
Katna

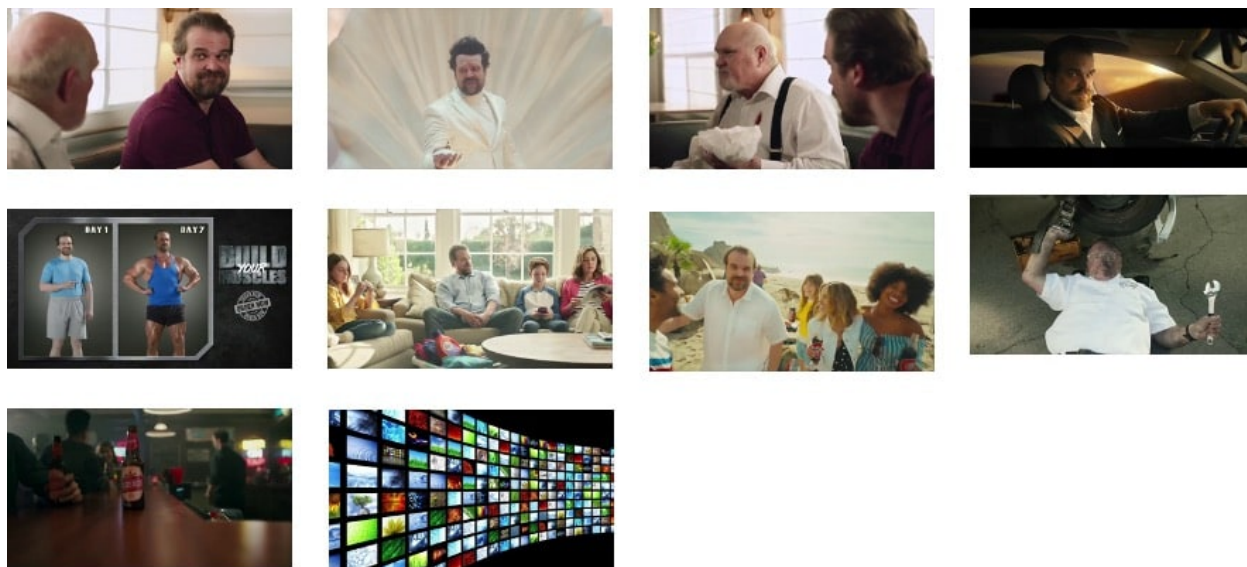
Mar 19, 2021

Contents

1	Video Module:	3
2	Image Module:	5
3	Indices and tables	41
	Python Module Index	43
	Index	45

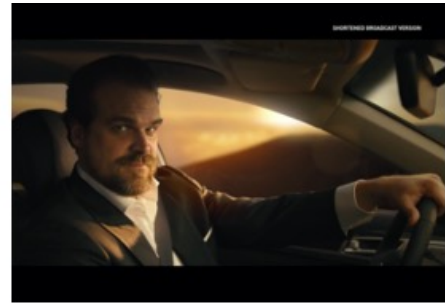
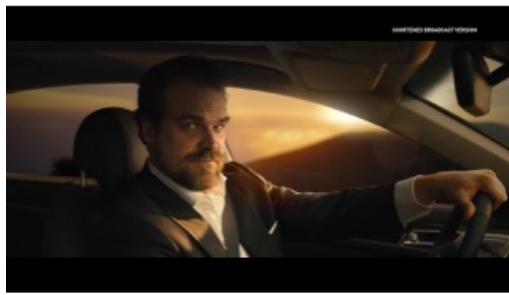
Katna automates the boring, error prone task of videos key/best frames extraction, video compression and manual time consuming task of image cropping. In summary you should consider using Katna library if you have following tasks which you want to automate:

1. You have video/videos from who you want to extract keyframe/keyframes. Please note Key-frames are defined as the representative frames of a video stream, the frames that provide the most accurate and compact summary of the video content. Take an example of this video and some of the top keyframes extracted using Katna.



2. You have video/videos you want to compress down to smaller size.
3. You have image/images which you want to smartly resize to a target resolution. (e.g. 500x500, 1080p (1920x1080) etc.)
4. You have image/images from which you want to intelligently extract a crop with a target resolution. (e.g. Get a crop of size 500x500 from image of size 1920x1080)
5. You want to extract a crop of particular aspect ratio e.g. 4:3 from your input image/images. (e.g. Get a crop of aspect ratio 1:1 from image of resolution 1920x1080 (16:9 aspect ratio image))
6. You want to resize a video into particular aspect ratio resize a 16:9 video to 4:3 aspect ratio video. Please note this feature is currently **experimental**.

Katna is divided into two modules namely video and image module. In next sections we will learn more about them in more details.



CHAPTER 1

Video Module:

This module handles the task(s) for key frame(s) extraction, video compression and **experimental** feature of video resize by aspect ratio.

Key-frames are defined as the representative frames of a video stream, the frames that provide the most accurate and compact summary of the video content.

Frame extraction and selection criteria

1. Frame that are sufficiently different from previous ones using absolute differences in LUV colorspace
2. Brightness score filtering of extracted frames
3. Entropy/contrast score filtering of extracted frames
4. K-Means Clustering of frames using image histogram
5. Selection of best frame from clusters based on and variance of laplacian (image blur detection)

Video compression is handled using ffmpeg library. Details about which could be read in [Katna.video_compressor module](#) section.

Since version 0.8.0 of Katna we are extending smart resize features to videos with the help of Google's Mediapipe project. To know more about this please refer to documentation [Katna.video_resize module](#). Please note that this feature is an experimental feature. And might be subject to removal/modifications at later versions. Also you also need to install Google's Mediapipe library, Specially autoflip binary for this to work. Please refer to [Smart video resize using katna](#) for how to install and configure mediapipe to be used with katna.

CHAPTER 2

Image Module:

This module handles the task(s) for smart cropping and smart image resizing.

The Smart crop feature tries to automatically identify important image areas where the user will focus more and tries to retain it while cropping. For a given input cropping dimension/final output image size, following selection and filtering criteria are used.

Crop selection and filtering criteria

1. Edge, saliency and Face features.
2. Distance of crop from Image edge, Rule of third
3. Crop filters. At the moment only text filter is supported. Text filter ensures that cropped rectangle contains text, also it ensures that texts present is not abruptly cropped.

Similar to Smart crop Katna image module supports **Smart image resizing** feature. Given an input image and target image resolution it will perform simple image image resize if if aspect ratio is same for input and target image. But in case of aspect ratio is different than smart image resize will extract a optimum crop in target resolution and then resizes image to target resolution. This ensures image resize without actually skewing output image. **Please not that if aspect ratio of input and output image are not same katna image_resize can lead to some loss of image content**

2.1 Supported Video and image file formats

All the major video formats like .mp4,.mov,.avi etc and image formats like .jpg, .png, .jpeg etc are supported. Only constraint is that video should be readable by ffmpeg library and images should be readable by opencv library.

2.1.1 How to install

Using pypi

- 1) Install Python 3
- 2) Install katna from pypi using this command

```
pip install katna
```

Install from source

- 1) Install git
- 2) Install Anaconda or Miniconda Python
- 3) Open terminal
- 4) Clone repo from here <https://github.com/keplerlab/Katna.git>
- 5) Change the directory to the directory where you have cloned your repo

```
cd path_to_the_folder_repo_cloned
```

- 6) Run the setup

```
python setup.py install
```

2.1.2 How to Use Katna.video

Extract keyframes for a video

Step 1

Import the video module

```
from Katna.video import Video
```

Step 2

Instantiate the video class inside your main module (necessary for multiprocessing in windows)

```
if __name__ == "__main__":  
    vd = Video()
```

Step 3

Call the **extract_video_keyframes** method. The method accepts two parameters and returns a list of numpy 2D array which are images. Refer to API reference for further details. Below are the two parameters required by the method

1. **no_of_frames**: Number of key frames to be extracted
2. **file_path**: Video file path.

```
imgs = vd.extract_video_keyframes(no_of_frames = no_of_frames_to_return, \  
file_path= video_file_path)
```

Step 4 (Optional)

In case you want to persist the extracted key frames then call the **save_frame_to_disk** method. The method accepts three parameters and returns nothing. Refer to API reference for further details. Below are the two parameters required by the method

1. **frame**: In-memory images generated by `extract_video_keyframes` method.
2. **file_name**: File name pattern for the persisted image.

3. **file_path**: Folder location where files needs to be saved
4. **file_ext**: File extension indicating the file type for example - '.jpg'

```
vd.save_frame_to_disk(img, file_path=output_folder_video_image, \
    file_name="test_"+str(counter), file_ext=".jpeg")
```

Code below is a complete example for a single video file.

```
1 from Katna.video import Video
2 import os
3
4 # For windows, the below if condition is must.
5 if __name__ == "__main__":
6
7     #instantiate the video class
8     vd = Video()
9
10    #number of key-frame images to be extracted
11    no_of_frames_to_return = 12
12
13    #Input Video file path
14    video_file_path = os.path.join(".", "tests","data", "pos_video.mp4")
15
16    #Call the public key-frame extraction method
17    imgs = vd.extract_video_keyframes(no_of_frames = no_of_frames_to_return, \
18        file_path= video_file_path)
19
20    # Make folder for saving frames
21    output_folder_video_image = 'selectedframes'
22    if not os.path.isdir(os.path.join(".", output_folder_video_image)):
23        os.mkdir(os.path.join(".", output_folder_video_image))
24
25    # Save all frames to disk
26    for counter,img in enumerate(imgs):
27        vd.save_frame_to_disk(img, file_path=output_folder_video_image, \
28            file_name="test_"+str(counter), file_ext=".jpeg")
```

Extract keyframes for all videos in a directory

Call the **extract_keyframes_from_videos_dir** method. The method accepts two parameters and return a dictionary with the file path as the key and list of numpy 2D array (which are images) as its value.

1. **no_of_frames**: Number of key frames to be extracted
2. **dir_path**: Directory path which has all the videos.

```
imgs = vd.extract_keyframes_from_videos_dir(no_of_frames = no_of_frames_to_return, \
    dir_path= dir_path_containing_videos)
```

Code below is a complete example for a directory containing videos.

```
1 from Katna.video import Video
2 import os
3 import ntpath
4
5 # For windows, the below if condition is must.
```

(continues on next page)

(continued from previous page)

```

6  if __name__ == "__main__":
7
8      #instantiate the video class
9      vd = Video()
10
11     #number of key-frame images to be extracted
12     no_of_frames_to_return = 3
13
14     #Input Video directory path
15     #All .mp4 and .mov files inside this directory will be used for keyframe extraction)
16     videos_dir_path = os.path.join(".", "tests", "data")
17
18     #Call the public key-frame extraction method
19     imgs = vd.extract_keyframes_from_videos_dir(no_of_frames = no_of_frames_to_return, \
20         dir_path = videos_dir_path)
21
22     # Make folder for saving frames
23     output_folder_video_image = 'selectedframes'
24     if not os.path.isdir(os.path.join(".", output_folder_video_image)):
25         os.mkdir(os.path.join(".", output_folder_video_image))
26
27     # Save all the frames to disk by segregating them into folders having the same name_
    ↪as the video file
28     for filepath, keyframe_data_li in imgs.items():
29
30         # name of the video file
31         filename = ntpath.basename(filepath)
32         name = filename.split(".")[0]
33
34         # folder path where the images will be stored
35         output_file_parent_folder_path = os.path.join(".", output_folder_video_image,
    ↪name)
36
37         # make folder with name of video if it doesnt exist
38         if not os.path.exists(output_file_parent_folder_path):
39             os.makedirs(output_file_parent_folder_path)
40
41         # save keyframes inside the folder
42         for counter, img in enumerate(keyframe_data_li):
43             vd.save_frame_to_disk(img, file_path=output_file_parent_folder_path,
44                 file_name=name + "_" + str(counter), file_ext=".jpeg")

```

Compress video using Katna

Step 1

Import the video module

```
from Katna.video import Video
```

Step 2

Instantiate the video class inside your main module (necessary for multiprocessing in windows)

```
if __name__ == "__main__":
    vd = Video()
```

Step 3

Call the **compress_video** method. The method accepts one required parameter that is path to input file returns status whether compression was done successfully or not. Refer to API reference for further details. Below are the parameters required by the method

1. **file_path**: Input video full file path. This is the only compulsory parameter

```
status = vd.compress_video(file_path= video_file_path)
```

Step 4 (Optional)

In case you play around with the different parameters like where to save compressed file etc. you can change optional parameters in compress_video function. Refer to API reference for further details. Below are the optional parameters supported by the method

1. **force_overwrite** (bool, optional) – optional parameter if True then if there is already a file in output file location function will overwrite it, defaults to False
2. **crf_parameter** (int, optional) – Constant Rate Factor Parameter for controlling amount of video compression to be applied, The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. It is recommend to keep this value between 20 to 30 A lower value is a higher quality, you can change default value by changing config.Video.video_compression_crf_parameter
3. **output_video_codec** (str, optional) – Type of video codec to choose, Currently supported options are libx264 and libx265, libx264 is default option. libx264 is more widely supported on different operating systems and platforms, libx265 uses more advanced x265 codec and results in better compression and even less output video sizes with same or better quality. Right now libx265 is not as widely compatible on older versions of MacOS and Widows by default. If wider video compatibility is your goal you should use libx264., you can change default value by changing Katna.config.Video.video_compression_codec
4. **out_dir_path** (str, optional) – output folder path where you want output video to be saved, defaults to ""
5. **out_file_name** (str, optional) – output filename, if not mentioned it will be same as input filename, defaults to ""

```
vd.compress_video(file_path, force_overwrite=False, \
crf_parameter=23, output_video_codec='libx264', out_dir_path='', out_file_name='')
```

Code below is a complete example for a single video file.

```
1  import os
2  from Katna.video import Video
3
4  def main():
5
6      vd = Video()
7
8      # folder to save extracted images
9      output_folder_for_compressed_videos= "compressed_folder"
10     out_dir_path = os.path.join(".", output_folder_for_compressed_videos)
11
12     if not os.path.isdir(out_dir_path):
13         os.mkdir(out_dir_path)
14
15     # Video file path
16     video_file_path = os.path.join(".", "tests", "data", "pos_video.mp4")
17     print(f"Input video file path = {video_file_path}")
18
19     status = vd.compress_video(
20         file_path=video_file_path,
```

(continues on next page)

(continued from previous page)

```

21         out_dir_path=out_dir_path
22     )
23
24
25     if __name__ == "__main__":
26         main()

```

Compress all videos in folder using Katna

Step 1

Import the video module

```
from Katna.video import Video
```

Step 2

Instantiate the video class inside your main module (necessary for multiprocessing in windows)

```
if __name__ == "__main__":
    vd = Video()
```

Step 3

Call the **compress_videos_from_dir** method. The method accepts one required parameter that is path to input folder where videos needs to be picked for compression returns status whether compression was done successfully or not. Refer to API reference for further details. Below are the parameters required by the method

1. **dir_path**: Input videos full folder path. This is the only compulsory parameter

```
status = vd.compress_videos_from_dir(dir_path=input_video_folder_path)
```

Step 4 (Optional)

In case you play around with the different parameters like where to save compressed file etc. you can change optional parameters in compress_video function. Refer to API reference for further details. Below are the optional parameters supported by the method

1. **force_overwrite** (bool, optional) – optional parameter if True then if there is already a file in output file location function will overwrite it, defaults to False
2. **crf_parameter** (int, optional) – Constant Rate Factor Parameter for controlling amount of video compression to be applied, The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. It is recommend to keep this value between 20 to 30 A lower value is a higher quality, you can change default value by changing config.Video.video_compression_crf_parameter
3. **output_video_codec** (str, optional) – Type of video codec to choose, Currently supported options are libx264 and libx265, libx264 is default option. libx264 is more widely supported on different operating systems and platforms, libx265 uses more advanced x265 codec and results in better compression and even less output video sizes with same or better quality. Right now libx265 is not as widely compatible on older versions of MacOS and Widows by default. If wider video compatibility is your goal you should use libx264., you can change default value by changing Katna.config.Video.video_compression_codec
4. **out_dir_path** (str, optional) – output folder path where you want output video to be saved, defaults to ""

```
vd.compress_videos_from_dir(dir_path, force_overwrite=False, \
crf_parameter=23, output_video_codec='libx264', out_dir_path='')

```

Code below is a complete example for a single video file.

```

1  import os
2  from Katna.video import Video
3
4  def main():
5
6      vd = Video()
7
8      # folder to save extracted images
9      output_folder_for_compressed_videos= "compressed_folder"
10     out_dir_path = os.path.join(".", output_folder_for_compressed_videos)
11
12     if not os.path.isdir(out_dir_path):
13         os.mkdir(out_dir_path)
14
15     # Video file path
16     video_folder_path = os.path.join(".", "tests", "data")
17     print(f"Input video folder path = {video_folder_path}")
18
19     status = vd.compress_videos_from_dir(
20         dir_path=video_folder_path,
21         out_dir_path=out_dir_path
22     )
23
24
25     if __name__ == "__main__":
26         main()

```

Smart video resize using katna

Please note that it is necessary to first install and initialize Google mediapipe autoflip solution before using Katna video resize (experimental) feature.

Install Google Mediapipe library and Autoflip solution.

1. Install Mediapipe by following these instructions : https://google.github.io/mediapipe/getting_started/install
2. Build Autoflip c++ solution by following these instructions: <https://google.github.io/mediapipe/solutions/autoflip>

Resize a single video using Katna (Using Experimental Mediapipe Autoflip bridge)

Step 1

Import the video module

```
from Katna.video import Video
```

Step 2

Instantiate the video class inside your main module (necessary for multiprocessing in windows)

```

autoflip_build_path = "/absolute/path/to/autoflip/build/folder
autoflip_model_path = "/absolute/path/to/autoflip/model/folder

if __name__ == "__main__":
    vd = Video(autoflip_build_path, autoflip_model_path)

```

Step 3 (Optional)

Configure the mediapipe autoflip properties. To check the list of configurable options, check *Katna.video_resize* module.

```

import Katna.config as app_config

# get the current configuration
conf = app_config.MediaPipe.AutoFlip.get_conf()

# set True for features which are required in output video
conf["ENFORCE_FEATURES"] = {
    "FACE_CORE_LANDMARKS": False,
    "FACE_ALL_LANDMARKS": False,
    "FACE_FULL": False,
    "HUMAN": False,
    "PET": False,
    "CAR": False,
    "OBJECT": False
}

# % stabalization threshold
conf["STABALIZATION_THRESHOLD"] = 0.5

# opacity of blur area
conf["BLUR_AREA_OPACITY"] = 0.6

# update configuration
app_config.MediaPipe.AutoFlip.set_conf(conf)

```

Step 4

Call the **resize_video** method. The method accepts three parameters and returns a status whether video resize is performed successfully or not. Refer to API reference for further details. Below are the four parameters required by the method

1. **file_path**: Video file path.
2. **abs_file_path_output**: absolute path for saving final output file.
3. **aspect_ratio**: required aspect ratio for output video. e.g. "4:3"

```

vd.resize_video(file_path = file_path, abs_file_path_output = abs_file_path_output,
↪aspect_ratio = aspect_ratio)

```

Code below is a complete example for a single video file.

```

1 from Katna.video import Video
2 import os
3
4 # For windows, the below if condition is must.
5 if __name__ == "__main__":
6

```

(continues on next page)

(continued from previous page)

```

7      # set the autoflip build and model path directory based on your installation
8      # usually autoflip build is located here : /mediapipe/repo/bazel-build/mediapipe/
      ↳ examples/desktop/autoflip
9      # usually mediapipe model is located here : /mediapipe/repo/mediapipe/models
10     autoflip_build_path = "/absolute/path/to/autoflip/build/folder
11     autoflip_model_path = "/absolute/path/to/autoflip/model/folder
12
13     # desired aspect ratio (e.g potrait mode - 9:16)
14     aspect_ratio = 9:16
15
16     # input video file path
17     file_path = os.path.join(".", "tests", "data", "pos_video.mp4")
18
19     # output file to save resized video
20     abs_file_path_output = os.path.join(".", "tests", "data", "pos_video_resize.mp4")
21
22     #instantiate the video class
23     vd = Video(autoflip_build_path, autoflip_model_path)
24
25     print(f"Input video file path = {file_path}")
26
27     vd.resize_video(file_path = file_path, abs_file_path_output = abs_file_path_
      ↳ output, aspect_ratio = aspect_ratio)
28
29     print(f"output resized video file path = {abs_file_path_output}")

```

NOTE : In case of subprocess.CalledProcessError, try running the `resize_video` method again.

Resize multiple videos in a directory using Katna (Using Experimental Mediapipe Autoflip bridge)

Call the **resize_video_from_dir** method. The method accepts three parameters and returns a status whether video resize is performed successfully or not. Refer to API reference for further details. Below are the four parameters required by the method

1. **dir_path:** Directory path where videos are stored.
2. **abs_dir_path_output:** absolute path to directory where resized videos will be dumped.
3. **aspect_ratio:** required aspect ratio for output video. e.g. "4:3"

```

vd.resize_video_from_dir(dir_path = dir_path, abs_dir_path_output = abs_dir_path_
↳ output, aspect_ratio = aspect_ratio)

```

Code below is a complete example for a folder full of video file.

```

1  from Katna.video import Video
2  import os
3
4  # For windows, the below if condition is must.
5  if __name__ == "__main__":
6
7      # folder where videos are located
8      dir_path = file_path = os.path.join(".", "tests", "data")
9
10     # output folder to dump videos after resizing
11     abs_dir_path_output = os.path.join(".", "tests", "data", "resize_results")

```

(continues on next page)

(continued from previous page)

```

12
13     # intialize video class
14     vd = Video(autoflip_build_path, autoflip_model_path)
15
16     # invoke resize for directory
17     try:
18         vd.resize_video_from_dir(dir_path = dir_path, abs_dir_path_output = abs_dir_
19 → path_output, aspect_ratio = aspect_ratio)
20     except Exception as e:
21         raise e
22
23     print(f"output resized video dir path = {abs_dir_path_output}")

```

2.1.3 How to Use Katna.image

Crop a single image

Step 1

Import the image module.

```
from Katna.image import Image
```

Step 2

Instantiate the image class.

```
img_module = Image()
```

Step 3

Call the **crop_image** method. This method accepts following parameters and returns a list of crop rectangles (in crop_rect data structure). Refer to API reference for further details. Below are the six parameters of the function

file_path: image file path from which crop has to be extracted

crop_width: width of crop to extract

crop_height: height of crop to extract

no_of_crops_to_return: number of crops rectangles to be extracted

filters: You can use this **optional** parameter to filter out unwanted crop rectangles according to some filtering criteria. At the moment only “text” detection filter is implemented and more filters will be added in future will be added in future. Passing on “text” detection filter ensures crop rectangle contains text, additionally it checks that detected “text” inside an image is not abruptly cropped by any crop_rectangle. By default, filters are not applied.

down_sample_factor: You can use this **optional** feature to specify the down sampling factor. For large images consider increasing this parameter for faster image cropping. By default input images are downsampled by factor of **8** before processing.

```

image_file_path = <Path where the image is stored>

crop_list = img_module.crop_image(
    file_path=image_file_path,
    crop_width=<crop_width>,
    crop_height=<crop_height>,

```

(continues on next page)

(continued from previous page)

```

num_of_crops=<no_of_crops_to_return>,
filters=<filters>,
down_sample_factor=<number_by_which_image_to_downsample>
)

```

Or

You can use **crop_image_from_cvimage** function in case you want to crop in-memory images. This method accepts opencv image as image source. Rest of the parameters are same as **crop_images** method. This function helps in connecting smart image cropping to any existing workflow.

```

img = cv2.imread(image_file_path)

crop_list = img_module.crop_image_from_cvimage(
input_image=img,
#Other parameters
)

```

If you want to get the crops by a specified aspect ratio. You can use **crop_image_with_aspect** function. This method accepts **crop_aspect_ratio** as parameter instead of height & width and returns a list of crop rectangles wrt to each crop dimension it finds with the specified aspect ratio.

crop_aspect_ratio: use this parameter to specify the aspect ratio by which crops need to be extracted. The parameter expects you to specify the aspect ratio in string format eg. '4:3' or '16:9'.

```

image_file_path = <Path where the image is stored>
crop_aspect_ratio = '4:3'

crop_list = img_module.crop_image_with_aspect(
    file_path=image_file_path,
    crop_aspect_ratio=<crop_aspect_ratio>,
    num_of_crops=<no_of_crops_to_return>,
    filters=<filters>,
    down_sample_factor=<number_by_which_image_to_downsample>
)

```

Step 4

To save the extracted crop rectangles call **save_crop_to_disk** method. The method accepts following parameters and doesn't returns anything. Refer to API reference for further details.

1. **crop_rect**: crop rect object from the extracted crops
2. **frame**: input image from which crops are extracted
3. **file_path**: Folder location where files needs to be saved
4. **file_name**: File name for the crop image to be saved.
5. **file_ext**: File extension indicating the file type for example - '.jpg'

```

img_module.save_crop_to_disk(crop_rect=<crop_rect>, frame=<image>, file_path=<output_
↪folder_cropped_image>,
    file_name=<file_name>,
    file_ext=<file_ext>,
)

```

Code below is a complete example for a single image.

```
1 import os.path
2 import cv2
3 from Katna.image import Image
4
5 img_module = Image()
6
7 # folder to save extracted images
8 output_folder_cropped_image = "selectedcrops"
9
10 if not os.path.isdir(os.path.join(".", \
11     output_folder_cropped_image)):
12
13     os.mkdir(os.path.join(".", \
14         output_folder_cropped_image))
15
16 # number of images to be returned
17 no_of_crops_to_return = 3
18
19 # crop dimensions
20 crop_width = 1000
21 crop_height = 600
22
23 # Filters
24 filters = ["text"]
25
26 # Image file path
27 image_file_path = os.path.join(".", "tests", "data", \
28     "image_for_text.png")
29
30 crop_list = img_module.crop_image(
31     file_path=image_file_path,
32     crop_width=crop_width,
33     crop_height=crop_height,
34     num_of_crops=no_of_crops_to_return,
35     filters= filters,
36     down_sample_factor=8
37 )
38
39 if len(crop_list) > 0:
40     top_crop = crop_list[0]
41     print("Top Crop", top_crop, " Score", top_crop.score)
42
43     img = cv2.imread(image_file_path)
44     img_module.save_crop_to_disk(top_crop, img,
45         file_path=output_folder_cropped_image,
46         file_name="cropped_image",
47         file_ext=".jpeg",
48     )
49
50
51 else:
52     print(
53         "No Perfect crop found for {0}x{1} with for Image {2}".format(
54             crop_width, crop_height ,image_file_path
55         )
56     )
```

Crop all images in a directory

To run crop image for all images in a directory, call the **crop_image_from_dir** method. This method accepts following parameters and returns a dictionary containing file path as key and list of crop rectangles (in crop_rect data structure) as its values. Below are the six parameters of the function

dir_path: directory path where images from which crop has to be extracted

crop_width: width of crop to extract

crop_height: height of crop to extract

no_of_crops_to_return: number of crops rectangles to be extracted

filters: You can use this **optional** parameter to filter out unwanted crop rectangles according to some filtering criteria. At the moment only “text” detection filter is implemented and more filters will be added in future will be added in future. Passing on “text” detection filter ensures crop rectangle contains text, additionally it checks that detected “text” inside an image is not abruptly cropped by any crop_rectangle. By default, filters are not applied.

down_sample_factor: You can use this **optional** feature to specify the down sampling factor. For large images consider increasing this parameter for faster image cropping. By default input images are downsampled by factor of **8** before processing.

```
input_dir_path = <Path to directory where images are stored>

crop_list = img_module.crop_image_from_dir(
    dir_path=input_dir_path,
    crop_width=<crop_width>,
    crop_height=<crop_height>,
    num_of_crops=<no_of_crops_to_return>,
    filters=<filters>,
    down_sample_factor=<number_by_which_image_to_downsample>
)
```

Code below is a complete example for a directory containing images.

```
1  import os.path
2  import cv2
3  import ntpath
4  from Katna.image import Image
5
6  img_module = Image()
7
8  # folder to save extracted images
9  output_folder_cropped_image = "selectedcrops"
10
11  if not os.path.isdir(os.path.join(".", \
12      output_folder_cropped_image)):
13
14      os.mkdir(os.path.join(".", \
15          output_folder_cropped_image))
16
17  # number of images to be returned
18  no_of_crops_to_return = 3
19
20  # crop dimensions
21  crop_width = 300
22  crop_height = 400
23
```

(continues on next page)

(continued from previous page)

```

24 # Filters
25 filters = ["text"]
26
27 # Directory containing images to be cropped
28 input_dir_path = os.path.join(".", "tests", "data")
29
30 crop_data = img_module.crop_image_from_dir(
31     dir_path=input_dir_path,
32     crop_width=crop_width,
33     crop_height=crop_height,
34     num_of_crops=no_of_crops_to_return,
35     filters=filters,
36     down_sample_factor=8
37 )
38
39 for filepath, crops in crop_data.items():
40
41     # name of the image file
42     filename = ntpath.basename(filepath)
43     name = filename.split(".")[0]
44
45     # folder path where the images will be stored
46     output_file_parent_folder_path = os.path.join(".", output_folder_cropped_image,
47 ↪ name)
48
49     if not os.path.exists(output_file_parent_folder_path):
50         os.makedirs(output_file_parent_folder_path)
51
52     if len(crops) > 0:
53         img = cv2.imread(filepath)
54         for count, crop in enumerate(crops):
55             img_module.save_crop_to_disk(crop, img, output_file_parent_folder_path,
56 ↪ name + "_cropped" + "_" + str(count), ".
57 ↪ jpeg")
58
59     else:
60         print(
61             "No Perfect crop found for {0}x{1} with for Image {2}".format(
62                 crop_width, crop_height, filepath
63             )
64         )

```

Resize a single image

Step 1

Import the image module.

```
from Katna.image import Image
```

Step 2

Instantiate the image class.

```
img_module = Image()
```

Step 3

Call the **resize_image** method. This method accepts following parameters and returns a in memory resized image in opencv format. Refer to API reference for further details. Below are the four parameters of the function

file_path: image file path from which crop has to be extracted

target_width: width of target image

target_height: height of target image

down_sample_factor: You can use this **optional** feature to specify the down sampling factor. For large images consider increasing this parameter for faster image resize and crop. By default input images are downsampled by factor of **8** before processing.

```
image_file_path = <Path where the image is stored>

crop_list = img_module.resize_image(
    file_path=image_file_path,
    target_width=<target_width>,
    target_height=<target_height>,
    down_sample_factor=<number_by_which_image_to_downsample>
)
```

Step 4

To save the extracted resized image call **save_image_to_disk** method. The method accepts following parameters and doesn't returns anything. Refer to API reference for further details.

1. **image:** output image to be saved
2. **file_path:** Folder location where files needs to be saved
3. **file_name:** File name for the crop image to be saved.
4. **file_ext:** File extension indicating the file type for example - '.jpg'

```
img_module.save_image_to_disk(image=<image>, file_path=<output_folder_cropped_image>,
    file_name=<file_name>,
    file_ext=<file_ext>,
)
```

Code below is a complete example for a single image.

```
1  import os.path
2  import cv2
3  from Katna.image import Image
4
5  def main():
6
7      # Extract specific number of key frames from video
8      img_module = Image()
9
10     # folder to save extracted images
11     output_folder_cropped_image = "resizedimages"
12
13     if not os.path.isdir(os.path.join(".", output_folder_cropped_image)):
14         os.mkdir(os.path.join(".", output_folder_cropped_image))
15
16     # crop dimentions
17     resize_width = 500
18     resize_height = 600
19
```

(continues on next page)

(continued from previous page)

```

20     # Image file path
21     image_file_path = os.path.join(".", "tests", "data", "bird_img_for_crop.jpg")
22     print(f"image_file_path = {image_file_path}")
23
24     resized_image = img_module.resize_image(
25         file_path=image_file_path,
26         target_width=resize_width,
27         target_height=resize_height,
28         down_sample_factor=8,
29     )
30     # cv2.imshow("resizedImage", resized_image)
31     # cv2.waitKey(0)
32
33     img_module.save_image_to_disk(
34         resized_image,
35         file_path=output_folder_cropped_image,
36         file_name="resized_image",
37         file_ext=".jpeg",
38     )
39
40     main()

```

Resize all images in a directory

To run resize image for all images in a directory, call the **resize_image_from_dir** method. This method accepts following parameters and returns a dictionary containing file path as key and resized image (in opencv numpy format) as its values. Below are the six parameters of the function

dir_path: directory path where images from which crop has to be extracted

target_width: width of output resized image

target_height: height of output resized image

down_sample_factor: You can use this **optional** feature to specify the down sampling factor. For large images consider increasing this parameter for faster image cropping and resizing. By default input images are downsampled by factor of **8** before processing.

```

input_dir_path = <Path to directory where images are stored>

crop_list = img_module.resize_image_from_dir(
    dir_path=input_dir_path,
    target_width=<target_width>,
    target_height=<target_height>,
    down_sample_factor=<number_by_which_image_to_downsample>
)

```

Code below is a complete example for a directory containing images.

```

1  import os.path
2  from Katna.image import Image
3  import os
4  import ntpath
5
6
7  def main():

```

(continues on next page)

(continued from previous page)

```

8
9     img_module = Image()
10
11     # folder to save resized images
12     output_folder_resized_image = "resizedimages"
13
14     if not os.path.isdir(os.path.join(".", output_folder_resized_image)):
15         os.mkdir(os.path.join(".", output_folder_resized_image))
16
17     # resized image dimensions
18     resize_width = 500
19     resize_height = 600
20
21     # Input folder file path
22     input_folder_path = os.path.join(".", "tests", "data")
23     print(f"input_folder_path = {input_folder_path}")
24
25     resized_images = img_module.resize_image_from_dir(
26         dir_path=input_folder_path,
27         target_width=resize_width,
28         target_height=resize_height,
29         down_sample_factor=8,
30     )
31
32     for filepath, resized_image in resized_images.items():
33         # name of the image file
34         filename = ntpath.basename(filepath)
35         name = filename.split(".")[0]
36         # folder path where the images will be stored
37         img_module.save_image_to_disk(
38             resized_image, output_folder_resized_image, name + "_resized" + "_", ".
↪ jpeg"
39         )
40
41
42     main()

```

2.1.4 Understanding katna

As mentioned in intro section katna consists of two modules, Video and Image, In this section we will go into details about each modules working.

Katna.video Module:

Video module handles the task(s) for key frame(s) extraction and video compression. This module has four primary public functions for keyframe extraction video_compression, which are **extract_video_keyframes**, **extract_keyframes_from_videos_dir**, **compress_video** and **compress_videos_from_dir**.

extract_video_keyframes is the primary function which given a video file extracts most important keyframe from a video. **extract_keyframes_from_videos_dir** actually runs **extract_video_frames** function for all video files in a directory recursively. Katna.video frame extraction feature first takes a video and divides a big video in smaller chunks of videos, it runs video frame extraction and frame selector tasks on these chunked videos in parallel. For each chunked video actual frame extraction is done in Katna by following two separate modules.

Details about public **compress_video** and **compress_videos_from_dir** functions is listed in [Katna.video_compressor module](#).

Katna.frame_extractor module

In frame extractor module given a input video all the video frames that are sufficiently different from previous ones using absolute differences in LUV colorspace are returned.

Katna.frame_selector module

In Katna.frame_selector module given list of frames returned from frame_extractor module following checks are performed:

1. Brightness score filtering of extracted frames.
2. Entropy/contrast score filtering of extracted frames.

Each of these properties are filtered based on threshold which you can check and edit in `Katna.config.ImageSelector` properties.

After frame filtering based on number of required frames N , N clusters are formed using K-Means clustering where $K=N$, clustering is done using image histogram based approach. After K-Means clustering, for each cluster selection of best frame from cluster is done using variance of laplacian sorting. In image processing world variance of laplacian method is often used for image blur detection. This sorting and selection ensures that least blurred image is selected from cluster.

Katna.video_compressor module

Apart from Frame extraction Katna.video module can also do efficient video compression. It is done by internal module called internally by Katna.video_compressor module and exposed publicly by two public functions: **compress_video** and **compress_videos_from_dir**. As the name suggests **compress_video** function does video compression on a single input video file and **compress_videos_from_dir** function recursively compresses all videos in a given input folder. Katna.video_compressor includes actual implementation of video compression using ffmpeg library.

As discussed **compress_video** functions can compress a given input video and saves the output in same folder with name=`original_input_file_name + "_compressed"` with mp4 extension. You can change this behavior and other Configurations using optional parameters.

In case you play around with the different parameters like where to save compressed file etc. you can change optional parameters in `compress_video` function. Below are the optional parameters supported by the method

1. **force_overwrite** (bool, optional) – optional parameter if True then if there is already a file in output file location function will overwrite it, defaults to False
2. **crf_parameter** (int, optional) – Constant Rate Factor Parameter for controlling amount of video compression to be applied, The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. It is recommend to keep this value between 20 to 30 A lower value is a higher quality, you can change default value by changing `config.Video.video_compression_crf_parameter`
3. **output_video_codec** (str, optional) – Type of video codec to choose, Currently supported options are libx264 and libx265, libx264 is default option. libx264 is more widely supported on different operating systems and platforms, libx265 uses more advanced x265 codec and results in better compression and even less output video sizes with same or better quality. Right now libx265 is not as widely compatible on older versions of MacOS and Windows by default. If wider video compatibility is your goal you should use libx264., you can change default value by changing `Katna.config.Video.video_compression_codec`

4. **out_dir_path** (str, optional) – output folder path where you want output video to be saved, defaults to “”
5. **out_file_name** (str, optional) – output filename, if not mentioned it will be same as input filename, defaults to “”

Katna.video_resize module

As mentioned in home section since version 0.8.0 of Katna we are extending smart resize features to videos with the help of Google’s Mediapipe project. In simple terms video resize functionality in Katna currently is a thin python wrapper around Google Mediapipe Autoflip solution. If you want to learn more about how it works under the hood Please refer to this blog post by Google AI: <https://ai.googleblog.com/2020/02/autoflip-open-source-framework-for.html> . Please refer to *Smart video resize using katna* for how to install and configure mediapipe to be used with katna. Right now following parameters are configurable using Katna video module:

1. **STABALIZATION_THRESHOLD** (float, optional) – defaults to “0.5” : This parameter controls extent of motion stabilization is applied while tracking an object or face across the frames for video resizing. Higher value Means more aggressive motion stabilization and vice versa.
2. **BLUR_AREA_OPACITY** (int, optional) – defaults to “0.6” In some cases e.g. while compulsorily including all faces across frames, it is not possible to do without rendering padding in video. In case overlay of same video content is used as padding. This parameter controls how much opacity to add to padded content.
3. **ENFORCE_FEATURES.FACE_CORE_LANDMARKS** (bool, optional) – defaults to **False** In case of this parameter set as true, It is ensured that all face present in video are compulsorily included in final resized video.
4. **ENFORCE_FEATURES.FACE_FULL** (bool, optional) – defaults to **False** In case of this parameter set as true, It is ensured that all full faces present in video are compulsorily included in final resized video.
5. **ENFORCE_FEATURES.HUMAN** (bool, optional) – defaults to **False** In case of this parameter set as true, It is ensured that all persons/humans present/detected in video are compulsorily included in final resized video.
6. **ENFORCE_FEATURES.PET** (bool, optional) – defaults to **False** In case of this parameter set as true, It is ensured that all PET’s like dogs and cats present/detected in video are compulsorily included in final resized video.
7. **ENFORCE_FEATURES.CAR** (bool, optional) – defaults to **False** In case of this parameter set as true, It is ensured that all CARs present/detected in video are compulsorily included in final resized video.
8. **ENFORCE_FEATURES.OBJECT** (bool, optional) – defaults to **False** In case of this parameter set as true, It is ensured that all objects detected in video are compulsorily included in final resized video.

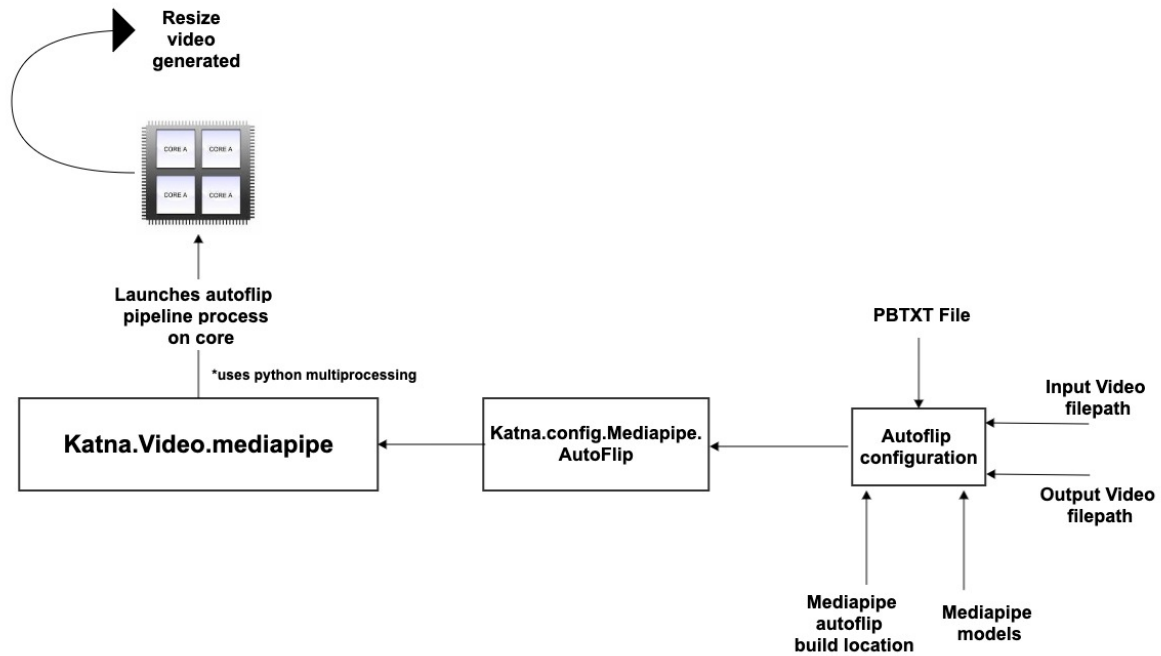
Mediapipe Autoflip Integration

Katna uses python multiprocessing to create a pool of processes based on the number of cpu cores on machine, and launch processes to resize video on these cores. For example on a 12 core machine, if 4 videos needs to be resized in parallel, Katna will launch mediapipe autoflip pipeline for 4 videos on 4 cores.

Since mediapipe is installed outside of Katna, mediaipie autoflip pipeline is launched using the python subprocess module. To ensure smooth run, Katna needs build autorun directory path (where binaries are build) and also the mediapipe models directory which containe tflite (tensorflow lite models for CPU). Internally, Katna creates a simlink to the models folder directory during its instance of execution. This allow Katna to access the models file required to run autoflip pipeline.

Below you can see a simple architecture describing the integration:

Katna provides an interface to configure autoflip. This enables Katna to hide the complexity of autoflip graphs configuration (.pbtxt file) and provide users will relevant configurations in the form of python friendly dictionary. Behind the scenes, Katna will create a temporary graph (.pbtxt) file based on the configuration provided and use it to run the autoflip pipeline.



Katna users can check **temp_pbtxt** directory when the pipeline is running to look at the mediapipe autoflip graph file. The folders gets deleted automatically when the pipeline finishes its execution.

To check the list of configurable options, check [Katna.video_resize module](#).

Katna.image Module:

This module handles the task(s) for smart cropping. The Smart crop feature tries to automatically identify important image areas where the user will focus more and tries to retain it while cropping. For a given input cropping dimension/final output image size, Katna.image works by first extracting all possible image crop given crop specification using `katna.crop_extractor` module, `Katna.crop_selector` module then uses various filtering and selection criteria to select best crops from list of image crops. Let's read more about these two modules in details.

Katna.crop_extractor module

`Katna.crop_extractor` module works by given a crop specification using a sliding window approach it first calculates all possible crop see `_get_all_possible_crops()` function inside `Katna.crop_extractor` module. Additionally it applies rule of third and crop rectangle distance from edge score. Configurations related to these scoring rules could be edited in `Katna.config.CropScorer` module.

Katna.crop_selector module

After returning candidate crops from `crop_extractor` module `Katna.crop_selector` module first does further filtering using `Katna.image_filters` filters. At the moment only text filter is supported. Text filter ensures that if cropped rectangle contains text, texts present is not abruptly cropped.

After performing `crop_filtering` crop selection is done by first calculating additional crop scoring is done based on following criteria: Saliency, edge features and Face features. This score is then combined with rule of third and crop

distance from edge feature calculated in crop_extractor module. Configurations related to these scoring rules could be edited in Katna.config.CropScorer, Katna.config.EdgeFeature, Katna.config.FaceFeature modules.

2.1.5 Tips and Troubleshooting

1) If input image is of very large size (larger than 2000x2000) it might take a long time to perform Automatic smart cropping. If you encounter this issue, consider changing down_sample_factor from default 8 to larger values (like 16 or 32). This will decrease processing time significantly.

2) If you get “AttributeError: module ‘cv2.cv2’ has no attribute ‘saliency’” error. then try re-installing opencv-contrib

```
python -m pip uninstall opencv-contrib-python
python -m pip install opencv-contrib-python
```

3) If you get “FileNotFoundError: frozen_east_text_detection.pb file not found”. Open python shell and follow the below commands.

```
from Katna.image_filters.text_detector import TextDetector
td = TextDetector()
td.download_data()
```

4) If you are running the code on windows, make sure to create the main file in the below format.

```
from Katna.video import Video

def main():
    vd = Video()
    # your code...

if __name__ == "__main__":
    main()
```

OR

```
from Katna.video import Video

if __name__ == "__main__":

    vd = Video()
    # your code
```

5) On windows, ensure that anaconda has admin rights if installing with anaconda as it fails with the write permission while installing some modules.

6) Python version 3.8 not supported due to the numpy and moviepy errors with this python version.

7) If you get “RuntimeError: No ffmpeg exe could be found. Install ffmpeg on your system, or set the IMAGEIO_FFMPEG_EXE environment variable”. Go to the **imageio-ffmpeg-*.egg** folder inside your **site-packages** folder, there’s ffmpeg file inside binaries folder set it’s path to environment variable.

Mediapipe Build Issues

1) If you are unable to run the “hello-world” example for MacOS, refer to the issue reported here: <https://github.com/bazelbuild/bazel/issues/8053#issuecomment-490793705> . Sometimes the build doesn’t work due to openCV version or dependencies on glog.

2) MediaPipe build can also give c++ compilations errors when building using Bazel. In some situations, this happens due to protobuf installation on system wherein Bazel accidentally picks up header files from the system when compiling

Bazel's checked in Protobuf C++ sources on macOS. The solution is to uninstall protobuf and is mentioned over here: <https://github.com/bazelbuild/bazel/issues/8053#issuecomment-490793705>

2.1.6 Katna

Main modules

Katna.video module

class `Katna.video.Video` (*autoflip_build_path=None, autoflip_model_path=None*)

Bases: `object`

Class for all video frames operations

Parameters `object` (class:*Object*) – base class inheritance

compress_video (*file_path*, *force_overwrite=False*, *crf_parameter=23*, *output_video_codec='libx264'*, *out_dir_path=""*, *out_file_name=""*)

Function to compress given input file

Parameters

- **file_path** (*str*) – Input file path
- **force_overwrite** (*bool*, *optional*) – optional parameter if True then if there is already a file in output file location function will overwrite it, defaults to False
- **crf_parameter** (*int*, *optional*) – Constant Rate Factor Parameter for controlling amount of video compression to be applied, The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. It is recommend to keep this value between 20 to 30 A lower value is a higher quality, you can change default value by changing `config.Video.video_compression_crf_parameter`
- **output_video_codec** (*str*, *optional*) – Type of video codec to choose, Currently supported options are libx264 and libx265, libx264 is default option. libx264 is more widely supported on different operating systems and platforms, libx265 uses more advanced x265 codec and results in better compression and even less output video sizes with same or better quality. Right now libx265 is not as widely compatible on older versions of MacOS and Windows by default. If wider video compatibility is your goal you should use libx264., you can change default value by changing `Katna.config.Video.video_compression_codec`
- **out_dir_path** (*str*, *optional*) – output folder path where you want output video to be saved, defaults to ""
- **out_file_name** (*str*, *optional*) – output filename, if not mentioned it will be same as input filename, defaults to ""

Raises Exception – raises `FileNotFoundError` Exception if input video file not found, also exception is raised in case output video file path already exist and `force_overwrite` is not set to True.

Returns Status code Returns True if video compression was successful else False

Return type `bool`

compress_videos_from_dir (*dir_path*, *force_overwrite=False*, *crf_parameter=23*, *output_video_codec='libx264'*, *out_dir_path=""*, *out_file_name=""*)

Function to compress input video files in a folder

Parameters

- **dir_path** (*str*) – Input folder path
- **force_overwrite** (*bool, optional*) – optional parameter if True then if there is already a file in output file location function will overwrite it, defaults to False
- **crf_parameter** (*int, optional*) – Constant Rate Factor Parameter for controlling amount of video compression to be applied, The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. It is recommend to keep this value between 20 to 30 A lower value is a higher quality, you can change default value by changing `config.Video.video_compression_crf_parameter`
- **output_video_codec** (*str, optional*) – Type of video codec to choose, Currently supported options are libx264 and libx265, libx264 is default option. libx264 is more widely supported on different operating systems and platforms, libx265 uses more advanced x265 codec and results in better compression and even less output video sizes with same or better quality. Right now libx265 is not as widely compatible on older versions of MacOS and Widows by default. If wider video compatibility is your goal you should use libx264., you can change default value by changing `Katna.config.Video.video_compression_codec`
- **out_dir_path** (*str, optional*) – output folder path where you want output video to be saved, defaults to ""

Raises Exception – raises FileNotFoundError Exception if input video file not found, also exception is raised in case output video file path already exist and `force_overwrite` is not set to True.

Returns Status code Returns True if video compression was successfull else False

Return type bool

extract_keyframes_from_videos_dir (*no_of_frames, dir_path*)

Returns best key images/frames from the videos in the given directory. you need to mention number of keyframes as well as directory path containing videos. Function returns python dictionary with key as filepath each dictionary element contains list of python numpy image objects as keyframes.

Parameters

- **no_of_frames** (*int, required*) – Number of key frames to be extracted
- **dir_path** (*str, required*) – Directory location with videos

Returns Dictionary with key as filepath and `numpy.2darray` Image objects

Return type dict

extract_video_keyframes (*no_of_frames, file_path*)

Returns a list of best key images/frames from a single video.

Parameters

- **no_of_frames** (*int, required*) – Number of key frames to be extracted
- **file_path** (*str, required*) – video file location

Returns List of `numpy.2darray` Image objects

Return type list

resize_video (*file_path, abs_file_path_output, aspect_ratio*)

Resize a single video file

Parameters

- **file_path** (*str*) – file path of the video to be resized
- **abs_file_path_output** (*str*) – absolute path to output video file
- **aspect_ratio** (*[type]*) – aspect ratio of the final video

Raises Exception – [description]

resize_video_from_dir (*dir_path, abs_dir_path_output, aspect_ratio*)

Resize all videos inside the directory

Parameters

- **dir_path** (*str*) – Directory path where videos are located
- **abs_dir_path_output** (*str*) – Absolute path to directory where output videos should to be dumped
- **aspect_ratio** (*[type]*) – desirable aspect ratio for the videos

Raises Exception – [description]

save_frame_to_disk (*frame, file_path, file_name, file_ext*)

saves an in-memory numpy image array on drive.

Parameters

- **frame** (*numpy.ndarray, required*) – In-memory image. This would have been generated by `extract_video_keyframes` method
- **file_name** (*str, required*) – name of the image.
- **file_path** (*str, required*) – Folder location where files needs to be saved
- **file_ext** (*str, required*) – File extension indicating the file type for example - `'jpg'`

Returns None

Katna.image module

class `Katna.image.Image` (*disable_text=True*)

Bases: `object`

Class for all image cropping operations

Parameters **object** (*class:Object*) – base class inheritance

Constructor for image files

crop_image (*file_path, crop_width, crop_height, num_of_crops, filters=[], down_sample_factor=8*)

smartly crops the imaged based on the specification - width and height

Parameters

- **file_path** (*str, required*) – Input file path
- **crop_width** (*int*) – output crop width
- **crop_height** (*int*) – output crop height
- **num_of_crops** (*int*) – number of crops required
- **filters** (*list (eg. ['text'])*) – filters to be applied for cropping (checks if image contains english text and the crop rectangle doesn't cut the text)

- Returns**
- crop list

crop_image_from_cvimage (*input_image*, *crop_width*, *crop_height*, *num_of_crops*, *filters*=[],
down_sample_factor=8)
 smartly crops the imaged based on the specification - width and height

- **input_image** (*numpy array, required*) – Input image
- **crop_width** (*int*) – output crop width
- **crop_height** (*int*) – output crop height
- **num_of_crops** (*int*) – number of crops required
- **filters** (*list (eg. ['text'])*) – filters to be applied for cropping (only returns crops containing english text where the crop rectangle doesn't cut the text)
- **down_sample_factor** (*int [default=8]*) – number by which you want to reduce image height & width (use it if image is large or to fasten the process)

Return type list of structure `crop_rect`

Parameters

- **dir_path** (*str*, *required*) – Input Directory path
- **crop_width** (*int*) – output crop width
- **crop_height** (*int*) – output crop height
- **num_of_crops** (*int*) – number of crops required
- **filters** (*list* (eg. ['text'])) – filters to be applied for cropping (checks if image contains english text and the crop rectangle doesn't cut the text)
- **down_sample_factor** (*int* [default=8]) – number by which you want to reduce image height & width (use it if image is large or to fasten the process)

Return type dict

crop_image_with_aspect (*file_path*, *crop_aspect_ratio*, *num_of_crops*, *filters*=[],
down_sample_factor=8)
 smartly crops the image based on the aspect ratio and returns number of specified crops for each crop
 spec found in the image with the specified aspect ratio

- **file_path**(*str*, *required*) – Input file path
- **crop_aspect_ratio**(*str* (eg. '4:3')) – output crop ratio
- **num_of_crops**(*int*) – number of crops required

- **filters** (*list* (eg. ['text'])) – filters to be applied for cropping (checks if image contains english text and the crop rectangle doesn't cut the text)
- **down_sample_factor** (*int* [default=8]) – number by which you want to reduce image height & width (use it if image is large or to fasten the process)

Returns crop list

Return type list of structure crop_rect

resize_image (*file_path, target_width, target_height, down_sample_factor=8*)
smartly resizes the image based on the specification - width and height

Parameters

- **file_path** (*str, required*) – Input file path
- **target_width** (*int*) – output image width
- **target_height** (*int*) – output image height
- **down_sample_factor** (*int* [default=8]) – number by which you want to reduce image height & width (use it if image is large or to fasten the process)

Returns resized image

Return type cv_image

resize_image_from_dir (*dir_path, target_width, target_height, down_sample_factor=8*)
smartly resizes all the images (inside a directory) based on the specification - width and height

Parameters

- **dir_path** (*str, required*) – Input Directory path
- **target_width** (*int*) – output width
- **target_height** (*int*) – output height
- **down_sample_factor** (*int* [default=8]) – number by which you want to reduce image height & width (use it if image is large or to fasten the process)

Returns dict with key as filepath and resized image as in opencv format as value

Return type dict

save_crop_to_disk (*crop_rect, frame, file_path, file_name, file_ext, rescale=False*)
saves an in-memory crop on drive.

Parameters

- **crop_rect** (*crop_rect, required*) – In-memory crop_rect.
- **frame** (*numpy.ndarray, required*) – In-memory input image.
- **file_name** (*str, required*) – name of the image.
- **file_path** (*str, required*) – Folder location where files needs to be saved
- **file_ext** (*str, required*) – File extension indicating the file type for example - 'jpg'

Returns None

save_image_to_disk (*image, file_path, file_name, file_ext*)
saves an in-memory image obtained from image resize on drive.

Parameters

- **image** (*numpy.ndarray*, *required*) – In-memory input image.
- **file_name** (*str*, *required*) – name of the image.
- **file_path** (*str*, *required*) – Folder location where files needs to be saved
- **file_ext** (*str*, *required*) – File extension indicating the file type for example -
'jpg'

Returns None

```
class Katna.image.UserFiltersEnum
    Bases: object

    Enum class for filters

    text = 'TextDetector'
```

Config module

```
class Katna.config.CropScorer
    Bases: object

    detail_weight = 0.2
    edge_radius = 0.4
    edge_weight = -20
    face_bias = 0.01
    face_weight = 3.4
    outside_importance = -0.5
    rects_weight = 1
    rule_of_thirds = True
    saliency_bias = 0.2
    saliency_weight = 1.3
```

```
class Katna.config.EdgeFeature
    Bases: object

    ksize = 3
    max_val_threshold = 200
    min_val_threshold = 100
```

```
class Katna.config.FaceFeature
    Bases: object

    cache_subdir = 'models'
    confidence = 0.5
    model_file = 'res10_300x300_ssd_iter_140000_fp16.caffemodel'
    modelfile_download_link = 'https://raw.githubusercontent.com/opencv/opencv_3rdparty/dn
    prototxt_download_link = 'https://raw.githubusercontent.com/opencv/opencv/master/sampl
    prototxt_file = 'deploy.prototxt'
```

```
class Katna.config.FrameExtractor
    Bases: object

    USE_LOCAL_MAXIMA = True

    len_window = 20

    max_frames_in_chunk = 2500

    window_type = 'hanning'

class Katna.config.Image
    Bases: object

    DEBUG = False

    crop_height_reduction_factor_in_each_iteration = 0.05

    down_sample_factor = 8

    min_image_to_crop_factor = 4

class Katna.config.ImageSelector
    Bases: object

    brightness_step = 2.0

    entropy_step = 0.5

    max_brightness_value = 90.0

    max_entropy_value = 10.0

    min_brightness_value = 10.0

    min_entropy_value = 1.0

class Katna.config.MediaPipe
    Bases: object

    class AutoFlip
        Bases: object

        BLUR_AREA_OPACITY = 0.6

        BLUR_AREA_OPACITY_KEYNAME = 'BLUR_AREA_OPACITY'

        BUILD_CMD = 'run_autoflip'

        CONFIG_FILE_PBTXT = '/home/docs/checkouts/readthedocs.org/user_builds/katna/checkou

        DEFAULT_BLUR_AREA_OPACITY = 0.6

        DEFAULT_FEATURE_SIGNAL_VALUE = 'false'

        DEFAULT_MOTION_STABALIZATION_THRESHOLD = 0.5

        ENFORCE_FEATURES = {'CAR': False, 'FACE_ALL_LANDMARKS': False, 'FACE_CORE_LANDMARKS

        ENFORCE_FEATURES_KEYNAME = 'ENFORCE_FEATURES'

        MODELS_FOLDER_LOCATION = '/home/docs/checkouts/readthedocs.org/user_builds/katna/ch

        STABALIZATION_THRESHOLD = 0.5

        STABALIZATION_THRESHOLD_KEYNAME = 'STABALIZATION_THRESHOLD'

        TMP_PBTXT_FOLDER_NAME = 'temp_pbtxt'

        TMP_PBTXT_FOLDER_PATH = '/home/docs/checkouts/readthedocs.org/user_builds/katna/che
```

```

    classmethod get_conf()
        Gets the current config
        Returns dictionary containing the current config
        Return type dict

    classmethod get_pbtxt_mapping()

    classmethod set_conf(config)
        Sets the config passed
        Parameters config(dict) – The configuration to set.

class Katna.config.TextDetector
    Bases: object

    cache_subdir = 'models'

    frozen_weights = 'frozen_east_text_detection.pb'

    layerNames = ['feature_fusion/Conv_7/Sigmoid', 'feature_fusion/concat_3']

    merge_threshold = 1

    min_confidence = 0.9

    model_download_link = 'https://github.com/ooyd/frozen_east_text_detection.pb/raw/master'

class Katna.config.Video
    Bases: object

    DEBUG = False

    compression_output_file_extension = 'mp4'

    min_video_duration = 5.0

    video_compression_codec = 'libx264'

    video_compression_crf_parameter = 23

```

Helper modules

Katna.decorators module

```

class Katna.decorators.DebugDecorators
    Bases: object

    File validation decorator

    Arguments: object {[type]} – [description]

    Raises: None:

    Returns: [] – [Decorated function]

    classmethod add_optional_debug_images_for_image_module(decorated)
        Add optional debug images in image_module class if DEBUG option is True in config

class Katna.decorators.FileDecorators
    Bases: object

    File validation decorator

    Raises FileNotFoundError – File or path is incorrect

```

classmethod validate_dir_path (*decorated*)

Validate if the input path is a valid dir or location

Parameters decorated (*function*, *required*) – decorated function

Returns function if the path is valid

Return type function object

classmethod validate_file_path (*decorated*)

Validate if the input path is a valid file or location

Parameters decorated (*function*, *required*) – decorated function

Returns function if the path is valid

Return type function object

class Katna.decorators.VideoDecorators

Bases: object

File validation decorator

Arguments: object {[type]} – [description]

Raises: FileNotFoundError: [Video File is missing]

Returns: [boolean] – [if the file exists and is valid]

classmethod validate_video (*decorated*)

Validate if the input video is a valid file

Katna.decorators.exception (*logger*)

A decorator that wraps the passed in function and logs exceptions should one occur

param logger: The logging object type logger: logger

Internal modules for Video

Katna.frame_extractor module

class Katna.frame_extractor.Frame (*frame*, *sum_abs_diff*)

Bases: object

Class for storing frame ref

class Katna.frame_extractor.FrameExtractor

Bases: object

Class for extraction of key frames from video : based on sum of absolute differences in LUV colorspace from given video

extract_candidate_frames (*videopath*)

Pubic function for this module , Given and input video path This functions Returns one list of all candidate key-frames

Parameters

- **object** (*class:Object*) – base class inheritance
- **videopath** (*str*) – inputvideo path

Returns opencv.Image.Image objects

Return type list

Katna.image_selector module

class Katna.image_selector.**ImageSelector** (*pool_obj*)

Bases: object

Class for selection of best top N images from input list of images, Currently following selection method are implemented: brightness filtering, contrast/entropy filtering, clustering of frames and variance of laplacian for non blurred images selection

Parameters **object** (class:*Object*) – base class inheritance

select_best_frames (*input_key_frames*, *number_of_frames*)

[summary] Public function for Image selector class: takes list of key-frames images and number of required frames as input, returns list of filtered keyframes

Parameters

- **object** (class:*Object*) – base class inheritance
- **input_key_frames** (*python list opencv images*) – list of input keyframes in list of opencv image format
- **number_of_frames** – Required number of images

Type int

Returns Returns list of filtered image files

Return type python list of images

Katna.video_compressor module

class Katna.video_compressor.**VideoCompressor**

Bases: object

Class for performing video compression task: based on ffmpeg library for doing video compression

compress_video (*file_path*, *force_overwrite*, *crf_parameter*, *output_video_codec*, *out_dir_path*, *out_file_name*)

Function to compress given input file

Parameters

- **file_path** (*str*) – Input file path
- **force_overwrite** (*bool*, *optional*) – optional parameter if True then if there is already a file in output file location function will overwrite it, defaults to False
- **crf_parameter** (*int*, *optional*) – Constant Rate Factor Parameter for controlling amount of video compression to be applied, The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. It is recommend to keep this value between 20 to 30 A lower value is a higher quality, you can change default value by changing config.Video.video_compression_crf_parameter
- **output_video_codec** (*str*, *optional*) – Type of video codec to choose, Currently supported options are libx264 and libx265, libx264 is default option. libx264 is more widely supported on different operating systems and platforms, libx265 uses more advanced x265 codec and results in better compression and even less output

video sizes with same or better quality. Right now libx265 is not as widely compatible on older versions of MacOS and Windows by default. If wider video compatibility is your goal you should use libx264., you can change default value by changing `Katna.config.Video.video_compression_codec`

- **out_dir_path** (*str, optional*) – output folder path where you want output video to be saved, defaults to “”
- **out_file_name** (*str, optional*) – output filename, if not mentioned it will be same as input filename, defaults to “”

Raises Exception – raises `FileNotFoundError` Exception if input video file not found, also exception is raised in case output video file path already exist and `force_overwrite` is not set to `True`.

Returns Status code Returns `True` if video compression was successful else `False`

Return type `bool`

Internal modules for Image

Katna.crop_rect module

class `Katna.crop_rect.CropRect` (*x, y, w, h*)

Bases: `object`

Data structure class for storing image crop rectangles

Parameters `object` (*class:Object*) – base class inheritance

get_image_crop (*input_image*)

public functions which for given input image and current crop rectangle object returns image cropped to crop rectangle specifications.

Parameters

- **object** (*class:Object*) – base class inheritance
- **image** (*Opencv Numpy Image*) – input image

Returns cropped image according to given spec

Return type `Opencv Numpy Image`

Katna.crop_extractor module

class `Katna.crop_extractor.CropExtractor`

Bases: `object`

Class for extraction and scoring of all possible crops from input image for input crop size. Currently features being used for scoring a given crop rectangle are: Edge, saliency and face detection. Additionally crop scoring also takes following metrics into account: Distance of pixel in crop rectangle from crop boundary and rule of third.

extract_candidate_crops (*inputImage, crop_width, crop_height, feature_list*)

Public Function for `crop_extractor` module, Given input image and desired crop width and height: function returns list of all candidate crop rectangles scored taking into account input Image Feature list

Parameters

- **object** (class:*Object*) – base class inheritance
- **inputImage** (*opencv. Image*) – input Image
- **crop_width** (*Int*) – input crop width
- **crop_height** (*Int*) – input crop height
- **feature_list** (*List of OpenCV numpy image type*) – list of input feature maps to be used for scoring a crop rectangle

Returns List of extracted crop rectangle objects

Return type list of crop_rect

Katna.crop_selector module

class Katna.crop_selector.**CropSelector**

Bases: object

Class for selection of top n crop rectangles from input list of crop rectangles ,It also implements filtering functionality, currently following filtering method are implemented: Text Detector filter

Parameters **object** (class:*Object*) – base class inheritance

select_candidate_crops (*input_image, num_of_crops, input_crop_list, defined_filters, filters=[]*)

Public Function for CropSelector class: takes list of crop rectangles and number of required crops as input and returns list of filtered crop rectangles as output. Also takes list of filters to be used for filtering out unwanted crop rectangles as optional input.

Parameters

- **object** (class:*Object*) – base class inheritance
- **input_image** (*Opencv Numpy Image*) – input image
- **input_crop_list** (*python list crop_rect data structure*) – list of input crop in list of crop_rect format
- **number_of_crop** – Required number of crop

Type int

Returns Returns list of filtered crop_rect

Return type python list of crop_rect data structure

Image Filters for Smart Cropping

Katna.image_filters.filter

class Katna.image_filters.filter.**Filter** (*weight*)

Bases: object

Base Class for Image filters

get_weight ()

gets weight of particular filter

Returns weight of the filter

Return type float

Katna.image_filters.text_detector

class Katna.image_filters.text_detector.**TextDetector** (*weight=1.0*)

Bases: Katna.image_filters.filter.Filter

TextDetector Class: Class for implementation of text detector filter, inherit from Filter class

Constructor for this class does following tasks, if not already downloaded , it first downloads text detector dnn weights file from public URL and save it at USER_HOME/.katna directory, or /tmp/.katna directory. After this initializer code initializes internal parameter: min_confidence (for text detection)

download_data ()

Public function for downloading the network weight from the URL link, to be used for text detection functionality. Troubleshooting tip: If you get FileNotFoundError during text detector initialization, initialize the text detector and call this function directly to download the model file from public URL link.

get_filter_result (*crop*)

Main public function of TextDetector filter class, this filter Returns false if crop contains no text, additionally checks for overlap between input crop rectangle and the detected text bounding box, returns True if No overlap (Filter will not discard input crop) otherwise returns False (signal for discarding input crop).

Parameters *crop* (*crop_rect*) – input crop rectangle to test

Returns True if No overlap (Filter will not discard input crop) otherwise returns False

Return type bool

set_image (*image*)

Public set_image function, This will detect all text boxes in input image and will saves them as internal list of text_rect to be used in get_filter_result

Parameters *image* (*numpy array (opencv)*) – input image from which needs to be cropped

Image Features for Smart Cropping

Katna.image_features.feature

class Katna.image_features.feature.**Feature** (*weight*)

Bases: object

Base Class: Base class for all Image features: To be used for Katna crop rectangle scoring, all image features inherit from this base class

get_weight ()

gets weight of particular feature

Returns weight of the feature

Return type float

Katna.image_features.edge_feature

class Katna.image_features.edge_feature.**EdgeFeature** (*weight=0.0*)

Bases: Katna.image_features.feature.Feature

Class for getting edge detector Feature, Constructor Parameters:- feature weight

Internal Parameters:-

`min_val_threshold` : min edge threshold

`max_val_threshold` : max edge threshold

`ksize`: size of Sobel kernel used for finding image gradients

`get_feature_map` (*image*)

Public function for getting Feature map image from edges detection

Parameters **`image`** (*numpy array*) – input image

Returns single channel opencv numpy image with feature map from edge detection

Return type numpy array

Katna.image_features.saliency_feature

class `Katna.image_features.saliency_feature.SaliencyFeature` (*weight=0.0*)

Bases: `Katna.image_features.feature.Feature`

Class for calculating saliency feature map from Input image

`get_feature_map` (*image*)

Public function for getting Feature map image from Image saliency detection

Parameters **`image`** (*numpy array*) – input image

Returns single channel opencv numpy image with feature map from saliency detection

Return type numpy array

Katna.image_features.face_feature

class `Katna.image_features.face_feature.FaceFeature` (*weight=1.0*)

Bases: `Katna.image_features.feature.Feature`

Class for calculating Face detection feature map from input image

Internal Parameters

`model_file` : Path for downloading model for face detection

`prototxt_file` : Path for downloading model description prototxt file

`confidence` : Face detection confidence threshold value

Constructor for this class does following tasks, if not already downloaded , it first downloads face detector model file and prototxt file from public URL and save it at `USER_HOME/.katna` directory, or `/tmp/.katna` directory. After this initializer code initializes internal parameter: `min_confidence` (for face detection)

`download_model` ()

Public function for downloading the network weight from the URL link, to be used for face detection functionality. Troubleshooting tip: If you get `FileNotFound` error during face detector initialization, initialize the face detector and call this function directly to download the model file from public URL link.

`download_proto` ()

Public function for downloading the network weight from the URL link, to be used for face detection functionality. Troubleshooting tip: If you get `FileNotFound` error during face detector initialization, initialize the face detector and call this function directly to download the model file from public URL link.

`get_feature_map` (*image*)

Public function for getting Feature map image from Face detection in input Image

Parameters **image** (*numpy array*) – input image

Returns single channel opencv numpy image with feature map from Face detection

Return type numpy array

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

k

`Katana.image_features.face_feature` (*OS X*), [39](#)
`Katna.config` (*Platform Independent*), [31](#)
`Katna.crop_extractor` (*OS X*), [36](#)
`Katna.crop_rect` (*OS X*), [36](#)
`Katna.crop_selector` (*OS X*), [37](#)
`Katna.decorators` (*OS X*), [33](#)
`Katna.frame_extractor` (*OS X*), [34](#)
`Katna.image` (*OS X*), [28](#)
`Katna.image_features.edge_feature` (*OS X*), [38](#)
`Katna.image_features.feature` (*Cross Platform*), [38](#)
`Katna.image_features.saliency_feature` (*Cross Platform*), [39](#)
`Katna.image_filters.filter` (*Cross Platform*), [37](#)
`Katna.image_filters.text_detector` (*OS X*), [38](#)
`Katna.image_selector` (*OS X*), [35](#)
`Katna.video` (*OS X*), [26](#)
`Katna.video_compressor` (*OS X*), [35](#)

K

`Katana.image_features.face_feature (module)`, 39

`Katna.config (module)`, 31

`Katna.crop_extractor (module)`, 36

`Katna.crop_rect (module)`, 36

`Katna.crop_selector (module)`, 37

`Katna.decorators (module)`, 33

`Katna.frame_extractor (module)`, 34

`Katna.image (module)`, 28

`Katna.image_features.edge_feature (module)`, 38

`Katna.image_features.feature (module)`, 38

`Katna.image_features.saliency_feature (module)`, 39

`Katna.image_filters.filter (module)`, 37

`Katna.image_filters.text_detector (module)`, 38

`Katna.image_selector (module)`, 35

`Katna.video (module)`, 26

`Katna.video_compressor (module)`, 35