A Project Report

on

Multimodal Content Annotation for Identification of Comic Mischief

Submitted by

Sumith Sai Budde[1], 4th Year,Dept. Of CSE,
IIIT Dharwad, Regd. No: 18BCS101

Shaik Fharook[2], 4th Year,Dept. Of CSE,
IIIT Dharwad, Regd. No: 18BCS091

Syed Sufyan Ahmed[3], 4th Year, Dept. Of CSE,
IIIT Dharwad, Regd. No: 18BCS103

Shubham Shinde[4], 4th Year, Dept. Of CSE,
IIIT Dharwad, Regd. No: 18BCS095

Under the guidance of

Dr. Sunil Saumya

Assistant Professor, Dept. Of CSE,

IIIT Dharwad.

Indian Institute of Information Technology Dharwad

Department of Computer Science and Engineering

CERTIFICATE

We hereby certify that the work which is being presented in the Mini Project II report entitled “Multimodal Content Annotation for Identification of Comic Mischief” in partial fulfilment of the requirements for the award of B.Tech degree and submitted to the Department of Computer Science and Engineering of Indian Institute of Information Technology Dharwad is an authentic record of our own work carried our during the period from August 2021 to November 2021 under the supervision of Dr. Sunil Saumya, Assistant Professor, Dept. Of CSE, IIIT Dharwad.

The matter proposed in this report has not been published earlier and has never been submitted by us for the award of any other degree elsewhere.

Name of the Students:

Sumith Sai Budde	18BCS101
Shaik Fharook	18BCS091
Syed Sufyan Ahmed	18BCS103
Shubham Shinde	18BCS095

Dr. Sunil Saumya	Dr. Sadhvi Manerikar	Dr. Uma Sheshadri
Project Supervisor	**Project Coordinator**	**HOD CSE**

Table of Contents

1. Introduction

Nowadays, streaming services are ubiquitous, and everyone has access to them. This poses serious issues for youngsters who are exposed to content that is not age-appropriate. Automated tools for labelling online content based on the existence of questionable content can help safeguard consumers from being exposed to unpleasant or improper content.

Many studies have shown no communities are fully immune on the impacts of modern new technologies where it becomes an influential force on the global social structure. Because of its fast-rapid spread in our community, it becomes a source of information entertainment and cultural trait. In addition, technology gave the opportunity to the community to learn more and absorb new concepts to stay up-to-date with the current world, while children being sensitive and more adaptive to these changes.

As children become more active online at a younger age, the possibility and probability that they'll see something inappropriate all depends on what they're doing online. Whether it's an explicit pop-up ad on a free game, videos showcasing children's cartoon characters in adult situations, or a forum promoting self-harm, an innocent search can expose children to content that can make them feel upset and confused. It can be difficult to monitor what a child is viewing as they can access this material through any internet enabled device, including mobile ones such as a phone or tablet. Hence there is a need for detecting mischievous content from videos and audios.

In this report, we approach the task of the automated detection of questionable comic mischief in videos. By comic mischief, we mean the appearance of mild harm inflicted on video characters in a manner that is intended to be humorous or funny. The content itself can be present in more than one single modality (soundtrack, dialogue, or video). We intend to motivate new multimodal research that can detect the presence of this type of content. Among the primary applications is the automated labelling of comic mischief in videos easily accessible to young viewers.

2. Literature Survey

Video Classification task has gained a significant popularity and success in the recent years after the emergence of deep learning models[1]. Video content is created and consumed by people all around the world. Every every day, nearly 1 billion hours of video are seen by different people on YouTube alone. With the vast amount of new content generated daily, children are highly exposed to the offensive content and thus creates an importance to automate the process of identifying the offensive content. Companies such as Google AI are investing in several competitions to solve the difficult challenge under confined constraints. Google AI has published a public dataset called YouTube-8M comprising millions of video attributes and over 3700 labels to enhance the advancement of the automatic video categorization work.

Many studies justify that Artificial Neural Network(ANN), an algorithm based on the interconnected nodes to recognize the relationships in a set of data, have shown a great success in modelling both the linear and non-linear relationship in the underlying data and these are extensively used in different real-time applications[1]-[5].

There are several video classification techniques available at the time. Many of these techniques are quantitative, and they provide a methodical way for categorising data and extracted characteristics to distinct aspects connected to requirements for computing a priority. Later in 2016 Z. Wu [7] gave a succinct overview of video classification using deep learning techniques. This paper covers deep learning models, feature extraction tools, a benchmark dataset, and a comparison of existing video categorization algorithms. Recurrent Neural Network (RNN) [6] has recently been expressed as an effective class of models for comprehending image contents. It provides cutting-edge results in image recognition, segmentation, detection, and retrieval.

3. Methodology

3.1 Dataset Description

In this project we consider using the dataset presented by MOCHA Grand Challenge @ ICMI 2021. The dataset is a collection of scenes from videos hosted in Youtube and IMDB. The dataset contains 998 scenes extracted from 347 videos. For this project, there are two tasks that aim at evaluating different settings of the problem.

Task 1: Binary detection of comic mischief at the scene level

We should assign a scene a binary label indicating the presence of comic mischief. The idea is to provide a localized labeling of content with coarse (present/absent) binary labels.

Task 2: Fine-grained detection of comic mischief at the scene level

The difference between this task and task 1 above is that instead of a binary labeling scheme, we will use the four fine-grained labels: mature humor, slapstick humor, gory humor and sarcasm. This makes the task a multi label classification.

3.1.1 Binary Classification

The binary classification is interested only in the presence of comic Mischief content in the scene and not with the type of comic mischief present. Usually,

- **Level 0:** Absence of Comic mischief in the scene.
- **Level 1:** Presence of Comic mischief in the scene.

The preliminary data exploration shows that approximately 300 scenes contain label 0 and 700 scenes contain label 1. An bar plot can shown in Figure 1: Presence of Mischief Content in Scene

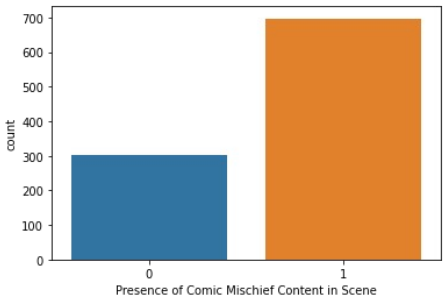


Figure 1: Presence of Mischief Content in Scene

3.1.2 MultiLabel Classification

A single scene may be associated to more than one comic-mischief label. The labels given in the dataset are as follows

- **Mature Humor:** Comic content with adult themes, such as content including political satire, sexual innuendos, or other themes that are not easily grasped by children.
- **Slapstick Humor:** Specific style of humor that depicts accidental or intentional violence, usually involving props (such as a slapstick).
- **Gory Humor:** This category refers to content that presents graphic violence, or the result of extremely violent actions, in a humoristic manner.
- **Sarcasm:** Content that relies on irony as the primary mechanism for delivering the comic content.

An illustration of numbers of each type can be shown at Figure 2: MultiLabel Classification

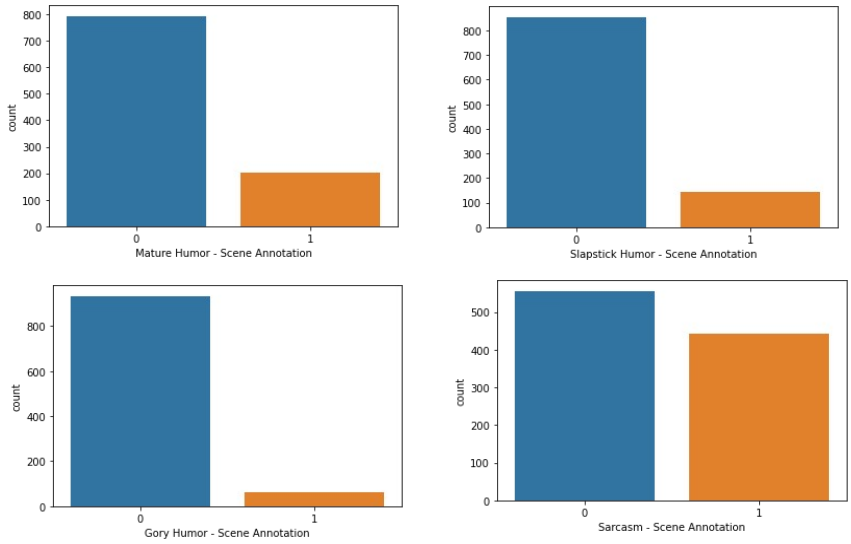


Figure 2: MultiLabel Classification

3.2 Data Pre-Processing

It's a common practise to pre-process the data before modelling. At this stage, different types of noise and corrupt data is/are removed and normalized for the best performance of the model. A Video is a sequence of images that have been arranged in a specific order. These sequences of images are also referred as frames. Unlike the image classification task, where images are processed to feature extractors like CNN to extract features and then classifying the image based on the features, Video Classification is a little bit tricky. Thus, we followed two approaches.

The typical preprocessing of video contains these following steps.

- Breaking all the scenes to frames.
- Applying the preprocessing techniques like resizing to the frames.

A visualisation of Video to Frames can be shown in Figure 3: Video to Frames

Figure 3: Video to Frames

3.2.1 Video Pre-processing – Approach 1: The Tradational way

In Approach 1, we follow the tradational video processing technique. Since a video is a sequence of images, we understand that video preprocessing is nothing but typical image classification with a little extra overhead of breaking of video to images or frames. The steps we followed in Approach 1 is as follows:

a) Download all the Youtube and IMDB Videos using the given links

b) Break every video into scenes of duration length 60 seconds each.

c) Clean the CSV file by removing the missing scene information.

d) Extract frames from each scene with required framerate. In this case, we used 1fps i.e., 60 frames per one scene of length 60 sec.

e) Create a dataframe to store the extracted frames and their respective labels.

f) Read the earlier extracted frames and apply the following image preprocessing techniques.

- Load the image
- Resize the image to (224,224,3)
- Convert the image to numpy array.
- Normalize the pixel values and store all the preprocessed data in a numpy array.

This creates the tensor shape (No. of images,224,224,3)

3.2.2 Video Pre-processing – Approach 2: Using Generators

After a few experiments conducted using the technique of Appraoch 1, we realised that we're missing an important factor i.e., sequence. Since the technique used in approach 1 doesn't help in modelling the entire sequence of frames at once and thus it leads us to do a bit more research on processing the entire sequence of frames at once for modelling.

As we gain access to an ever-increasing amount of data, large datasets become more prevalent in our lives. Even the state-of-art configuration computing machines fails to load such large amounts of data in memory, thus leads us to use Data Generators. Data Generators helps us to load the data in batches and to reduce the high memory consumpion.

With the help of Data Generators, we specifically built a data pipeline that loads scenes in batches, pre-process and pass to modelling in batches by minimizing the case of Resource Allocation errors. The steps used in this approach are as follows:

a) Clean the CSV file by removing the missing scene entries.

b) Read the Configuration file and load the scenes with their respective labels.

c) Break each scene (max duration = 60sec) into required MAX FRAMES. In this case, the max frames are set to 60.

   (a) If duration of scene >= 60 sec:

   - Break the scene into frames with framerate 1fps.

   (b) If duration of scene < 60 sec:

   - Break the scene into frames with frames 1fps and append leftover with blank frames to maintain uniformity.

d) Apply the image preproccessing techniques discussed in Approach 1 to all the frames.

e) Append all the preprocessed frames to a numpy array and the label.

f) Pass the numpy array for modelling and repeat for the next batch.

A detailed flowchart of Approach 2 using DataGenerators can be visualized at Figure 4: Flow Chart of Video Preprocessing Approach 2

3.2.3 Audio Pre-processing

Similar to video preprocessing, Audio preprocessing is a bit tricky. By taking the advantage of ffmpeg library and moviepy editor, we first seperate the audio from video and store in .wav form and seperate the extracted audio into their respective labels. By using audio sampling techniques, features from .wav formatted audio waves can be extracted i.e., converting the analog wave form to digital representable values to perform digital signal analysis. The steps used for audio preprocessing are as follows

a) Clean the CSV file by removing the missing scene entries.

b) Extract audio from video in .wav format using ffmpeg.

c) Apply Audio Augmentation Techniques to balance the data.

d) Analyse the spectogramof each audio signal and pass for modelling.

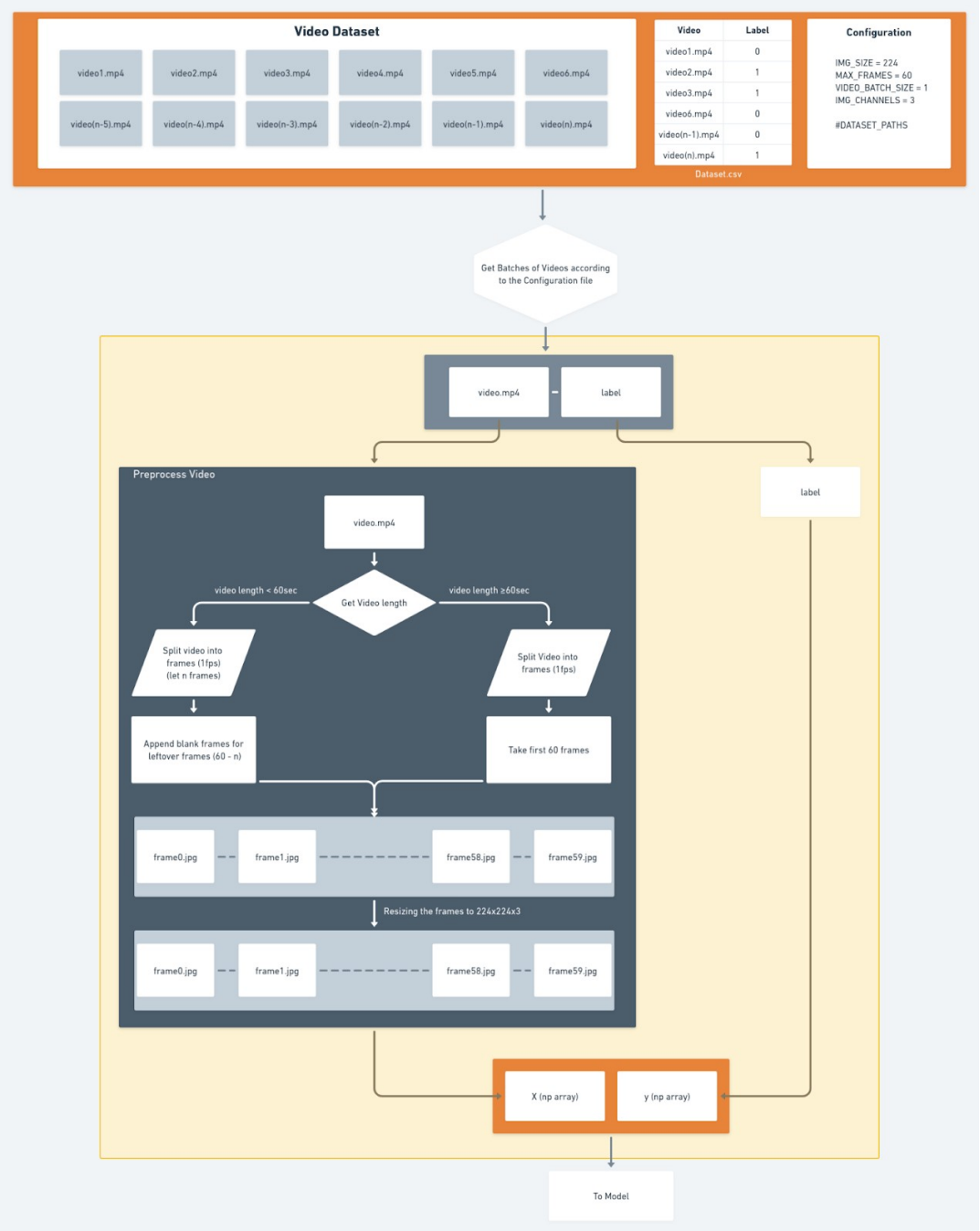


Figure 4: Flow Chart of Video Preprocessing Approach 2

3.3 Model Description

In this project, we've experimented 7 models of both binary and multilabel classification. These models include both video classification and audio classification.

3.3.1 Binary Classification (Approach 1)

3.3.1.1 CNN + Dense Model

Convolution Neural Network, shortly CNN is a type of neural network designed to extract higher representations specially for image content. Typical CNN's takes the 2D image pixel values as input and extract the features useful for classification tasks. CNN floows a heirarchical model which works on building a network, like a funnel, and finally gives out a fully connected dense network. The general CNN architecture can be shown at Figure 5: CNN Architecture

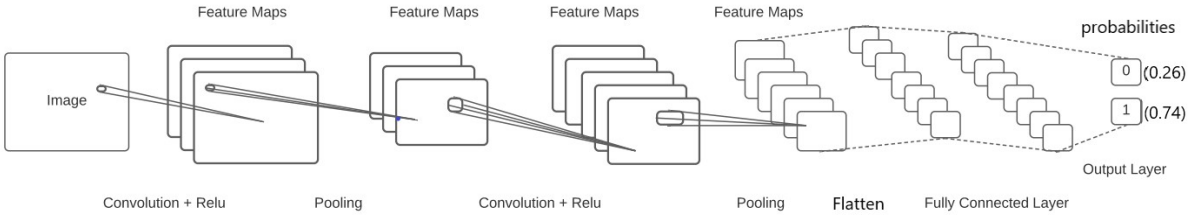


Figure 5: CNN Architecture

Each image input is passed through a convolution layers with filters then pooling, finally to fully connected dense network. Here, the softmax activation is applied at the last layer of dense network to get the classification results in probalistic values between 0 and 1. The Model's architecture with it's respective weights can be shown at Figure 6: CNN + Dense Approach 1

CNN + Dense	
Layers	**Weights**
Input Layer	(224,224,3)
Conv2D	Filters=64, kernel_size=(3,3)
Activation	Relu
Maxpooling2D	pool_size=(2,2)
Dropout	0.25
Conv2D	Filters=64, kernel_size=(3,3)
Activation	Relu
Maxpooling2D	pool_size=(2,2)
Dropout	0.25
Flatten	
Dense	128
Dropout	0.25
Dense	2
Activation	Softmax

Figure 6: CNN + Dense Approach 1

3.3.1.2 VGG 16 + Dense

VGG16 is a convolution neural network (CNN) architecture named after the Visual Geometry Group from Oxford. It follows an arrangement of convolution and maxpool layers consistently throughout the whole architecture.

The input to the network is image of dimensions (224,224,3). The first two layers have 64 channels of 3*3 filter size and same padding. Then after a max pool layer of stride (2,2), two layers which have convolution layers of 128 channels of filter size (3,3). This is followed by a max pooling layer of stride (2,2) which is same as the previous layer. Then there are 3 convolution layers of filter size (3,3) and 256 channel and a max pool layer. After that there are 2 sets of 3 convolution layers and a max pool layer. Each have 512 channels of (3,3) filters with same padding. They are followed by fully connected layers with last layer having a softmax activation for predicting the probabilities in the range of 0 to 1.

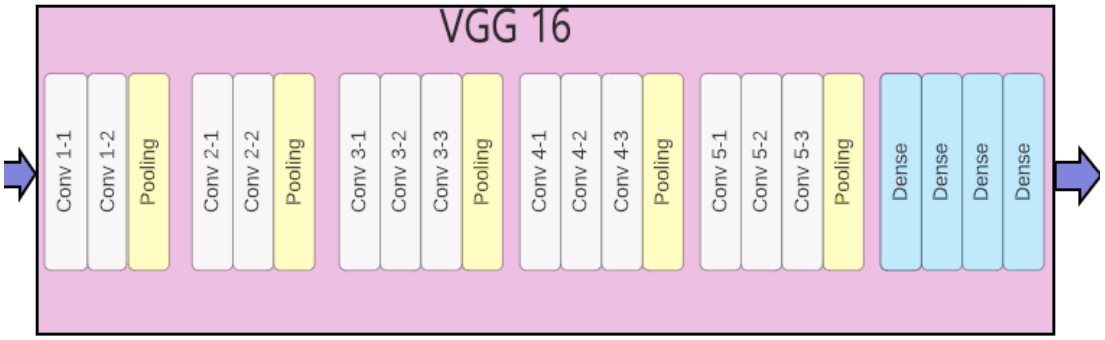


Figure 7: VGG16 Architecture

VGG16 + Dense	
Layers	**Weights**
Input Layer	(224,224,3)
VGG16	imagenet
Flatten	
Dense	1024
Dropout	0.5
Dense	512
Dropout	0.5
Dense	256
Dropout	0.5
Dense	128
Dropout	0.5
Dense	2
Activation	Softmax

Figure 8: VGG16 + Dense (Approach 1)

The architecture of VGG16 can be show at Figure 7: VGG16 Architecture and the VGG16 + Dense Model along with the weights can be shown at Figure 8: VGG16 + Dense (Approach 1)

3.3.2 Binary Classification (Approach 2)

We know that using the technique discussed in approach 2, we get the input as a sequence of preprocessed frames. We're well known that CNN and other state-of-art frameworks like VGG16, ResNet50 etc. Work great on images, but video has an extra little dimension called time and this makes clear that a video classification task is a sequence task. For a sequence classification task, we desperately use RNN as

- RNN's use the output from the previous step as input to the current step.
- An RNN also provides the opportunity to work with sequences of vectors both in the input and output. This feature allows an RNN to remember a sequence of data.

LSTM (Long Short Term Memory)

LSTM is a type of artificial neural network architecture used to process multiple input data points in images, speech, audio as well as text. It consists of a cell and three gates, an input gate, forget gate and output gate. Research studies have proven that LSTM is a well suited RNN framework for modelling time series / sequence classification jobs.

Unlike other architectures, LSTM has connections for feedback which are helpful regulating the information flow through the gates. The architecture is designed in such a way that it can remember the long-term dependencies of the data being presented to it. It could over- come the vanishing gradient problem that arises when using Recurrent Neural Networks. These models can be trained in both super- vised and unsupervised manner.

The LSTM Cell Structure can be shown at Figure 9: LSTM Cell Structure. The following discussed models uses LSTM as a base model and gets the input from the TimeDistributed Layer, a specially designed layer to hold sequences of data and gets features from feature extractor. Two experiments were conducted with changing of feature extractor keeping the constant base model. A detailed flow chart of how this model works can be shown at Figure 10: LSTM for Video Classification

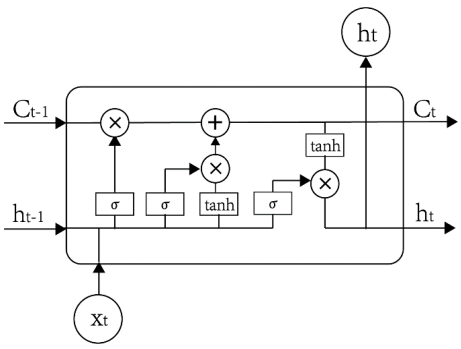


Figure 9: LSTM Cell Structure

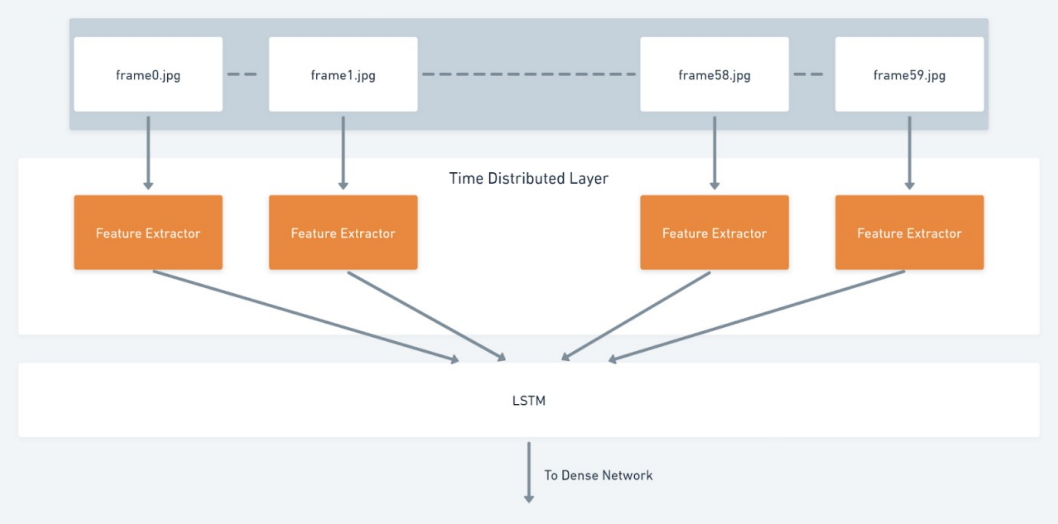


Figure 10: LSTM for Video Classification

3.3.2.1 CNN + LSTM + Dense

Keeping the Base LSTM Model constant, the complete model with simple CNN as feature extractor along with respective used weights can be shown at Figure 11: CNN + LSTM + Dense (Approach 2)

CNN + LSTM + Dense	
Layers	**Weights**
Input Layer	(224,224,3)
Conv2D	Filters=32, kernel_size=(3,3)
Activation	Relu
Conv2D	Filters=32, kernel_size=(3,3)
Activation	Relu
Batch Normalization	Momentum=0.9
Max Pooing	pool_size=(2,2)
Conv2D	Filters=32, kernel_size=(3,3)
Activation	Relu
Conv2D	Filters=32, kernel_size=(3,3)
Activation	Relu
Batch Normalization	Momentum=0.9
Max Pooing	pool_size=(2,2)
Time Distributed	
LSTM	64
Dense	128
Dropout	0.5
Dense	64
Dropout	0.5
Dense	32
Dropout	0.5
Dense	1
Activation	Sigmoid

Figure 11: CNN + LSTM + Dense (Approach 2)

3.3.2.2 VGG16 + LSTM + Dense

Keeping the Base LSTM Model constant, the complete model with simple CNN as feature extractor along with respective used weights can be shown atFigure 12: VGG16 + LSTM + Dense (Approach 2)

VGG16 + LSTM + Dense	
Layers	**Weights**
Input Layer	(224,224,3)
VGG16	imagenet
Time Distributed	
LSTM	64
Dense	128
Dropout	0.5
Dense	64
Dropout	0.5
Dense	32
Dropout	0.5
Dense	1
Activation	Sigmoid

Figure 12: VGG16 + LSTM + Dense (Approach 2)

3.3.3 MultiLabel Classification (Approach 1)

3.3.3.1 CNN + Dense

CNN + Dense	
Layers	**Weights**
Input Layer	(224,224,3)
Conv2D	Filters=64, kernel_size=(3,3), Activation= Relu
MaxPooling	pool_size=(2,2)
Dropout	0.25
Conv2D	Filters=64, kernel_size=(3,3), Activation= Relu
MaxPooling	pool_size=(2,2)
Dropout	0.25
Conv2D	Filters=64, kernel_size=(3,3), Activation= Relu
MaxPooling	pool_size=(2,2)
Dropout	0.25
Conv2D	Filters=64, kernel_size=(3,3), Activation= Relu
MaxPooling	pool_size=(2,2)
Dropout	0.25
Flatten	
Dense	128
Dropout	0.25
Dense	64
Dropout	0.25
Dense	4
Activation	Sigmoid

Figure 13: CNN + Dense Multilabel Classification

3.3.3.2 VGG16 + Dense

VGG16 + Dense	
Layers	**Weights**
Input Layer	(224,224,3)
VGG16	imagenet
Flatten	
Dense	1024
Dropout	0.5
Dense	512
Dropout	0.5
Dense	256
Dropout	0.5
Dense	128
Dropout	0.5
Dense	4
Activation	Sigmoid

Figure 14: VGG + Dense Multilabel Classification

3.3.4 Audio SVM Classifier

Support Vector Machine(SVM) is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs.

We train an SVM classifier to binary annotate our audio data. Here we train n-SVM's with different constraint(i.e, C - values) and select the best performing model which is giving high accuracy. We chose SVM to try kernel methods for our task.

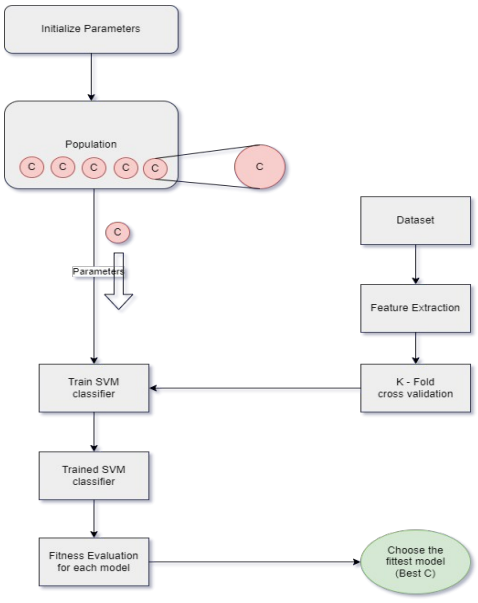


Figure 15: SVM Classifier

We perform data augmentation to tackle class imbalance and overfitting by applying audio data augmentation methods used in Digital Signal Processing like

1. Gaussian Noise Addition
2. Shifting of Sound Wave
3. Time Stretching of Wave
4. Pitch Shifting of Wave

Here we will not be able to apply a method similar to flipping of an image because for us humans if we hear an audio in reverse manner it doesn't make any sense both syntactically and semantically.

3.4 Optimizers, Loss Functions, Performance Metrics

3.4.1 Optimizer Adam (Learning Rate=1e^{-5})

Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

3.4.2 Loss Functions

Loss functions measure how far an estimated value is from its true value. A loss function maps decisions to their associated costs. If predictions deviates too much from actual results, loss function would cough up a very large number. Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.

3.4.2.1 Binary Crossentropy

Binary cross entropy compares each of the predicted probabilities to actual class output and then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. Binary Cross Entropy is the negative average of the log of corrected predicted probabilities.

$$\log(loss) = 1/N \sum_{i=1}^{N} -(y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Here , p_i is the probability of class 1 and $(1 - p_i)$ is the probability of class 0. When the observation belongs to class 1, the first part of the formula becomes active and the second part vanishes and vice versa in case the observation belongs to class 0.

3.4.3 Metrics

The only metrics used is Accuracy.

4. Result & Analysis

4.1 Binary Classification (Approach 1)

4.1.1 CNN + Dense

We have trained the model on images of all the scenes together hence the output will also be frame level. We have to process the frame level output probabilities and get scene level labels.

Image level Probablilities:

```
array([[0.00142997, 0.99856997],
       [0.98282653, 0.0171735 ],
       [0.9859698 , 0.01403028],
       ...,
       [0.7418433 , 0.2581567 ],
       [0.00523358, 0.9947665 ],
       [0.70084447, 0.2991555 ]], dtype=float32)
```

Extracting Labels from predicted probabilities:

If probability of image belonging to class 0 is higher then assign label 0 and vice versa.
ex. Label([0.00142997,0.99856997]) = 1

```
array(['1', '0', '0', ..., '0', '1', '0'], dtype='<U1')
```

Slicing the Output Vector:

The Output and results which we got from the model are frame level predictions. But we need scene level predictions and have to calculate scene level accuracy. For this we have sliced the output vector according to number of frames of a particular scene. Then calculated the “mode” of each slice to get the most occurring label in that slice and assign it to the corresponding scene as its class.

Output Scene Labels:

```
array(['0', '0', '0', '1', '1', '0', '0', '1', '0', '0', '0', '0', '0',
       '0', '0', '0', '1', '0', '0', '0', '0', '1', '1'], dtype='<U1')
```

Accuracy:

- Image Level Training Accuracy : 0.79
- Image Level Training Loss : 0.50
- Image Level Validation Accuracy : 0.55
- Image Level Validation Loss : 0.80
- Scene Level Accuracy : 0.47

4.1.2 VGG16 + Dense

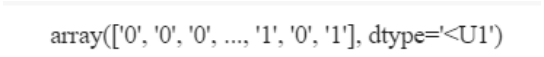
We have to process the frame level output probabilities and get scene level labels.

Image level Probabilities:

```
array([[0.8911456 , 0.10885445],
       [0.95572567, 0.04427435],
       [0.9606761 , 0.03932391],
       ...,
       [0.42520702, 0.57479304],
       [0.540679  , 0.459321  ],
       [0.40838382, 0.5916162 ]], dtype=float32)
```

Extracting Labels from predicted probabilities:

If probability of image belonging to class 0 is higher then assign label 0 and vice versa.

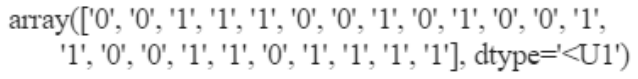


Slicing the Output Vector:

The Output and results which we got from the model are frame level predictions. But we need scene level predictions and have to calculate scene level accuracy. For this we have sliced the output vector according to number of frames of a particular scene.

Then calculated the “mode” of each slice to get the most occurring label in that slice and assign it to the corresponding scene as its class.

Output Scene Labels:



Accuracy:

- Image Level Training Accuracy : 0.75
- Image Level Training Loss : 0.55
- Image Level Validation Accuracy : 0.50
- Image Level Validation Loss : 1.03
- Scene Level Accuracy : 0.60

4.2 Binary Classification (Approach 2)

4.2.1 CNN + LSTM + Dense

As we have trained the model using generator and passed each scene in batches, the output we get is scene level output. We don’t need any processing of the results.

Accuracy:

- Training Accuracy : 0.56
- Training Loss : 0.68
- Validation Accuracy : 0.56
- Validation Loss : 0.69

4.2.2 VGG16 + LSTM + Dense

As we have trained the model using generator and passed each scene in batches, the output we get is scene level output. We don’t need any processing of the results.

Accuracy:

- Training Accuracy : 0.48
- Training Loss : 0.69
- Validation Accuracy : 0.50
- Validation Loss : 0.69

4.3 Multilabel Classification (Approach 1)

4.3.1 CNN + Dense

We have trained the model on images of all the scenes together hence the output will also be frame level. We have to process the frame level output probabilities and get scene level labels.

Image level Probabilities:

```
array([[0.76528645, 0.10269621, 0.2660396 , 0.02793568],
       [0.76528645, 0.10269621, 0.2660396 , 0.02793568],
       [0.5637543 , 0.13654071, 0.4716452 , 0.12816086],
       ...,
       [0.7832081 , 0.18032071, 0.27500397, 0.07065243],
       [0.78296196, 0.17847168, 0.277574  , 0.06934941],
       [0.7851854 , 0.18249878, 0.26646796, 0.07127509]], dtype=float32)
```

Extracting Labels from predicted probabilities:

If probability of image belonging to a label is greater than 0.5, then assign the label to image.

ex. [0.76, 0.10, 0.62, 0.23] = [1 , 0 , 1 , 0]

```
array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       ...,
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0]])
```

Slicing the Output Vector:

The Output and results which we got from the model are frame level predictions. But we need scene level predictions and have to calculate scene level accuracy. For this we have sliced the output vector according to number of frames of a particular scene. Then calculated the “mode” of each slice to get the most occurring label in that slice and assign it to the corresponding scene as its class.

Output Scene Labels:

```
array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 1],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 1, 1],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 1, 0],
       [1, 0, 0, 0]])
```

Accuracy:

- Image Level Training Accuracy : 0.82
- Image Level Training Loss : 0.31
- Image Level Validation Accuracy : 0.86
- Image Level Validation Loss : 0.79
- Scene Level Accuracy : 0.25

4.3.2 VGG16 + Dense

We have trained the model on images of all the scenes together hence the output will also be frame level. We have to process the frame level output probabilities and get scene level labels.

Image level Probabilities:

```
array([[0.7889411 , 0.10799405, 0.30510044, 0.04266861],
       [0.7889411 , 0.10799405, 0.30510044, 0.04266861],
       [0.7906928 , 0.10659963, 0.30364725, 0.04169437],
       ...,
       [0.79084057, 0.10648248, 0.30352435, 0.04161295],
       [0.79084194, 0.10648137, 0.30352324, 0.04161218],
       [0.7908391 , 0.10648367, 0.3035256 , 0.04161373]], dtype=float32)
```

Extracting Labels from predicted probabilities:

If probability of image belonging to a label is greater than 0.5, then assign the label to image.

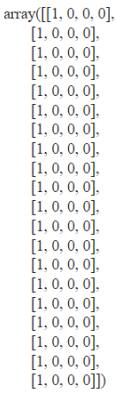
Ex: [0.76, 0.10, 0.62, 0.23] = [1 , 0 , 1 , 0]

```
array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       ...,
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0]])
```

Slicing the Output Vector:

The Output and results which we got from the model are frame level predictions. But we need scene level predictions and have to calculate scene level accuracy. For this we have sliced the output vector according to number of frames of a particular scene. Then calculated the “mode” of each slice to get the most occurring label in that slice and assign it to the corresponding scene as its class.

Output Scene Labels:



Accuracy:

- Image Level Training Accuracy : 0.91
- Image Level Training Loss : 0.15
- Image Level Validation Accuracy : 0.80
- Image Level Validation Loss : 0.69
- Scene Level Accuracy : 0.25

4.4 SVM Audio Classifier

For each audio we get the prediction of following type:

```
(0.0, array([0.60788104, 0.39211896]), ['class_0_Audio', 'class_1_Audio'])
```

Here, probability of audio belonging to class 0 is higher hence label is 0.Similarly for the entire validation set :

True labels : [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

Predicted Labels : [1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

Accuracy:

- Audio Train Accuracy = 0.75
- Audio Validation Accuracy = 0.778

4.5 Comparision of Results

			Image Level				Scene Level
Video Model Name	**Model Type**	**Epochs**	**Training Accuracy**	**Training Loss**	**Validation Accuracy**	**Validation Loss**	**Validation Accuracy**
Simple CNN + Dense (Approach 1)	Binary	10	0.79	0.51	0.55	0.81	0.47
VGG 16 + Dense (Approach 1)	Binary	10	0.75	0.55	0.50	1.03	0.60
Simple CNN + Dense (Approach 1)	Multilabel	10	0.82	0.32	0.86	0.79	0.25
VGG 16 + Dense (Approach 1)	Multilabel	10	0.91	0.15	0.8	0.69	0.25

Video Model Name	**Model Type**	**Epochs**	**Training Accuracy**	**Training Loss**	**Validation Accuracy**	**Validation Loss**
Simple CNN + LSTM (Approach 2)	Binary	10	0.56	0.68	0.56	0.69
VGG 16 + LSTM (Approach 2)	Binary	5	0.48	0.69	0.50	0.69

Audio Model Name	**Model Type**	**Training Accuracy**	**Validation Accuracy**
SVM	Binary	0.75	0.77

5. Conclusion & Future Scope

In this report we proposed various approaches to predict comic mischief from videos using different types of models and compared their results.

The accuracy of models was low as we have trained them on a small subset of data. Giving more data and fine tuning the parameters might improve the accuracy of the models. Improvements can be made in the models and also using ensemble learning on one or more models could also possibly increase accuracy. Audio modality can also be explored and used for prediction. Ensemble learning on both audio and video models could produce good results.

In future, several avenues may be explored to further improve performance. Further research on video classification could come up with various other solutions for tagging online video content.

6. References

[1] A. Rehman and S. B. Belhaouari, “Deep Learning for Video Classification: A Review”. TechRxiv, 16-Aug-2021 [Online]. Available: https://www.techrxiv.org/articles/preprint/Deep_Learning_for_Video_Classification_A_Review/15172920/1.

[2] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K. R. Müller, “Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications,” Proc. IEEE, vol. 109, no. 3, pp. 247– 278, 2021.

[3] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, “1D convolutional neural networks and applications: A survey,” Mech. Syst. Signal Process., vol. 151, 2021.

[4] N. Minallah, M. Tariq, N. Aziz, W. Khan, A. ur Rehman, and S. B. Belhaouari, “On the performance of fusion based planet-scope and Sentinel-2 data for crop classification using inception inspired deep convolutional neural network,” PLoS One, vol. 15, no. 9 September, 2020.

[5] A. U. Rehman and A. Bermak, “Averaging neural network ensembles model for quantification of volatile organic compound,” 2019 15th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2019, pp. 848– 852, 2019.

[6] D. Brezeale, D. J. Cook, and S. Member, \Automatic video Classification: A survey of the literature," IEEE Transactions on Systems, Man, and Cybernetics, Part C.

[7] Z. Wu, T. Yao, Y. Fu, and Y.-G. Jiang, “Deep learning for video classification and captioning,” Front. Multimed. Res., pp. 3–29, 2017.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-Scale Video Classification with Convolutional Neural Networks," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1725-1732, doi: 10.1109/CVPR.2014.223.

[9] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. ArXiv 1409.1556.

[10] Sun, Chen & Myers, Austin & Vondrick, Carl & Murphy, Kevin & Schmid, Cordelia. (2019). VideoBERT: A Joint Model for Video and Language Representation Learning. 7463-7472. 10.1109/ICCV.2019.00756.

[11] A Flexible CNN Framework for Multi-Label Image Classification Yunchao Wei, Wei Xia, Min Lin, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao, Senior Member, IEEE, and Shuicheng Yan, Senior Member, IEEE.