UNIVERSITY SCHOOL OF INFORMATION, COMMUNICATION AND TECHNOLOGY

To Detect Face And Calculate Screen Time Of Characters In a Video

SUBMITTED BY:

Akshay Mishra

00716403217

Btech Cse 7th Sem

SUBMITTED TO:

DR RL UJJWAL

INDEX

ABSTRACT

We present an end-to-end system for counting actor screen time in movies through detecting and clustering faces in a Video . We break the problem apart into two parts: face segmentation and facial recognition. Our data consists of trailers that we divide into different frames (images). Then, we solve the first part of the problem using state-of-the-art pre-trained models in order to detect bounding boxes around where the Actor faces are. Given this output, we feed the information into the second model which detects which Actor it is. We present results the show our quality predictions with different models.

INTRODUCTION

In the 4 minutes of Video from movie Jurrasic Park:, one of the highest grossing blockbuster of that time, which characters and stories were given the most minutes to deliver a performance worthy of entertainment news cycles, word of mouth reviews, and pop culture canon? To satisfy this curiosity, we apply deep learning and CNNs to Video to breakdown the screen time allocated to each actor or character. For our project, we will build an application to recognize faces and count how many seconds each character appears in a single video.

PROBLEM STATEMENT

As one of the most successful applications of image analysis and understanding, face recognition has recently gained significant attention, especially during the past several years. This is evidenced by the emergence of specific face recognition conferences such as AFGR and AVBPA, and systematic empirical evaluation of face recognition techniques (FRT), including the FERET and XM2VTS protocols. There are at least two reasons for such a trend: the first is the wide range of commercial and law enforcement applications and the second is the availability of feasible technologies after years of research.

A general statement of face recognition problem can be formulated as follows:
Given still or video images of a scene identify or verify one or more persons in the scene using a stored database of faces and calculate screen time of faces in video. The solution of the problem involves segmentation of faces (face detection) from cluttered scenes, feature extraction from the face region recognition or verification. In identification problems, the input to the system is an unknown face and the system reports back the decided identity from a database of known individuals, whereas in verification problems, the system needs to confirm or reject the claimed identity of the input face.

Proposed Suggestive Solution

1. Face Recognition in A video

Here we use technique called deep metric learning.

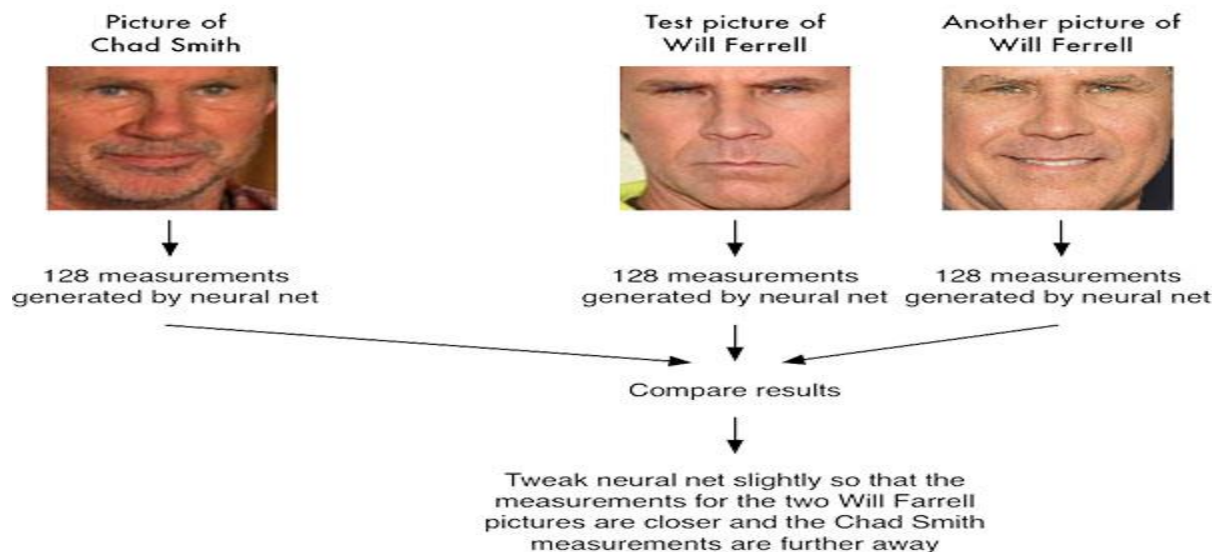In, deep learning you know that we typically train a network to:

- Accept a single input image

- And output a classification/label for that image

However, deep metric learning is different.

instead, of trying to output a single label (or even the coordinates/bounding box of objects in an image), we are instead outputting a real-valued feature vector.

For the dlib facial recognition network, the output feature vector is 128-d (i.e., a list of 128 real-valued numbers) that is used to quantify the face. Training the network is done using triplets

A single 'triplet' training step:

Picture of Chad Smith — 128 measurements generated by neural net

Test picture of Will Ferrell — 128 measurements generated by neural net

Another picture of Will Ferrell — 128 measurements generated by neural net

Compare results

Tweak neural net slightly so that the measurements for the two Will Farrell pictures are closer and the Chad Smith measurements are further away
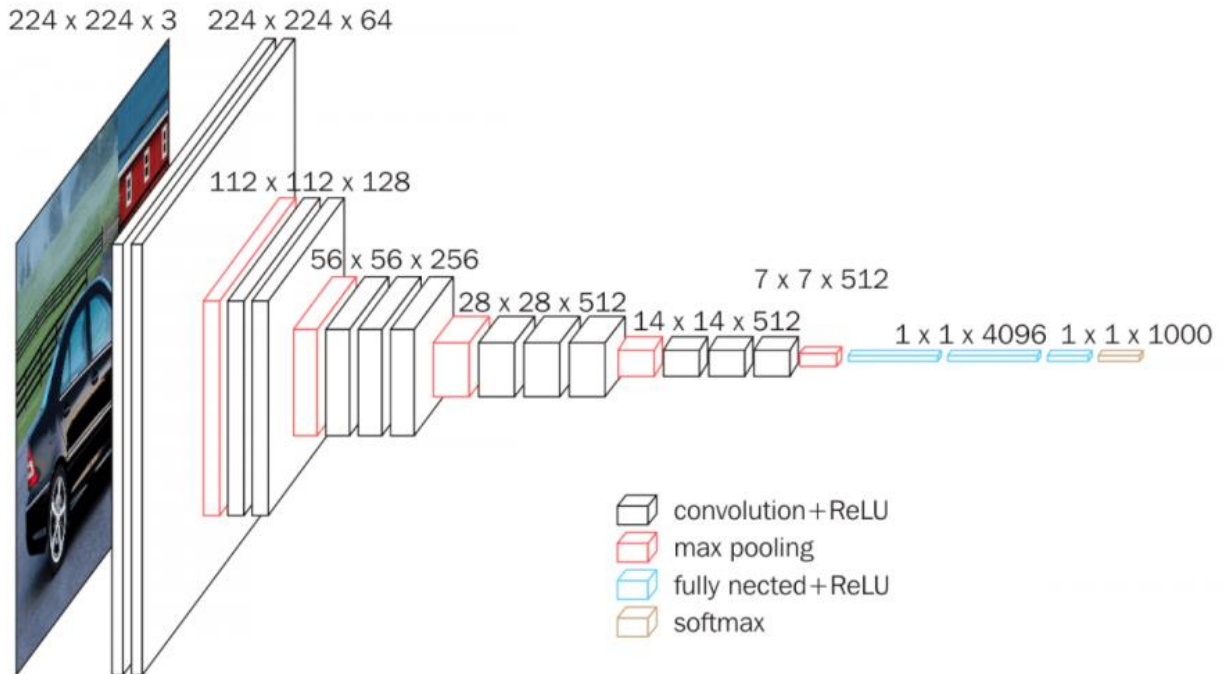
2.Calculate Screen Time Of characters

1 Preprocessing

Model should take images, not video, the first step is to convert movies into frames. Break down the video into frames taken one second apart. Then apply our deep learning model to each image in order to identify the faces that appear in each frame. The total screen time for each character is the sum of 1-second frames that they appear in.

2 Transfer Learning

Since we have only 4559 images, so it will be difficult to train a neural network with this little dataset. Here comes the concept of transfer learning.

With the help of transfer learning, we can use features generated by a model trained on a large dataset into our model. Here we will use VGG16 model trained on "imagenet" dataset. For this, we are using tensorflow high-level API Keras. With keras, you can directly import VGG16 model as shown in the code below.

224 x 224 x 3   224 x 224 x 64

112 x 112 x 128

56 x 56 x 256

28 x 28 x 512

7 x 7 x 512

14 x 14 x 512

1 x 1 x 4096   1 x 1 x 1000

convolution+ReLU
max pooling
fully nected+ReLU
softmax

VGG16 has around 138 million parameters. But in this project we will calculate screen time of only 2 actors. Therefore we are not including fully connected layers. After removing fully connected layers,we are left with 14 million parameters.

Since video is made up of 24 frames per second, we will count the number of frames which has been predicted for having "Alan Grant" in it and then divide it by 24 to count the number of seconds "Alan Grant" was on screen.

SYSTEM REQUIREMENTS

1 Hardware Requirements

- 4 GB RAM(min)
- Intel core processor(any processor ranging from i5 to i9)/Dual Core.
- 512 GB SSD/HDD(80 GB min).
- Camera.

1   Software Requirements

- Windows XP/7/8/10, Linux(Ubuntu)
- OpenCv
- Pandas
- Numpy
- Jupyter Notebook/Spyder
- Anaconda

PROJECT SNAPSHOTS

1.Face Recognition in Video

```python
#%%
import face_recognition
import argparse
import imutils
import pickle
import time
import cv2

# construct the argument parser and parse the arguments
#%%
ap = argparse.ArgumentParser()
ap.add_argument("-e", "--encodings", required=True,
    help="path to serialized db of facial encodings")
ap.add_argument("-i", "--input", required=True,
    help="path to input video")
ap.add_argument("-o", "--output", type=str,
    help="path to output video")
ap.add_argument("-y", "--display", type=int, default=1,
    help="whether or not to display output frame to screen")
ap.add_argument("-d", "--detection-method", type=str, default="hog",
    help="face detection model to use: either `hog` or `cnn`")
args = vars(ap.parse_args())
```

```python
#%%
# load the known faces and embeddings
print("[INFO] loading encodings...")
data = pickle.loads(open(args["encodings"], "rb").read())

# initialize the pointer to the video file and the video writer
print("[INFO] processing video...")
stream = cv2.VideoCapture(args["input"])
writer = None

# loop over frames from the video file stream
while True:
    # grab the next frame
    (grabbed, frame) = stream.read()

    # if the frame was not grabbed, then we have reached the
    # end of the stream
    if not grabbed:
        break

    # convert the input frame from BGR to RGB then resize it to have
    # a width of 750px (to speedup processing)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    rgb = imutils.resize(frame, width=750)
    r = frame.shape[1] / float(rgb.shape[1])

    # detect the (x, y)-coordinates of the bounding boxes
    # corresponding to each face in the input frame, then compute
    # the facial embeddings for each face
    boxes = face_recognition.face_locations(rgb,
        model=args["detection_method"])
    encodings = face_recognition.face_encodings(rgb, boxes)
    names = []

    # loop over the facial embeddings
    for encoding in encodings:
        # attempt to match each face in the input image to our known
        # encodings
        matches = face_recognition.compare_faces(data["encodings"],
            encoding)
```

```python
		name = "Unknown"

		# check to see if we have found a match
		if True in matches:
			# find the indexes of all matched faces then initialize a
			# dictionary to count the total number of times each face
			# was matched
			matchedIdxs = [i for (i, b) in enumerate(matches) if b]
			counts = {}

			# loop over the matched indexes and maintain a count for
			# each recognized face face
			for i in matchedIdxs:
				name = data["names"][i]
				counts[name] = counts.get(name, 0) + 1

			# determine the recognized face with the largest number
			# of votes (note: in the event of an unlikely tie Python
			# will select first entry in the dictionary)
			name = max(counts, key=counts.get)

		# update the list of names
		names.append(name)

	# loop over the recognized faces
	for ((top, right, bottom, left), name) in zip(boxes, names):
		# rescale the face coordinates
		top = int(top * r)
		right = int(right * r)
		bottom = int(bottom * r)
		left = int(left * r)

		# draw the predicted face name on the image
		cv2.rectangle(frame, (left, top), (right, bottom),
			(0, 255, 0), 2)
		y = top - 15 if top - 15 > 15 else top + 15
		cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
			0.75, (0, 255, 0), 2)

	# if the video writer is None *AND* we are supposed to write
```

```python
        # if the writer is not None, write the frame with recognized
        # faces t odisk
        if writer is not None:
            writer.write(frame)

        # check to see if we are supposed to display the output frame to
        # the screen
        if args["display"] > 0:
            cv2.imshow("Frame", frame)
            key = cv2.waitKey(1) & 0xFF

            # if the `q` key was pressed, break from the loop
            if key == ord("q"):
                break

# close the video file pointers
stream.release()

# check to see if the video writer point needs to be released
if writer is not None:
    writer.release()
```
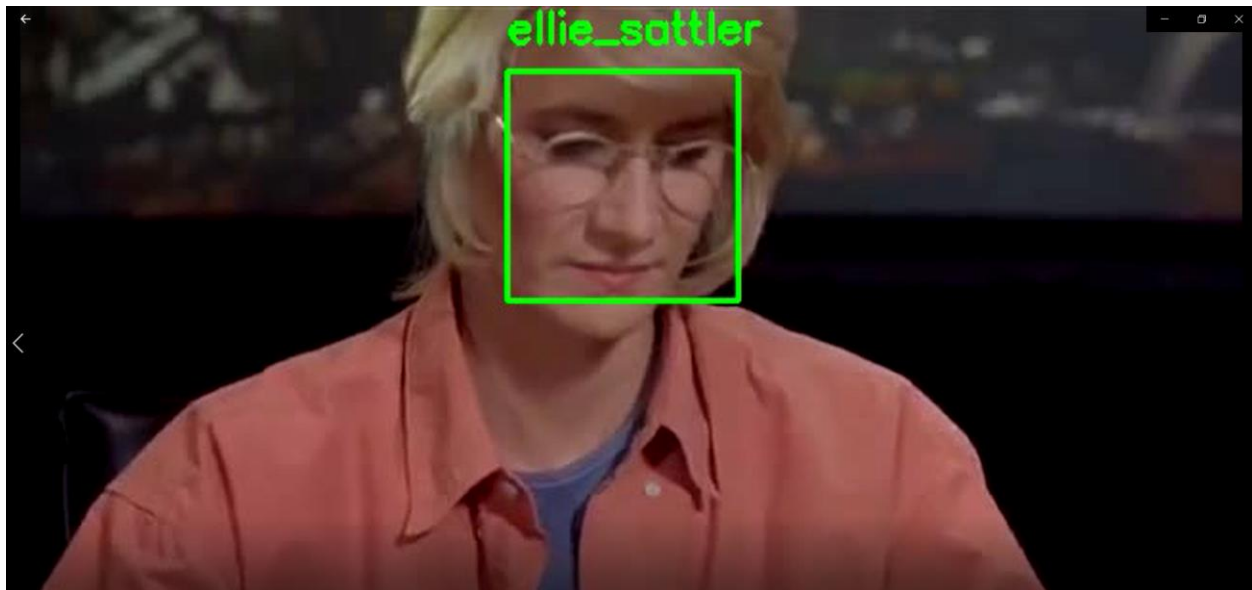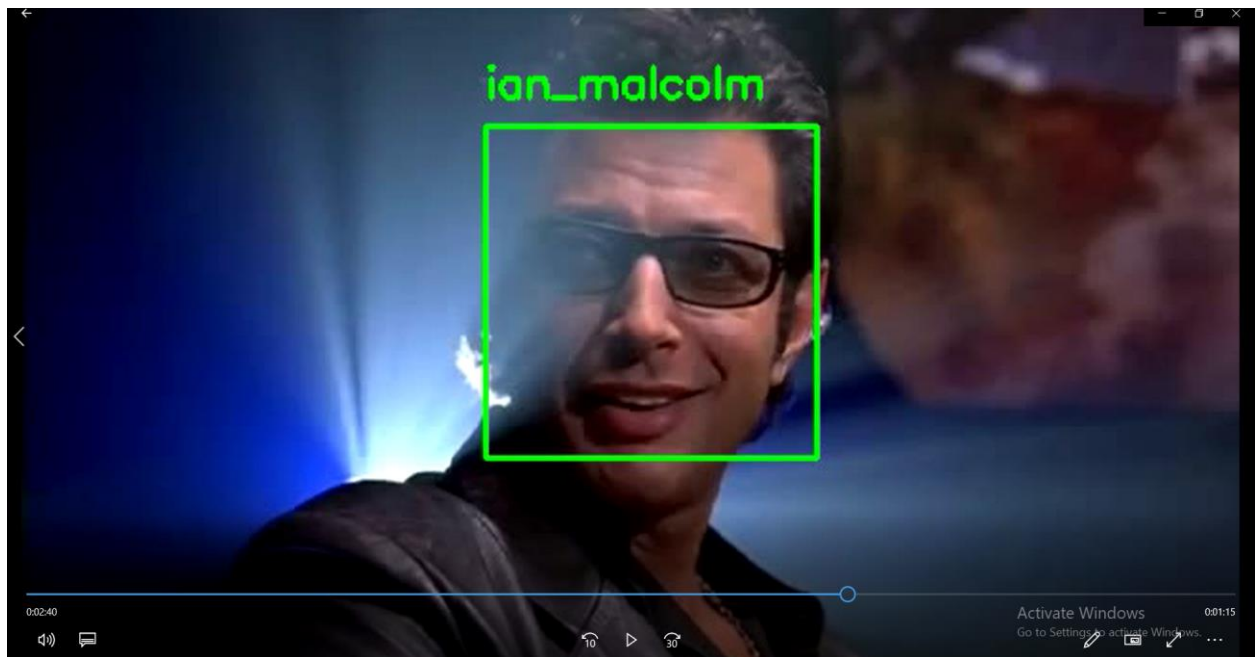
2.Calculate Screen time of Actors

```python
from tqdm import tqdm
import cv2
import os
import numpy as np
```

```python
img_path = '/content/drive/MyDrive/Colab Notebooks/Minor Project/data'

class1_data = []
class2_data = []
class3_data = []
class4_data = []
class5_data = []
for classes in os.listdir(img_path):
        fin_path = os.path.join(img_path, classes)
        for fin_classes in tqdm(os.listdir(fin_path)):
            img = cv2.imread(os.path.join(fin_path, fin_classes))
            img = cv2.resize(img, (224,224))
            img = img/255.
            if classes == 'alan_grant':
                class1_data.append(img)
            #elif classes == 'ellie_sattler':
            #    class2_data.append(img)
            elif classes == 'ian_malcom':
                class3_data.append(img)
            #elif classes == 'john_hammond':
            #    class4_data.append(img)
            #elif classes == 'Unknown':
            #    class5_data.append(img)

class1_data = np.array(class1_data)
#class2_data = np.array(class2_data)
class3_data = np.array(class3_data)
#class4_data = np.array(class4_data)
```

```
100%|          | 1672/1672 [00:10<00:00, 159.48it/s]
100%|          | 2461/2461 [00:15<00:00, 163.42it/s]
100%|          | 3019/3019 [00:19<00:00, 151.21it/s]
100%|          | 1194/1194 [00:07<00:00, 162.80it/s]
100%|          | 1540/1540 [00:10<00:00, 152.07it/s]
```

```python
from keras.applications import VGG16

model = VGG16(include_top=False,input_shape = (224,224,3), weights='imagenet')
```

```python
import tensorflow_addons as tfa
tqdm_callback = tfa.callbacks.TQDMProgressBar()
vgg_class1 = model.predict(class1_data,verbose = 1)
#vgg_class2 = model.predict(class2_data)
vgg_class3 = model.predict(class3_data,verbose = 1)
#vgg_class4 = model.predict(class4_data)
#vgg_class5 = model.predict(class5_data)
```

```
49/49 [==============================] - 14s 118ms/step
95/95 [==============================] - 11s 119ms/step
```

```python
from keras.layers import Input, Dense, Dropout
from keras.models import Model

inputs = Input(shape=(7*7*512,))

dense1 = Dense(1024, activation = 'relu')(inputs)
drop1 = Dropout(0.5)(dense1)
dense2 = Dense(512, activation = 'relu')(drop1)
drop2 = Dropout(0.5)(dense2)
outputs = Dense(1, activation = 'sigmoid')(drop2)

model1 = Model(inputs, outputs)
model1.summary()
```

```
Model: "model"

Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 25088)]           0

dense (Dense)                (None, 1024)              25691136

dropout (Dropout)            (None, 1024)              0

dense_1 (Dense)              (None, 512)               524800

dropout_1 (Dropout)          (None, 512)               0

dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 26,216,449
Trainable params: 26,216,449
Non-trainable params: 0
```

```python
train_data = np.concatenate((vgg_class1[:1078], vgg_class3[:2113]), axis = 0)
train_data = train_data.reshape(train_data.shape[0],7*7*512)

valid_data = np.concatenate((vgg_class1[1078:], vgg_class3[2113:]), axis = 0)
valid_data = valid_data.reshape(valid_data.shape[0],7*7*512)
```

```python
train_label = np.array([0]*vgg_class1[:1078].shape[0] + [1]*vgg_class3[:2113].shape[0])
valid_label = np.array([0]*vgg_class1[1078:].shape[0] + [1]*vgg_class3[2113:].shape[0])
```

```python
import tensorflow as tf
from keras.callbacks import ModelCheckpoint

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
model1.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])

filepath="best_model.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

```python
model1.fit(train_data, train_label, epochs = 10, batch_size = 64, validation_data = (valid_data, valid_label), verbose = 1, callbacks = callbacks_list)
```

```
Epoch 1/10
50/50 [==============================] - 3s 39ms/step - loss: 0.6526 - accuracy: 0.6942 - val_loss: 0.3396 - val_accuracy: 0.8655
Epoch 2/10
50/50 [==============================] - 1s 10ms/step - loss: 0.1026 - accuracy: 0.9786 - val_loss: 0.3866 - val_accuracy: 0.8173
Epoch 3/10
50/50 [==============================] - 1s 10ms/step - loss: 0.0336 - accuracy: 0.9993 - val_loss: 0.4707 - val_accuracy: 0.8173
Epoch 4/10
50/50 [==============================] - 0s 10ms/step - loss: 0.0184 - accuracy: 0.9999 - val_loss: 0.5659 - val_accuracy: 0.8173
Epoch 5/10
50/50 [==============================] - 0s 9ms/step - loss: 0.0132 - accuracy: 0.9997 - val_loss: 0.6802 - val_accuracy: 0.8173
Epoch 6/10
50/50 [==============================] - 0s 10ms/step - loss: 0.0096 - accuracy: 1.0000 - val_loss: 0.7955 - val_accuracy: 0.8173
Epoch 7/10
50/50 [==============================] - 1s 12ms/step - loss: 0.0070 - accuracy: 1.0000 - val_loss: 0.8202 - val_accuracy: 0.8173
Epoch 8/10
50/50 [==============================] - 0s 10ms/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 0.8311 - val_accuracy: 0.8173
Epoch 9/10
50/50 [==============================] - 0s 10ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.9062 - val_accuracy: 0.8173
Epoch 10/10
50/50 [==============================] - 0s 10ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 0.8975 - val_accuracy: 0.8173
<tensorflow.python.keras.callbacks.History at 0x7f55146f4588>
```

```python
[ ]  import os
     import numpy as np

     alan_images = []
     ian_images = []

     test_path = '/content/drive/MyDrive/Colab Notebooks/Minor Project/img'

     for test in tqdm(os.listdir(test_path)):
         test_img = cv2.imread(os.path.join(test_path, test))
         test_img = cv2.resize(test_img, (224,224))
         test_img = test_img/255.
         test_img = np.expand_dims(test_img, 0)
         pred_img = model.predict(test_img)
         pred_feat = pred_img.reshape(1, 7*7*512)
         out_class = model1.predict(pred_feat)
         if out_class < 0.5:
             alan_images.append(out_class)
         else:
             ian_images.append(out_class)

     100%|██████████| 3859/3859 [1:03:51<00:00,  1.01it/s]

[ ]  len(alan_images)

     1706

 ⏵   len(ian_images)

     2153
```
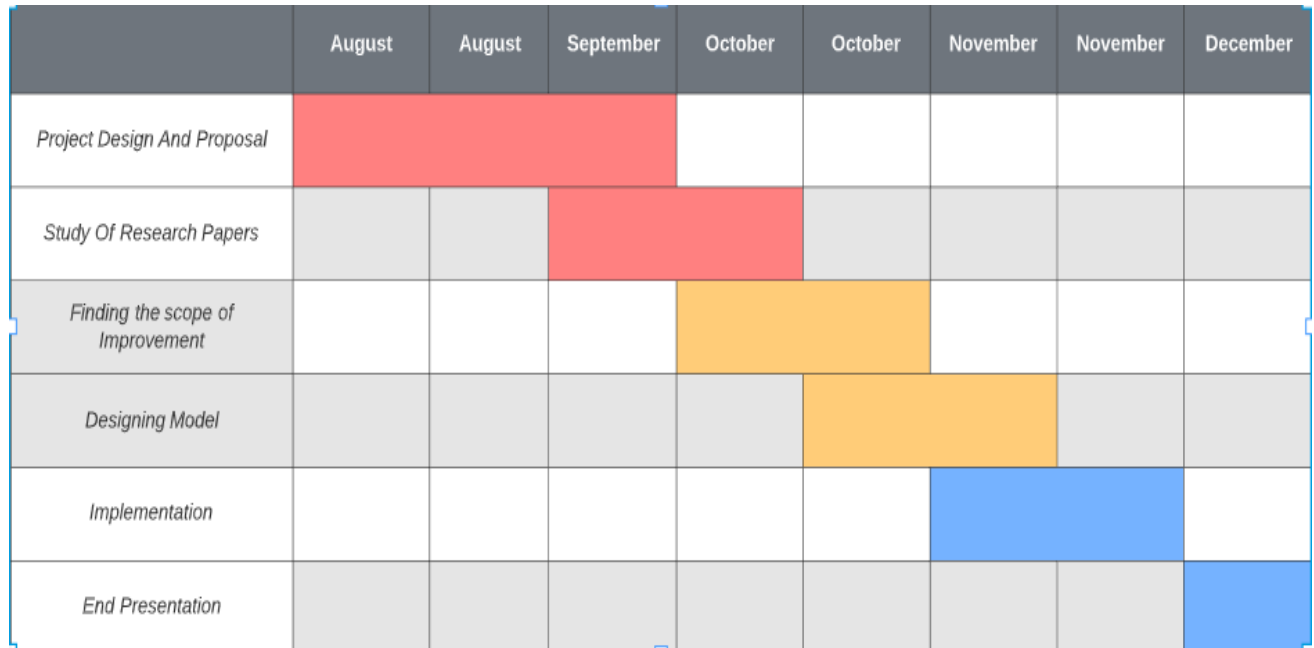
This test video clip is made up of 24 frames per second and number of images predicted for having "Alan Grant" in it are 1706. So the screen time for Alan Grant will be 1706/24 = 71 seconds

Number of images predicted for having "Ian Malcom" in it are 2153. So the screen time for Ian Malcom will be 2153/24 = 90 seconds

GANTT CHART

	August	August	September	October	October	November	November	December
Project Design And Proposal								
Study Of Research Papers								
Finding the scope of Improvement								
Designing Model								
Implementation								
End Presentation								

REFERENCE

1. https://stackoverflow.com/
2. https://stackexchange.com/
3. https://www.geeksforgeeks.org/

RL Ujjwal

to me ▾

2:38 PM (1 hour ago)

Minor project report approved.

•••

On Sun, Jan 10, 2021 at 2:07 PM Akshay Mishra <akshaymishra1515@gmail.com> wrote:

> 📁 **Project**
>
> Respected Sir,
> I have added more pages in report with proper format.
>
> On Sun, Jan 10, 2021 at 12:56 PM RL Ujjwal <ujjwal@ipu.ac.in> wrote:
>
>> Need more pages in proper format.
>>
>> On Sat, Jan 9, 2021 at 8:47 PM Akshay Mishra <akshaymishra1515@gmail.com> wrote:

RL Ujjwal

to me ▾

2:38 PM (1 hour ago)

Minor project report approved.