Document index

Abstract

Our document is structured as follows.
In the first section we have the benchmarking of available APIs, whether open-source or closed-source, free or paid, and the conclusions on which we've chosen.
In the second section we have all the information about prototyping the solution, where and how to get and use the program. Every part of the codes are explained in the 'implementation' section and in the last part of the document we've described how the output is formatted and how to read it.

Goal

The goal of this project's part is to implement a system that is able to detect the scene ('blackboard', 'slide', 'slide-and-talk', 'talk'), the main speaker and various objects present in a certain frame, along with texts (if there are any). With all this information, be able to divide into segments grouped by the scene.

Motivation

In the first part about benchmarking APIs, we've chosen the Tensorflow Object Detection API because it is an open-source API and has features that are useful for our project.
During the implementation of the project, we found that training the model with a pre-trained model (MobileNet) and a small dataset resulted in poor performance.
Then we tested with YoloV3, which we found other issues, like detecting multiple people in the same person. So we changed to the latest version, YoloV5 and it performed well, so we've included it into the project.
Then we used this pre-trained YoloV5 model to obtain a custom model that is able to detect blackboards. It worked sometimes and it didn't work the other times. One of the issues was that it detected a projected slide as a blackboard, which is a big problem for the project.
In the end we followed a paper (with some little tweaks) by Masneri and Schreer, where their main approach was shallow learning and worked best.
The paper used the SVM algorithm to train the model for the classification problem. We've tried it with Optuna (hyperparameter optimization framework) and it performed around 60-70% of accuracy.
So we chose to use the XGBoost algorithm along with Optuna. First time with 150 trials, we obtained 83,9% of accuracy with a lot of uncertainty in the classification and the second time with 500 trials, we obtained 81,6% of accuracy but with less uncertainty.
To identify the speaker we took all the faces, compared to each other by checking if it's the same person, then the face which appeared the most, we assigned it as the speaker.
Detecting texts works best with slide's characters and doesn't work well with hand-written characters.

Benchmark API

Video intelligence API (Google)

https://cloud.google.com/video-intelligence

Features

- Livestream mode
- Annotate videos saved locally or on cloud
- Explicit content detection
- Face detection
- Label detection
- Logo recognition
- Object tracking
- Person detection
- Shot change detection
- **Speech transcription**
- Text detection
- Costo: Free per 16,5 ore di video al mese se non vado oltre il limite

Prices

Prices and functions per offline videos

Funzionalità	Primi 1000 minuti	Oltre i 1000 minuti
Rilevamento etichette	Nessun costo	$ 0,10 al minuto
Rilevamento inquadrature	Nessun costo	$ 0,05 al minuto o nessun costo con rilevamento etichette
Rilevamento di contenuti espliciti	Nessun costo	$ 0,10 al minuto
Trascrizione del parlato	Nessun costo	$ 0,048 al minuto (addebiti solo per trascrizioni en-US)
Monitoraggio oggetti	Nessun costo	$ 0,15 al minuto
Rilevamento testo	Nessun costo	$ 0,15 al minuto
Rilevamento loghi	Nessun costo	$ 0,15 al minuto
Rilevamento facciale	Nessun costo	$ 0,10 al minuto
Rilevamento persone	Nessun costo	$ 0,10 al minuto
Riconoscimento di volti celebri	Nessun costo	$ 0,10 al minuto

Prices and functions per live videos

Funzionalità	Primi 1000 minuti	Oltre i 1000 minuti
Rilevamento etichette	Nessun costo	$ 0,12 al minuto
Rilevamento inquadrature	Nessun costo	$ 0,07 al minuto
Rilevamento di contenuti espliciti	Nessun costo	$ 0,12 al minuto
Monitoraggio oggetti	Nessun costo	$ 0,17 al minuto

Rekognition (Amazon)

https://aws.amazon.com/it/rekognition

Features

- Scene objects and activities detection
- Explicit content detection
- Text detection
- Celebrity face detection
- Face analysis and detection
- Face recognition
- People movement detection and livestream video analysis.

CONs: no speech-recognition (?)

Prices

12 months free with 1000 minutes per month (16,5 hours)

Tabella dei prezzi

Analisi di video archiviate

Regione: Europa (Francoforte)

Caratteristica	Prezzi
Rilevamento delle etichette	0,12 USD/min
Moderazione dei contenuti	0,12 USD/min
Rilevamento del testo	0,12 USD/min
Rilevamento facciale	0,12 USD/min
Riconoscimento di volti celebri	0,12 USD/min
Ricerca facciale	0,12 USD/min
Rilevamento dei movimenti delle persone	0,12 USD/min

Analisi multimediale

Caratteristica	Prezzi
Rilevamento dei fotogrammi	0,06 USD/min
Rilevamento di fotogrammi neri, titoli di coda, barre colore ("segnali d'azione tecnici")	0,06 USD/min

Analisi di video in live stream

Caratteristica	Prezzi
Ricerca facciale	0,144 USD/min

I costi di utilizzo applicati sono mensili e ripartiti proporzionalmente nei minuti di utilizzo parziale.

Storage di metadati facciali	0.00001 USD/metadati facciali al mese**
**I costi di storage applicati sono mensili e ripartiti proporzionalmente nei mesi di utilizzo parziale	

TensorFlow Object detection API

https://github.com/tensorflow/models/tree/master/research/object_detection
https://www.youtube.com/watch?v=yqkISICHH-U

Features

Natural language processing

- Text classification
- Real time semantic search
- Multilingual question answering

Images

- Image classification
- Object recognition
- Human faces generation

Audio

- Pitch recognition
- Sound classification

Video

- Action recognition
- Video interpolation
- Text-to-video retrieval

CONs: ma solo Object detection senza Speech recognition / transcript.

Prices

Free (open source)

Ximilar

https://www.ximilar.com/

Features

- Image recognition
- Photo similarity
- Product similarity

Prices

Free

€0 $0

+ €0.9 or $0.95 per 1k credits

3,000 API credits free*

Sequential processing

Limited email support

Available services:
— Image Recognition
— Generic Tagging
— Photos Similarity
— Product Similarity
Contact us to test other services.

Image Recognition: 3 tasks

Recognition Flows: 1 flow

Similarity: 1 collection, 3,000 images

TRY IT NOW

Quick setup
No credit card required

Business

€55 $59 or €149 $175

monthly

100k or 300k API credits*

Fast parallel processing

Standard support
+ one consultation with ML expert

Free services plus:
— Advanced Annotation System
— Object Detection
— Fashion Tagging
Contact us to test other services.

Image Recognition: 10 tasks

Recognition Flows: 3 flows

Object Detection: 3 diferent objects

Similarity: 1 collection, 100,000 images

UPGRADE IN APP

Easy & quick setup
Credit card or PayPal

Professional

€450 $499

monthly

1M API credits*

Fast parallel processing

Priority support
+ regular consultations with ML expert

Business services plus:
— Team Collaboration
— Visual Fashion Search

Image Recognition: 20 tasks, unlim. flows
— ML loop: add processed images to training data

Object Detection: 20 different objects

Similarity: 3 collections, unlim. images
— nightly sync of your DB
— custom similarity model

Fashion Tagging: map our taxonomy to yours

UPGRADE IN APP

Easy & quick setup

Valossa

https://valossa.com/

Features

- Video recognition
- Face recognition
- Emotion & sentiment detection
- Object detection

- **Audio sound tags**
- Explicit content detection
- Speech analysis
- Video categories and topics

Prices

Price to be agreed with the staff.

Watson Visual Recognition (IBM)

https://cloud.ibm.com/docs/visual-recognition?topic=visual-recognition-index

It will shut down on 31st December 2021, from 7th January 2021 it is no longer available.

Image AI

http://imageai.org/

Features

- Object detection
- Video Tracking
- Video analysis and tags
- Livestream mode
- Detection speed

Prices

Free (open source)

TorchVision

https://pytorch.org/vision/0.9/io.html#video

Features

- Image classification
- Object detection
- Instance segmentation
- Keypoint detection (simplified human skeleton)
- **Video classification**

- Semantic segmentation

Prices

Free (open source)

Sensifai

https://sensifai.com/#services
https://demo.sensifai.com/

Features

- Video recognition
- Video segmentation
- Explicit content detection
- Celebrity detection
- Object, scene, attribute recognition
- Action, sport recognition
- Logo recognition
- Landmark recognition
- Audio, sound, speech recognition

Prices

Prices to be agreed with the staff.

Conclusion

The ideal choices would be Valossa or Sensifai, but since they need to be agreed on the price, a good trade-off could be Google's Video Intelligence API.
If our goal is just video recognition, then we could use some open-source frameworks. But if we also need speech and audio recognition, then Google's Video Intelligence API is the safe bet.

SOLUTION PROTOTYPING

Our problem was to detect “BLACKBOARD” or in general to detect the scene whether it's BLACKBOARD, SLIDE, SLIDE-AND-TALK or TALK. We’ve tried to develop a machine learning model by prototyping the following three methods.

Project’s repository

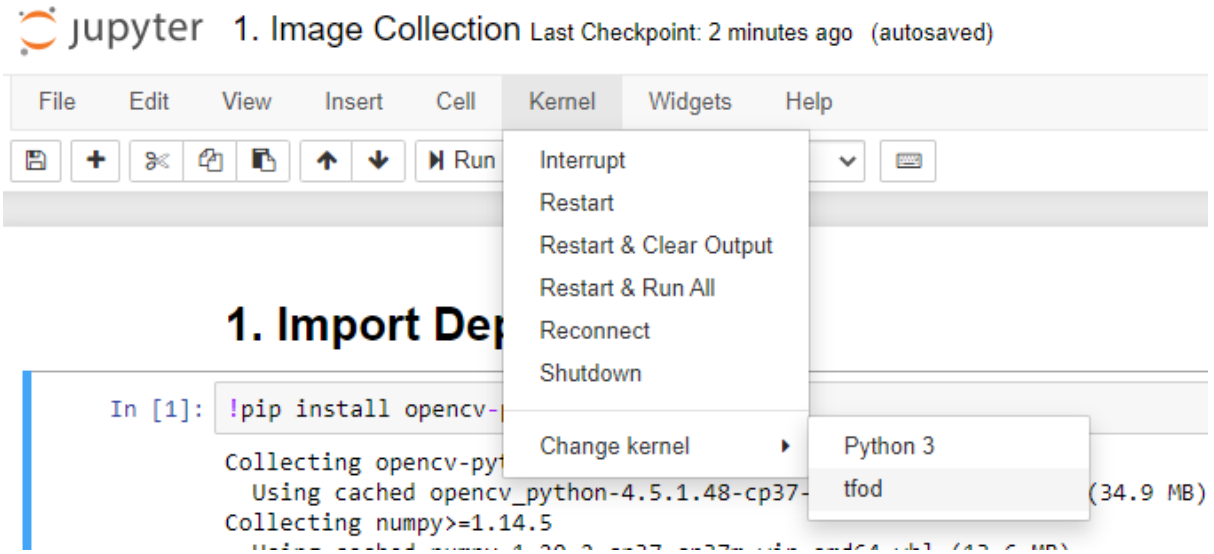
https://github.com/Gabbosaur/SWT-AggregateLD

TENSORFLOW: OBJECT DETECTION API

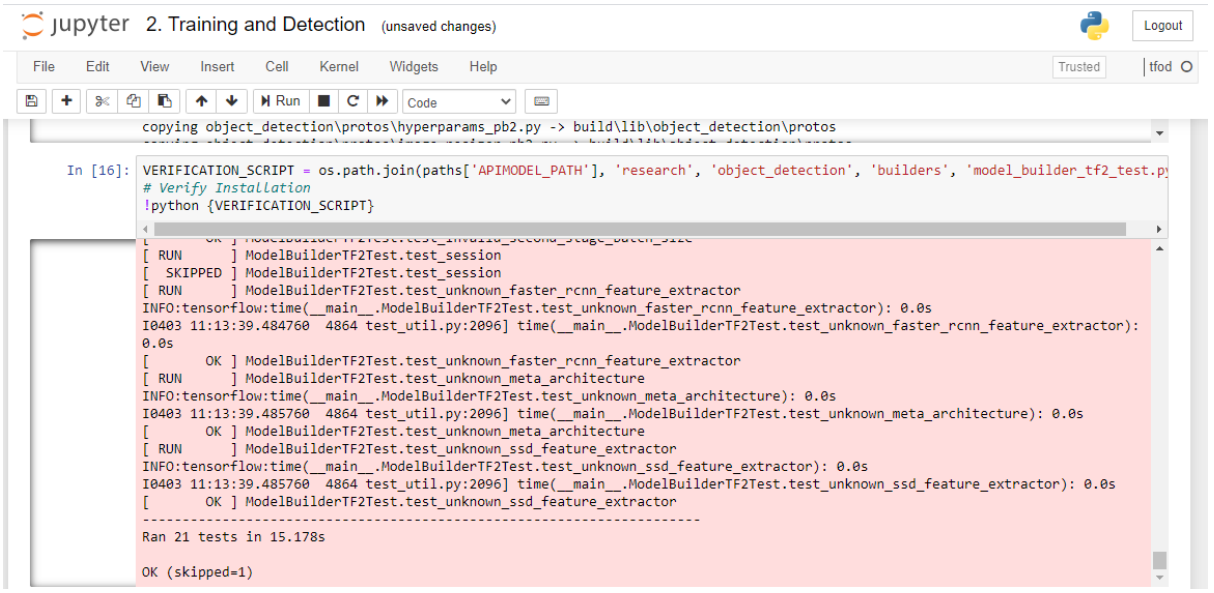
We chose this framework because the stakeholder preferred open-source APIs and it contains the features we needed.
The following steps are aimed at obtaining **custom detections**. For instance, gesture recognition.

WORKFLOW STEPS

1. Clone this repository: https://github.com/nicknochnack/TFODCourse
2. Create a new virtual environment using `python -m venv tfod`
3. Activate your virtual environment
    - `source tfod/bin/activate # Linux`
    - `.\tfod\Scripts\activate # Windows`
4. Install dependencies and add virtual environment to the Python Kernel
    - `python -m pip install --upgrade pip`
    - `pip install ipykernel`
    - `python -m ipykernel install --user --name=tfodj`
5. Collect images using the Notebook 1. Image Collection.ipynb - ensure you change the kernel to the virtual environment as shown below

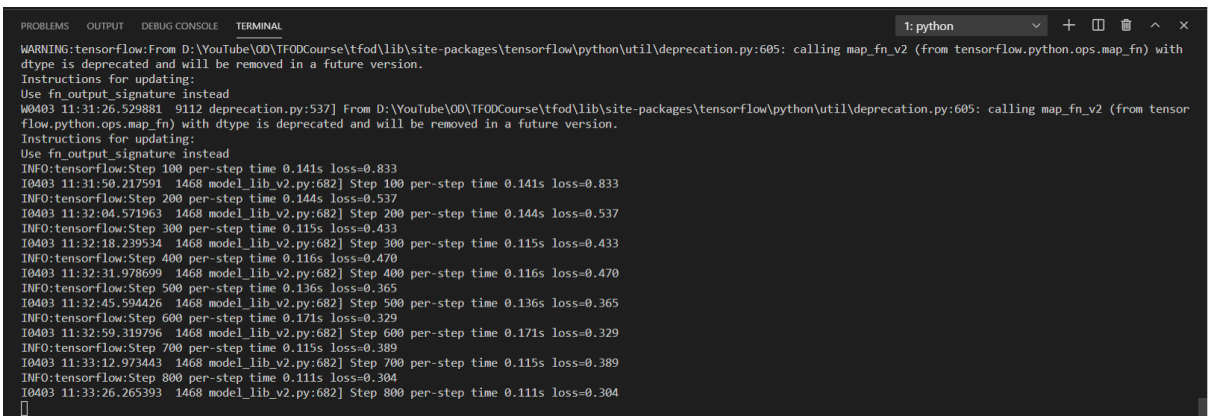


6. Manually divide collected images into two folders: train and test. So now all folders and annotations should be split between the following two folders.
    - \TFODCourse\Tensorflow\workspace\images\train
    - \TFODCourse\Tensorflow\workspace\images\test
7. Begin the training process by opening 2. Training and Detection.ipynb, this notebook will walk you through installing Tensorflow Object Detection, making detections, saving and exporting your model.
8. During this process the Notebook will install Tensorflow Object Detection. You should ideally receive a notification indicating that the API has installed successfully at Step 8 with the last line stating OK.



If not, resolve installation errors by referring to the Error Guide.md in this folder.

9. Once you get to step 6. Train the model, inside of the notebook, you may choose to train the model from within the notebook. I have noticed however that training inside of a separate terminal on a Windows machine you're able to display live loss metrics.



(screenshot during training phase)

10. You can optionally evaluate your model inside of Tensorboard. Once the model has been trained and you have run the evaluation command under Step 7. Navigate to the evaluation folder for your trained model e.g.
    - `cd Tensorlfow/workspace/models/my_ssd_mobnet/eval`

    and open Tensorboard with the following command

    ```
    tensorboard --logdir=.
    ```

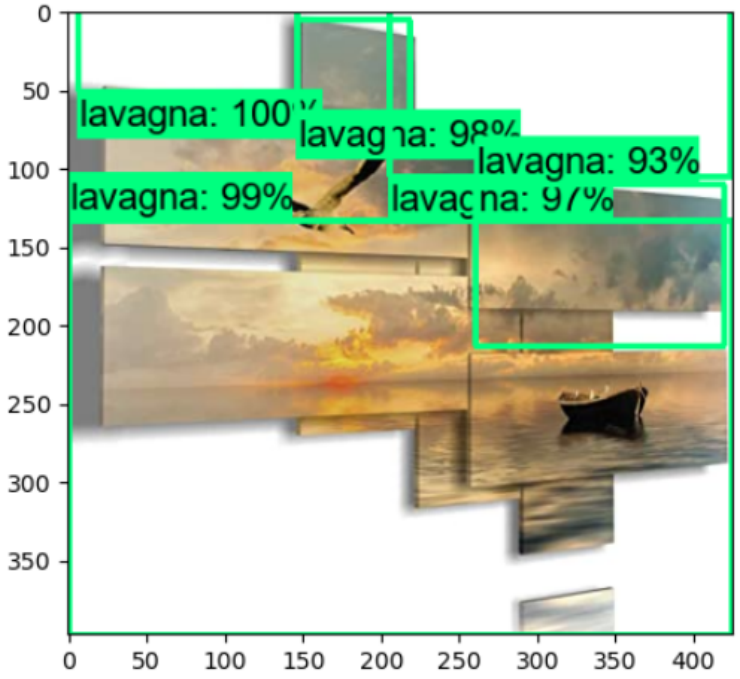
    Tensorboard will be accessible through your browser and you will be able to see metrics including mAP - mean Average Precision, and Recall.

TESTING PROBLEMS

We tested with a few images taken with a webcam and using the model SSD MobileNet V2 FPNLite 320x320, the performance was okay.
After that, we tried to use other images taken from the web (with different backgrounds and hand positions), but the performance got worse.

PERFORMANCE

Testing performance of TF model custom trained MobileNet, which as you can see performed poorly.



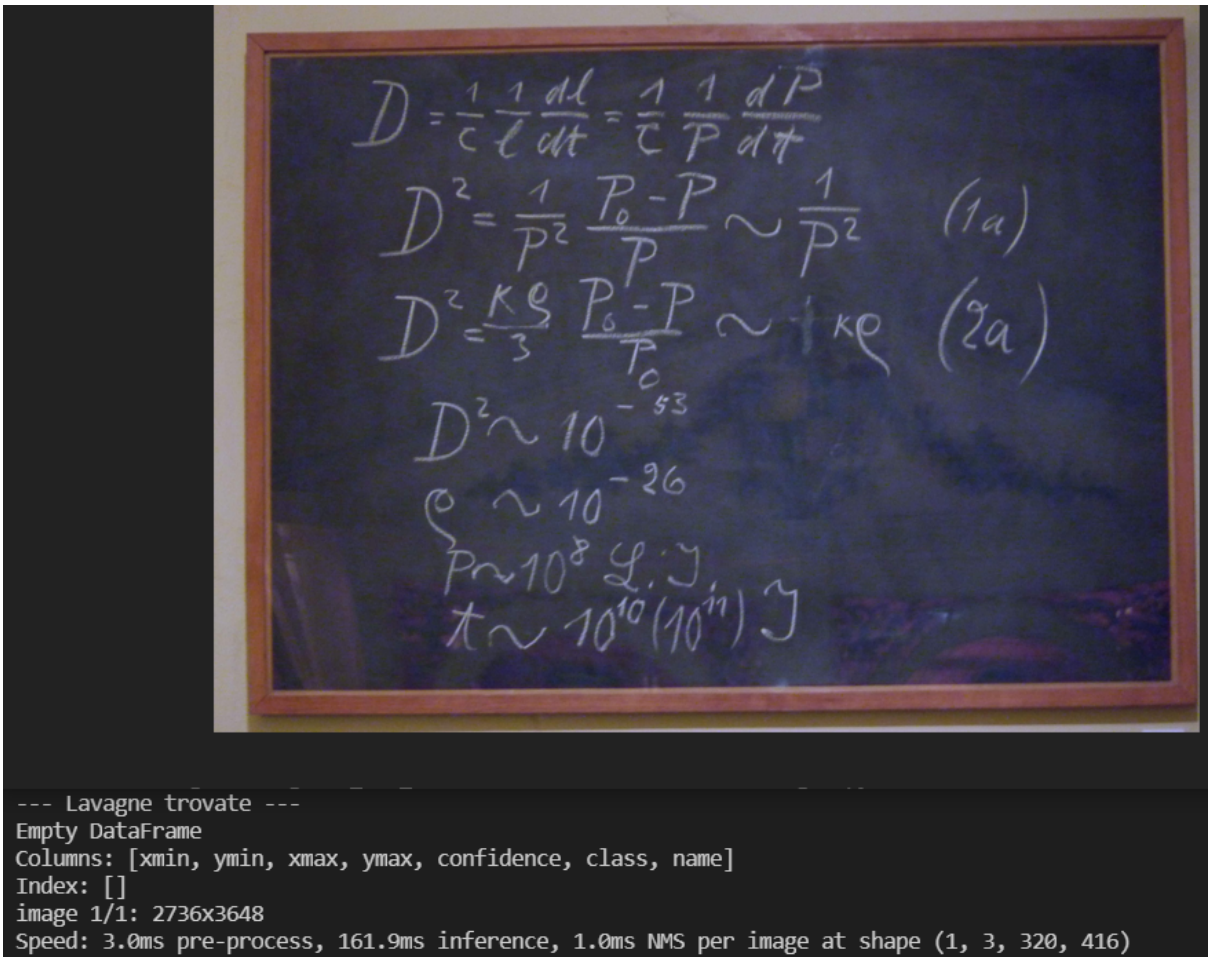
YOLOv5

https://github.com/ultralytics/yolov5
Pre-trained model used to detect the most common objects in the real world.

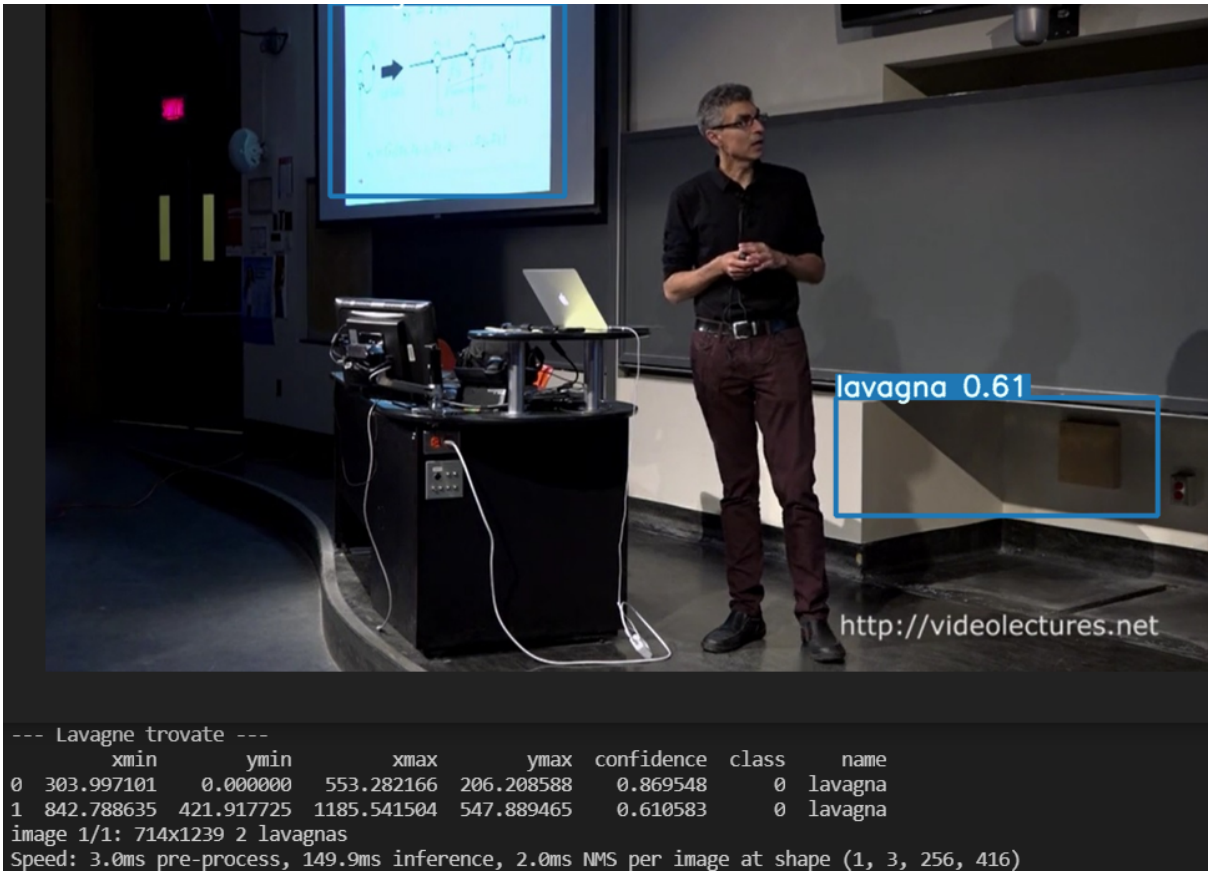
PERFORMANCE

Sometimes it works

Sometimes it doesn’t.




```
--- Lavagne trovate ---
Empty DataFrame
Columns: [xmin, ymin, xmax, ymax, confidence, class, name]
Index: []
image 1/1: 2736x3648
Speed: 3.0ms pre-process, 161.9ms inference, 1.0ms NMS per image at shape (1, 3, 320, 416)
```




```
--- Lavagne trovate ---
         xmin        ymin         xmax        ymax  confidence  class     name
0  303.997101    0.000000   553.282166  206.208588    0.869548      0  lavagna
1  842.788635  421.917725  1185.541504  547.889465    0.610583      0  lavagna
image 1/1: 714x1239 2 lavagnas
Speed: 3.0ms pre-process, 149.9ms inference, 2.0ms NMS per image at shape (1, 3, 256, 416)
```

Sometimes the problem is that the value of confidence detected can be ambigue, because sometimes when we have blackboards, it has low confidence and when we don't have blackboards, it has high confidence (for example the previous screenshot has detected the projected slide as blackboard with high confidence 86,9%).

SHALLOW LEARNING

In the end, we opted for this approach since we didn't have a big dataset to train our model. We've followed this paper and adapted to our problem: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=AvJA648AAAAJ&alert_preview_top_rm=2&citation_for_view=AvJA648AAAAJ:nb7KW1ujOQ8C

Data collection

We took screenshots from various videos from the web and split them into 4 categories.

- Blackboard
- Slide
- Slide-and-talk
- Talk

Feature engineering

We analyzed the data we gathered and extracted 19 normalized features with respect to the image resolution.

- Color histogram divided into 16 groups of colors
- People's face presence (0 = no people's face, 1 = one face, 2 = two or more faces)
- Position (xmedio) of the person's face
- Faces' size (area of the bounding box)

ML Algorithm

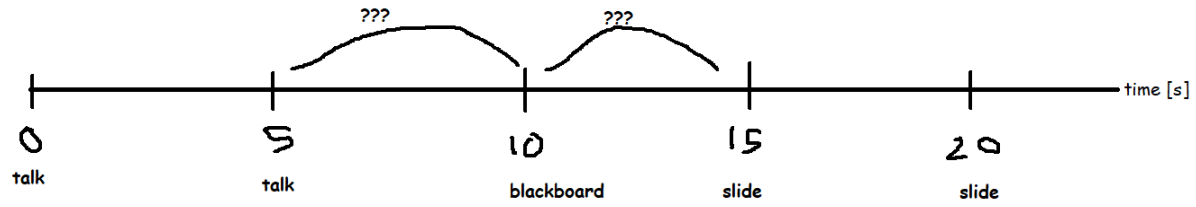
The paper used the SVM algorithm to train the model for the classification problem. We've tried it with Optuna (hyperparameter optimization framework) and it performed around 60-70% of accuracy.

So we've changed our approach and used the XGBoost algorithm along with Optuna:

- First time with 150 trials, we obtained 83,9% of accuracy with a lot of uncertainty in the classification
- Second time with 500 trials, we obtained 81,6% of accuracy but with less uncertainty

Split in segments

The stakeholder wanted to split the input video into segments of the same detected scenes. The split has a starting time and an ending time and between the segments we have an instance of time (secondsForFrameSkip variable) where it isn't classified.



In this example we have the first two frames classified as “TALK” but when we detected the third frame as “BLACKBOARD” we don’t know how the frames between 5 and 10 are classified. We can suppose many things like taking the average (from time 5 to 7,5 classified as TALK and time 7,5 to 10 as BLACKBOARD). It depends on how precise we want to achieve.

CONCLUSION

Since we **don’t have much data**, the **optimal solution** is the **shallow learning** approach.

TEXT RECOGNITION

TESSERACT OCR

Since TensorFlow Object Detection API doesn't contain text detection, we opted to use Tesseract OCR as an extension module to detect texts in images.

WORKFLOW STEPS

1. Setup
    a. pip install pytesseract
    b. download tesseract.exe https://github.com/UB-Mannheim/tesseract/wiki
    c. install it
2. In VScode

```
import cv2
import pytesseract

pytesseract.pytesseract.tesseract_cmd = "C:\\Program
Files\\Tesseract-OCR\\tesseract.exe"
img = cv2.imread('image.jpg')
# img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
print(pytesseract.image_to_string(img))
cv2.imshow("Result", img)
cv2.waitKey(0)
```

This code prints out the texts present in the given image. It works well with good contrasts and standard texts.

IMPLEMENTATION

The code is splitted into many Python files.

- main.py is the main script which connects all the modules used
- calculateFeatureAndTrain_module.py
- svm_module.py
- xgboost_module.py
- testTFmodel.py
- testYOLOv5-lavagne.py

main.py

```
4   # Controllo degli argomenti passati in linea di comando
5   if (len(sys.argv)<3):
6       print("ERRORE: inserisci il percorso del video da elaborare e ogni quanti secondi prendere un frame.")
7       sys.exit()
8
9   try:
10      secondsForFrameSkip=int(sys.argv[2])
11  except:
12      print("ERRORE: inserire un numero intero come secondo parametro.")
13      sys.exit()
14
15  PATH_VIDEO = sys.argv[1]
16  if not os.path.exists(PATH_VIDEO):
17      print("ERRORE: video inesistente.")
18      sys.exit()
```

The first lines of code are used to check the arguments if they are valid or not.

```
33  matplotlib.use("TkAgg", force=True)
34  #%matplotlib inline
35  #plt.show()
36
37
38  CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
39  PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
40  PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_co
41  TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
42  LABEL_MAP_NAME = 'label_map.pbtxt'
43
44  paths = {···
58
59  files = {···
67  }
68
69  ########################################YOLO V3, non utilizzata perché rileva oggetti
70  def testConYoloV3():···
```

These lines of code are no longer used; they were used to test Tensorflow's approach which gave poor performance.

The function "testConYoloV3()"

```python
#######################################YOLO V3, non utilizzata perché rileva oggetti
def testConYoloV3():
    # Load Yolo
    print("LOADING YOLO")
    net = cv2.dnn.readNet(files['YOLOV3_SPP_WEIGHTS'], files['YOLOV3_CFG'])
    #save all the names in file o the list classes
    classes = []
    with open(files['COCO_NAMES'], "r") as f:      You, 2 months ago • trovato facce e salva le facce in cartella per
        classes = [line.strip() for line in f.readlines()]

    print(classes)

    #get layers of the network
    layer_names = net.getLayerNames()
    #Determine the output layer names from the YOLO model
    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
    print("YOLO LOADED")


    # Capture frame-by-frame
    IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'thumbsdown.b1f20c56-b4d4-11eb-ae88-240a64b78789.jpg')
    #IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'thumbsup.fd7cdb14-b4d4-11eb-b662-240a64b78789.jpg')
    img = cv2.imread(IMAGE_PATH)
    #img=cv2.imread("test_img.jpg")
    img = cv2.resize(img, None, fx=0.4, fy=0.4)
    height, width, channels = img.shape

    # USing blob function of opencv to preprocess image
    #blob = cv2.dnn.blobFromImage(img, 1 / 255.0, (416, 416),swapRB=True, crop=False)

    blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(320, 320), mean=(0, 0, 0), swapRB=True, crop=False)

    #for b in blob:
    #   for n,img_blob in enumerate(b):
    #       cv2.imshow(str(n),img_blob)

    #Detecting objects
    net.setInput(blob)
```

loads YoloV3's model and COCO's classes name, then we load a test image and apply some image filtering, this result is given to the model which we obtain the detection results to draw the bounding box.

The problem with YoloV3 is that if we have a person in the frame, the model detects the person multiple times. So we tried with the latest version YoloV5.

```python
import yolov5
#import torch
# YOLOV5_model=os.path.join('tfod', 'Lib','site-packages', 'yolov5' ,'models', 'yolov5s.yaml'),
# print(os.getcwd()) # returns the currently working directory of a process
# model = yolov5.load(YOLOV5_model) # carico modello tramite formato yaml, però non funziona
# model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True) # scarico il modello
model = yolov5.load('yolov5s.pt') # se ho il modello in locale

def rilevaPersona(model, frame):
    ########################################YOLO V5

    isPerson=False
    ## Immagine per test rilevazione persona
    # IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'livelong.fed895dc-b264-11eb-bccc-086266b476b9.jpg')

    # img = cv2.imread(frame)


    # inference
    #results = model(img)

    results = model(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB), size=400)

    labels, cord_thres = results.xyxyn[0][:, -1].numpy(), results.xyxyn[0][:, :-1].numpy() # estraggo labels e coordinate dei
    classes = model.names
    label_class = []

    # show results
    for i in labels:
        # print(classes[int(i)])
        label_class.append(classes[int(i)])
        if(classes[int(i)]=="person"):
            isPerson=True

    # show results
```

This function “rilevaPersona(model, frame)” is used in the project to detect people and takes as parameters the model which is “yolov5s.pt” and the image to be analyzed.
We’ve done the same thing as done for YoloV3 and if we detect a person we set our variable isPerson as True. This function returns isPerson variable and all the objects detected by the model.

class Record

```python
@dataclass
class Record: # 1 frame
    num_frame: int
    time: float
    list_words: list
    isPersonDetected: bool
    isSpeaker: bool
    # path_faces: list      # ['faccia1', 'faccia2', 'faccia3']
    idFaces: list           # [0,1,2] in questo caso ci sono 3 persone distinte e alla fine contare le occorrenze e scegliere il massimo
    scene: any              # blackboard || slide || slide-and-talk || talk
    objects_detected: any   # lista di oggetti rilevati tramite YoloV5

                            # punti mediapipe upperbody position
    def to_dict(self):
            return {"num_frame": self.num_frame, "time": self.time, "list_words": self.list_words, "isPersonDetected": self.isPersonDete
```

This class contains the information for each frame.

- num_frame: is the number of frame of the video that is going to be analyzed
- time: a float number that represents the time position of the frame in the video
- list_words: list of the words detected by the OCR
- isPersonDetected: boolean value that represents if there is a person in that frame
- isSpeaker: boolean value that represents if the main speaker is present in that frame
- idFaces: list of faces based on unique id
- scene: string value that represents the actual scene in the frame (‘blackboard’, ‘slide’, ‘slide-and-talk’, ‘talk’)
- objects_detected: list of objects detected by YoloV5

- function to_dict(self): used to format the data in json format

class FrameWithFaces

```
282  @dataclass
283  class FrameWithFaces:
284      num_frame: int
285      num_faces: int
286      isProcessed: list
287      time: float
```

This class contains the information of a certain frame that contains faces. It is used to assign IDs to detected faces.

- num_frame: is the number of frame of the video that is going to be analyzed
- num_faces: is the number of faces in that frame
- isProcessed: list of boolean value that represents faces that are already processed, for example if the first face of the frame is already assigned to a specific ID, the first value of isProcessed list will be True
- time: a float number that represents the time position of the frame in the video

class Segment

```
289  @dataclass
290  class Segment:
291      id_label: any
292      start_time: float
293      end_time: float
294      list_of_same_scene: list    # lista di Records di quel tipo di scena
295
296      def to_dict(self):
297          return {"id_label": self.id_label, "start_time": self.start_time, "end_time": self.end_time, "list_of_same_scene":[ obj.to_dict
```

This class contains the information of specific segments of the video based on the scene detected.

- id_label: name of the scene (‘blackboard’, ‘slide’, ‘slide-and-talk’, ‘talk’)
- start_time: when the segment starts
- end_time: when the segment finishes
- list_of_same_scene: list of Record class that has the same ‘scene’ consecutively
- function to_dict(self): used to format the data in json format

Video processing

```
324  # Carico il modello migliore per scene prediction
325  scene_model = pickle.load(open("xgboost500.sav", 'rb'))
326
327  while video.isOpened():
328      ret, frame = video.read()
329      if not ret:
330          break
331      totalFrame += 1
332      if i > frame_skip - 1: # In questo caso ogni 5 secondi
333          duration = frame_counter*temp
334          minutes = int(duration/60)
335          seconds = int(duration%60)
336          print(" ------------ Frame at " + str(minutes) + " min and " + str(seconds) + " seconds ----------- ")
337          n_frame_analyzed+=1
338          #cv2.imwrite('test_'+str(i)+'.jpg', frame)
339          #print(pytesseract.image_to_string(frame))
340          temp_list_words=[]
341          boxes=pytesseract.image_to_data(frame)
342          frame_counter+=frame_skip
343          #print("frame number: " + str(frame_counter))
344
345          # Riconoscimento persona con YoloV5
346          isPersonDetected, oggetti_rilevati = rilevaPersona(model, frame)
347          print("OGGETTI RILEVATI: ", oggetti_rilevati)
348          if isPersonDetected == True:
349              '''
350              import cv2
351              import mediapipe as mp
352              import numpy as np
353              mp_drawing = mp.solutions.drawing_utils
354              mp_drawing_styles = mp.solutions.drawing_styles
355              mp_pose = mp.solutions.pose
```

First of all we load the model we've previously trained to classify the scene.
Then we read the video given as input and analyze each frame. We use the pytesseract OCR to detect and extract the words present in the actual frame and store them in our data structure.
We use the function "rilevaPersona(model,frame)" as previously described and we check if there is any person; if yes, we launch mediapipe to extract their faces and save into a temporary image file, else we skip the extraction.

The commented code at line 349 can be used to extract the body landmarks points for further applications.

```
453          # Lista delle parole con OCR
454          for x,b in enumerate(boxes.splitlines()):
455              if x!=0:
456                  b=b.split()
457                  if len(b)==12: # gli indici prima del 12 sono altre informazioni del testo (colore, posizione, etc..)
458                      #print(b[11])
459                      #lista delle parole
460                      temp_list_words.append(b[11])
```

OCR: detects words in the frame and stores them in our data structure.

```
462          # Inizializzo la lista delle facce per le facce univoche trovate con isProcessed
463          list_idFaces = []
464          for i in range(lunghezza_isProcessed):
465              list_idFaces.append(-1)
466
467          lunghezza_isProcessed = 1 # se non ci sono facce, la lista idFaces è -1 (anziché vuota)
```

All detected faces are initialized as value -1.

Then we have the scene prediction.

```
474         # SCENE PREDICTION
475         image_feature = calculateFeatureAndTrain_module.singleImageFeatureExtraction(img=frame)
476         image_feature = np.array([image_feature])
477         # Predict the response for test dataset
478         y_pred = scene_model.predict(image_feature)
479         y_pred_proba = scene_model.predict_proba(image_feature)
480 
481         counter_scenes[int(y_pred)] += 1
482 
483         print("Probabilità del tipo di scena:\t", y_pred_proba)
484 
485         sorted_y_pred_proba = y_pred_proba
486         sorted_y_pred_proba.sort()
487         differenza_prob = sorted_y_pred_proba[0][-1]-sorted_y_pred_proba[0][-2]
488         print("Differenza tra le due probabilità più grandi:\t{:.5f} ({:.2f}%)".format(differenza_prob, differenza_prob*100))
489         # print("Varianza della probabilità:\t", np.var(y_pred_proba))
490         print("Scene:\t", scenes[int(y_pred)])
```

By using the previously loaded model, we predict the actual scene by analyzing and passing the features of the actual frame.

We store all the information gathered to our value “rec” of type dataclass Record, then we check if we already have the same frame based on words detected and if there is a person.

```
494         #struct con numero del frame e la lista di parole
495         rec=Record(frame_counter,frame_counter*temp,temp_list_words, isPersonDetected, False, list_idFaces, scenes[int(y_pred)], oggett
496         #controlliamo se questo identico record è già contenuto nella lista
497         #potrebbe essere dispendioso, facciamo solo il compare con l'ultimo frame in listOfRecords?
498         flag=False
499 
500         # conta se ci sono frames con la lista delle parole uguali
501         for y in listOfRecords:
502             if(y.list_words == rec.list_words):
503                 flag=True
504                 count_frame_doppi += 1
505                 break
506 
507         # listOfRecords.append(rec) # Inserisco ogni Record anche se doppione
508 
509 
510         if rec.isPersonDetected == True: #se c'è una persona
511             listOfRecords.append(rec)
512         else: # se NON c'è una persona
513             if flag == False: # ma le parole sono diverse
514                 listOfRecords.append(rec)
515 
516 
517         # # SCOMMENTARE PER AVERE I RECORD UNIVOCI IN BASE ALLE PAROLE
518         # if(flag==False):
519         #   listOfRecords.append(rec) # avremo listOfRecords UNIVOCI
```

Comparing faces

```python
# CICLO CHE VA A COMPARARE TUTTE LE FACCE
# Compara le due foto e definisce se le due persone trovate sono la stessa persona
# result = DeepFace.verify(img1_path = "images/lec3.jpg", img2_path = IMAGE_FACES_PATH+"face_of_lec2_face0.jpg")
# print(result)
# print(frame_with_faces)

from os import listdir
from os.path import isfile, join

mypath = 'images/faces/'
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
# print(onlyfiles)


FILENAME = os.path.splitext(NOME_VIDEO)[0]
idFace = -1
totalFace=[]

# Loop che compara le facce trovate e conta il numero di occorrenze della relativa faccia
for i in range(len(frame_with_faces)):
    frame = frame_with_faces[i]
    for n_face in range(frame.num_faces):
        # La prima immagine di faccia
        path_img = 'images/faces/face_of_' + FILENAME + '_frame_' + str(frame.num_frame) + '_face_' + str(n_face) + '.jpg'

        if frame_with_faces[i].isProcessed[n_face] == False: # assegnamo una faccia se non è stata processata

            try:
                obj = DeepFace.verify(img1_path=path_img, img2_path=path_img) # se trova una faccia continua, se no exception
                idFace+=1
                totalFace.append(1)
                frame_with_faces[i].isProcessed[n_face] = True
                for k in range(i,len(listOfRecords)):
                    if(listOfRecords[k].time == frame_with_faces[i].time):
                        listOfRecords[k].idFaces[n_face] = idFace
                        break


                for j in range(i+1, len(frame_with_faces)):
```

Once we've analyzed all the frames, we start to compare the faces to detect who is the host/speaker.
Our assumption is that the person who appears the most in the video is the host/speaker.
At this point we will have a folder full of faces to be compared to each other. We use "DeepFace" to check if a face belongs to a certain person.
We start by picking the first image face and compare it to all the rest of the image faces. When we find that the comparison fits together, we assign the same ID to the face we are comparing. We keep track of the faces that are already processed to optimize the algorithm. After all the comparisons, all the faces have an ID assigned.

```python
idSpeaker=totalFace.index(max(totalFace))
maxSpeakerApparence=max(totalFace) # numero delle apparizioni dello speaker

counter=0
# Setta isSpeaker a True per ogni record che contiene gli ID della faccia che è comparsa più volte
for i in range(len(listOfRecords)):
    for j in range(len(listOfRecords[i].idFaces)):
        if listOfRecords[i].idFaces[j]==idSpeaker:
            listOfRecords[i].isSpeaker=True
            counter+=1
            break
    if counter==maxSpeakerApparence:
        break
```

We extract the ID of the speaker by taking the index of the highest value in the "totalFace" array. For example if totalFace is [7,10,3] it means that we have 3 different faces, the first one appeared 7 times, the second 10 times and third 3 times. Then we take the highest value which is 10 in position 1 of the array. So the face with ID = 1 is the speaker's face.

Now we set in our list of Records, all the “isSpeaker” to True if the record has the ID face = 1.

Video segment

```
# Segment
listOfSameSegment=[]
startSegment=0.0
endSegment=0.0
for i in range(0,len(listOfRecords)):
    if i==0:
        #primo elemento
        actualSegmentScene=listOfRecords[i].scene
        listOfSameSegment.append(listOfRecords[i])
        endSegment=listOfRecords[i].time
    else:
        if actualSegmentScene==listOfRecords[i].scene:
            listOfSameSegment.append(listOfRecords[i])
            endSegment=listOfRecords[i].time
        else:
            endSegment=listOfRecords[i].time-secondsForFrameSkip
            listOfSegments.append(Segment(actualSegmentScene,startSegment,endSegment,listOfSameSegment[:]))
            actualSegmentScene=listOfRecords[i].scene
            startSegment=endSegment+secondsForFrameSkip
            endSegment=listOfRecords[i].time
            listOfSameSegment.clear()
            listOfSameSegment.append(listOfRecords[i])
listOfSegments.append(Segment(actualSegmentScene,startSegment,totalFrame*temp,listOfSameSegment[:]))


print(" - - - - - VIDEO SUMMARY - - - - - ")
print("Le facce trovate sono: ", totalFace)
print("La faccia con più apparizioni è quella con id: "+str(idSpeaker)+" ed ha avuto "+str(maxSpeakerApparence)+" apparizioni con una pe

print("\nScene:")
for i in range(len(counter_scenes)):
    print(scenes[i] + ": " + str(counter_scenes[i]) + " frames (" + str(round((counter_scenes[i]/n_frame_analyzed)*100,2)) + "%)")
print("\nSegmenti:")
for indice,segment in enumerate(listOfSegments):
    print("Segmento {}: {}   da {} a {}".format(indice+1,segment.id_label,round(segment.start_time,0),round(segment.end_time,0)))
print(" - - - - - - - - - - - - - - - - - - ")
```

We create the Segment structure and group all the consecutive frames with the same scene. In the end, we print the video summary which includes the array of all the detected faces, the id of the speaker, the percentage of the scenes and the information related to the segments.

After all those face file images generated, we proceed to delete them because they’re no longer used.

```
# DELETE FACE IMAGES
import os, shutil
for filename in os.listdir(IMAGE_FACES_PATH):
    file_path = os.path.join(IMAGE_FACES_PATH, filename)
    try:
        if os.path.isfile(file_path) or os.path.islink(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)
    except Exception as e:
        print('Failed to delete %s. Reason: %s' % (file_path, e))
```

We had to save the images because DeepFace doesn’t take as input the image itself but only the path to the image.

In conclusion, we write the information of the list of segments which contains all the information of the segments and the related frame's data to a text file in JSON format.

```
693  # # Write video data in json output file
694  import json
695  results = [obj.to_dict() for obj in listOfSegments]
696  print(results)
697  with open('data_'+FILENAME+'.txt', 'w') as outfile:
698      json.dump(results, outfile)
699
700
701  # # Read video data from json file
702  # with open('data_prova2persone.txt') as f:
703  #   data = json.load(f)
704
705  # print(type(data))
706
707  # # Example of reading data structure from JSON file
708  # for i in data:
709  #   for j in i["list_of_same_scene"]:
710  #       for k in j["list_words"]:
711  #           if k=='\\':
712  #               print("!!!!!!!!!!!!")
713  #   #print(i)
```

We've also included some lines of code to read the text file and to access the data structure.

calculateFeatureAndTrain_module.py

This module is used to calculate our features for the machine learning model.

```python
# Plot the color histogram divided in 16 groups
def plotList(array):
    objects = ('0', '1', '2', '3', '4', '5','6','7','8','9','10','11','12','13','14','15')
    y_pos = np.arange(len(objects))
    performance = array

    plt.bar(y_pos, performance, align='center', alpha=0.5)
    plt.xticks(y_pos, objects)

    plt.show()
```

This function plots the color histogram.

```python
# Print the confusion matrix
def confusionMatrix(y_test, y_pred, actions):
    print("\nCONFUSION MATRIX\n")

    # print(y_pred)
    # print(y_pred.argmax(axis=1))

    # Model Accuracy, how often is the classifier correct?
    # print("metrics.accuracy score 1D (non tiene conto dei null):\t",metrics.accuracy_score(y_test.argmax(axis=1), y_pred.argmax(axis=1
    print("metrics.accuracy score normale (considera i null sbagliati) - Testing score:\t",metrics.accuracy_score(y_test, y_pred))

    matrix = [[0 for x in range(len(actions))] for y in range(len(actions))]
    errori = 0
    predNull = 0
    for i in range(0,len(y_test)):
        if 1 in y_pred[i]:
            if(np.argmax(y_pred[i]) != np.argmax(y_test[i])):
                matrix[np.argmax(y_test[i])][np.argmax(y_pred[i])]=matrix[np.argmax(y_test[i])][np.argmax(y_pred[i])] + 1
                errori=errori+1
            #print("valore predetto per campione "+ str(i)+ ": "+str(actions[np.argmax(y_pred[i])])) #prediction
            #print("valore effettivo per campione "+ str(i)+ ": "+str(actions[np.argmax(y_test[i])])+"\n") #valore effettivo
            else:
                matrix[np.argmax(y_test[i])][np.argmax(y_pred[i])]=matrix[np.argmax(y_test[i])][np.argmax(y_pred[i])] + 1
        else:
            errori+=1
            predNull+=1

    mat=pd.DataFrame(np.row_stack(matrix))
    col=[]
    for i in range(len(actions)):
        col.append(actions[i])

    mat.columns=col
    mat.index=actions

    print("numero campioni di test: "+str(len(y_pred))+"   campioni erroneamente classificati: "+str(errori) + "   campioni classificati
    print(mat)

    # print("\nmetrics.confusion_matrix")
    # print(confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1)))
```

This confusionMatrix function calculates and prints the confusion matrix.

```python
# Extract features from multiple images in a folder
def featureExtraction(IMAGE_TRAIN_PATH):
    PROJECT_PATH=pathlib.Path(__file__).parent.resolve() #restituisce il path del progetto
    FINAL_IMAGE_TRAIN_PATH=os.path.join(PROJECT_PATH,IMAGE_TRAIN_PATH)

    list_subfolders_with_paths = [f.path for f in os.scandir(FINAL_IMAGE_TRAIN_PATH) if f.is_dir()]

    numero_gruppi = 16


    X_train = []
    y_train = []

    for directory in list_subfolders_with_paths:

        #cancella le 2 righe sotto al commento
        #cartella=os.path.basename(os.path.normpath(directory))
        #if cartella=="noAlzateLaterali":
            #directory=FULL_VIDEO_PATH
        print("sto processando le img in: "+ directory )
        for file in os.listdir(directory):
            path_directory = os.path.join(FINAL_IMAGE_TRAIN_PATH, directory)
            filename = os.fsdecode(file)
            final_file_path = os.path.join(path_directory, filename)

            img = cv.imread(final_file_path,0)
            hist = cv.calcHist([img],[0],None,[256],[0,256])

            dimensione_gruppo = len(hist) / numero_gruppi

            counter = 0
            array_of_hist = []
            sum = 0
            for i in range(0, len(hist)):
                counter+=1

                sum = sum + hist[i][0].astype(int)
```

This is one of our main functions used to extract the features from multiple images contained in the ‘images/train’ folder within the workspace.
In this ‘train’ folder we have other 4 folders called ‘blackboard’, ‘slide’, ‘slide-and-talk’, ‘talk’ and inside these folders we have all the images used to train the model.
The function selects all these images and extracts their features and annotates them to their category.
The features extracted are:

- Color histogram divided into 16 groups of colors
- People’s face presence (0 = no people’s face, 1 = one face, 2 = two or more faces)
- Position (xmedio) of the person’s face
- Faces’ size (area of the bounding box)

This function returns two lists:

- X_train: contains all the features of the images
- y_train: contains the category which is associated (0 = blackboard, 1 = slide, 2 = slide-and-talk, 3 = talk)

```python
# Extract features from a single image (either image path or the image already read)
def singleImageFeatureExtraction(final_file_path = None, img = None):
    numero_gruppi = 16
    flagException = False

    try:
        if (final_file_path == None) and (img == None):
            flagException = True
    except:
        pass

    try:
        if ((final_file_path == None) and (img.all() == None)):
            flagException = True
    except:
        pass

    if flagException == True:
        raise Exception("inserire un parametro alla funzione singleImageFeatureExtraction")

    if final_file_path != None:
        img = cv.imread(final_file_path,0)

    hist = cv.calcHist([img],[0],None,[256],[0,256])

    dimensione_gruppo = len(hist) / numero_gruppi

    counter = 0
    array_of_hist = []
    sum = 0
    for i in range(0, len(hist)):
        counter+=1

        sum = sum + hist[i][0].astype(int)
        if counter == dimensione_gruppo:
            array_of_hist.append(sum)
            counter = 0
            sum = 0
```

This function is similar to the previous one, but it takes only 1 image path or 1 image already read.
It is used to extract the single image's features, then passed to the prediction of the model.
This function returns the list of features.

```python
# Print the model's score
def print_model_score(model):
    X_train, y_train = featureExtraction(IMAGE_TRAIN_PATH)
    cross_score = cross_val_score(model, X_train, y_train, cv=5)
    print("Model's score: %f accuracy with a standard deviation of %f" % (cross_score.mean(), cross_score.std()))
```

This function is used to evaluate our model.

```python
# Function to train the XGBoost model
def train():
    X_train, y_train = featureExtraction(IMAGE_TRAIN_PATH)
    X_test, y_test = featureExtraction(IMAGE_TEST_PATH)


    # # SVM training
    # print("# SVM --- Best HP training: ")
    # study=svm_module.findBestHyperparameters(X_train, y_train)
    # model=svm_module.train(X_train, y_train, study.best_params)
    # y_pred = model.predict(X_test)
    # print("------ y_pred")
    # print(y_pred)

    scenes = [0,1,2,3]
    # # confusionMatrix(y_test, y_pred, scenes)
    # print(confusion_matrix(y_true=y_test, y_pred=y_pred, labels=scenes))



    # XGBoost training
    print("# XGBoost --- Best HP training: ")
    study=xgboost_module.findBestHyperparameters(X_train, y_train)
    model=xgboost_module.train(X_train, y_train, study.best_params)
    y_pred = model.predict(X_test)
    print_model_score(model)
    print(confusion_matrix(y_true=y_test, y_pred=y_pred, labels=scenes))
```

This function is used to train our machine learning model. We use specifically the extreme gradient boosting algorithm along with Optuna (hyperparameters optimization framework) to train the model.

svm_module.py

This module was used the first time to test the paper's approach. But later we switched to 'xgboost_module.py'.

```
# Transform from one hot encoding to 1-dimensional list
def oneHot_to_1D(y):

    y_final = []
    for i in range(0,len(y)):
        for j in range(0, 4):
            if y[i,j] == 1:
                y_final.append(j)
                break

    return y_final


# Transform from 1-dimensional list to one-hot-encoding
def oneD_to_oneHot(y):
    y_final = []
    for i in range(0,len(y)):
        if y[i] == 0:
            y_final.append([1,0,0,0])
        elif y[i] == 1:
            y_final.append([0,1,0,0])
        elif y[i] == 2:
            y_final.append([0,0,1,0])
        else:
            y_final.append([0,0,0,1])
    return y_final
```

These functions are used to switch from one-hot-encoding to 1-dimensional list and viceversa.

```python
# Used for cross validation score
def train_and_score(X_train, X_test, y_train, y_test):

    y_train_new = y_train
    y_test_new = y_test

    clf = svm.SVC(random_state=0)
    clf.fit(X=X_train, y=y_train_new)
    y_pred = clf.predict(X_test)
    score = clf.score(X_test,y_test_new)

    ac_score = metrics.accuracy_score(y_test_new, y_pred)
    print("\nSVM score:\t", score)
    print("metrics.accuracy score SVM:\t"+ str(ac_score))

    cross_score = cross_val_score(clf, X_train, y_train_new, cv=5)
    print("cross val score SVM:\t\t" + str(cross_score))
    print("cross val score SVM mean:\t" + str(cross_score.mean()))
    print("%f accuracy with a standard deviation of %f" % (cross_score.mean(), cross_score.std()))

    y_pred = oneD_to_oneHot(y_pred)
    return y_pred
```

This function is used to calculate the cross validation score and returns the prediction.

```python
# Train and save to file the model
def train(X_train,y_train,best_params):

    model = svm.SVC(**best_params, random_state=0)
    model.fit(X_train, y_train) # Training del modello con i dati

    cross_score = cross_val_score(model, X_train, y_train, cv=5) # training accuracy
    print("best: %f accuracy with a standard deviation of %f" % (cross_score.mean(), cross_score.std()))
    # save the model to disk
    filename = 'svm.sav'
    pickle.dump(model, open(filename, 'wb'))

    return model
```

This is used to train and obtain our model.

```python
# Find the best hyperparameters with Optuna
def findBestHyperparameters(X_train, y_train):

    def objective(trial):
        dtc_params = dict(
            kernel=trial.suggest_categorical('kernel',['rbf','poly','linear','sigmoid']),
            C=trial.suggest_float("C",0.1,3.0,log=True),
            gamma=trial.suggest_categorical('gamma',['auto','scale']),
            degree=trial.suggest_int("degree",1,3,log=True),
        )
        DTC = svm.SVC(**dtc_params, random_state=0)
        cross_score = cross_val_score(DTC, X_train, y_train, cv=5)
        error = 1.0 - cross_score.mean()
        return error


    # 3. Create a study object and optimize the objective function.
    study = optuna.create_study() # di default è minimize, quindi di minimizzare l'errore
    study.optimize(objective, n_trials=1000)

    print(study.best_params) # Printa i migliori parametri
    print(1.0 - study.best_value) # Printa l'accuracy
    return study
```

This function is used to find the best hyperparameters using Optuna, which tries to find the optimal hyperparameters combination by minimizing the error.

xgboost_module.py

This module is used to train our xgboost model.

```python
def oneHot_to_1D(y): ···

def oneD_to_oneHot(y): ···
```

These are the same functions used by svm_module.py to switch between one-hot-encoding and 1-dimensional list.

```python
def train_and_score(X_train, X_test, y_train, y_test):

    y_train_new = y_train
    y_test_new = y_test

    clf = XGBClassifier(use_label_encoder=False, eval_metric = 'mlogloss', random_state=0)
    clf.fit(X=X_train, y=y_train_new)
    y_pred = clf.predict(X_test)
    score = clf.score(X_test,y_test_new) # testing accuracy

    ac_score = metrics.accuracy_score(y_test_new, y_pred)
    print("\nExtreme gradient boosting score:\t", score)
    print("metrics.accuracy score XGBoost:\t"+ str(ac_score))

    cross_score = cross_val_score(clf, X_train, y_train_new, cv=5) # training accuracy
    print("cross val score XGBoost:\t\t" + str(cross_score)) # array di 5 elementi
    print("cross val score XGBoost mean:\t" + str(cross_score.mean()))
    print("%f accuracy with a standard deviation of %f" % (cross_score.mean(), cross_score.std()))

    y_pred = oneD_to_oneHot(y_pred)

    return y_pred # testing accuracy
```

This function is used to calculate the cross validation score and returns the prediction.

```python
def train(X_train,y_train,best_params):

    model = XGBClassifier(**best_params, use_label_encoder=False, eval_metric = 'mlogloss', random_state=0)
    model.fit(X_train, y_train) # Training del modello con i dati

    cross_score = cross_val_score(model, X_train, y_train, cv=5) # training accuracy
    print("best: %f accuracy with a standard deviation of %f" % (cross_score.mean(), cross_score.std()))
    # save the model to disk
    filename = 'xgboost500.sav'
    pickle.dump(model, open(filename, 'wb'))

    return model
```

This function is used to train with XGBoost algorithm and save the model

```python
def findBestHyperparameters(X_train, y_train):

    def objective(trial):
        dtc_params = dict(
            max_depth=trial.suggest_int("max_depth", 2, 10),
            learning_rate=trial.suggest_float("learning_rate", 1e-4, 1e-1, log=True),
            n_estimators=trial.suggest_int("n_estimators", 100, 2000),
            min_child_weight=trial.suggest_int("min_child_weight", 1, 10),
            colsample_bytree=trial.suggest_float("colsample_bytree", 0.2, 1.0),
            subsample=trial.suggest_float("subsample", 0.2, 1.0),
            reg_alpha=trial.suggest_float("reg_alpha", 1e-4, 1e2, log=True),
            reg_lambda=trial.suggest_float("reg_lambda", 1e-4, 1e2, log=True),
            gamma=trial.suggest_float("gamma", 0, 50),
        )
        DTC = XGBClassifier(**dtc_params, use_label_encoder=False, eval_metric = 'mlogloss', random_state=42)
        cross_score = cross_val_score(DTC, X_train, y_train, cv=5)
        error = 1.0 - cross_score.mean()
        return error


    # 3. Create a study object and optimize the objective function.
    study = optuna.create_study() # di default è minimize, quindi di minimizzare l'errore
    study.optimize(objective, n_trials=500)

    print(study.best_params) # Printa i migliori parametri
    print(1.0 - study.best_value) # Printa l'accuracy
    return study
```

This is used to find the best hyperparameters for our model by minimizing the error.

testTFmodel.py

This script is used to test the Tensorflow model trained to recognize the blackboards as previously cited.

```python
IMAGE = "images/cas1.jpg"       You, a day ago • final touch
img = cv2.imread(IMAGE)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)


num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
                for key, value in detections.items()}
detections['num_detections'] = num_detections

print("Num detections: ", num_detections)
```

To test an image, just change the path given to “IMAGE” variable and run the script.

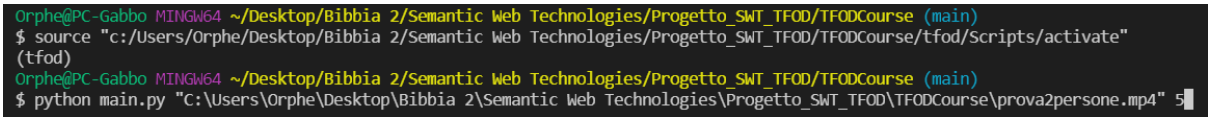
testYOLOv5-lavagne.py

This script is used to test the YoloV5 custom trained to detect blackboards.

```
4   fileName = "models/best-senzaAug.pt"
5   model = yolov5.load(fileName) # se ho il modello in locale
6
7   # non funziona bene, rileva lavagne in ogni immagine. Il problema può essere la scarsità e qualità del dataset.
8   def rilevaLavagna(model, frame):
9       model.conf = 0.6
10
11      isLavagna = False
12
13      results = model(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB), size=400)
14
15      print("--- Lavagne trovate ---")
16      print(results.pandas().xyxy[0])
17
18      labels, cord_thres = results.xyxyn[0][:, -1].numpy(), results.xyxyn[0][:, :-1].numpy() # estraggo labels e coo
19      classes = model.names
20      # show results
21      for i in labels:
22          # print(classes[int(i)])
23          if(classes[int(i)]=="lavagna"):
24              isLavagna=True
25
26      # show results
27      results.print()
28      results.show()
29      print(isLavagna)
30
31
32  frame = cv2.imread("images/lec8.jpg")
33  rilevaLavagna(model, frame)      You, 2 months ago • per testare fare print(framewithfaces e listofrec…
```

The model ‘best-senzaAug.pt’ is trained using a Google Colab script where we’ve loaded the dataset.
You can test the model by changing the parameter at line 32.

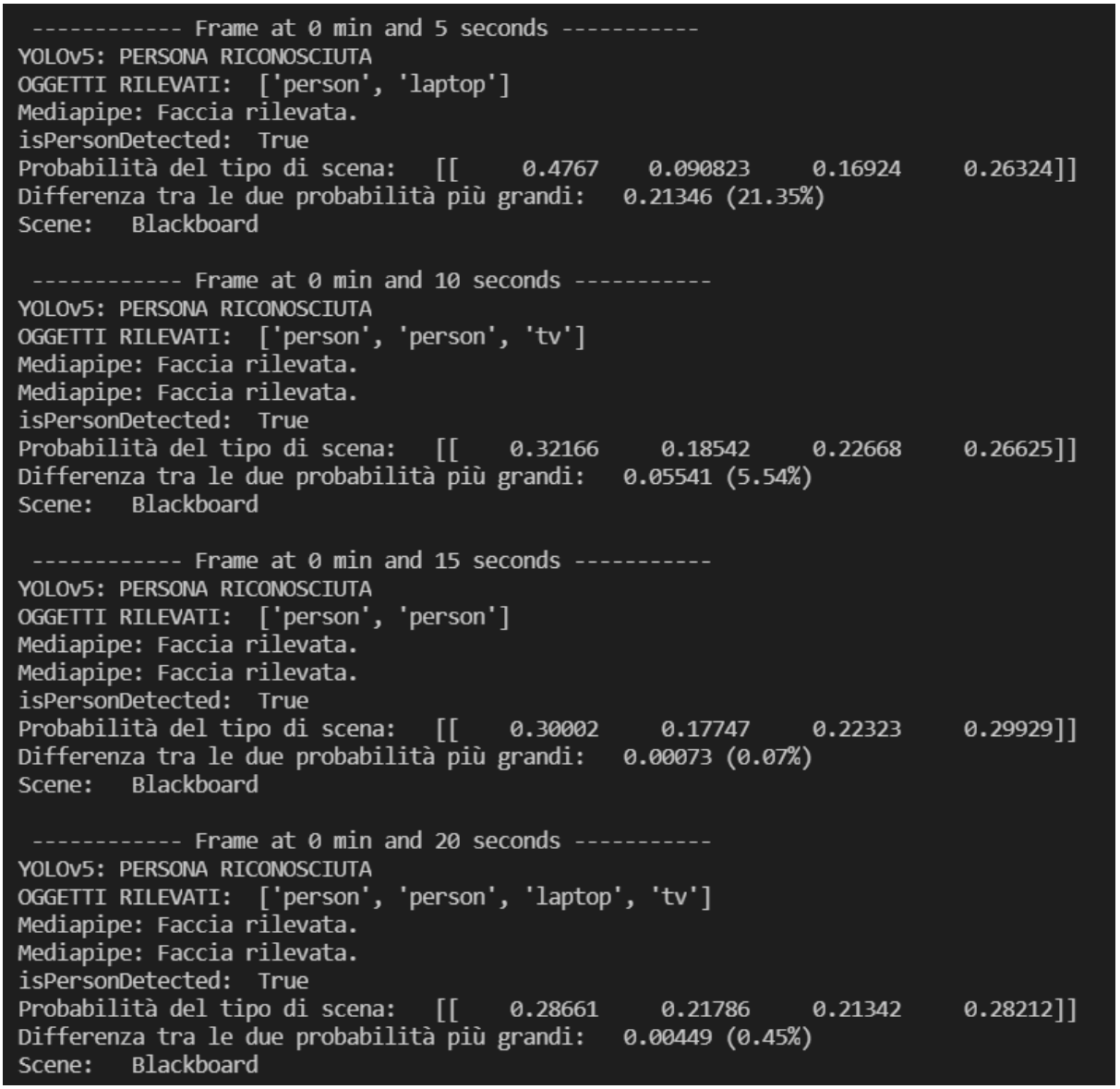
HOW TO USE



Run the main.py script in the terminal with 2 arguments: the first is the path of the video file (for example “C:/User/Video/filename.mp4”) and the second is an integer (in seconds) that represents the number of seconds we need to wait for the frame to be processed.
(Remember to activate your virtual environment before running the script)

OUTPUT FORMAT

Terminal output



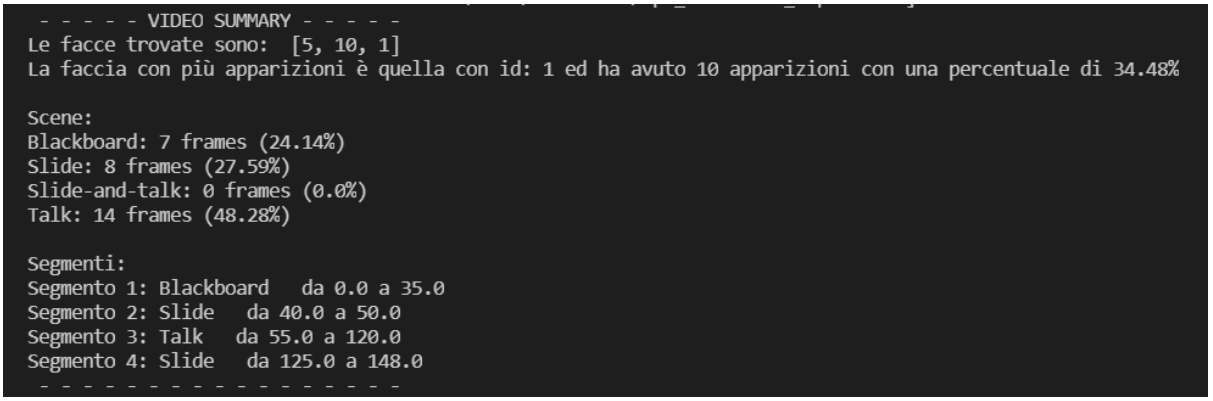
The screenshot above illustrates the output obtained from running the main script.
It tells you that at second 5, YoloV5 has found a person in the frame, 2 objects detected based on COCO classes, you can notice that on seconds 10 we have 2 rows of “Mediapipe: Faccia rilevata.” this means that Mediapipe has found 2 faces in that frame.
“Probabilità del tipo di scena” represents the probability of the scene in the given frame. As stated before, we have:

- first index as ‘blackboard’
- second index as ‘slide’
- third index as ‘slide-and-talk’
- forth index as ‘talk’

Then we have “Differenza tra le due probabilità più grandi” which is the difference between the top 2 largest probabilities obtained and this gives us a measure of certainty. If the

difference is big, then we will have a better certainty of the scene's prediction, else if the difference is small, then we will have a bit of uncertainty of the prediction.

Video summary



At the end of video elaboration, we will have a video summary which gives us:

- Faces' found are: [5, 10, 1] — this means that in the entire video 3 unique faces are found, where the first face appeared 5 times, the second face appeared 10 times and the third face appeared once. Hence, our host/speaker will be the second face that appeared the most
- Scene: a summary of the scenes predicted for the entire video
- Segments: prints out each segment with its starting and ending time

JSON data structure format

List of objects (objects are Segment type)
|-- (for each Segment)
|-- id_label
|-- start_time
|-- end_time
|-- list_of_same_scene (list of Record objects)
|-- (for each Record)
|-- num_frame
|-- time
|-- list_words
|-- isPersonDetected
|-- isSpeaker
|-- idFaces
|-- scene
|-- objects_detected

This data structure is saved as JSON format in a text file and you can read it with the following snippet code.

```
# Read video data from json file
with open('data_prova2persone.txt') as f:
    data = json.load(f)

print(type(data))

# Example of reading data structure from JSON file
for i in data:
    for j in i["list_of_same_scene"]:
        for k in j["list_words"]:
            if k=='\\':
                print("!!!!!!!!!!!")
    #print(i)
```