3min 48s
```

```
import numpy as np
X_original = np.array(X_original).reshape(-1 , IMG_SIZE * IMG_SIZE * 3)
y_original = np.array(y_original)
len(X_original)
```

Output Window:

```
13583
```

```
from sklearn.model_selection import StratifiedShuffleSplit
stratified_sample = StratifiedShuffleSplit(n_splits=2, test_size=0.3, random_state=73)
for train_index, test_index in stratified_sample.split(X_original, y_original):
    X_train, X_test = X_original[train_index], X_original[test_index]
    y_train, y_test = y_original[train_index], y_original[test_index]
X_train_nn = X_train.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255
X_test_nn = X_test.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255
```

Model Training

```
!pip install imutils
clear_output()
```

```
import cv2
import os
import numpy as np
import pickle
import matplotlib
matplotlib.use("Agg")
from keras.layers import Input
from keras.models import Model
from keras.layers.core import Dropout,Flatten,Dense
import matplotlib.pyplot as plt
epochs = 150
from keras import regularizers
kernel_regularizer = regularizers.l2(0.0001)
from keras.applications import MobileNetV2
def load_layers():
    input_tensor = Input(shape=(IMG_SIZE, IMG_SIZE, ColorChannels))
    baseModel = MobileNetV2(pooling='avg',
                            include_top=False,
                            input_tensor=input_tensor)

    headModel = baseModel.output
    headModel = Dense(1, activation="sigmoid")(headModel)
    model = Model(inputs=baseModel.input, outputs=headModel)

    for layer in baseModel.layers:
        layer.trainable = False
    print("Compiling model...")
    model.compile(loss="binary_crossentropy",
                    optimizer='adam',
                    metrics=["accuracy"])
    return model
```

```
if TPU_INIT:
    with tpu_strategy.scope():
        model = load_layers()
else:
    model = load_layers()
model.summary()
from tensorflow.keras.callbacks import Callback, ModelCheckpoint, LearningRateScheduler,
TensorBoard, EarlyStopping, ReduceLROnPlateau
patience = 3
start_lr = 0.00001
min_lr = 0.00001
max_lr = 0.00005
batch_size = 4
if TPU_INIT:
    max_lr = max_lr * tpu_strategy.num_replicas_in_sync
    batch_size = batch_size * tpu_strategy.num_replicas_in_sync
rampup_epochs = 5
sustain_epochs = 0
exp_decay = .8

def lrfn(epoch):
   if epoch < rampup_epochs:
        return (max_lr - start_lr)/rampup_epochs * epoch + start_lr
    elif epoch < rampup_epochs + sustain_epochs:
        return max_lr
    else:
        return (max_lr - min_lr) * exp_decay**(epoch-rampup_epochs-sustain_epochs) + min_lr
class myCallback(Callback):
    def on_epoch_end(self, epoch, logs={}):
        if ((logs.get('accuracy')>=0.999)):
            print("\nLimits Reached cancelling training!")
        self.model.stop_training = True
end_callback = myCallback()
lr_callback = LearningRateScheduler(lambda epoch: lrfn(epoch), verbose=False)

early_stopping = EarlyStopping(patience = patience, monitor='val_loss',
                          mode='min', restore_best_weights=True,
                          verbose = 1, min_delta = .00075)
PROJECT_DIR = MyDrive + '/RiskDetection'
lr_plat = ReduceLROnPlateau(patience = 2, mode = 'min')
os.system('rm -rf ./logs/')
import datetime
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir = log_dir, write_graph=True, histogram_freq=1)
checkpoint_filepath = 'ModelWeights.h5'
model_checkpoints = ModelCheckpoint(filepath=checkpoint_filepath,
                              save_weights_only=True,
                              monitor='val_loss',
                              mode='min',
                              verbose = 1,
                              save_best_only=True)
callbacks = [end_callback, lr_callback, model_checkpoints, tensorboard_callback,
early_stopping, lr_plat]
if TPU_INIT:
  callbacks = [end_callback, lr_callback, model_checkpoints, early_stopping, lr_plat]
print('Training head...')
#model.load_weights('./Model_Weights.h5')
history = model.fit(X_train_nn ,y_train, epochs=epochs,
                    callbacks=callbacks,
                    validation_data = (X_test_nn, y_test),
                    batch_size=batch_size)
print('\nRestoring best Weights for MobileNetV2')
```

```
model.load_weights(checkpoint_filepath)
```

Output Window:

```
Training head...
Epoch 1/150
2377/2377 [==============================] - 28s 10ms/step - loss: 0.8119 -
accuracy: 0.5507 - val_loss: 0.6402 - val_accuracy: 0.6466
Epoch 00001: val_loss improved from inf to 0.64018, saving model to ModelWeights.h5
Epoch 2/150
2377/2377 [==============================] - 19s 8ms/step - loss: 0.6116 - accuracy:
0.6719 - val_loss: 0.5003 - val_accuracy: 0.7504
Epoch 00002: val_loss improved from 0.64018 to 0.50028, saving model to ModelWeights.h5
Epoch 3/150
2377/2377 [==============================] - 20s 8ms/step - loss: 0.4879 - accuracy:
0.7587 - val_loss: 0.4211 - val_accuracy: 0.8091
Epoch 00003: val_loss improved from 0.50028 to 0.42114, saving model to ModelWeights.h5
Epoch 4/150
2377/2377 [==============================] - 20s 8ms/step - loss: 0.4039 - accuracy:
0.8131 - val_loss: 0.3778 - val_accuracy: 0.8312
Epoch 00004: val_loss improved from 0.42114 to 0.37776, saving model to ModelWeights.h5
Epoch 5/150
2377/2377 [==============================] - 19s 8ms/step - loss: 0.3658 - accuracy:
0.8322 - val_loss: 0.3500 - val_accuracy: 0.8520
Epoch 00005: val_loss improved from 0.37776 to 0.35003, saving model to ModelWeights.h5
Epoch 6/150
2377/2377 [==============================] - 20s 8ms/step - loss: 0.3401 - accuracy:
0.8518 - val_loss: 0.3323 - val_accuracy: 0.8591
Epoch 00006: val_loss improved from 0.35003 to 0.33229, saving model to ModelWeights.h5
Epoch 7/150
2377/2377 [==============================] - 19s 8ms/step - loss: 0.3100 - accuracy:
```

```
0.8658 - val_loss: 0.3155 - val_accuracy: 0.8690
Epoch 00007: val_loss improved from 0.33229 to 0.31555, saving model to ModelWeights.h5
Epoch 8/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2960 - accuracy: 0.8764 - val_loss: 0.3087 - val_accuracy: 0.8726
Epoch 00008: val_loss improved from 0.31555 to 0.30868, saving model to ModelWeights.h5
Epoch 9/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2900 - accuracy: 0.8774 - val_loss: 0.3015 - val_accuracy: 0.8766
Epoch 00009: val_loss improved from 0.30868 to 0.30154, saving model to ModelWeights.h5
Epoch 10/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2887 - accuracy: 0.8755 - val_loss: 0.2947 - val_accuracy: 0.8790
Epoch 00010: val_loss improved from 0.30154 to 0.29472, saving model to ModelWeights.h5
Epoch 11/150
2377/2377 [==============================] - 20s 8ms/step - loss: 0.2891 - accuracy:
0.8750 - val_loss: 0.2910 - val_accuracy: 0.8810

 Epoch 00011: val_loss improved from 0.29472 to 0.29102, saving model to ModelWeights.h5
Epoch 12/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2671 - accuracy: 0.8915 - val_loss: 0.2890 - val_accuracy: 0.8822

Epoch 00012: val_loss improved from 0.29102 to 0.28900, saving model to ModelWeights.h5
Epoch 13/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2642 - accuracy: 0.8927 - val_loss: 0.2846 - val_accuracy: 0.8829

Epoch 00013: val_loss improved from 0.28900 to 0.28458, saving model to ModelWeights.h5
Epoch 14/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2761 - accuracy: 0.8863 - val_loss: 0.2836 - val_accuracy: 0.8854

Epoch 00014: val_loss improved from 0.28458 to 0.28362, saving model to ModelWeights.h5
Epoch 15/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2667 - accuracy: 0.8897 - val_loss: 0.2807 - val_accuracy: 0.8861

Epoch 00015: val_loss improved from 0.28362 to 0.28072, saving model to ModelWeights.h5
Epoch 16/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2574 - accuracy: 0.8918 - val_loss: 0.2790 - val_accuracy: 0.8869

Epoch 00016: val_loss improved from 0.28072 to 0.27903, saving model to ModelWeights.h5
Epoch 17/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2554 - accuracy: 0.8931 - val_loss: 0.2765 - val_accuracy: 0.8869

Epoch 00017: val_loss improved from 0.27903 to 0.27648, saving model to ModelWeights.h5
Epoch 18/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2506 - accuracy: 0.8964 - val_loss: 0.2778 - val_accuracy: 0.8888

Epoch 00018: val_loss did not improve from 0.27648 Epoch 19/150 2377/2377
[==============================] - 20s 8ms/step - loss: 0.2538 - accuracy: 0.8954 -
val_loss: 0.2747 - val_accuracy: 0.8883

Epoch 00019: val_loss improved from 0.27648 to 0.27467, saving model to ModelWeights.h5
Epoch 20/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2460 - accuracy: 0.8969 - val_loss: 0.2728 - val_accuracy: 0.8879

Epoch 00020: val_loss improved from 0.27467 to 0.27283, saving model to ModelWeights.h5
Epoch 21/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2609 - accuracy: 0.8897 - val_loss: 0.2711 - val_accuracy: 0.8896

Epoch 00021: val_loss improved from 0.27283 to 0.27108, saving model to ModelWeights.h5
Epoch 22/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2536 - accuracy: 0.8953 - val_loss: 0.2720 - val_accuracy: 0.8893

Epoch 00022: val_loss did not improve from 0.27108 Epoch 23/150 2377/2377
[==============================] - 19s 8ms/step - loss: 0.2442 - accuracy: 0.8976 -
val_loss: 0.2688 - val_accuracy: 0.8913

Epoch 00023: val_loss improved from 0.27108 to 0.26883, saving model to ModelWeights.h5
Epoch 24/150 2377/2377 [==============================] - 19s 8ms/step - loss:
```

```
0.2503 - accuracy: 0.8967 - val_loss: 0.2695 - val_accuracy: 0.8910

Epoch 00024: val_loss did not improve from 0.26883 Epoch 25/150 2377/2377
[==============================] - 20s 8ms/step - loss: 0.2473 - accuracy: 0.9001 -
val_loss: 0.2670 - val_accuracy: 0.8915

Epoch 00025: val_loss improved from 0.26883 to 0.26700, saving model to ModelWeights.h5
Epoch 26/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2354 - accuracy: 0.9083 - val_loss: 0.2658 - val_accuracy: 0.8923

Epoch 00026: val_loss improved from 0.26700 to 0.26583, saving model to ModelWeights.h5
Epoch 27/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2443 - accuracy: 0.9006 - val_loss: 0.2652 - val_accuracy: 0.8923

Epoch 00027: val_loss improved from 0.26583 to 0.26520, saving model to ModelWeights.h5
Epoch 28/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2378 - accuracy: 0.9005 - val_loss: 0.2638 - val_accuracy: 0.8940

Epoch 00028: val_loss improved from 0.26520 to 0.26376, saving model to ModelWeights.h5
Epoch 29/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2284 - accuracy: 0.9087 - val_loss: 0.2636 - val_accuracy: 0.8928

Epoch 00029: val_loss improved from 0.26376 to 0.26360, saving model to ModelWeights.h5
Epoch 30/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2359 - accuracy: 0.9052 - val_loss: 0.2637 - val_accuracy: 0.8937

Epoch 00030: val_loss did not improve from 0.26360
Epoch 31/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2351 - accuracy: 0.9069 - val_loss: 0.2612 - val_accuracy: 0.8937
Epoch 00031: val_loss improved from 0.26360 to 0.26125, saving model to ModelWeights.h5
Epoch 32/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2327 - accuracy: 0.9061 - val_loss: 0.2607 - val_accuracy: 0.8947
```

```
Epoch 00032: val_loss improved from 0.26125 to 0.26069, saving model to ModelWeights.h5
Epoch 33/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2324 - accuracy: 0.9102 - val_loss: 0.2598 - val_accuracy: 0.8942

Epoch 00033: val_loss improved from 0.26069 to 0.25977, saving model to ModelWeights.h5
Epoch 34/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2350 - accuracy: 0.9058 - val_loss: 0.2594 - val_accuracy: 0.8960

Epoch 00034: val_loss improved from 0.25977 to 0.25942, saving model to ModelWeights.h5
Epoch 35/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2372 - accuracy: 0.9032 - val_loss: 0.2586 - val_accuracy: 0.8960

Epoch 00035: val_loss improved from 0.25942 to 0.25859, saving model to ModelWeights.h5
Epoch 36/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2334 - accuracy: 0.9048 - val_loss: 0.2573 - val_accuracy: 0.8972

Epoch 00036: val_loss improved from 0.25859 to 0.25732, saving model to ModelWeights.h5
Epoch 37/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2350 - accuracy: 0.9047 - val_loss: 0.2574 - val_accuracy: 0.8962

Epoch 00037: val_loss did not improve from 0.25732 Epoch 38/150 2377/2377
[==============================] - 20s 8ms/step - loss: 0.2293 - accuracy: 0.9068 -
val_loss: 0.2562 - val_accuracy: 0.8974

Epoch 00038: val_loss improved from 0.25732 to 0.25622, saving model to ModelWeights.h5
Epoch 39/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2392 - accuracy: 0.9030 - val_loss: 0.2553 - val_accuracy: 0.8984
Epoch 00039: val_loss improved from 0.25622 to 0.25532, saving model to ModelWeights.h5
Epoch 40/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2316 - accuracy: 0.9033 - val_loss: 0.2553 - val_accuracy: 0.8982
```

```
Epoch 00040: val_loss did not improve from 0.25532 Epoch 41/150 2377/2377
[==============================] - 19s 8ms/step - loss: 0.2310 - accuracy: 0.9062 -
val_loss: 0.2538 - val_accuracy: 0.8989

Epoch 00041: val_loss improved from 0.25532 to 0.25381, saving model to ModelWeights.h5
Epoch 42/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2278 - accuracy: 0.9119 - val_loss: 0.2535 - val_accuracy: 0.8989

Epoch 00042: val_loss improved from 0.25381 to 0.25347, saving model to ModelWeights.h5
Epoch 43/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2272 - accuracy: 0.9073 - val_loss: 0.2533 - val_accuracy: 0.8984

Epoch 00043: val_loss improved from 0.25347 to 0.25327, saving model to ModelWeights.h5
Epoch 44/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2207 - accuracy: 0.9140 - val_loss: 0.2519 - val_accuracy: 0.9009

Epoch 00044: val_loss improved from 0.25327 to 0.25195, saving model to ModelWeights.h5
Epoch 45/150 2377/2377 [==============================] - 20s 9ms/step - loss:
0.2226 - accuracy: 0.9091 - val_loss: 0.2517 - val_accuracy: 0.8989

Epoch 00045: val_loss improved from 0.25195 to 0.25169, saving model to ModelWeights.h5
Epoch 46/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2179 - accuracy: 0.9133 - val_loss: 0.2513 - val_accuracy: 0.8989

Epoch 00046: val_loss improved from 0.25169 to 0.25134, saving model to ModelWeights.h5
Epoch 47/150 2377/2377 [==============================] - 20s 9ms/step - loss:
0.2179 - accuracy: 0.9134 - val_loss: 0.2512 - val_accuracy: 0.8969
Epoch 00047: val_loss improved from 0.25134 to 0.25117, saving model to ModelWeights.h5
Epoch 48/150 2377/2377 [==============================] - 20s 9ms/step - loss:
0.2250 - accuracy: 0.9098 - val_loss: 0.2502 - val_accuracy: 0.8989
```

```
Epoch 00048: val_loss improved from 0.25117 to 0.25020, saving model to ModelWeights.h5
Epoch 49/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2235 - accuracy: 0.9107 - val_loss: 0.2491 - val_accuracy: 0.9001

Epoch 00049: val_loss improved from 0.25020 to 0.24913, saving model to ModelWeights.h5
Epoch 50/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2164 - accuracy: 0.9140 - val_loss: 0.2499 - val_accuracy: 0.8972

Epoch 00050: val_loss did not improve from 0.24913 Epoch 51/150 2377/2377
[==============================] - 19s 8ms/step - loss: 0.2198 - accuracy: 0.9135 -
val_loss: 0.2493 - val_accuracy: 0.8979
Epoch 00051: val_loss did not improve from 0.24913 Epoch 52/150 2377/2377
[==============================] - 19s 8ms/step - loss: 0.2132 - accuracy: 0.9136 -
val_loss: 0.2479 - val_accuracy: 0.8989

Epoch 00052: val_loss improved from 0.24913 to 0.24795, saving model to ModelWeights.h5
Epoch 53/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2123 - accuracy: 0.9154 - val_loss: 0.2484 - val_accuracy: 0.8991

Epoch 00053: val_loss did not improve from 0.24795 Epoch 54/150 2377/2377
[==============================] - 19s 8ms/step - loss: 0.2191 - accuracy: 0.9122 -
val_loss: 0.2462 - val_accuracy: 0.9023

Epoch 00054: val_loss improved from 0.24795 to 0.24619, saving model to ModelWeights.h5
Epoch 55/150 2377/2377 [==============================] - 20s 8ms/step - loss:
0.2127 - accuracy: 0.9184 - val_loss: 0.2457 - val_accuracy: 0.9028

Epoch 00055: val_loss improved from 0.24619 to 0.24569, saving model to ModelWeights.h5
Epoch 56/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2203 - accuracy: 0.9086 - val_loss: 0.2454 - val_accuracy: 0.9013
```

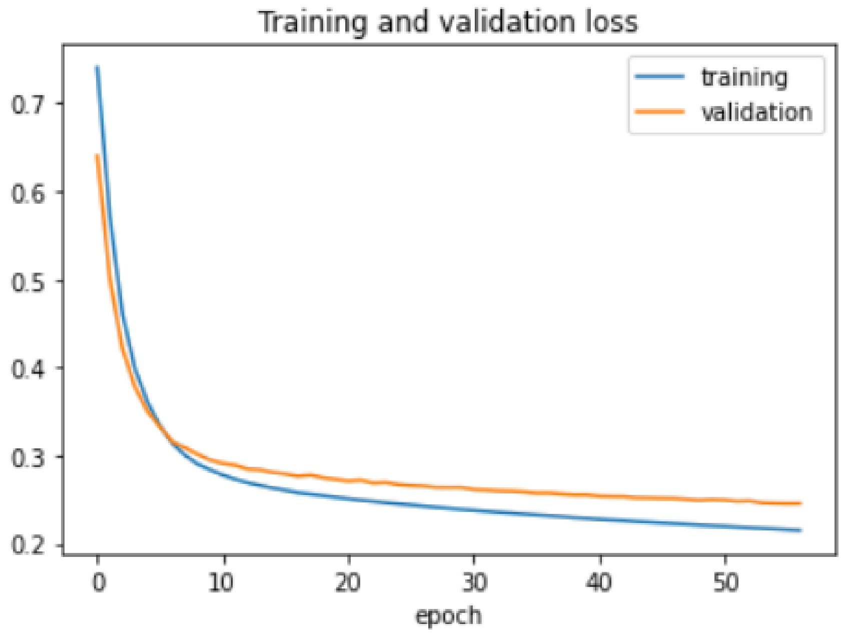
```
Epoch 00056: val_loss improved from 0.24569 to 0.24539, saving model to ModelWeights.h5
Epoch 57/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2204 - accuracy: 0.9112 - val_loss: 0.2455 - val_accuracy: 0.8999

Epoch 00057: val_loss did not improve from 0.24539 Epoch 58/150 2377/2377
[==============================] - 20s 8ms/step - loss: 0.2148 - accuracy: 0.9158 -
val_loss: 0.2442 - val_accuracy: 0.9026

Epoch 00058: val_loss improved from 0.24539 to 0.24418, saving model to ModelWeights.h5
Epoch 59/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2196 - accuracy: 0.9118 - val_loss: 0.2438 - val_accuracy: 0.9021

Epoch 00059: val_loss improved from 0.24418 to 0.24377, saving model to ModelWeights.h5
Epoch 60/150 2377/2377 [==============================] - 19s 8ms/step - loss:
0.2103 - accuracy: 0.9176 - val_loss: 0.2441 - val_accuracy: 0.9006

Epoch 00060: val_loss did not improve from 0.24377 Epoch 61/150 2377/2377
[==============================] - 20s 8ms/step - loss: 0.2192 - accuracy: 0.9165 -
val_loss: 0.2451 - val_accuracy: 0.8999
```

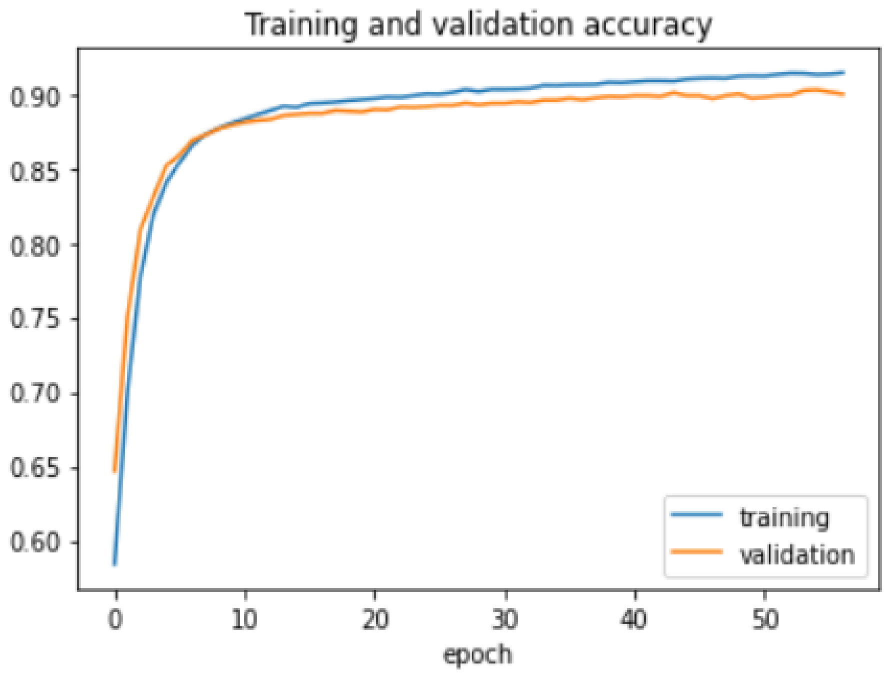
```
Epoch 00061: val_loss did not improve from 0.24377
Restoring model weights from the end of the best epoch.
Epoch 00061: early stopping
Restoring best Weights for MobileNetV2
```

```
%matplotlib inline
def print_graph(item, index, history):
    plt.figure()
    train_values = history.history[item][0:index]
    plt.plot(train_values)
    test_values = history.history['val_' + item][0:index]
    plt.plot(test_values)
    plt.legend(['training','validation'])
    plt.title('Training and validation '+ item)
    plt.xlabel('epoch')
    plt.show()
    plot = '{}.png'.format(item)
    plt.savefig(plot)
def get_best_epoch(test_loss, history):
    for key, item in enumerate(history.history.items()):
        (name, arr) = item
        if name == 'val_loss':
            for i in range(len(arr)):
                if round(test_loss, 2) == round(arr[i], 2):
                    return i
def model_summary(model, history):
    print('---'*30)
    test_loss, test_accuracy = model.evaluate(X_test_nn, y_test, verbose=0)
    if history:
        index = get_best_epoch(test_loss, history)
        print('Best Epochs: ', index)
        train_accuracy = history.history['accuracy'][index]
        train_loss = history.history['loss'][index]
        print('Accuracy on train:',train_accuracy,'\tLoss on train:',train_loss)
        print('Accuracy on test:',test_accuracy,'\tLoss on test:',test_loss)
        print_graph('loss', index, history)
        print_graph('accuracy', index, history)
        print('---'*30)
model_summary(model, history)
```

Output Window:

```
------------------------------------------------------------------------------------------

Best Epochs:  57

Accuracy on train: 0.9151241183280945       Loss on train: 0.2140893042087555

Accuracy on test: 0.9020859003067017        Loss on test: 0.2437741458415985
```





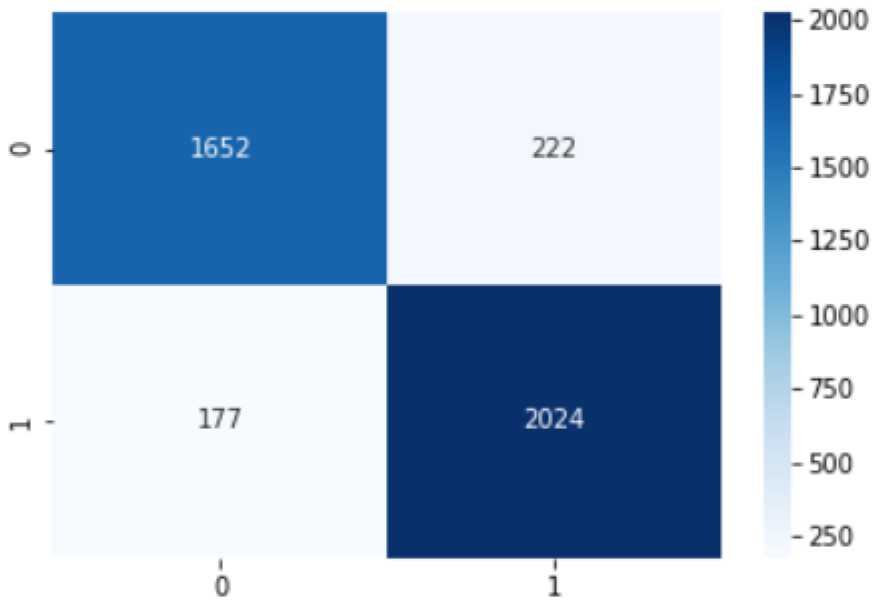
Evaluation on Test Set

```
# evaluate the network
print("Evaluating network...")
predictions = model.predict(X_test_nn)
preds = predictions > 0.5
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import roc_curve, roc_auc_score, plot_roc_curve, accuracy_score,
classification_report, confusion_matrix
corr_pred = metrics.confusion_matrix(y_test, preds)

n_correct = np.int((corr_pred[0][0] + corr_pred[1][1]))
```

```
print('> Correct Predictions:', n_correct)
n_wrongs = np.int((corr_pred[0][1] + (corr_pred[1][0])))
print('> Wrong Predictions:', n_wrongs)
sns.heatmap(corr_pred,annot=True, fmt="d",cmap="Blues")
plt.show()
print(metrics.classification_report(y_test, preds,
                        target_names=["NonViolence", "Violence"]))
```

Output Window:

```
> Correct Predictions: 3676

> Wrong Predictions: 399
```



Classification metrics:

When performing classification predictions, there's four types of outcomes that could occur.

True positives are when you predict an observation belongs to a class and it actually does belong to that class.

True negatives are when you predict an observation does not belong to a class and it actually does not belong to that class.

False positives occur when you predict an observation belongs to a class when in reality it does not.

False negatives occur when you predict an observation does not belong to a class when in fact it does.

These four outcomes are often plotted on a confusion matrix. The following confusion matrix is an example for the case of binary classification. You would generate this matrix after making predictions on your test data and then identifying each prediction as one of the four possible outcomes described above.

	precision	**recall**	**f1-score**	**support**
NonViolence	0.90	0.88	0.89	1874
Violence	0.90	0.92	0.91	2201
accuracy			0.90	4075
macro avg	0.90	0.90	0.90	4075
weighted avg	0.90	0.90	0.90	4075

Table 1: Classification report for the MobileNetV2 model.

The three main metrics used to evaluate a classification model are accuracy, precision, and recall.

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

Precision is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Recall is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class.

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Precision and recall are useful in cases where classes aren't evenly distributed.

We can evaluate the performance of the model using the measures shown in Fig 3 above. Precision, that is the measure of correctly predicted positive observations than the amount of total predicted positive observations. Here in Fig 4 it can be seen that precision score for non-violence category is 91 percent and for violence category its 90 percent.

We can observe in Table 1 the recall for non-violence category is 88 percent and violence category is 93 percent. Here the violence category has more recall, which is good for the model as it increases scope for detecting violence.

Here in Table 1 we can observe that the F1 score for non-violence category is 89 percent and for violence category is 91 percent.

The support is the total number of samples of that class, that are observed in the dataset. Here during the testing of the model, the model has observed 1874 occurrences of non-violence class and 2201 occurrences of violence class out of total 4075 occurrences of both the classes included.

In the Fig 2, we can see the gradual decrease of training loss and validation loss up till 57 epochs. The training loss and validation loss are in decreasing motion which is a good result from the model.

In the Fig 2, we can see the gradual increase of training accuracy and validation accuracy up till 57 epochs. The training loss and validation loss are in increasing motion which is a good result from the model.

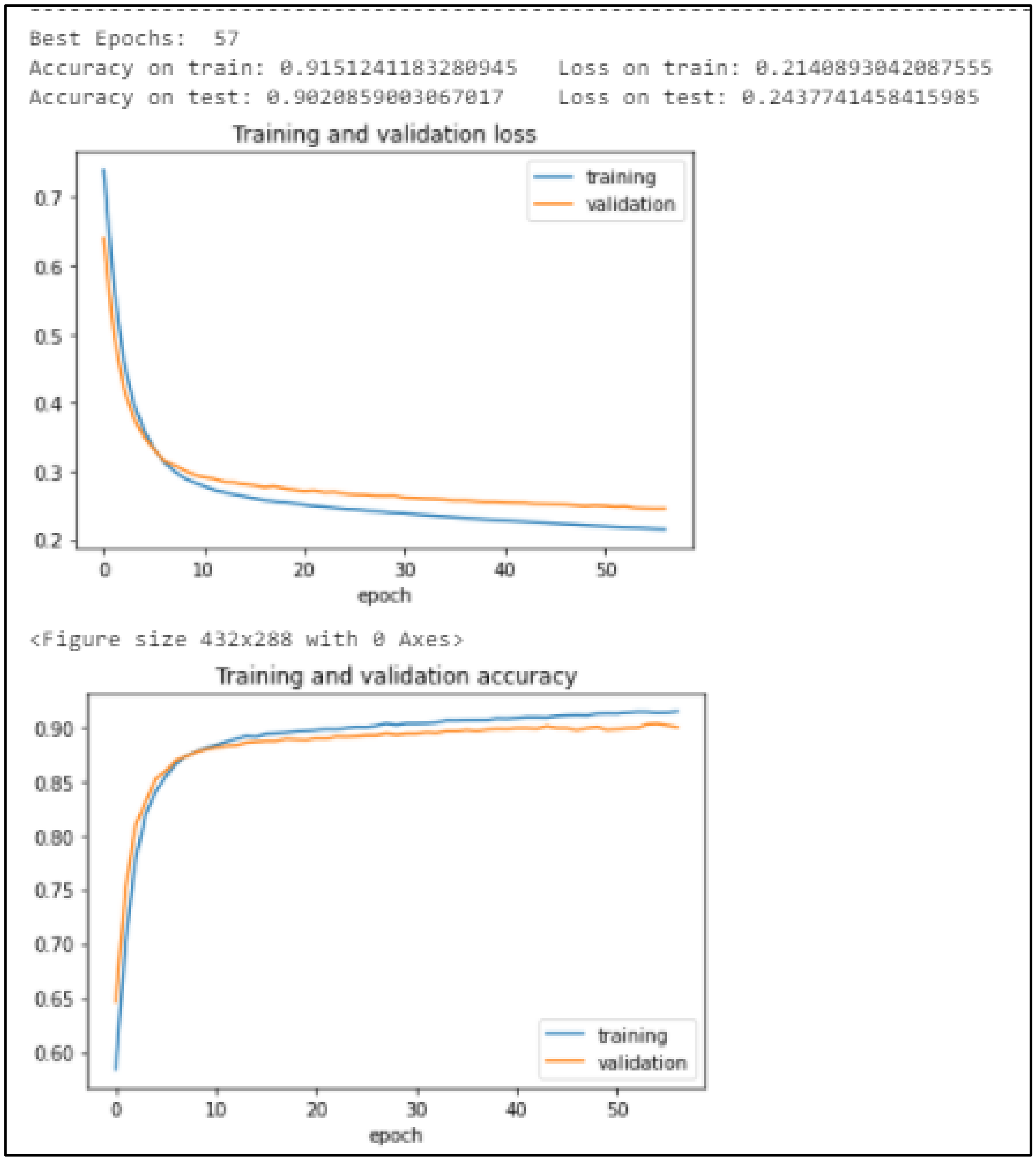
CHAPTER-4

Results and Discussions

The purpose of frame-main grouping is to efficiently extract relevant features from multiple frames utilising current CNN backbones. To determine a sampling method, we looked at the appropriate time gap between consecutive frames. Because most of the datasets we looked at were compiled from YouTube, even within the same dataset, the frame rate of each video varied. As a result, establishing the optimal time period is impossible. Some motion pictures, for example, are in slow movement, but others have abnormally low frame rates. MobileNets focuses on reducing latency and parameter count in mobile and embedded applications with limited resources.

The problem with most violence detection algorithms is that they must analyse a high number of nonsensical frames, which consumes a lot of memory and takes a long time. Given this considerable constraint, we chose a pre-trained MobileNet convolutional neural network model because the InceptionV3 model did not perform well due to overfitting. Due to a lack of data in typical datasets for violence detection and a frequently low accuracy rate, existing mainstream approaches are unable to learn effective patterns. Inspired by the concept of transfer learnings, in both interior and outdoor surveillance, a 3D-Convolutional neural network was fine-tuned for violence detection using publicly accessible standard datasets. It outperforms traditional hand-engineered feature extraction algorithms in terms of accuracy in tests.

Rather than processing the full video stream, we process only those sequences that contain individuals, discarding unimportant frames. Instead of normal convolutions, MobileNet uses depth wise separable convolutions to detect items. If pointwise and depth wise convolutions are counted individually, there are 28 layers. with nonlinearity batch norm and Sigmoid after each layer except the last fully linked layer.

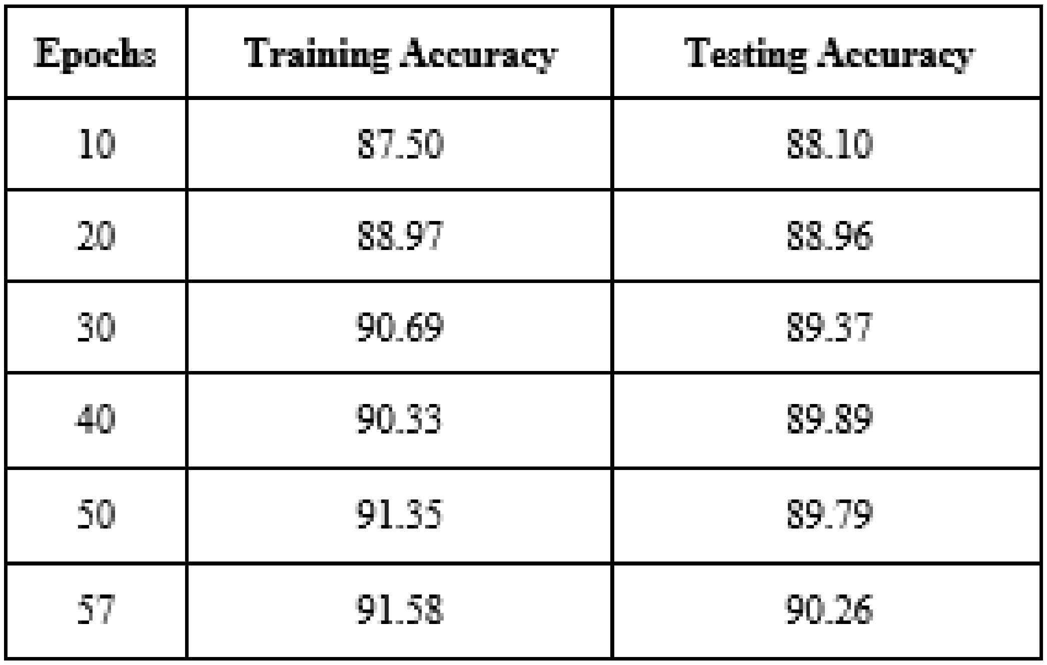


```
--------------------------------------------------------------------------------
Best Epochs:  57
Accuracy on train: 0.9151241183280945   Loss on train: 0.2140893042087555
Accuracy on test: 0.9020859003067017    Loss on test: 0.2437741458415985
```

```
<Figure size 432x288 with 0 Axes>
```

Figure 2: Training and Validation Loss and Accuracy Graphs

For the MobileNetV2 model: A training accuracy of 91% and the loss is 21%. A test accuracy of 90% and loss on the test set is 24%.



Epochs	Training Accuracy	Testing Accuracy
10	87.50	88.10
20	88.97	88.96
30	90.69	89.37
40	90.33	89.89
50	91.35	89.79
57	91.58	90.26

Table 2: MobileNetV2 Model Performance with respect to accuracy.

The lower the loss, the better a model (unless the model has over-fitted to the training data). The loss is calculated on training and validation and its interperation is how well the model is doing for these two sets. Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each example in training or validation sets.

In the case of neural networks, the loss is usually negative log-likelihood and residual sum of squares for classification and regression respectively. Then naturally, the main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as backpropagation in neural networks.

Loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s).

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then the percentage of misclassification is calculated.

In the above table we can observe the total number of epochs for which the model has executed. Since validation loss was not improving with increase of epochs, with the execution of the model, so the best epochs were estimated to be 57 in total.

Due to its 3D convolution and pooling capabilities, a 3D Convolutional neural network is well-suited for retrieving spatio-temporal properties and can preserve temporal data better. Furthermore, 2D CNNs only collect spatial information, A 3D Convolutional neural network, on the other hand, can collect all of the temporal information about the input sequence. To increase the number of feature maps in late layers, convolutional neural networks leverage the concept of creating many different kinds of features from the similar feature maps. A succession of frames serves as the network's input data. The volume mean of training and testing data is computed before beginning the training operation. The network's design has been optimised to accept these sequences as inputs. The Sigmoid layer determines whether the final forecast is violent or not.

One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE. Since one epoch is too big to feed to the computer at once we divide it in several smaller batches. An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the amount of data is very large).

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions. In other words, the test (or testing) accuracy often refers to the validation accuracy, that is, the accuracy you calculate on the data set you do not use for training, but you use (during the training process) for validating (or "testing") the generalisation ability of your model or for "early stopping".

CHAPTER-5

Conclusion and Future Scope

To construct a viable violence detection system, we demonstrated frame-grouping methods and spatial-temporal attention modules. Frame-grouping was introduced as a way of averaging the channels. It was able to represent short-term dynamics, which was a key aspect in classifying aggressive behaviours like kicking and punching. In our research MobileNetV2 showed a better performance in detection with a validation accuracy of 90.26%. To train a more robust model, in the future, we'll train the model with more data and test a variety of data augmentation tactics. In addition, for a more versatile use, we will widen our research to encompass a variety of action recognition tasks.

References

1. W. Sultani, C. Chen and M. Shah, "Real-world anomaly detection in surveillance videos", Proc. IEEE Conf. Comput. Vis. Pattern Recognit., pp. 6479-6488, 2018.
2. O. Deniz, I. Serrano, G. Bueno and T.-K. Kim, "Fast violence detection in video", Proc. Int. Conf. Comput. Vis. Theory Appl. (VISAPP), vol. 2, pp. 478-485, Jan. 2014.
3. P. C. Ribeiro, R. Audigier and Q. C. Pham, "RIMOC a feature to discriminate unstructured motions: Application to violence detection for video-surveillance", Comput. Vis. Image Understand., vol. 144, pp. 121-143, Mar. 2016.
4. E. Y. Fu, H. Va Leong, G. Ngai and S. Chan, "Automatic fight detection in surveillance videos", Proc. 14th Int. Conf. Adv. Mobile Comput. MultiMedia, pp. 225-234, Nov. 2016.
5. T. Senst, V. Eiselein, A. Kuhn and T. Sikora, "Crowd violence detection using global motion-compensated Lagrangian features and scale-sensitive video-level representation", IEEE Trans. Inf. Forensics Security, vol. 12, no. 12, pp. 2945-2956, Dec. 2017.
6. T. Hassner, Y. Itcher and O. Kliper-Gross, "Violent flows: Real-time detection of violent crowd behavior", Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops, pp. 1-6, Jun. 2012.
7. Dai, Qi, Jian Tu, Ziqiang Shi, Yu G. Jiang, and Xiangyang Xue. “Violent Scenes Detection Using Motion Features and Part-Level Attributes.” MediaEval Workshop, Oct. 2013.
8. Jiang, Yu G., Qi Dai, Xiangyang Xue, Wei Liu, and Chong W. Ngo. “Trajectory-Based Modeling of Human Actions with Motion Reference

Points.” ECCV, 2012.

9. M. Soliman, M. Kamal, M. Nashed, Y. Mostafa, B. Chawky, D. Khattab, “ Violence Recognition from Videos using Deep Learning Techniques", Proc. 9th International Conference on Intelligent Computing and Information Systems (ICICIS'19), Cairo, pp. 79-84, 2019
10. Shakil Ahmed Sumon, Raihan Goni, Niyaz Bin Hashem, Tanzil Shahria and Rashedur M. Rahman, “Violence Detection by Pretrained Modules with Different Deep Learning Approaches”, Vietnam Journal of Computer ScienceVol. 07, No. 01, pp. 19-40, 2020.
11. Enrique Bermejo NievasOscar Deniz SuarezGloria Bueno GarcíaRahul Sukthankar, “Violence Detection in Video Using Computer Vision Techniques”,International Conference on Computer Analysis of Images and Patterns (Springer), 2011.
12. Chunhui DingShouke FanMing ZhuWeiguo FengBaozhi Jia, “Violence Detection in Video by Using 3D Convolutional Neural Networks”, International Symposium on Visual Computing ISVC, 2014.
13. Abdali, Almamon & Al-Tuma, Rana, “Robust Real-Time Violence Detection in Video Using CNN And LSTM”, 2nd Scientific Conference of Computer Sciences (SCCS), 2019.
14. Zhou Peipei, Ding Qinghai, Luo Haibo, Hou Xinglin, “Violent Interaction Detection in Video Based on Deep Learning”, Journal of Physics: Conference Series, 2017.