DETECTING VIOLENT BEHAVIOR USING CCTV FOOTAGE

COMPUTER VISION

Business Use Case

BUSINESS PROBLEM

For this project, the team decided to make use of the CCTV footage to reduce the hours spend by workers on watching the surveillance cameras. The CCTV software should automatically detect violent behavior or situation where weapons are involved.

BUSINESS REQUIREMENTS

Feature & Dataset: CCTV Footage(YouTube), customizable Dataset

Model Development: The ML life-cycle - constantly test & improve

Monitoring: Feedback on detected violent behavior

Infrastructure: Pre-Trained Model

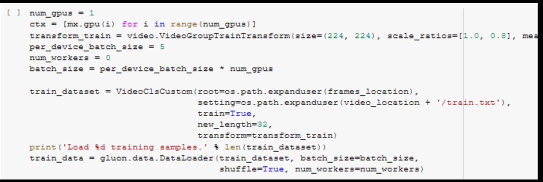
Our Solution

- **Fine-tuning a pre-trained I3D** on a custom dataset of classified CCTV video-recordings **for dangerous action estimation & classification**, using:
    - Gluon API
    - mxnet library (by Gluon)
- **I3D is a pre-trained action detection model** (CNN) obtained from GluonCV's Model Zoo. Accounts for images (2D) in a 3D setting (2D-images + time-dimension)
- **I3D was pre-trained on Kinetics400** - action recognition dataset of 306,245 realistic action videos from YouTube (from 400 action categories)

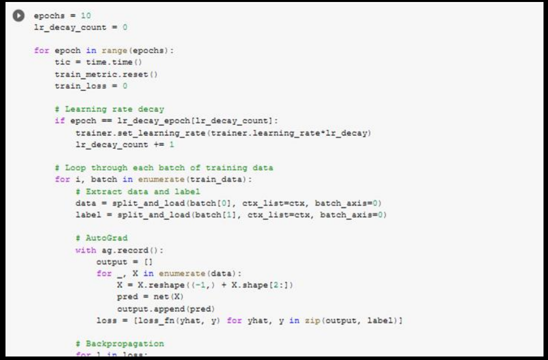
GluonCV Demo (1'20min)

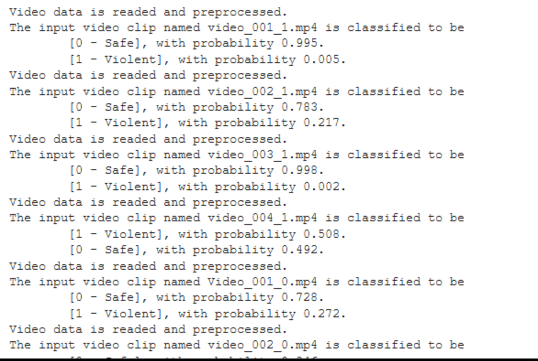
Developing the Model

MODEL


```python
num_gpus = 1
ctx = [mx.gpu(i) for i in range(num_gpus)]
transform_train = video.VideoGroupTrainTransform(size=(224, 224), scale_ratios=[1.0, 0.8], mea
per_device_batch_size = 5
num_workers = 0
batch_size = per_device_batch_size * num_gpus

train_dataset = VideoClsCustom(root=os.path.expanduser(frames_location),
                               setting=os.path.expanduser(video_location + '/train.txt'),
                               train=True,
                               new_length=32,
                               transform=transform_train)
print('Load %d training samples.' % len(train_dataset))
train_data = gluon.data.DataLoader(train_dataset, batch_size=batch_size,
                                   shuffle=True, num_workers=num_workers)
```


```python
net = get_model(name='i3d_resnet50_v1_custom', nclass=101)
net.collect_params().reset_ctx(ctx)
print(net)
```


```python
epochs = 10
lr_decay_count = 0

for epoch in range(epochs):
    tic = time.time()
    train_metric.reset()
    train_loss = 0

    # Learning rate decay
    if epoch == lr_decay_epoch[lr_decay_count]:
        trainer.set_learning_rate(trainer.learning_rate*lr_decay)
        lr_decay_count += 1

    # Loop through each batch of training data
    for i, batch in enumerate(train_data):
        # Extract data and label
        data = split_and_load(batch[0], ctx_list=ctx, batch_axis=0)
        label = split_and_load(batch[1], ctx_list=ctx, batch_axis=0)

        # AutoGrad
        with ag.record():
            output = []
            for _, X in enumerate(data):
                X = X.reshape((-1,) + X.shape[2:])
                pred = net(X)
                output.append(pred)
            loss = [loss_fn(yhat, y) for yhat, y in zip(output, label)]

        # Backpropagation
```


```
Video data is readed and preprocessed.
The input video clip named video_001_1.mp4 is classified to be
        [0 - Safe], with probability 0.995.
        [1 - Violent], with probability 0.005.
Video data is readed and preprocessed.
The input video clip named video_002_1.mp4 is classified to be
        [0 - Safe], with probability 0.783.
        [1 - Violent], with probability 0.217.
Video data is readed and preprocessed.
The input video clip named video_003_1.mp4 is classified to be
        [0 - Safe], with probability 0.998.
        [1 - Violent], with probability 0.002.
Video data is readed and preprocessed.
The input video clip named video_004_1.mp4 is classified to be
        [1 - Violent], with probability 0.508.
        [0 - Safe], with probability 0.492.
Video data is readed and preprocessed.
The input video clip named Video_001_0.mp4 is classified to be
        [0 - Safe], with probability 0.728.
        [1 - Violent], with probability 0.272.
Video data is readed and preprocessed.
The input video clip named video_002_0.mp4 is classified to be
```

1 Loading the Data

Defining a custom **DataLoader** to load the custom dataset from a train.txt which specifies the label of each video

2 Custom Net Architecture

Loading the pre-trained I3D model and **adding a single fully-connected layer** to its architecture

This dense layer translates the action detection out of 400 categories to the particular task of detecting violent behavior

3 Training & Tuning

Setting all relevant **training parameters** and **SoftMax** as loss function (Classification Task with accuracy as evaluation metric and balanced classes)

Training the model with the specified parameters

4 Model Evaluation

Evaluating the trained model on a validation set of videos through prediction on the validation videos.

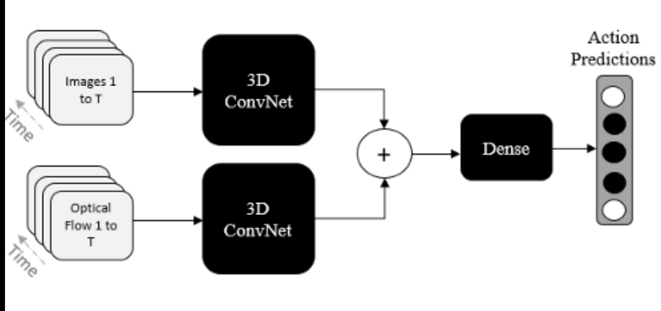
1 Custom Dataset

MODEL

- **Video Source & Video Format:** CCTV Clips (YouTube)
  Clipped into 10-15 seconds, HD 1080
- **Storing:** Video format or already decoded to frames
- **Categorized:** Train Set Classified in 1 = Violent, 0 = Non-violent CCTV Clips, set as balanced
- **Toolkit:** GluonCV, no format import, only requirement to define train.txt file.

Loading Dataset

- GluonCV allows the user to specify the custom dataset
- User specifies the label of each of the videos that are to be used for training in a file set called train.txt
- Based on this, Class VideoClsCustom generates a customized data loader that can be used for later fine-tuning



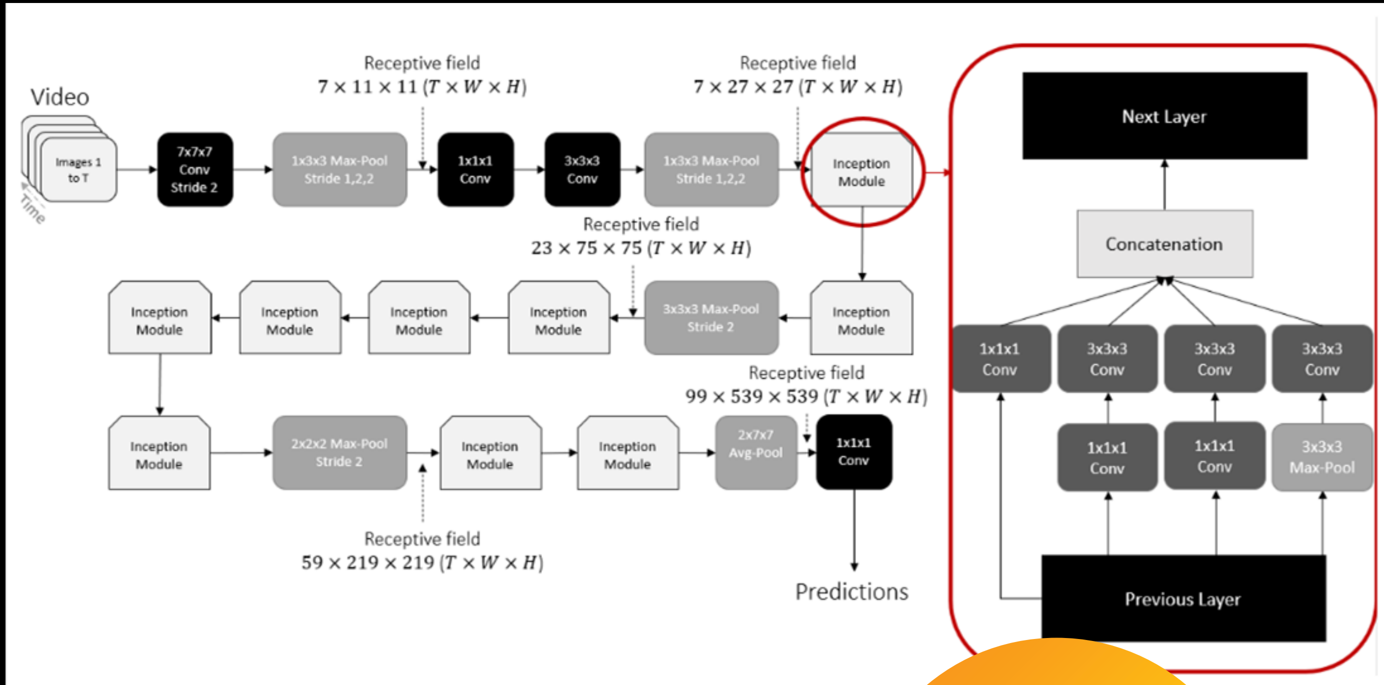


Figure 2. Image by author, adapted from Carreira and Zisserman (20

② Architecture

MODEL

Model Architecture

- **Pre-trained I3D:**
  - 3-dimensions: 2D-images (height/width) + time
  - Convolutional Network, using asymmetrical receptive fields
  - Spatial information is pooled separately from time-dimension (asymmetrical pooling in the beginning - 1x3x3 kernel)
- **+ fully-connected dense layer** for binary classification

Optimizer

- SGD
- Loss-function: Softmax

③ Train & Tune

MODEL

8
Epochs

Epochs: defines the number of times that the learning algorithm will work through the entire training dataset

0.001
Learning Rate

Learning Rate: how much to change the model in response to the estimated error each time the model weights are updated

0.1
LR Decay

Learning Rate Decay: defines the gradual reduction of the learning rate as training progresses

10/20/30
LR Decay Epochs

Learning Rate Decay Epochs: Epochs where learning rate decays (learning rate schedule)

0.09
Momentum

Momentum: using an exponentially weighted average of the prior updates to the weight upon updating the weights

0.0001
WD

Weight Decay: regularization technique by adding a small penalty to the loss function after each update to prevent overfitting and to prevent the weights from growing too large

4. Training

Now that the pre-trained model has been loaded, let's proceed to prepare and carry out the fine-tuning of the additional classifier layers.

First, let's set all relevant training parameters. Note that SoftMax is used as a loss function, since this is a classifier task, and accuracy is employed as a performance metric. This last selection is valid since the classes in the custom dataset are balanced.

Note that all of these hyperparameters are selected based on their effect on the test set. A series of tuning trainings have been carried out in order to select the learning rate, its rate of decay, and the epochs at which it decays. In order to select the number of epochs for training, the point at which the accuracy of the test set diverges with respect to the accuracy of the training set is selected.

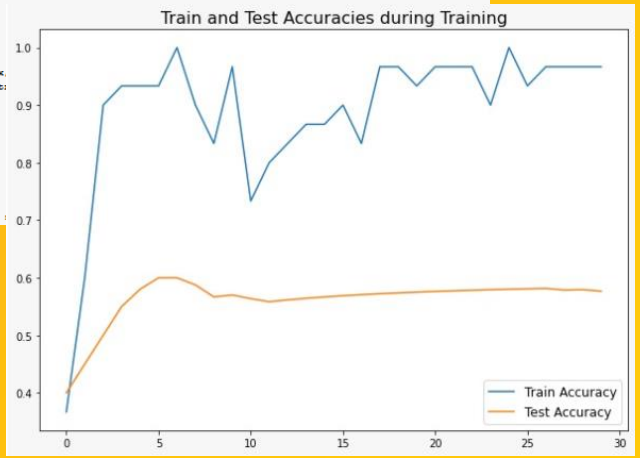
```
# Learning rate decay factor
lr_decay = 0.1
# Epochs where learning rate decays
lr_decay_epoch = [10, 20, 30]

# Stochastic gradient descent
optimizer = 'sgd'
# Set parameters
optimizer_params = {'learning_rate': 0.005, 'wd': 0.0001, 'momentum': 0.9}

# Define our trainer for net
trainer = gluon.Trainer(net.collect_params(), optimizer, optimizer_params)
```

```
loss_fn = gluon.loss.SoftmaxCrossEntropyLoss()
```

```
train_metric = mx.metric.Accuracy()
test_metric = mx.metric.Accuracy()
train_history = TrainingHistory(['training-acc', 'testing-acc'])
```

Now, let's carry out the training following the specified parameters.

```
epochs = 8
lr_decay_count = 0
test_accuracy = []
train_accuracy = []

for epoch in range(epochs):
    tic = time.time()
    train_metric.reset()
    train_loss = 0

    # Learning rate decay
    if epoch == lr_decay_epoch[lr_decay_count]:
        trainer.set_learning_rate(trainer.learning_rate*lr_decay)
        lr_decay_count += 1

    # Loop through each batch of training data
    for i, batch in enumerate(train_data):
        # Extract data and label
        data = split_and_load(batch[0], ctx_list=ctx,
        label = split_and_load(batch[1], ctx_list=ct

        # AutoGrad
        with ag.record():
            output = []
            for _, X in enumerate(data):
                X = X.reshape((-1,) + X.shape[2:])
                pred = net(X)
                output.append(pred)
            loss = [loss_fn(yhat, y) for yhat, y in
```

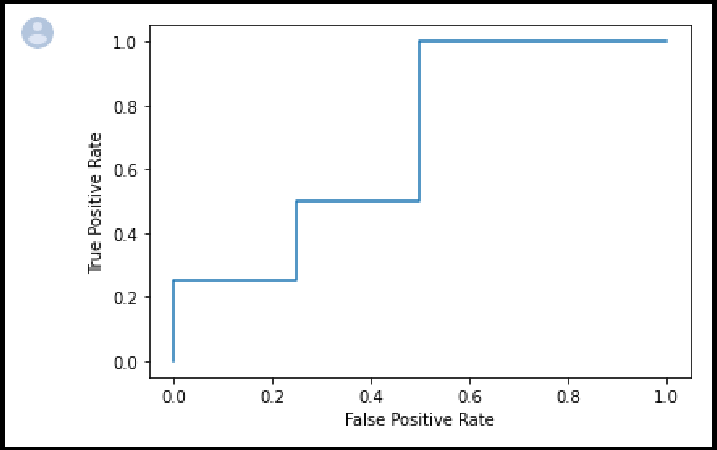


Model Evaluation

Model

Model Accuracy

62.5%



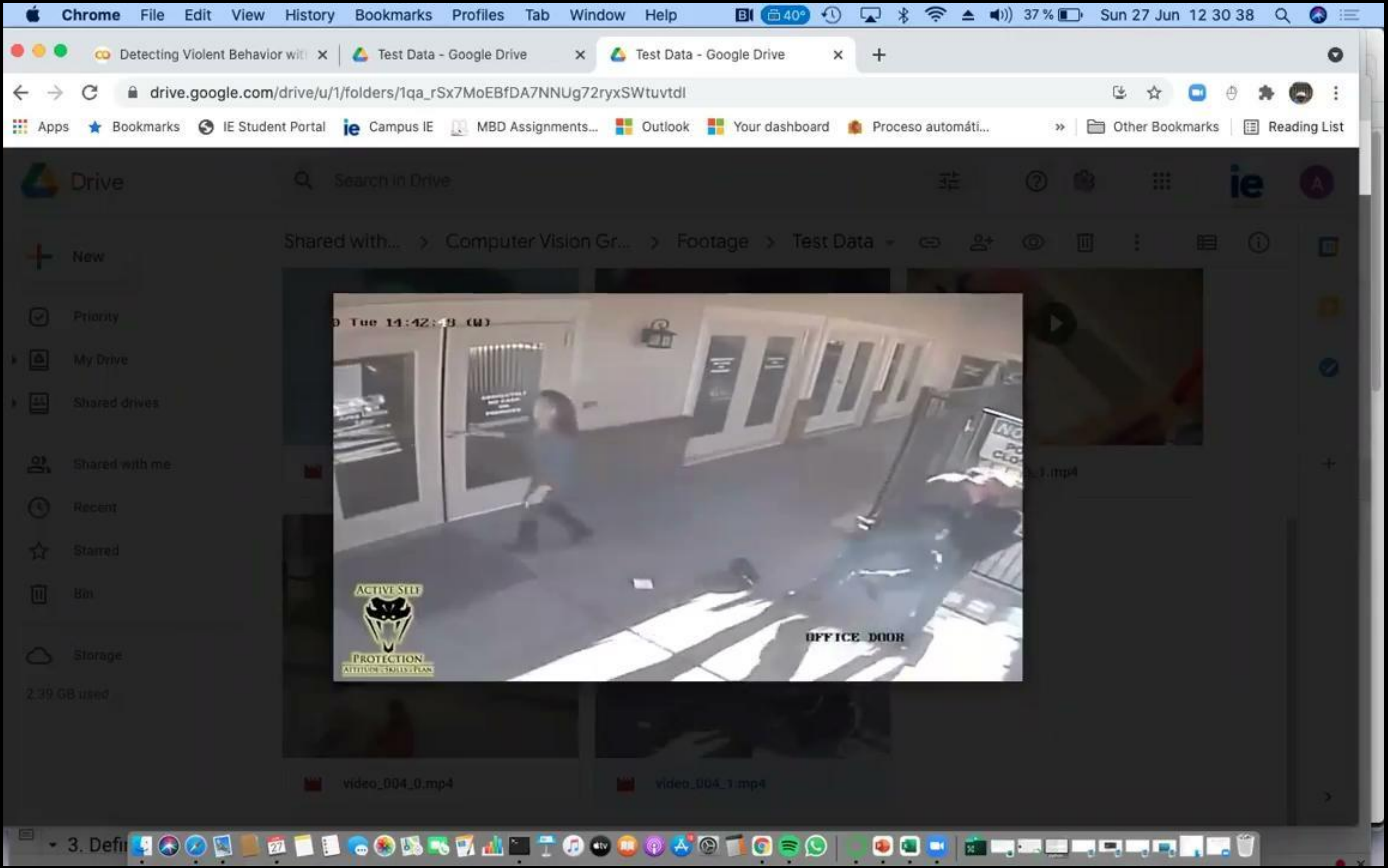
- Testing Model Performance by predicting on a new set of videos, a **validation sample with balanced classes**
- Our model obtained an **Accuracy of 62.5 %** on a never before seen validation set - only slightly better than random guessing

ROC Curve:

- Objective: minimize false negatives (wrongfully classified as non-violent) at the cost of a larger amount of false positives (wrongfully classified as violent)
- Result: ROC curve of our model advances step-wise

Classification Example

Demo


Chrome File Edit View History Bookmarks Profiles Tab Window Help
37 %
Sun 27 Jun 12 30 38
Detecting Violent Behavior wit
Test Data - Google Drive
Test Data - Google Drive
drive.google.com/drive/u/1/folders/1qa_rSx7MoEBfDA7NNUg72ryxSWtuvtdl
Apps
Bookmarks
IE Student Portal
Campus IE
MBD Assignments...
Outlook
Your dashboard
Proceso automáti...
Other Bookmarks
Reading List
Tue 14:42:
ACTIVE SELF
PROTECTION
OFFICE DOOR
video_004_0.mp4
video_004_1.mp4

Critical Evaluation

CURRENT LIMITATIONS & FUTURE IMPROVEMENTS

Current Limitations
of our model

- **Low Accuracy:** model currently only achieves an accuracy of 62.5 %
- **Training Data Quantity:** model trained on a small data set only due to (a) time- & capacity constraints, and (b) lack of a standardized process for extracting & labelling data
- **Representative Training Data:** training samples contain multiple persons carrying out different activities - pre-trained I3D model was trained on the Kinetics 400 dataset with few people carrying out a similar activity

Future Improvements
of our model

- Increase the amount of **training data and training time** and assess impact on model performance and probability threshold
- Perform **systematic parameter search** with a larger dataset
- Expand **experience with MXNET library** to fully utilize all parameters


CREATIVE PRESENT
THANK YOU
COMPUTER VISION