faceswap

Release 0.99

Apr 20, 2021

Contents:

CHAPTER 1

faceswap

1.1 lib package

The lib package holds core functionality used throughout Faceswap.

1.1.1 align package

The align Package handles detected faces, their alignments and masks.

Contents

aligned_face module

Handles aligned faces and corresponding pose estimates

Module Summary

`AlignedFace`	Class to align a face.
`get_matrix_scaling`	Given a matrix, return the cv2 Interpolation method and inverse interpolation method for applying the matrix on an image.

Continued on next page

Table 1 – continued from previous page

PoseEstimate	Estimates pose from a generic 3D head model for the given 2D face landmarks.
transform_image	Perform transformation on an image, applying the given size and padding to the matrix.

Module

Aligner for faceswap.py

class lib.align.aligned_face.**AlignedFace**(*landmarks*, *image=None*, *centering='face'*, *size=64*, *coverage_ratio=1.0*, *dtype=None*, *is_aligned=False*)

Bases: `object`

Class to align a face.

Holds the aligned landmarks and face image, as well as associated matrices and information about an aligned face.

Parameters

- **landmarks** (`numpy.ndarray`) – The original 68 point landmarks that pertain to the given image for this face
- **image** (`numpy.ndarray`, optional) – The original frame that contains the face that is to be aligned. Pass *None* if the aligned face is not to be generated, and just the co-ordinates should be calculated.
- **centering** (*`["legacy", "face", "head"], optional`*) – The type of extracted face that should be loaded. "legacy" places the nose in the center of the image (the original method for aligning). "face" aligns for the nose to be in the center of the face (top to bottom) but the center of the skull for left to right. "head" aligns for the center of the skull (in 3D space) being the center of the extracted image, with the crop holding the full head. Default: *"face"*
- **size** (*`int, optional`*) – The size in pixels, of each edge of the final aligned face. Default: *64*
- **coverage_ratio** (*`float, optional`*) – The amount of the aligned image to return. A ratio of 1.0 will return the full contents of the aligned image. A ratio of 0.5 will return an image of the given size, but will crop to the central 50%% of the image.
- **dtype** (*`str, optional`*) – Set a data type for the final face to be returned as. Passing `None` will return a face with the same data type as the original `image`. Default: `None`
- **is_aligned_face** (*`bool, optional`*) – Indicates that the `image` is an aligned face rather than a frame. Default: `False`

adjusted_matrix

The 3x2 transformation matrix for extracting and aligning the core face area out of the original frame with padding and sizing applied.

Type `numpy.ndarray`

extract_face(*image*)

Extract the face from a source image and populate *face*. If an image is not provided then `None` is returned.

Parameters **image** (`numpy.ndarray` or `None`) – The original frame to extract the face from. `None` if the face should not be extracted

Returns The extracted face at the given size, with the given coverage of the given dtype or `None` if no image has been provided.

Return type `numpy.ndarray` or `None`

face
The aligned face at the given *size* at the specified `coverage` in the given `dtype`. If an `image` has not been provided then an the attribute will return `None`.

Type `numpy.ndarray`

get_cropped_roi(*centering*)
Obtain the region of interest within an aligned face set to centered coverage for an alternative centering

Parameters **centering** (*["legacy", "face"]*) – The type of centering to obtain the region of interest for. "legacy" places the nose in the center of the image (the original method for aligning). "face" aligns for the nose to be in the center of the face (top to bottom) but the center of the skull for left to right.

Returns The (*left*, *top*, *right*, *bottom* location of the region of interest within an aligned face centered on the head for the given centering

Return type `numpy.ndarray`

interpolators
(*interpolator* and *reverse interpolator*) for the `adjusted matrix`.

Type tuple

landmarks
The 68 point facial landmarks aligned to the extracted face box.

Type `numpy.ndarray`

matrix
The 3x2 transformation matrix for extracting and aligning the core face area out of the original frame, with no padding or sizing applied. The returned matrix is offset for the given `centering`.

Type `numpy.ndarray`

original_roi
The location of the extracted face box within the original frame.

Type `numpy.ndarray`

padding
The amount of padding (in pixels) that is applied to each side of the extracted face image for the selected extract type.

Type int

pose
The estimated pose in 3D space.

Type `lib.align.PoseEstimate`

size
The size (in pixels) of one side of the square extracted face image.

Type int

transform_points(*points*, *invert=False*)
Perform transformation on a series of (x, y) co-ordinates in world space into aligned face space.

Parameters

- **points** (numpy.ndarray) – The points to transform
- **invert** (*bool, optional*) – True to reverse the transformation (i.e. transform the points into world space from aligned face space). Default: False

Returns The transformed points

Return type numpy.ndarray

class lib.align.aligned_face.**PoseEstimate**(*landmarks*)
Bases: object

Estimates pose from a generic 3D head model for the given 2D face landmarks.

Parameters **landmarks** (numpy.ndarry) – The original 68 point landmarks aligned to 0.0 - 1.0 range

References

Head Pose Estimation using OpenCV and Dlib - https://www.learnopencv.com/tag/solvepnp/ 3D Model points - http://aifi.isr.uc.pt/Downloads/OpenGL/glAnthropometric3DModel.cpp

offset
The amount to offset a standard 0.0 - 1.0 umeyama transformation matrix for a from the center of the face (between the eyes) or center of the head (middle of skull) rather than the nose area.

Type dict

pitch
The pitch of the aligned face in eular angles

Type float

xyz_2d
numpy.ndarray projected (x, y) coordinates for each x, y, z point at a constant distance from adjusted center of the skull (0.5, 0.5) in the 2D space.

yaw
The yaw of the aligned face in eular angles

Type float

lib.align.aligned_face.**get_centered_size**(*source_centering*, *target_centering*, *size*)
Obtain the size of a cropped face from an aligned image.

Given an image of a certain dimensions, returns the dimensions of the sub-crop within that image for the requested centering.

Notes

"legacy" places the nose in the center of the image (the original method for aligning). *"face"* aligns for the nose to be in the center of the face (top to bottom) but the center of the skull for left to right. *"head"* places the center in the middle of the skull in 3D space.

The ROI in relation to the source image is calculated by rounding the padding of one side to the nearest integer then applying this padding to the center of the crop, to ensure that any dimensions always have an even number of pixels.

Parameters

- **source_centering** (*["head", "face", "legacy"]*) – The centering that the original image is aligned at

- **target_centering** (*["head", "face", "legacy"]*) – The centering that the sub-crop size should be obtained for
- **size** (*int*) – The size of the source image to obtain the cropped size for

Returns The pixel size of a sub-crop image from a full head aligned image

Return type int

lib.align.aligned_face.**get_matrix_scaling**(*matrix*)
Given a matrix, return the cv2 Interpolation method and inverse interpolation method for applying the matrix on an image.

Parameters **matrix** (numpy.ndarray) – The transform matrix to return the interpolator for

Returns The interpolator and inverse interpolator for the given matrix. This will be (Cubic, Area) for an upscale matrix and (Area, Cubic) for a downscale matrix

Return type tuple

lib.align.aligned_face.**transform_image**(*image*, *matrix*, *size*, *padding=0*)
Perform transformation on an image, applying the given size and padding to the matrix.

Parameters

- **image** (numpy.ndarray) – The image to transform
- **matrix** (numpy.ndarray) – The transformation matrix to apply to the image
- **size** (*int*) – The final size of the transformed image
- **padding** (*int, optional*) – The amount of padding to apply to the final image. Default: *0*

Returns The transformed image

Return type numpy.ndarray

alignments module

Handles alignments stored in a serialized alignments.fsa file

Module Summary

Alignments	The alignments file is a custom serialized .fsa file that holds information for each frame for a video or series of images.
Thumbnails	Thumbnail images stored in the alignments file.

Module

Alignments file functions for reading, writing and manipulating the data stored in a serialized alignments file.

class lib.align.alignments.**Alignments**(*folder*, *filename='alignments'*)
Bases: object

The alignments file is a custom serialized .fsa file that holds information for each frame for a video or series of images.

Specifically, it holds a list of faces that appear in each frame. Each face contains information detailing their

detected bounding box location within the frame, the 68 point facial landmarks and any masks that have been extracted.

Additionally it can also hold video meta information (timestamp and whether a frame is a key frame.)

Parameters

- **folder** (*str*) – The folder that contains the alignments `.fsa` file
- **filename** (*str, optional*) – The filename of the `.fsa` alignments file. If not provided then the given folder will be checked for a default alignments file filename. Default: "alignments"

add_face (*frame_name*, *face*)
Add a new face for the given frame_name in *data* and return it's index.

Parameters

- **frame_name** (*str*) – The frame name to add the face to. This should be the base name of the frame, not the full path
- **face** (*dict*) – The face information to add to the given frame_name, correctly formatted for storing in *data*

Returns The index of the newly added face within *data* for the given frame_name

Return type int

backup ()
Create a backup copy of the alignments *file*.

Creates a copy of the serialized alignments *file* appending a timestamp onto the end of the file name and storing in the same folder as the original *file*.

data
The loaded alignments *file* in dictionary form.

Type dict

delete_face_at_index (*frame_name*, *face_index*)
Delete the face for the given frame_name at the given face index from *data*.

Parameters

- **frame_name** (*str*) – The frame name to remove the face from. This should be the base name of the frame, not the full path
- **face_index** (*int*) – The index number of the face within the given frame_name to remove

Returns `True` if a face was successfully deleted otherwise `False`

Return type bool

faces_count
The total number of faces that appear in the alignments *data*.

Type int

file
The full path to the currently loaded alignments file.

Type str

filter_faces (*filter_dict*, *filter_out=False*)
Remove faces from *data* based on a given filter list.

Parameters

- **filter_dict** (*dict*) – Dictionary of source filenames as key with a list of face indices to filter as value.
- **filter_out** (*bool, optional*) – `True` if faces should be removed from *data* when there is a corresponding match in the given filter_dict. `False` if faces should be kept in *data* when there is a corresponding match in the given filter_dict, but removed if there is no match. Default: `False`

frame_exists(*frame_name*)

Check whether a given frame_name exists within the alignments *data*.

Parameters **frame_name** (*str*) – The frame name to check. This should be the base name of the frame, not the full path

Returns `True` if the given frame_name exists within the alignments *data* otherwise `False`

Return type bool

frame_has_faces(*frame_name*)

Check whether a given frame_name exists within the alignments *data* and contains at least 1 face.

Parameters **frame_name** (*str*) – The frame name to check. This should be the base name of the frame, not the full path

Returns `True` if the given frame_name exists within the alignments *data* and has at least 1 face associated with it, otherwise `False`

Return type bool

frame_has_multiple_faces(*frame_name*)

Check whether a given frame_name exists within the alignments *data* and contains more than 1 face.

Parameters **frame_name** (*str*) – The frame_name name to check. This should be the base name of the frame, not the full path

Returns `True` if the given frame_name exists within the alignments *data* and has more than 1 face associated with it, otherwise `False`

Return type bool

frames_count

The number of frames that appear in the alignments *data*.

Type int

get_faces_in_frame(*frame_name*)

Obtain the faces from *data* associated with a given frame_name.

Parameters **frame_name** (*str*) – The frame name to return faces for. This should be the base name of the frame, not the full path

Returns The list of face dictionaries that appear within the requested frame_name

Return type list

hashes_to_alignment

The SHA1 hash of the face mapped to the alignment for the face that the hash corresponds to. The structure of the dictionary is:

Notes

This method is depractated and exists purely for updating legacy hash based alignments to new png header storage in `lib.align.update_legacy_png_header`.

The first time this property is referenced, the dictionary will be created and cached. Subsequent references will be made to this cached dictionary.

Type dict

hashes_to_frame
The SHA1 hash of the face mapped to the frame(s) and face index within the frame that the hash corresponds to. The structure of the dictionary is:

{**SHA1_hash** (*str*): {**filename** (*str*): **face_index** (*int*)}}.

Notes

This method is depractated and exists purely for updating legacy hash based alignments to new png header storage in `lib.align.update_legacy_png_header`.

The first time this property is referenced, the dictionary will be created and cached. Subsequent references will be made to this cached dictionary.

Type dict

have_alignments_file
`True` if an alignments file exists at location *file* otherwise `False`.

Type bool

mask_is_valid(*mask_type*)
Ensure the given `mask_type` is valid for the alignments *data*.

Every face in the alignments *data* must have the given mask type to successfully pass the test.

Parameters **mask_type** (*str*) – The mask type to check against the current alignments *data*

Returns `True` if all faces in the current alignments possess the given `mask_type` otherwise `False`

Return type bool

mask_summary
The mask type names stored in the alignments *data* as key with the number of faces which possess the mask type as value.

Type dict

save()
Write the contents of *data* and `_meta` to a serialized `.fsa` file at the location *file*.

save_video_meta_data(*pts_time*, *keyframes*)
Save video meta data to the alignments file.

If the alignments file does not have an entry for every frame (e.g. if Extract Every N was used) then the frame is added to the alignments file with no faces, so that they video meta data can be stored.

Parameters

- **pts_time** (*list*) – A list of presentation timestamps (*float*) in frame index order for every frame in the input video

- **keyframes** (*list*) – A list of frame indices corresponding to the key frames in the input video

thumbnails
The low resolution thumbnail images that exist within the alignments file

Type Thumbnails

update_face(*frame_name*, *face_index*, *face*)
Update the face for the given frame_name at the given face index in *data*.

Parameters

- **frame_name** (*str*) – The frame name to update the face for. This should be the base name of the frame, not the full path
- **face_index** (*int*) – The index number of the face within the given frame_name to update
- **face** (*dict*) – The face information to update to the given frame_name at the given face_index, correctly formatted for storing in *data*

version
The alignments file version number.

Type float

video_meta_data
The frame meta data stored in the alignments file. If data does not exist in the alignments file then None is returned for each Key

Type dict

yield_faces()
Generator to obtain all faces with meta information from *data*. The results are yielded by frame.

Notes

The yielded order is non-deterministic.

Yields

- **frame_name** (*str*) – The frame name that the face belongs to. This is the base name of the frame, as it appears in *data*, not the full path
- **faces** (*list*) – The list of face *dict* objects that exist for this frame
- **face_count** (*int*) – The number of faces that exist within *data* for this frame
- **frame_fullname** (*str*) – The full path (folder and filename) for the yielded frame

class lib.align.alignments.**Thumbnails**(*alignments*)
Bases: object

Thumbnail images stored in the alignments file.

The thumbnails are stored as low resolution (64px), low quality jpg in the alignments file and are used for the Manual Alignments tool.

Parameters **alignments** (*:class:'~lib.align.Alignments`*) – The parent alignments class that these thumbs belong to

add_thumbnail(*frame*, *face_index*, *thumb*)
Add a thumbnail for the given face index for the given frame.

Parameters

- **frame** (*str*) – The name of the frame to add the thumbnail for
- **face_index** (*int*) – The face index within the given frame to add the thumbnail for
- **thumb** (numpy.ndarray) – The encoded jpg thumbnail at 64px to add to the alignments file

get_thumbnail_by_index (*frame_index*, *face_index*)
Obtain a jpg thumbnail from the given frame index for the given face index

Parameters

- **frame_index** (*int*) – The frame index that contains the thumbnail
- **face_index** (*int*) – The face index within the frame to retrieve the thumbnail for

Returns The encoded jpg thumbnail

Return type numpy.ndarray

has_thumbnails
True if all faces in the alignments file contain thumbnail images otherwise False.

Type bool

detected_face module

Handles detected face objects and their associated masks.

Module Summary

BlurMask	Factory class to return the correct blur object for requested blur type.
DetectedFace	Detected face and landmark information
Mask	Face Mask information and convenience methods
update_legacy_png_header	Update a legacy extracted face from pre v2.1 alignments by placing the alignment data for the face in the png exif header for the given filename with the given alignment data.

Module

Face and landmarks detection for faceswap.py

class lib.align.detected_face.**BlurMask** (*blur_type*, *mask*, *kernel*, *is_ratio=False*, *passes=1*)
Bases: object

Factory class to return the correct blur object for requested blur type.

Works for square images only. Currently supports Gaussian and Normalized Box Filters.

Parameters

- **blur_type** (*["gaussian", "normalized"]*) – The type of blur to use
- **mask** (numpy.ndarray) – The mask to apply the blur to

- **kernel** (*int or float*) – Either the kernel size (in pixels) or the size of the kernel as a ratio of mask size
- **is_ratio** (*bool, optional*) – Whether the given `kernel` parameter is a ratio or not. If `True` then the actual kernel size will be calculated from the given ratio and the mask size. If `False` then the kernel size will be set directly from the `kernel` parameter. Default: `False`
- **passes** (*int, optional*) – The number of passes to perform when blurring. Default: `1`

Example

```
>>> print(mask.shape)
(128, 128, 1)
>>> new_mask = BlurMask("gaussian", mask, 3, is_ratio=False, passes=1).blurred
>>> print(new_mask.shape)
(128, 128, 1)
```

blurred
The final mask with blurring applied.

Type `numpy.ndarray`

class lib.align.detected_face.**DetectedFace**(*image=None*, *x=None*, *w=None*, *y=None*, *h=None*, *landmarks_xy=None*, *mask=None*, *filename=None*)

Bases: `object`

Detected face and landmark information

Holds information about a detected face, it's location in a source image and the face's 68 point landmarks.

Methods for aligning a face are also callable from here.

Parameters

- **image** (*numpy.ndarray, optional*) – Original frame that holds this face. Optional (not required if just storing coordinates)
- **x** (*int*) – The left most point (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`
- **w** (*int*) – The width (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`
- **y** (*int*) – The top most point (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`
- **h** (*int*) – The height (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`
- **landmarks_xy** (*list*) – The 68 point landmarks as discovered in `plugins.extract.align`. Should be a `list` of 68 *(x, y)* `tuples` with each of the landmark co-ordinates.
- **mask** (*dict*) – The generated mask(s) for the face as generated in `plugins.extract.mask`. Must be a dict of {**name** (*str*): *Mask*}.

image
This is a generic image placeholder that should not be relied on to be holding a particular image. It may

hold the source frame that holds the face, a cropped face or a scaled image depending on the method using this object.

Type numpy.ndarray, optional

x
The left most point (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`

Type int

w
The width (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`

Type int

y
The top most point (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`

Type int

h
The height (in pixels) of the face's bounding box as discovered in `plugins.extract.detect`

Type int

landmarks_xy
The 68 point landmarks as discovered in `plugins.extract.align`.

Type list

mask
The generated mask(s) for the face as generated in `plugins.extract.mask`. Is a dict of {**name** (*str*): *Mask*}.

Type dict

add_mask(*name*, *mask*, *affine_matrix*, *interpolator*, *storage_size=128*)
Add a *Mask* to this detected face

The mask should be the original output from `plugins.extract.mask` If a mask with this name already exists it will be overwritten by the given mask.

Parameters

- **name** (*str*) – The name of the mask as defined by the `plugins.extract.mask._base.name` parameter.
- **mask** (*numpy.ndarray*) – The mask that is to be added as output from `plugins.extract.mask` It should be in the range 0.0 - 1.0 ideally with a `dtype` of `float32`
- **affine_matrix** (*numpy.ndarray*) – The transformation matrix required to transform the mask to the original frame.
- **int** (*interpolator,*) – The CV2 interpolator required to transform this mask to it's original frame.
- **int (optional)** (*storage_size,*) – The size the mask is to be stored at. Default: 128

bottom
Bottom point (in pixels) of face detection bounding box within the parent image

Type int

from_alignment(*alignment*, *image=None*, *with_thumb=False*)
Set the attributes of this class from an alignments file and optionally load the face into the `image` attribute.

Parameters

- **alignment** (*dict*) – A dictionary entry for a face from an alignments file containing the keys `x`, `w`, `y`, `h`, `landmarks_xy`. Optionally the key `thumb` will be provided. This is for use in the manual tool and contains the compressed jpg thumbnail of the face to be allocated to `thumbnail. Optionally the key ``mask` will be provided, but legacy alignments will not have this key.
- **image** (*numpy.ndarray, optional*) – If an image is passed in, then the `image` attribute will be set to the cropped face based on the passed in bounding box co-ordinates
- **with_thumb** (*bool, optional*) – Whether to load the jpg thumbnail into the detected face object, if provided. Default: `False`

from_png_meta(*alignment*)

Set the attributes of this class from alignments stored in a png exif header.

Parameters **alignment** (*dict*) – A dictionary entry for a face from alignments stored in a png exif header containing the keys `x`, `w`, `y`, `h`, `landmarks_xy` and `mask`

get_landmark_mask(*size*, *area*, *aligned=True*, *centering='head'*, *dilation=0*, *blur_kernel=0*, *as_zip=False*)

Obtain a single channel mask based on the face's landmark points.

Parameters

- **size** (*int or tuple*) – The size of the aligned mask to retrieve. Should be an *int* if an aligned face is being requested, or a ('height', 'width') shape tuple if a full frame is being requested
- **area** (*["mouth", "eyes"]*) – The type of mask to obtain. *face* is a full face mask the others are masks for those specific areas
- **aligned** (*bool, optional*) – `True` if the returned mask should be for an aligned face. `False` if a full frame mask should be returned. Default `True`
- **centering** (*["legacy", "face", "head"], optional*) – Only used if *aligned*=*'True'*. The centering for the landmarks based mask. Should be the same as the centering used for the extracted face that this mask will be applied to. "legacy" places the nose in the center of the image (the original method for aligning). "face" aligns for the nose to be in the center of the face (top to bottom) but the center of the skull for left to right. "head" aligns for the center of the skull (in 3D space) being the center of the extracted image, with the crop holding the full head. Default: *"face"*
- **dilation** (*int, optional*) – The amount of dilation to apply to the mask. *0* for none. Default: *0*
- **blur_kernel** (*int, optional*) – The kernel size for applying gaussian blur to apply to the mask. *0* for none. Default: *0*
- **as_zip** (*bool, optional*) – `True` if the mask should be returned zipped otherwise `False`

Returns The mask as a single channel image of the given `size` dimension. If `as_zip` is `True` then the `numpy.ndarray` will be contained within a zipped container

Return type `numpy.ndarray` or zipped array

left

Left point (in pixels) of face detection bounding box within the parent image

Type int

load_aligned(*image*, *size=256*, *dtype=None*, *centering='head'*, *force=False*)
Align a face from a given image.

Aligning a face is a relatively expensive task and is not required for all uses of the `DetectedFace` object, so call this function explicitly to load an aligned face.

This method plugs into `lib.align.AlignedFace` to perform face alignment based on this face's `landmarks_xy`. If the face has already been aligned, then this function will return having performed no action.

Parameters

- **image** (*numpy.ndarray*) – The image that contains the face to be aligned
- **size** (*int*) – The size of the output face in pixels
- **dtype** (*str, optional*) – Optionally set a `dtype` for the final face to be formatted in. Default: `None`
- **centering** (*["legacy", "face", "head"], optional*) – The type of extracted face that should be loaded. "legacy" places the nose in the center of the image (the original method for aligning). "face" aligns for the nose to be in the center of the face (top to bottom) but the center of the skull for left to right. "head" aligns for the center of the skull (in 3D space) being the center of the extracted image, with the crop holding the full head. Default: *"head"*
- **force** (*bool, optional*) – Force an update of the aligned face, even if it is already loaded. Default: `False`

Notes

This method must be executed to get access to the following an `AlignedFace` object

right
Right point (in pixels) of face detection bounding box within the parent image

Type int

to_alignment()
Return the detected face formatted for an alignments file

Returns alignment – The alignment dict will be returned with the keys `x`, `w`, `y`, `h`, `landmarks_xy`, `mask`. The additional key `thumb` will be provided if the detected face object contains a thumbnail.

Return type dict

to_png_meta()
Return the detected face formatted for insertion into a png itxt header.

returns: dict The alignments dict will be returned with the keys `x`, `w`, `y`, `h`, `landmarks_xy` and `mask`

top
Top point (in pixels) of face detection bounding box within the parent image

Type int

class lib.align.detected_face.**Mask**(*storage_size=128*)
Bases: `object`

Face Mask information and convenience methods

Holds a Faceswap mask as generated from `plugins.extract.mask` and the information required to transform it to its original frame.

Holds convenience methods to handle the warping, storing and retrieval of the mask.

Parameters **storage_size** (*int, optional*) – The size (in pixels) that the mask should be stored at. Default: 128.

stored_size
The size, in pixels, of the stored mask across its height and width.

Type int

add(*mask*, *affine_matrix*, *interpolator*)
Add a Faceswap mask to this *Mask*.

The mask should be the original output from `plugins.extract.mask`

Parameters

- **mask** (*numpy.ndarray*) – The mask that is to be added as output from `plugins.extract.mask` It should be in the range 0.0 - 1.0 ideally with a `dtype` of `float32`
- **affine_matrix** (*numpy.ndarray*) – The transformation matrix required to transform the mask to the original frame.
- **int** (*interpolator,*) – The CV2 interpolator required to transform this mask to it's original frame

affine_matrix
numpy.ndarray: The affine matrix to transpose the mask to a full frame.

Type class

from_dict(*mask_dict*)
Populates the *Mask* from a dictionary loaded from an alignments file.

Parameters **mask_dict** (*dict*) – A dictionary stored in an alignments file containing the keys `mask`, `affine_matrix`, `interpolator`, `stored_size`

get_full_frame_mask(*width*, *height*)
Return the stored mask in a full size frame of the given dimensions

Parameters

- **width** (*int*) – The width of the original frame that the mask was extracted from
- **height** (*int*) – The height of the original frame that the mask was extracted from

Returns **numpy.ndarray**

Return type The mask affined to the original full frame of the given dimensions

interpolator
The cv2 interpolator required to transpose the mask to a full frame.

Type int

mask
The mask at the size of *stored_size* with any requested blurring and threshold amount applied.

Type numpy.ndarray

original_roi
numpy.ndarray: The original region of interest of the mask in the source frame.

Type class

replace_mask(*mask*)
Replace the existing _mask with the given mask.

Parameters **mask** (*numpy.ndarray*) – The mask that is to be added as output from plugins.extract.mask. It should be in the range 0.0 - 1.0 ideally with a dtype of float32

set_blur_and_threshold(*blur_kernel=0*, *blur_type='gaussian'*, *blur_passes=1*, *threshold=0*)
Set the internal blur kernel and threshold amount for returned masks

Parameters

- **blur_kernel** (*int, optional*) – The kernel size, in pixels to apply gaussian blurring to the mask. Set to 0 for no blurring. Should be odd, if an even number is passed in (outside of 0) then it is rounded up to the next odd number. Default: 0
- **blur_type** (*["gaussian", "normalized"], optional*) – The blur type to use. gaussian or normalized box filter. Default: gaussian
- **blur_passes** (*int, optional*) – The number of passed to perform when blurring. Default: 1
- **threshold** (*int, optional*) – The threshold amount to minimize/maximize mask values to 0 and 100. Percentage value. Default: 0

set_sub_crop(*offset*)
Set the internal crop area of the mask to be returned.

This impacts the returned mask from *mask* if the requested mask is required for different face centering than what has been stored.

Parameters **offset** (numpy.ndarray) – The (x, y) offset from the center point to return the mask for

Notes

All masks are currently stored with *face* centering and all crops are for 'legacy' centering. This may change in future

to_dict()
Convert the mask to a dictionary for saving to an alignments file

Returns The *Mask* for saving to an alignments file. Contains the keys mask, affine_matrix, interpolator, stored_size

Return type dict

to_png_meta()
Convert the mask to a dictionary supported by png itxt headers.

Returns The *Mask* for saving to an alignments file. Contains the keys mask, affine_matrix, interpolator, stored_size

Return type dict

lib.align.detected_face.**update_legacy_png_header**(*filename*, *alignments*)
Update a legacy extracted face from pre v2.1 alignments by placing the alignment data for the face in the png exif header for the given filename with the given alignment data.

If the given file is not a .png then a png is created and the original file is removed

Parameters

- **filename** (*str*) – The image file to update
- **alignments** (*lib.align.alignments.Alignments*) – The alignments data the contains the information to store in the image header. This must be a v2.0 or less alignments file as later versions no longer store the face hash (unrequired)

Returns The metadata that has been applied to the given image

Return type dict

1.1.2 cli package

The CLI Package handles the Command Line Arguments that act as the entry point into Faceswap.

Contents

args module

Module Summary

ConvertArgs	Creates the command line arguments for conversion.
ExtractArgs	Creates the command line arguments for extraction.
ExtractConvertArgs	Parent class to capture arguments that will be used in both extract and convert processes.
FaceSwapArgs	Faceswap argument parser functions that are universal to all commands.
FullHelpArgumentParser	Extends `argparse.ArgumentParser` to output full help on bad arguments.
GuiArgs	Creates the command line arguments for the GUI.
SmartFormatter	Extends the class `argparse.HelpFormatter` to allow custom formatting in help text.
TrainArgs	Creates the command line arguments for training.

Module

The Command Line Argument options for faceswap.py

class lib.cli.args.**ConvertArgs**(*subparser*, *command*, *description='default'*)

Bases: *lib.cli.args.ExtractConvertArgs*

Creates the command line arguments for conversion.

This class inherits base options from *ExtractConvertArgs* where arguments that are used for both Extract and Convert should be placed.

Commands explicit to Convert should be added in *get_optional_arguments()*

static get_info()

The information text for the Convert command.

Returns The information text for the Convert command.

Return type str

static get_optional_arguments()
Returns the argument list unique to the Convert command.

Returns The list of optional command line options for the Convert command

Return type list

class lib.cli.args.**ExtractArgs**(*subparser*, *command*, *description='default'*)
Bases: *lib.cli.args.ExtractConvertArgs*

Creates the command line arguments for extraction.

This class inherits base options from *ExtractConvertArgs* where arguments that are used for both Extract and Convert should be placed.

Commands explicit to Extract should be added in *get_optional_arguments()*

static get_info()
The information text for the Extract command.

Returns The information text for the Extract command.

Return type str

static get_optional_arguments()
Returns the argument list unique to the Extract command.

Returns The list of optional command line options for the Extract command

Return type list

class lib.cli.args.**ExtractConvertArgs**(*subparser*, *command*, *description='default'*)
Bases: *lib.cli.args.FaceSwapArgs*

Parent class to capture arguments that will be used in both extract and convert processes.

Extract and Convert share a fair amount of arguments, so arguments that can be used in both of these processes should be placed here.

No further processing is done in this class (this is handled by the children), this just captures the shared arguments.

static get_argument_list()
Returns the argument list for shared Extract and Convert arguments.

Returns The list of command line options for the given Extract and Convert

Return type list

class lib.cli.args.**FaceSwapArgs**(*subparser*, *command*, *description='default'*)
Bases: object

Faceswap argument parser functions that are universal to all commands.

This is the parent class to all subsequent argparsers which holds global arguments that pertain to all commands.

Process the incoming command line arguments, validates then launches the relevant faceswap script with the given arguments.

Parameters

- **subparser** (argparse._SubParsersAction) – The subparser for the given command

- **command** (*str*) – The faceswap command that is to be executed
- **description** (*str, optional*) – The description for the given command. Default: "default"

static get_argument_list()
Returns the argument list for the current command.

The argument list should be a list of dictionaries pertaining to each option for a command. This function should be overridden with the actual argument list for each command's argument list.

See existing parsers for examples.

Returns The list of command line options for the given command

Return type list

static get_info()
Returns the information text for the current command.

This function should be overridden with the actual command help text for each commands' parser.

Returns The information text for this command.

Return type str

static get_optional_arguments()
Returns the optional argument list for the current command.

The optional arguments list is not always required, but is used when there are shared options between multiple commands (e.g. convert and extract). Only override if required.

Returns The list of optional command line options for the given command

Return type list

class lib.cli.args.**FullHelpArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)
Bases: argparse.ArgumentParser

Extends argparse.ArgumentParser to output full help on bad arguments.

error(*message: string*)
Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

class lib.cli.args.**GuiArgs**(*subparser*, *command*, *description='default'*)
Bases: *lib.cli.args.FaceSwapArgs*

Creates the command line arguments for the GUI.

static get_argument_list()
Returns the argument list for GUI arguments.

Returns The list of command line options for the GUI

Return type list

class lib.cli.args.**SmartFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: argparse.HelpFormatter

Extends the class argparse.HelpFormatter to allow custom formatting in help text.

Adapted from: https://stackoverflow.com/questions/3853722

Notes

Prefix help text with "R|" to override default formatting and use explicitly defined formatting within the help text. Prefixing a new line within the help text with "L|" will turn that line into a list item in both the cli help text and the GUI.

class lib.cli.args.**TrainArgs**(*subparser*, *command*, *description='default'*)

Bases: *lib.cli.args.FaceSwapArgs*

Creates the command line arguments for training.

static get_argument_list()

Returns the argument list for Train arguments.

Returns The list of command line options for training

Return type list

static get_info()

The information text for the Train command.

Returns The information text for the Train command.

Return type str

actions module

Module Summary

ContextFullPaths	Adds support for context sensitive browser dialog opening in the GUI.
DirFullPaths	Adds support for a Directory browser in the GUI.
DirOrFileFullPaths	Adds support to the GUI to launch either a file browser or a folder browser.
FileFullPaths	Adds support for a File browser to select a single file in the GUI.
FilesFullPaths	Adds support for a File browser to select multiple files in the GUI.
MultiOption	Adds support for multiple option checkboxes in the GUI.
Radio	Adds support for a GUI Radio options box.
SaveFileFullPaths	Adds support for a Save File dialog in the GUI.
Slider	Adds support for a slider in the GUI.

Module

Custom argparse.Action objects for Faceswap's Command Line Interface.

The custom actions within this module allow for custom manipulation of Command Line Arguments as well as adding

a mechanism for indicating to the GUI how specific options should be rendered.

class lib.cli.actions.**ContextFullPaths**(**args*, *filetypes=None*, *action_option=None*, ***kwargs*)

Bases: *lib.cli.actions.FileFullPaths*

Adds support for context sensitive browser dialog opening in the GUI.

For some tasks, the type of action (file load, folder open, file save etc.) can vary depending on the task to be performed (a good example of this is the effmpeg tool). Using this action indicates to the GUI that the type of dialog to be launched can change depending on another option. As well as the standard parameters, the below parameters are required. NB: `nargs` are explicitly disallowed.

Parameters

- **filetypes** (*str*) – The accepted file types for this option. This is the key for the GUIs lookup table which can be found in *lib.gui.utils.FileHandler*
- **action_option** (*str*) – The command line option that dictates the context of the file dialog to be opened. Bespoke actions are set in *lib.gui.utils.FileHandler*

Example

Assuming an argument has already been set with option string *-a* indicating the action to be performed, the following will pop a different type of dialog depending on the action selected:

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--input_video"),
>>>         action=ContextFullPaths,
>>>         filetypes="video",
>>>         action_option="-a"))
```

class lib.cli.actions.**DirFullPaths**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `lib.cli.actions._FullPaths`

Adds support for a Directory browser in the GUI.

This is a standard `argparse.Action` (with stock parameters) which indicates to the GUI that a dialog box should be opened in order to browse for a folder.

No additional parameters are required.

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--folder_location"),
>>>         action=DirFullPaths)),
```

class lib.cli.actions.**DirOrFileFullPaths**(**args*, *filetypes=None*, ***kwargs*)

Bases: *lib.cli.actions.FileFullPaths*

Adds support to the GUI to launch either a file browser or a folder browser.

Some inputs (for example source frames) can come from a folder of images or from a video file. This indicates to the GUI that it should place 2 buttons (one for a folder browser, one for a file browser) for file/folder browsing.

The standard `argparse.Action` is extended with the additional parameter `filetypes`, indicating to the GUI that it should pop a file browser, and limit the results to the file types listed. As well as the standard parameters, the following parameter is required:

Parameters **`filetypes`** (*str*) – The accepted file types for this option. This is the key for the GUIs lookup table which can be found in *lib.gui.utils.FileHandler*. NB: This parameter is only used for the file browser and not the folder browser

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--input_frames"),
>>>         action=DirOrFileFullPaths,
>>>         filetypes="video))"
```

class lib.cli.actions.**FileFullPaths**(**args*, *filetypes=None*, ***kwargs*)
Bases: `lib.cli.actions._FullPaths`

Adds support for a File browser to select a single file in the GUI.

This extends the standard `argparse.Action` and adds an additional parameter `filetypes`, indicating to the GUI that it should pop a file browser for opening a file and limit the results to the file types listed. As well as the standard parameters, the following parameter is required:

Parameters **`filetypes`** (*str*) – The accepted file types for this option. This is the key for the GUIs lookup table which can be found in *lib.gui.utils.FileHandler*

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--video_location"),
>>>         action=FileFullPaths,
>>>         filetypes="video))"
```

class lib.cli.actions.**FilesFullPaths**(**args*, *filetypes=None*, ***kwargs*)
Bases: *lib.cli.actions.FileFullPaths*

Adds support for a File browser to select multiple files in the GUI.

This extends the standard `argparse.Action` and adds an additional parameter `filetypes`, indicating to the GUI that it should pop a file browser, and limit the results to the file types listed. Multiple files can be selected for opening, so the `nargs` parameter must be set. As well as the standard parameters, the following parameter is required:

Parameters **`filetypes`** (*str*) – The accepted file types for this option. This is the key for the GUIs lookup table which can be found in *lib.gui.utils.FileHandler*

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--images"),
>>>         action=FilesFullPaths,
```

(continues on next page)

(continued from previous page)

```
>>>         filetypes="image",
>>>         nargs="+"))
```

class lib.cli.actions.**MultiOption**(**args*, ***kwargs*)

Bases: argparse.Action

Adds support for multiple option checkboxes in the GUI.

This is a standard argparse.Action (with stock parameters) which indicates to the GUI that the options passed should be rendered as a group of Radio Buttons rather than a combo box.

The choices parameter must be provided as this provides the valid option choices.

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--foobar"),
>>>         action=MultiOption,
>>>         choices=["foo", "bar"))
```

class lib.cli.actions.**Radio**(**args*, ***kwargs*)

Bases: argparse.Action

Adds support for a GUI Radio options box.

This is a standard argparse.Action (with stock parameters) which indicates to the GUI that the options passed should be rendered as a group of Radio Buttons rather than a combo box.

No additional parameters are required, but the choices parameter must be provided as these will be the Radio Box options. nargs are explicitly disallowed.

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--foobar"),
>>>         action=Radio,
>>>         choices=["foo", "bar"))
```

class lib.cli.actions.**SaveFileFullPaths**(**args*, *filetypes=None*, ***kwargs*)

Bases: *lib.cli.actions.FileFullPaths*

Adds support for a Save File dialog in the GUI.

This extends the standard argparse.Action and adds an additional parameter filetypes, indicating to the GUI that it should pop a save file browser, and limit the results to the file types listed. As well as the standard parameters, the following parameter is required:

Parameters **filetypes** (*str*) – The accepted file types for this option. This is the key for the GUIs lookup table which can be found in *lib.gui.utils.FileHandler*

Example

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--video_out"),
>>>         action=SaveFileFullPaths,
>>>         filetypes="video"))
```

class lib.cli.actions.**Slider**(**args*, *min_max=None*, *rounding=None*, ***kwargs*)
Bases: argparse.Action

Adds support for a slider in the GUI.

The standard argparse.Action is extended with the additional parameters listed below. The default value must be supplied and the type must be either int or float. nargs are explicitly disallowed.

Parameters

- **min_max** (*tuple*) – The (*min*, *max*) values that the slider's range should be set to. The values should be a pair of *float* or *int* data types, depending on the data type of the slider. NB: These min/max values are not enforced, they are purely for setting the slider range. Values outside of this range can still be explicitly passed in from the cli.
- **rounding** (*int*) – If the underlying data type for the option is a *float* then this value is the number of decimal places to round the slider values to. If the underlying data type for the option is an *int* then this is the step interval between each value for the slider.

Examples

For integer values:

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--foobar"),
>>>         action=Slider,
>>>         min_max=(0, 10)
>>>         rounding=1
>>>         type=int,
>>>         default=5))
```

For floating point values:

```
>>> argument_list = []
>>> argument_list.append(dict(
>>>         opts=("-f", "--foobar"),
>>>         action=Slider,
>>>         min_max=(0.00, 1.00)
>>>         rounding=2
>>>         type=float,
>>>         default=5.00))
```

launcher module

Launches the correct script with the given Command Line Arguments

class lib.cli.launcher.**ScriptExecutor**(*command*)
Bases: object

Loads the relevant script modules and executes the script.

This class is initialized in each of the argparsers for the relevant command, then execute script is called within their set_default function.

Parameters **command** (*str*) – The faceswap command that is being executed

execute_script(*arguments*)
Performs final set up and launches the requested _command with the given command line arguments.

Monitors for errors and attempts to shut down the process cleanly on exit.

Parameters **arguments** (argparse.Namespace) – The command line arguments to be passed to the executing script.

1.1.3 convert module

Converter for Faceswap

class lib.convert.**Converter**(*output_size*, *coverage_ratio*, *centering*, *draw_transparent*, *pre_encode*, *arguments*, *configfile=None*)
Bases: object

The converter is responsible for swapping the original face(s) in a frame with the output of a trained Faceswap model.

Parameters

- **output_size** (*int*) – The size of the face, in pixels, that is output from the Faceswap model
- **coverage_ratio** (*float*) – The ratio of the training image that was used for training the Faceswap model
- **centering** (*str*) – The extracted face centering that the model was trained on (*"face"* or "*legacy*")
- **draw_transparent** (*bool*) – Whether the final output should be drawn onto a transparent layer rather than the original frame. Only available with certain writer plugins.
- **pre_encode** (*python function*) – Some writer plugins support the pre-encoding of images prior to saving out. As patching is done in multiple threads, but writing is done in a single thread, it can speed up the process to do any pre-encoding as part of the converter process.
- **arguments** (argparse.Namespace) – The arguments that were passed to the convert process as generated from Faceswap's command line arguments
- **configfile** (*str, optional*) – Optional location of custom configuration ini file. If None then use the default config location. Default: None

cli_arguments
The command line arguments passed to the convert process

Type argparse.Namespace

process(*in_queue*, *out_queue*)
Main convert process.

Takes items from the in queue, runs the relevant adjustments, patches faces to final frame and outputs patched frame to the out queue.

Parameters

- **in_queue** (queue.Queue) – The output from scripts.convert.Predictor. Contains detected faces from the Faceswap model as well as the frame to be patched.
- **out_queue** (queue.Queue) – The queue to place patched frames into for writing by one of Faceswap's plugins.convert.writer plugins.

reinitialize(*config*)
Reinitialize this *Converter*.

Called as part of the preview tool. Resets all adjustments then loads the plugins as specified in the given config.

Parameters **config** (lib.config.FaceswapConfig) – Pre-loaded lib.config.FaceswapConfig. used over any configuration on disk.

1.1.4 gpu_stats module

Collects and returns Information on available GPUs.

The information returned from this module provides information for both Nvidia and AMD GPUs. However, the information available for Nvidia is far more thorough than what is available for AMD, where we need to plug into plaidML to pull stats. The quality of this data will vary depending on the OS' particular OpenCL implementation.

class lib.gpu_stats.**GPUStats**(*log=True*)
Bases: object

Holds information and statistics about the GPU(s) available on the currently running system.

Parameters **log** (*bool, optional*) – Whether the class should output information to the logger. There may be occasions where the logger has not yet been set up when this class is queried. Attempting to log in these instances will raise an error. If GPU stats are being queried prior to the logger being available then this parameter should be set to False. Otherwise set to True. Default: True

cli_devices
List of available devices for use in faceswap's command line arguments

Type list

device_count
The number of GPU devices discovered on the system.

Type int

exclude_all_devices
True if all GPU devices have been explicitly disabled otherwise False

Type bool

get_card_most_free()
Obtain statistics for the GPU with the most available free VRAM.

Returns

The dictionary contains the following data:

card_id (*int*): The index of the card as pertaining to _handles

device (*str*): The name of the device

free (*float*): The amount of available VRAM on the GPU

total (*float*): the total amount of VRAM on the GPU

If a GPU is not detected then the **card_id** is returned as `-1` and the amount of free and total RAM available is fixed to 2048 Megabytes.

Return type dict

sys_info

GPU Stats that are required for system information logging.

The dictionary contains the following data:

vram (*list*): the total amount of VRAM in Megabytes for each GPU as pertaining to `_handles`

driver (*str*): The GPU driver version that is installed on the OS

devices (*list*): The device name of each GPU on the system as pertaining to `_handles`

devices_active (*list*): The device name of each active GPU on the system as pertaining to `_handles`

Type dict

`lib.gpu_stats.`**set_exclude_devices**(*devices*)

Add any explicitly selected GPU devices to the global list of devices to be excluded from use by Faceswap.

Parameters **devices** (*list*) – list of indices corresponding to the GPU devices connected to the computer

1.1.5 gui package

The GUI Package contains the entire code base for Faceswap's optional GUI. The GUI itself is largely self-generated from the command line options specified in *lib.cli.args*.

Contents

analysis package

stats module

Package Summary

Calculations	Class that performs calculations on the GlobalSession raw data for the given session id.
GlobalSession	Holds information about a loaded or current training session by accessing a model's state file and Tensorboard logs.
SessionsSummary	Performs top level summary calculations for each session ID within the loaded or currently training Session for display in the Analysis tree view.
TensorBoardLogs	Parse data from TensorBoard logs.

stats Module

Stats functions for the GUI.

Holds the globally loaded training session. This will either be a user selected session (loaded in the analysis tab) or the currently training session.

class lib.gui.analysis.stats.**Calculations**(*session_id*, *display='loss'*, *loss_keys='loss'*, *selections='raw'*, *avg_samples=500*, *smooth_amount=0.9*, *flatten_outliers=False*)

Bases: object

Class that performs calculations on the *GlobalSession* raw data for the given session id.

Parameters

- **session_id** (int or None) – The session id number for the selected session from the Analysis tab. Should be None if all sessions are being calculated
- **display** (*{"loss", "rate"}, optional*) – Whether to display a graph for loss or training rate. Default: *"loss"*
- **loss_keys** (*list, optional*) – The list of loss keys to display on the graph. Default: *["loss"]*
- **selections** (*list, optional*) – The selected annotations to display. Default: *["raw"]*
- **avg_samples** (*int, optional*) – The number of samples to use for performing moving average calculation. Default: *500*.
- **smooth_amount** (*float, optional*) – The amount of smoothing to apply for performing smoothing calculation. Default: *0.9*.
- **flatten_outliers** (*bool, optional*) – True if values significantly away from the average should be excluded, otherwise False. Default: False

iterations
The number of iterations in the data set.

Type int

refresh()
Refresh the stats

set_iterations_limit(*limit*)
Set the number of iterations to display in the calculations.

If a value greater than 0 is passed, then the latest iterations up to the given limit will be calculated.

Parameters **limit** (*int*) – The number of iterations to calculate data for. *0* to calculate for all data

set_smooth_amount(*amount*)
Set the amount of smoothing to apply to smoothed graph.

Parameters **amount** (*float*) – The amount of smoothing to apply to smoothed graph

start_iteration
The starting iteration number of a limit has been set on the amount of data.

Type int

stats
The final calculated statistics

Type dict

update_selections(*selection*, *option*)
Update the type of selected data.

Parameters

- **selection** (*str*) – The selection to update (as can exist in `_selections`)
- **option** (*bool*) – `True` if the selection should be included, `False` if it should be removed

class lib.gui.analysis.stats.**GlobalSession**
Bases: `object`

Holds information about a loaded or current training session by accessing a model's state file and Tensorboard logs. This class should not be accessed directly, rather through `lib.gui.analysis.Session`

batch_sizes
The batch sizes for each session_id for the model.

Type dict

clear()
Clear the currently loaded session.

full_summary
List of dictionaries containing summary statistics for each session id.

Type list

get_loss(*session_id*)
Obtain the loss values for the given session_id.

Parameters **session_id** (int or `None`) – The session ID to return loss for. Pass `None` to return loss for all sessions.

Returns Loss names as key, `numpy.ndarray` as value. If No session ID was provided all session's losses are collated

Return type dict

get_loss_keys(*session_id*)
Obtain the loss keys for the given session_id.

Parameters **session_id** (int or `None`) – The session ID to return the loss keys for. Pass `None` to return loss keys for all sessions.

Returns The loss keys for the given session. If `None` is passed as session_id then a unique list of all loss keys for all sessions is returned

Return type list

get_timestamps(*session_id*)
Obtain the time stamps keys for the given session_id.

Parameters **session_id** (int or `None`) – The session ID to return the time stamps for. Pass `None` to return time stamps for all sessions.

Returns If a session ID has been given then a single `numpy.ndarray` will be returned with the session's time stamps. Otherwise a 'dict' will be returned with the session IDs as key with `numpy.ndarray` of timestamps as values

Return type dict or `numpy.ndarray`

initialize_session(*model_folder*, *model_name*, *is_training=False*)
Initialize a Session.

Load's the model's state file, and sets the paths to any underlying Tensorboard logs, ready for access on request.

Parameters

- **model_folder** (*str, optional*) – If loading a session manually (e.g. for the analysis tab), then the path to the model folder must be provided. For training sessions, this should be left at `None`
- **model_name** (*str, optional*) – If loading a session manually (e.g. for the analysis tab), then the model filename must be provided. For training sessions, this should be left at `None`
- **is_training** (*bool, optional*) – `True` if the session is being initialized for a training session, otherwise `False`. Default: `False`

is_loaded
`True` if session data is loaded otherwise `False`

Type bool

is_training
`True` if the loaded session is the currently training model, otherwise `False`

Type bool

logging_disabled
`True` if logging is enabled for the currently training session otherwise `False`.

Type bool

model_filename
The full model filename

Type str

session_ids
The sorted list of all existing session ids in the state file

Type list

stop_training()
Clears the internal training flag. To be called when training completes.

class lib.gui.analysis.stats.**SessionsSummary**(*session*)
Bases: object

Performs top level summary calculations for each session ID within the loaded or currently training Session for display in the Analysis tree view.

Parameters **session** (*GlobalSession*) – The loaded or currently training session

get_summary_stats()
Compile the individual session statistics and calculate the total.

Format the stats for display

Returns A list of summary statistics dictionaries containing the Session ID, start time, end time, elapsed time, rate, batch size and number of iterations for each session id within the loaded data as well as the totals.

Return type list

event_reader Module

Handles the loading and collation of events from Tensorflow event log files.

class lib.gui.analysis.event_reader.**TensorBoardLogs**(*logs_folder*, *is_training*)
Bases: object

Parse data from TensorBoard logs.

Process the input logs folder and stores the individual filenames per session.

Caches timestamp and loss data on request and returns this data from the cache.

Parameters

- **logs_folder** (*str*) – The folder that contains the Tensorboard log files
- **is_training** (*bool*) – True if the events are being read whilst Faceswap is training otherwise False

get_loss(*session_id=None*)
Read the loss from the TensorBoard event logs

Parameters **session_id** (*int, optional*) – The Session ID to return the loss for. Set to None to return all session losses. Default None

Returns The session id(s) as key, with a further dictionary as value containing the loss name and list of loss values for each step

Return type dict

get_timestamps(*session_id=None*)
Read the timestamps from the TensorBoard logs.

As loss timestamps are slightly different for each loss, we collect the timestamp from the *batch_loss* key.

Parameters **session_id** (*int, optional*) – The Session ID to return the timestamps for. Set to None to return all session timestamps. Default None

Returns The session id(s) as key with list of timestamps per step as value

Return type dict

session_ids
Sorted list of integers of available session ids.

Type list

set_training(*is_training*)
Set the internal training flag to the given *is_training* value.

If a new training session is being instigated, refresh the log filenames

Parameters **is_training** (*bool*) – `True` to indicate that the logs to be read are from the currently training session otherwise `False`

custom_widgets module

Module Summary

ConsoleOut	The Console out section of the GUI.
ContextMenu	A Pop up menu to be triggered when right clicking on widgets that this menu has been applied to.
MultiOption	Similar to the standard `ttk.Radio` widget, but with the ability to select multiple pre-defined options.
RightClickMenu	A Pop up menu that can be bound to a right click mouse event to bring up a context menu
StatusBar	Status Bar for displaying the Status Message and Progress Bar at the bottom of the GUI.
Tooltip	Create a tooltip for a given widget as the mouse goes on it.

Module

Custom widgets for Faceswap GUI

class lib.gui.custom_widgets.**ConsoleOut**(*parent*, *debug*)
Bases: `tkinter.ttk.Frame`

The Console out section of the GUI.

A Read only text box for displaying the output from stdout/stderr.

All handling is internal to this method. To clear the console, the stored tkinter variable in `tk_vars` `console_clear` should be triggered.

Parameters

- **parent** (*tkinter object*) – The Console's parent widget
- **debug** (*bool*) – `True` if console output should not be directed to this widget otherwise `False`

class lib.gui.custom_widgets.**ContextMenu**(*widget*)
Bases: `tkinter.Menu`

A Pop up menu to be triggered when right clicking on widgets that this menu has been applied to.

This widget provides a simple right click pop up menu to the widget passed in with *Cut*, *Copy*, *Paste* and *Select all* menu items.

Parameters **widget** (*tkinter object*) – The widget to apply the *ContextMenu* to

Example

```
>>> text_box = ttk.Entry(parent)
>>> text_box.pack()
>>> right_click_menu = ContextMenu(text_box)
>>> right_click_menu.cm_bind()
```

cm_bind()
Bind the menu to the given widgets Right Click event

After associating a widget with this *ContextMenu* this function should be called to bind it to the right click button

class lib.gui.custom_widgets.**MultiOption**(*parent*, *value*, *variable*, ***kwargs*)
Bases: tkinter.ttk.Checkbutton

Similar to the standard ttk.Radio widget, but with the ability to select multiple pre-defined options. Selected options are generated as *nargs* for the argument parser to consume.

Parameters

- **parent** (ttk.Frame) – The tkinter parent widget for the check button
- **value** (*str*) – The raw option value for this check button
- **variable** (tkinter.StingVar) – The master variable for the group of check buttons that this check button will belong to. The output of this variable will be a string containing a space separated list of the selected check button options

class lib.gui.custom_widgets.**PopupProgress**(*title*, *total*)
Bases: tkinter.Toplevel

A simple pop up progress bar that appears of the center of the root window.

When this is called, the root will be disabled until the close() method is called.

Parameters

- **title** (*str*) – The title to appear above the progress bar
- **total** (*int or float*) – The total count of items for the progress bar

Example

```
>>> total = 100
>>> progress = PopupProgress("My title...", total)
>>> for i in range(total):
>>>     progress.update(1)
>>> progress.close()
```

progress_bar
The progress bar object within the pop up window.

Type tkinter.ttk.Progressbar

step(*amount*)
Increment the progress bar.

Parameters **amount** (*int or float*) – The amount to increment the progress bar by

stop()
Stop the progress bar, re-enable the root window and destroy the pop up window.

update_title(*title*)
Update the title that displays above the progress bar.

Parameters **title** (*str*) – The title to appear above the progress bar

class lib.gui.custom_widgets.**RightClickMenu**(*labels*, *actions*, *hotkeys=None*)
Bases: tkinter.Menu

A Pop up menu that can be bound to a right click mouse event to bring up a context menu

Parameters

- **labels** (*list*) – A list of label titles that will appear in the right click menu
- **actions** (*list*) – A list of python functions that are called when the corresponding label is clicked on
- **hotkeys** (*list, optional*) – The hotkeys corresponding to the labels. If using hotkeys, then there must be an entry in the list for every label even if they don't all use hotkeys. Labels without a hotkey can be an empty string or `None`. Passing `None` instead of a list means that no actions will be given hotkeys. NB: The hotkey is not bound by this class, that needs to be done in code. Giving hotkeys here means that they will be displayed in the menu though. Default: `None`

popup(*event*)
Pop up the right click menu.

Parameters **event** (class:*tkinter.Event*) – The tkinter mouse event calling this popup

class lib.gui.custom_widgets.**StatusBar**(*parent*, *hide_status=False*)
Bases: tkinter.ttk.Frame

Status Bar for displaying the Status Message and Progress Bar at the bottom of the GUI.

Parameters

- **parent** (*tkinter object*) – The parent tkinter widget that will hold the status bar
- **hide_status** (*bool, optional*) – `True` to hide the status message that appears at the far left hand side of the status frame otherwise `False`. Default: `False`

message
The variable to hold the status bar message on the left hand side of the status bar.

Type tkinter.StringVar

progress_update(*message*, *position*, *update_position=True*)
Update the GUIs progress bar and position.

Parameters

- **message** (*str*) – The message to display next to the progress bar
- **position** (*int*) – The position that the progress bar should be set to
- **update_position** (*bool, optional*) – If `True` then the progress bar will be updated to the position given in `position`. If `False` the progress bar will not be updates. Default: `True`

start(*mode*)
Set progress bar mode and display,

Parameters **mode** (*["indeterminate", "determinate"]*) – The mode that the progress bar should be executed in

stop()
Reset progress bar and hide

class lib.gui.custom_widgets.**ToggledFrame**(*parent*, **args*, *text=''*, *theme='CPanel'*, *toggle_var=None*, ***kwargs*)

Bases: tkinter.ttk.Frame

A collapsible and expandable frame.

The frame contains a header given in the text argument, and adds an expand contract button. Clicking on the header will expand and contract the sub-frame below

Parameters

- **text** (*str*) – The text to appear in the Toggle Frame header
- **theme** (*str, optional*) – The theme to use for the panel header. Default: *"CPanel"*
- **subframe_style** (*str, optional*) – The name of the ttk Style to use for the sub frame. Default: None
- **toggle_var** (tk.BooleanVar, optional) – If provided, this variable will control the expanded (True) and minimized (False) state of the widget. Set to None to create the variable internally. Default: None

is_expanded
True if the Toggle Frame is expanded. False if it is minimized.

Type bool

class lib.gui.custom_widgets.**Tooltip**(*widget*, ***, *pad=(5, 3, 5, 3)*, *text='widget info'*, *text_variable=None*, *wait_time=400*, *wrap_length=250*)

Bases: object

Create a tooltip for a given widget as the mouse goes on it.

Parameters

- **widget** (*tkinter object*) – The widget to apply the tool-tip to
- **pad** (*tuple, optional*) – (left, top, right, bottom) padding for the tool-tip. Default: (5, 3, 5, 3)
- **text** (*str, optional*) – The text to be displayed in the tool-tip. Default: 'widget info'
- **text_variable** (tkinter.strVar, optional) – The text variable to use for dynamic help text. Appended after the contents of text if provided. Default: None
- **wait_time** (*int, optional*) – The time in milliseconds to wait before showing the tool-tip. Default: 400
- **wrap_length** (*int, optional*) – The text length for each line before wrapping. Default: 250

Example

```
>>> button = ttk.Button(parent, text="Exit")
>>> Tooltip(button, text="Click to exit")
>>> button.pack()
```

Notes

Adapted from StackOverflow: http://stackoverflow.com/questions/3221956 and http://www.daniweb.com/programming/software-development/code/484591/a-tooltip-class-for-tkinter

display module

Display Frame of the Faceswap GUI

This is the large right hand area of the GUI. At default, the Analysis tab is always displayed here. Further optional tabs will also be displayed depending on the currently executing Faceswap task.

class lib.gui.display.**DisplayNotebook**(*parent*)
Bases: tkinter.ttk.Notebook

The tkinter Notebook that holds the display items.

Parameters **parent** (tk.PanedWindow) – The paned window that holds the Display Notebook

runningtask
The global tkinter variable that indicates whether a Faceswap task is currently running or not.

Type tkinter.BooleanVar

display_analysis module

Module Summary

Analysis	Session Analysis Tab.
StatsData	Stats frame of analysis tab.

Module

Analysis tab of Display Frame of the Faceswap GUI

class lib.gui.display_analysis.**Analysis**(*parent*, *tab_name*, *helptext*)
Bases: lib.gui.display_page.DisplayPage

Session Analysis Tab.

The area of the GUI that holds the session summary stats for model training sessions.

Parameters

- **parent** (*lib.gui.display.DisplayNotebook*) – The ttk.Notebook that holds this session summary statistics page
- **tab_name** (*str*) – The name of the tab to be displayed in the notebook
- **helptext** (*str*) – The help text to display for the summary statistics page

on_tab_select()
Callback for when the analysis tab is selected.

If Faceswap is currently training a model, then update the statistics with the latest values.

set_vars()
Set the analysis specific tkinter variables to vars.

The tracked variables are the global variables that:

- Trigger when a graph refresh has been requested.
- Trigger training is commenced or halted
- The variable holding the location of the current Tensorboard log folder.

Returns The dictionary of variable names to tkinter variables

Return type dict

class lib.gui.display_analysis.**StatsData**(*parent*, *selected_id*, *helptext*)
Bases: tkinter.ttk.Frame

Stats frame of analysis tab.

Holds the tree-view containing the summarized session statistics in the Analysis tab.

Parameters

- **parent** (tkinter.Frame) – The frame within the Analysis Notebook that will hold the statistics
- **selected_id** (tkinter.IntVar) – The tkinter variable that holds the currently selected session ID
- **helptext** (*str*) – The help text to display for the summary statistics page

tree_clear()
Clear all of the summary data from the tree-view.

tree_insert_data(*sessions_summary*)
Insert the summary data into the statistics tree-view.

Parameters **sessions_summary** (*list*) – List of session summary dicts for populating into the tree-view

popup_configure module

The pop-up window of the Faceswap GUI for the setting of configuration options.

class lib.gui.popup_configure.**DisplayArea**(*parent*, *configurations*, *tree*, *theme*)
Bases: tkinter.ttk.Frame

The option configuration area of the pop up options.

Parameters

- **parent** (tkinter.ttk.Frame) – The parent frame that holds the Display Area of the pop up configuration window
- **tree** (tkinter.ttk.TreeView) – The Tree View navigator for the pop up configuration window
- **configurations** (*dict*) – Dictionary containing the FaceswapConfig object for each configuration section for the requested pop-up window
- **theme** (*dict*) – The color mapping for the settings pop-up theme

reset(*page_only=False*)
Reset all configuration options to their default values.

Parameters **page_only** (*bool, optional*) – True resets just the currently selected page's options to default, False resets all plugins within the currently selected config to default. Default: False

save(*page_only=False*)
Save the configuration file to disk.

Parameters **page_only** (*bool, optional*) – `True` saves just the currently selected page's options, `False` saves all the plugins options within the currently selected config. Default: `False`

select_options(*section*, *subsections*)
Display the page for the given section and subsections.

Parameters

- **section** (*str*) – The main section to be navigated to (or root node)
- **subsections** (*list*) – The full list of subsections ending on the required node

popup_session module

Pop-up Graph launched from the Analysis tab of the Faceswap GUI

class lib.gui.popup_session.**SessionPopUp**(*session_id*, *data_points*)
Bases: `tkinter.Toplevel`

Pop up for detailed graph/stats for selected session.

session_id: int or *"Total"* The session id number for the selected session from the Analysis tab. Should be the string *"Total"* if all sessions are being graphed

data_points: int The number of iterations in the selected session

project module

Module Summary

LastSession	Faceswap Last Session handling.
Project	Faceswap `.fsw` Project File handling.
Tasks	Faceswap `.fst` Task File handling.

Module

Handling of Faceswap GUI Projects, Tasks and Last Session

class lib.gui.project.**LastSession**(*config*)
Bases: `lib.gui.project._GuiSession`

Faceswap Last Session handling.

Faceswap *LastSession* handles saving the state of the Faceswap GUI at close and reloading the state at launch.

Last Session behavior can be configured in `config.gui.ini`.

Parameters **config** (*lib.gui.utils.Config*) – The master GUI config

ask_load()
Pop a message box to ask the user if they wish to load their last session.

from_dict(*options*)
Set the `_options` property based on the given options dictionary and update the GUI to use these values.

This function is required for reloading the GUI state when the GUI has been force refreshed on a config change.

Parameters **options** (*dict*) – The options to set. Should be the output of *to_dict()*

load()
Load the last session.

Loads the last saved session options. Checks if a previous project was loaded and whether there have been changes since the last saved version of the project. Sets the display and *Project* and Task objects accordingly.

save()
Save a snapshot of currently set GUI config options.

Called on Faceswap shutdown.

to_dict()
Collect the current GUI options and place them in a dict for retrieval or storage.

This function is required for reloading the GUI state when the GUI has been force refreshed on a config change.

Returns **dict**

Return type The current cli options ready for saving or retrieval by *from_dict()*

class lib.gui.project.**Project**(*config*, *file_handler*)
Bases: lib.gui.project._GuiSession

Faceswap .fsw Project File handling.

Faceswap projects handle the management of all task tabs in the GUI and updates the main Faceswap title bar with the project name and modified state.

Parameters

- **config** (*lib.gui.utils.Config*) – The master GUI config
- **file_handler** (*lib.gui.utils.FileHandler*) – A file handler object

cli_options
the raw cli options from _options with project fields removed.

Type dict

close(**args*)
Clear the current project and set all options to default.

Parameters ***args** (*tuple*) – Unused, but needs to be present for arguments passed by tkinter event handling

confirm_close()
Pop a message box to get confirmation that an unsaved project should be closed

Returns **bool**

Return type True if user confirms close, False if user cancels close

filename
The currently active project filename.

Type str

load(**args*, *filename=None*)
Load a project from a saved .fsw project file.

Parameters

- ***args** (*tuple*) – Unused, but needs to be present for arguments passed by tkinter event handling
- **filename** (*str, optional*) – If a filename is passed in, This will be used, otherwise a file handler will be launched to select the relevant file.

new(**args*)
Create a new project with default options.

Pops a file handler to select location.

Parameters ***args** (*tuple*) – Unused, but needs to be present for arguments passed by tkinter event handling

reload(**args*)
Reset all GUI's option tabs to their last saved state.

Parameters ***args** (*tuple*) – Unused, but needs to be present for arguments passed by tkinter event handling

save(**args*, *save_as=False*)
Save the current GUI state to a `.fsw` project file.

Parameters

- ***args** (*tuple*) – Unused, but needs to be present for arguments passed by tkinter event handling
- **save_as** (*bool, optional*) – Whether to save to the stored filename, or pop open a file handler to ask for a location. If there is no stored filename, then a file handler will automatically be popped.

set_default_options()
Set the default options. The Default GUI options are stored on Faceswap startup.

Exposed as the `_default_options` for a project cannot be set until after the main Command Tabs have been loaded.

set_modified_callback()
Adds a callback to each of the `_modified_vars` tkinter variables When one of these variables is changed, triggers `_modified_callback()` with the command that was changed.

This is exposed as the callback can only be added after the main Command Tabs have been drawn, and their options' initial values have been set.

class lib.gui.project.**Tasks**(*config*, *file_handler*)
Bases: `lib.gui.project._GuiSession`

Faceswap `.fst` Task File handling.

Faceswap tasks handle the management of each individual task tab in the GUI. Unlike `Projects`, Tasks contains all the active tasks currently running, rather than an individual task.

Parameters

- **config** (*lib.gui.utils.Config*) – The master GUI config
- **file_handler** (*lib.gui.utils.FileHandler*) – A file handler object

add_project_task(*filename*, *command*, *options*)
Add an individual task from a loaded *Project* to the internal `_tasks` dict.

Project tasks take priority over any other tasks, so the individual tasks from a new project must be placed in the _tasks dict.

Parameters

- **filename** (*str*) – The filename of the session project file
- **command** (*str*) – The tab that this task's options belong to
- **options** (*dict*) – The options for this task loaded from the project

clear()
Reset all GUI options to their default values for the active tab.

clear_tasks()
Clears all of the stored tasks.

This is required when loading a task stored in a legacy project file, and is only to be called by *Project* when a project has been loaded which is in fact a task.

load(**args*, *filename=None*, *current_tab=True*)
Load a task into this *Tasks* class.

Tasks can be loaded from project `.fsw` files or task `.fst` files, depending on where this function is being called from.

Parameters

- ***args** (*tuple*) – Unused, but needs to be present for arguments passed by tkinter event handling
- **filename** (*str, optional*) – If a filename is passed in, This will be used, otherwise a file handler will be launched to select the relevant file.
- **current_tab** (*bool, optional*) – `True` if the task to be loaded must be for the currently selected tab. `False` if loading a task into any tab. If current_tab is *True* then tasks can be loaded from `.fsw` and `.fst` files, otherwise they can only be loaded from `.fst` files. Default: `True`

reload()
Reset currently selected tab GUI options to their last saved state.

save(*save_as=False*)
Save the current GUI state for the active tab to a `.fst` faceswap task file.

Parameters **save_as** (*bool, optional*) – Whether to save to the stored filename, or pop open a file handler to ask for a location. If there is no stored filename, then a file handler will automatically be popped.

theme module

Module

functions for implementing themes in Faceswap's GUI

class lib.gui.theme.**Style**(*default_font*, *root*, *path_cache*)
Bases: `object`

Set the overarching theme and customize widgets.

Parameters

- **default_font** (*tuple*) – The name and size of the default font

- **root** (tkinter.Tk) – The root tkinter object
- **path_cache** (*str*) – The path to the GUI's cache

user_theme
The currently selected user theme.

Type dict

utils module

Module Summary

Config	The centralized configuration class for holding items that should be made available to all parts of the GUI.
FileHandler	Handles all GUI File Dialog actions and tasks.
Images	The centralized image repository for holding all icons and images required by the GUI.
LongRunningTask	Runs long running tasks in a background thread to prevent the GUI from becoming unresponsive.
get_config	Get the Master GUI configuration.
get_images	Get the Master GUI Images handler.
initialize_config	Initialize the GUI Master Config and add to global constant.
initialize_images	Initialize the Images handler and add to global constant.

Module

Utility functions for the GUI

class lib.gui.utils.**Config**(*root*, *cli_opts*, *statusbar*)
Bases: object

The centralized configuration class for holding items that should be made available to all parts of the GUI.

This class should be initialized on GUI startup through *initialize_config()*. Any further access to this class should be through *get_config()*.

Parameters

- **root** (tkinter.Tk) – The root Tkinter object
- **cli_opts** (lib.gui.options.CliOpts) – The command line options object
- **statusbar** (*lib.gui.custom_widgets.StatusBar*) – The GUI Status bar

cli_opts
The command line options for this GUI Session.

Type lib.gui.options.CliOptions

command_notebook
The main Faceswap Command Notebook.

Type lib.gui.command.CommandNoteboook

default_font
The selected font as configured in user settings. First item is the font (*str*) second item the font size (*int*).

Type tuple

default_options
The default options for all tabs

Type dict

modified_vars
The command notebook modified tkinter variables.

Type dict

pathcache
The path to the GUI cache folder

Type str

project
The project session handler.

Type *lib.gui.project.Project*

refresh_config()
Reload the user config from file.

root
The root tkinter window.

Type tkinter.Tk

scaling_factor
The scaling factor for current display.

Type float

set_active_tab_by_name(*name*)
Sets the *command_notebook* or *tools_notebook* to active based on given name.

Parameters **name** (*str*) – The name of the tab to set active

set_command_notebook(*notebook*)
Set the command notebook to the *command_notebook* attribute and enable the modified callback for *project*.

Parameters **notebook** (lib.gui.command.CommandNotebook) – The main command notebook for the Faceswap GUI

set_cursor_busy(*widget=None*)
Set the root or widget cursor to busy.

Parameters **widget** (*tkinter object, optional*) – The widget to set busy cursor for. If the provided value is None then sets the cursor busy for the whole of the GUI. Default: None.

set_cursor_default(*widget=None*)
Set the root or widget cursor to default.

Parameters **widget** (*tkinter object, optional*) – The widget to set default cursor for. If the provided value is None then sets the cursor busy for the whole of the GUI. Default: None

set_default_options()
Set the default options for lib.gui.projects

The Default GUI options are stored on Faceswap startup.

Exposed as the `_default_opts` for a project cannot be set until after the main Command Tabs have been loaded.

set_geometry(*width*, *height*, *fullscreen=False*)
Set the geometry for the root tkinter object.

Parameters

- **width** (*int*) – The width to set the window to (prior to scaling)
- **height** (*int*) – The height to set the window to (prior to scaling)
- **fullscreen** (*bool, optional*) – Whether to set the window to full-screen mode. If `True` then `width` and `height` are ignored. Default: `False`

set_modified_true(*command*)
Set the modified variable to `True` for the given command in *modified_vars*.

Parameters **command** (*str*) – The command to set the modified state to `True`

set_root_title(*text=None*)
Set the main title text for Faceswap.

The title will always begin with 'Faceswap.py'. Additional text can be appended.

Parameters **text** (*str, optional*) – Additional text to be appended to the GUI title bar. Default: `None`

statusbar
The GUI StatusBar `tkinter.ttk.Frame`.

Type *lib.gui.custom_widgets.StatusBar*

tasks
The session tasks handler.

Type *lib.gui.project.Tasks*

tk_vars
The global tkinter variables.

Type dict

tools_notebook
The Faceswap Tools sub-Notebook.

Type `lib.gui.command.ToolsNotebook`

user_config
The GUI config in dict form.

Type dict

user_config_dict
The GUI config in dict form.

Type dict

user_theme
The GUI theme selection options.

Type dict

class lib.gui.utils.**FileHandler**(*handle_type*, *file_type*, *title=None*, *initial_folder=None*, *command=None*, *action=None*, *variable=None*)
Bases: `object`

Handles all GUI File Dialog actions and tasks.

Parameters

- **handle_type** (['open', 'save', 'filename', 'filename_multi', 'save_filename', 'context', *dir*]) – The type of file dialog to return. *open* and *save* will perform the open and save actions and return the file. *filename* returns the filename from an *open* dialog. *filename_multi* allows for multi-selection of files and returns a list of files selected. *save_filename* returns the filename from a *save as* dialog. *context* is a context sensitive parameter that returns a certain dialog based on the current options. *dir* asks for a folder location.
- **file_type** (*['default', 'alignments', 'config_project', 'config_task', 'config_all', 'csv', 'image', 'ini', 'state', 'log', 'video']*) – The type of file that this dialog is for. *default* allows selection of any files. Other options limit the file type selection
- **title** (*str, optional*) – The title to display on the file dialog. If *None* then the default title will be used. Default: `None`
- **initial_folder** (*str, optional*) – The folder to initially open with the file dialog. If *None* then tkinter will decide. Default: `None`
- **command** (*str, optional*) – Required for context handling file dialog, otherwise unused. Default: `None`
- **action** (*str, optional*) – Required for context handling file dialog, otherwise unused. Default: `None`
- **variable** (`tkinter.StringVar`, optional) – Required for context handling file dialog, otherwise unused. The variable to associate with this file dialog. Default: `None`

return_file
The return value from the file dialog

Type str or object

Example

```
>>> handler = FileHandler('filename', 'video', title='Select a video...')
>>> video_file = handler.return_file
>>> print(video_file)
'/path/to/selected/video.mp4'
```

class lib.gui.utils.**Images**
Bases: `object`

The centralized image repository for holding all icons and images required by the GUI.

This class should be initialized on GUI startup through *initialize_images()*. Any further access to this class should be through *get_images()*.

delete_preview()
Delete the preview files in the cache folder and reset the image cache.

Should be called when terminating tasks, or when Faceswap starts up or shuts down.

icons
The faceswap icons for all parts of the GUI. The dictionary key is the icon name (*str*) the value is the icon sized and formatted for display (`PIL.ImageTK.PhotoImage`).

Example

```
>>> icons = get_images().icons
>>> save = icons["save"]
>>> button = ttk.Button(parent, image=save)
>>> button.pack()
```

Type dict

load_latest_preview(*thumbnail_size*, *frame_dims*)
Load the latest preview image for extract and convert.

Retrieves the latest preview images from the faceswap output folder, resizes to thumbnails and lays out for display. Places the images into *previewoutput* for loading into the display panel.

Parameters

- **thumbnail_size** (*int*) – The size of each thumbnail that should be created
- **frame_dims** (*tuple*) – The (width (*int*), height (*int*)) of the display panel that will display the preview

load_training_preview()
Load the training preview images.

Reads the training image currently stored in the cache folder and loads them to *previewtrain* for retrieval in the GUI.

previewoutput
First item in the tuple is the extract or convert preview image (`PIL.Image`), the second item is the image in a format that tkinter can display (`PIL.ImageTK.PhotoImage`).

The value of the property is `None` if no extract or convert task is running or there are no files available in the output folder.

Type Tuple or `None`

previewtrain
The training preview images. Dictionary key is the image name (*str*). Dictionary values are a *list* of the training image (`PIL.Image`), the image formatted for tkinter display (`PIL.ImageTK.PhotoImage`), the last modification time of the image (*float*).

The value of this property is `None` if training is not running or there are no preview images available.

Type dict or `None`

resize_image(*name*, *frame_dims*)
Resize the training preview image based on the passed in frame size.

If the canvas that holds the preview image changes, update the image size to fit the new canvas and refresh *previewtrain*.

Parameters

- **name** (*str*) – The name of the training image to be resized
- **frame_dims** (*tuple*) – The (width (*int*), height (*int*)) of the display panel that will display the preview

set_faceswap_output_path(*location*)
Set the path that will contain the output from an Extract or Convert task.

Required so that the GUI can fetch output images to display for return in *previewoutput*.

Parameters **location** (*str*) – The output location that has been specified for an Extract or Convert task

class lib.gui.utils.**LongRunningTask**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, ***, *daemon=True*, *widget=None*)

Bases: threading.Thread

Runs long running tasks in a background thread to prevent the GUI from becoming unresponsive.

This is sub-classed from Threading.Thread so check documentation there for base parameters. Additional parameters listed below.

Parameters **widget** (*tkinter object, optional*) – The widget that this *LongRunningTask* is associated with. Used for setting the busy cursor in the correct location. Default: None.

complete
Event is set if the thread has completed its task, otherwise it is unset.

Type threading.Event

get_result()
Return the result from the given task.

Returns The result of the thread will depend on the given task. If a call is made to *get_result()* prior to the thread completing its task then None will be returned

Return type varies

run()
Commence the given task in a background thread.

class lib.gui.utils.**PreviewTrigger**

Bases: object

Trigger to indicate to underlying Faceswap process that the preview image should be updated.

Writes a file to the cache folder that is picked up by the main process.

clear()
Remove the trigger file from the cache folder

set()
Place the trigger file into the cache folder

lib.gui.utils.**get_config**()
Get the Master GUI configuration.

Returns The Master GUI Config

Return type *Config*

lib.gui.utils.**get_images**()
Get the Master GUI Images handler.

Returns The Master GUI Images handler

Return type *Images*

lib.gui.utils.**initialize_config**(*root*, *cli_opts*, *statusbar*)
Initialize the GUI Master *Config* and add to global constant.

This should only be called once on first GUI startup. Future access to *Config* should only be executed through *get_config()*.

Parameters

- **root** (tkinter.Tk) – The root Tkinter object
- **cli_opts** (lib.gui.options.CliOpts) – The command line options object
- **statusbar** (*lib.gui.custom_widgets.StatusBar*) – The GUI Status bar

lib.gui.utils.**initialize_images**()
Initialize the *Images* handler and add to global constant.

This should only be called once on first GUI startup. Future access to *Images* handler should only be executed through *get_images()*.

lib.gui.utils.**preview_trigger**()
Set the global preview trigger if it has not always been set and return.

Returns The trigger to indicate to the main faceswap process that it should perform a training preview update

Return type *PreviewTrigger*

1.1.6 image module

Handles loading and manipulation of images in Faceswap.

Module Summary

FacesLoader	Loads faces from a faces folder along with the face's Faceswap metadata.
FfmpegReader	Monkey patch imageio ffmpeg to use keyframes whilst seeking
ImageIO	Perform disk IO for images or videos in a background thread.
ImagesLoader	Perform image loading from a folder of images or a video.
ImagesSaver	Perform image saving to a destination folder.
SingleFrameLoader	Allows direct access to a frame by filename or frame index.
batch_convert_color	Convert a batch of images from one color space to another.
count_frames	Count the number of frames in a video file
encode_image	Encode an image.
generate_thumbnail	Generate a jpg thumbnail for the given image.
hex_to_rgb	Convert a hex number to it's RGB counterpart.
png_read_meta	Read the Faceswap information stored in a png's iTXt field.
png_write_meta	Write Faceswap information to a png's iTXt field.
read_image	Read an image file from a file location.
read_image_batch	Load a batch of images from the given file locations.
read_image_meta	Read the Faceswap metadata stored in an extracted face's exif header.
read_image_meta_batch	Read the Faceswap metadata stored in a batch extracted faces' exif headers.
rgb_to_hex	Convert an RGB tuple to it's hex counterpart.

Module

Utilities for working with images and videos

class lib.image.**FacesLoader**(*path*, *skip_list=None*, *count=None*)
Bases: *lib.image.ImagesLoader*

Loads faces from a faces folder along with the face's Faceswap metadata.

Examples

Loading faces with their Faceswap metadata:

```
>>> loader = FacesLoader('/path/to/faces/folder')
>>> for filename, face, metadata in loader.load():
>>>     <do processing>
```

class lib.image.**FfmpegReader**(*format*, *request*)
Bases: imageio.plugins.ffmpeg.Reader

Monkey patch imageio ffmpeg to use keyframes whilst seeking

get_frame_info(*frame_pts=None*, *keyframes=None*)
Store the source video's keyframes in _frame_info" for the current video for use in :func:`initialize.

Parameters

- **frame_pts** (*list, optional*) – A list corresponding to the video frame count of the pts_time per frame. If this and *keyframes* are provided, then analyzing the video is skipped and the values from the given lists are used. Default: None
- **keyframes** (*list, optional*) – A list containing the frame numbers of each key frame. if this and *frame_pts* are provided, then analyzing the video is skipped and the values from the given lists are used. Default: None

class lib.image.**ImageIO**(*path*, *queue_size*, *args=None*)
Bases: object

Perform disk IO for images or videos in a background thread.

This is the parent thread for *ImagesLoader* and *ImagesSaver* and should not be called directly.

Parameters

- **path** (*str or list*) – The path to load or save images to/from. For loading this can be a folder which contains images, video file or a list of image files. For saving this must be an existing folder.
- **queue_size** (*int*) – The amount of images to hold in the internal buffer.
- **args** (*tuple, optional*) – The arguments to be passed to the loader or saver thread. Default: None

See also:

lib.image.ImagesLoader Background Image Loader inheriting from this class.

lib.image.ImagesSaver Background Image Saver inheriting from this class.

close()
Closes down and joins the internal threads

location
The folder or video that was passed in as the `path` parameter.

Type str

class lib.image.**ImagesLoader**(*path*, *queue_size=8*, *fast_count=True*, *skip_list=None*, *count=None*)

Bases: *lib.image.ImageIO*

Perform image loading from a folder of images or a video.

Images will be loaded and returned in the order that they appear in the folder, or in the video to ensure deterministic ordering. Loading occurs in a background thread, caching 8 images at a time so that other processes do not need to wait on disk reads.

See also *ImageIO* for additional attributes.

Parameters

- **path** (*str or list*) – The path to load images from. This can be a folder which contains images a video file or a list of image files.
- **queue_size** (*int, optional*) – The amount of images to hold in the internal buffer. Default: 8.
- **fast_count** (*bool, optional*) – When loading from video, the video needs to be parsed frame by frame to get an accurate count. This can be done quite quickly without guaranteed accuracy, or slower with guaranteed accuracy. Set to `True` to count quickly, or `False` to count slower but accurately. Default: `True`.
- **skip_list** (*list, optional*) – Optional list of frame/image indices to not load. Any indices provided here will be skipped when executing the *load()* function from the given location. Default: `None`
- **count** (*int, optional*) – If the number of images that the loader will encounter is already known, it can be passed in here to skip the image counting step, which can save time at launch. Set to `None` if the count is not already known. Default: `None`

Examples

Loading from a video file:

```
>>> loader = ImagesLoader('/path/to/video.mp4')
>>> for filename, image in loader.load():
>>>     <do processing>
```

add_skip_list(*skip_list*)
Add a skip list to this *ImagesLoader*

Parameters **skip_list** (*list*) – A list of indices corresponding to the frame indices that should be skipped by the *load()* function.

count
The number of images or video frames in the source location. This count includes any files that will ultimately be skipped if a `skip_list` has been provided. See also: *process_count*

Type int

file_list
A full list of files in the source location. This includes any files that will ultimately be skipped if a `skip_list` has been provided. If the input is a video then this is a list of dummy filenames as corresponding to an alignments file

Type list

fps
For an input folder of images, this will always return 25fps. If the input is a video, then the fps of the video will be returned.

Type float

is_video
`True` if the input is a video, `False` if it is not

Type bool

load()
Generator for loading images from the given `location`

If *FacesLoader* is in use then the Faceswap metadata of the image stored in the image exif file is added as the final item in the output *tuple*.

Yields

- **filename** (*str*) – The filename of the loaded image.
- **image** (*numpy.ndarray*) – The loaded image.
- **metadata** (dict, (*FacesLoader* only)) – The Faceswap metadata associated with the loaded image.

process_count
The number of images or video frames to be processed (IE the total count less items that are to be skipped from the `skip_list`)

Type int

class lib.image.**ImagesSaver**(*path*, *queue_size=8*, *as_bytes=False*)
Bases: *lib.image.ImageIO*

Perform image saving to a destination folder.

Images are saved in a background ThreadPoolExecutor to allow for concurrent saving. See also *ImageIO* for additional attributes.

Parameters

- **path** (*str*) – The folder to save images to. This must be an existing folder.
- **queue_size** (*int, optional*) – The amount of images to hold in the internal buffer. Default: 8.
- **as_bytes** (*bool, optional*) – `True` if the image is already encoded to bytes, `False` if the image is a `numpy.ndarray`. Default: `False`.

Examples

```
>>> saver = ImagesSaver('/path/to/save/folder')
>>> for filename, image in <image_iterator>:
>>>     saver.save(filename, image)
>>> saver.close()
```

close()
Signal to the Save Threads that they should be closed and cleanly shutdown the saver

save(*filename*, *image*)
Save the given image in the background thread

Ensure that *close()* is called once all save operations are complete.

Parameters

- **filename** (*str*) – The filename of the image to be saved
- **image** (*numpy.ndarray*) – The image to be saved

class lib.image.**SingleFrameLoader**(*path*, *video_meta_data=None*)
Bases: *lib.image.ImagesLoader*

Allows direct access to a frame by filename or frame index.

As we are interested in instant access to frames, there is no requirement to process in a background thread, as either way we need to wait for the frame to load.

Parameters **video_meta_data** (*dict, optional*) – Existing video meta information containing the pts_time and iskey flags for the given video. Used in conjunction with single_frame_reader for faster seeks. Providing this means that the video does not need to be scanned again. Set to `None` if the video is to be scanned. Default: `None`

image_from_index(*index*)
Return a single image from `file_list` for the given index.

Parameters **index** (*int*) – The index number (frame number) of the frame to retrieve. NB: The first frame is index *0*

Returns

- **filename** (*str*) – The filename of the returned image
- **image** (`numpy.ndarray`) – The image for the given index

Notes

Retrieving frames from video files can be slow as the whole video file needs to be iterated to retrieve the requested frame. If a frame has already been retrieved, then retrieving frames of a higher index will be quicker than retrieving frames of a lower index, as iteration needs to start from the beginning again when navigating backwards.

We do not use a background thread for this task, as it is assumed that requesting an image by index will be done when required.

video_meta_data
For videos contains the keys *frame_pts* holding a list of time stamps for each frame and *keyframes* holding the frame index of each key frame.

Notes

Only populated if the input is a video and single frame reader is being used, otherwise returns `None`.

Type dict

lib.image.**batch_convert_color**(*batch*, *colorspace*)
Convert a batch of images from one color space to another.

Converts a batch of images by reshaping the batch prior to conversion rather than iterating over the images. This leads to a significant speed up in the convert process.

Parameters

- **batch** (*numpy.ndarray*) – A batch of images.
- **colorspace** (*str*) – The OpenCV Color Conversion Code suffix. For example for BGR to LAB this would be `'BGR2LAB'`. See https://docs.opencv.org/4.1.1/d8/d01/group__imgproc__color__conversions.html for a full list of color codes.

Returns The batch converted to the requested color space.

Return type numpy.ndarray

Example

```
>>> images_bgr = numpy.array([image1, image2, image3])
>>> images_lab = batch_convert_color(images_bgr, "BGR2LAB")
```

Notes

This function is only compatible for color space conversions that have the same image shape for source and destination color spaces.

If you use *batch_convert_color()* with 8-bit images, the conversion will have some information lost. For many cases, this will not be noticeable but it is recommended to use 32-bit images in cases that need the full range of colors or that convert an image before an operation and then convert back.

lib.image.**count_frames**(*filename*, *fast=False*)
Count the number of frames in a video file

There is no guaranteed accurate way to get a count of video frames without iterating through a video and decoding every frame.

count_frames() can return an accurate count (albeit fairly slowly) or a possibly less accurate count, depending on the `fast` parameter. A progress bar is displayed.

Parameters

- **filename** (*str*) – Full path to the video to return the frame count from.
- **fast** (*bool, optional*) – Whether to count the frames without decoding them. This is significantly faster but accuracy is not guaranteed. Default: `False`.

Returns The number of frames in the given video file.

Return type int

Example

```
>>> filename = "/path/to/video.mp4"
>>> frame_count = count_frames(filename)
```

lib.image.**encode_image**(*image*, *extension*, *metadata=None*)
Encode an image.

Parameters

- **image** (*numpy.ndarray*) – The image to be encoded in *BGR* channel order.

- **extension** (*str*) – A compatible *cv2* image file extension that the final image is to be saved to.
- **metadata** (*dict, optional*) – Metadata for the image. If provided, and the extension is png, this information will be written to the PNG itxt header. Default:`None`

Returns **encoded_image** – The image encoded into the correct file format

Return type bytes

Example

```
>>> image_file = "/path/to/image.png"
>>> image = read_image(image_file)
>>> encoded_image = encode_image(image, ".jpg")
```

`lib.image.`**`generate_thumbnail`**(*image*, *size=96*, *quality=60*)
Generate a jpg thumbnail for the given image.

Parameters

- **image** (`numpy.ndarray`) – Three channel BGR image to convert to a jpg thumbnail
- **size** (*int*) – The width and height, in pixels, that the thumbnail should be generated at
- **quality** (*int*) – The jpg quality setting to use

Returns The given image encoded to a jpg at the given size and quality settings

Return type `numpy.ndarray`

`lib.image.`**`hex_to_rgb`**(*hexcode*)
Convert a hex number to it's RGB counterpart.

Parameters **hexcode** (*str*) – The hex code to convert (e.g. *"#0d25ac"*)

Returns The hex code as a 3 integer (*R*, *G*, *B*) tuple

Return type tuple

`lib.image.`**`pack_to_itxt`**(*metadata*)
Pack the given metadata dictionary to a PNG iTXt header field.

Parameters **metadata** (*dict or bytes*) – The dictionary to write to the header. Can be pre-encoded as utf-8.

Returns A byte encoded PNG iTXt field, including chunk header and CRC

Return type bytes

`lib.image.`**`png_read_meta`**(*png*)
Read the Faceswap information stored in a png's iTXt field.

Parameters **png** (*bytes*) – The bytes encoded png file to read header data from

Returns The Faceswap information stored in the PNG header

Return type dict

Notes

This is a very stripped down, non-robust and non-secure header reader to fit a very specific task. OpenCV will not write any iTXt headers to the PNG file, so we make the assumption that the only iTXt header that exists is the one that Faceswap created for storing alignments.

lib.image.**png_write_meta**(*png*, *data*)
Write Faceswap information to a png's iTXt field.

Parameters

- **png** (*bytes*) – The bytes encoded png file to write header data to
- **data** (*dict or bytes*) – The dictionary to write to the header. Can be pre-encoded as utf-8.

Notes

This is a fairly stripped down and non-robust header writer to fit a very specific task. OpenCV will not write any iTXt headers to the PNG file, so we make the assumption that the only iTXt header that exists is the one that we created for storing alignments.

References

PNG Specification: https://www.w3.org/TR/2003/REC-PNG-20031110/

lib.image.**read_image**(*filename*, *raise_error=False*, *with_metadata=False*)
Read an image file from a file location.

Extends the functionality of `cv2.imread()` by ensuring that an image was actually loaded. Errors can be logged and ignored so that the process can continue on an image load failure.

Parameters

- **filename** (*str*) – Full path to the image to be loaded.
- **raise_error** (*bool, optional*) – If `True` then any failures (including the returned image being `None`) will be raised. If `False` then an error message will be logged, but the error will not be raised. Default: `False`
- **with_metadata** (*bool, optional*) – Only returns a value if the images loaded are extracted Faceswap faces. If `True` then returns the Faceswap metadata stored with in a Face images .png exif header. Default: `False`

Returns If `with_metadata` is `False` then returns a *numpy.ndarray* of the image in *BGR* channel order. If `with_metadata` is `True` then returns a *tuple* of (*numpy.ndarray*" of the image in *BGR*, *dict* of face's Faceswap metadata)

Return type numpy.ndarray or tuple

Example

```
>>> image_file = "/path/to/image.png"
>>> try:
>>>     image = read_image(image_file, raise_error=True, with_metadata=False)
>>> except:
>>>     raise ValueError("There was an error")
```

lib.image.**read_image_batch**(*filenames*, *with_metadata=False*)
Load a batch of images from the given file locations.

Leverages multi-threading to load multiple images from disk at the same time leading to vastly reduced image read times.

Parameters

- **filenames** (*list*) – A list of `str` full paths to the images to be loaded.
- **with_metadata** (*bool, optional*) – Only returns a value if the images loaded are extracted Faceswap faces. If `True` then returns the Faceswap metadata stored with in a Face images .png exif header. Default: `False`

Returns The batch of images in *BGR* channel order returned in the order of `filenames`

Return type numpy.ndarray

Notes

As the images are compiled into a batch, they must be all of the same dimensions.

Example

```
>>> image_filenames = ["/path/to/image_1.png", "/path/to/image_2.png", "/path/to/
↪image_3.png"]
>>> images = read_image_batch(image_filenames)
```

lib.image.**read_image_meta**(*filename*)
Read the Faceswap metadata stored in an extracted face's exif header.

Parameters **filename** (*str*) – Full path to the image to be retrieve the meta information for.

Returns The output dictionary will contain the *width* and *height* of the png image as well as any *itxt* information.

Return type dict

Example

```
>>> image_file = "/path/to/image.png"
>>> metadata = read_image_meta(image_file)
>>> width = metadata["width]
>>> height = metadata["height"]
>>> faceswap_info = metadata["itxt"]
```

lib.image.**read_image_meta_batch**(*filenames*)
Read the Faceswap metadata stored in a batch extracted faces' exif headers.

Leverages multi-threading to load multiple images from disk at the same time leading to vastly reduced image read times. Creates a generator to retrieve filenames with their metadata as they are calculated.

Notes

The order of returned values is non-deterministic so will most likely not be returned in the same order as the filenames

Parameters **filenames** (*list*) – A list of `str` full paths to the images to be loaded.

Yields *tuple* – (**filename** (*str*), **metadata** (*dict*))

Example

```
>>> image_filenames = ["/path/to/image_1.png", "/path/to/image_2.png", "/path/to/
↪image_3.png"]
>>> for filename, meta in read_image_meta_batch(image_filenames):
>>>         <do something>
```

`lib.image.`**`rgb_to_hex`**(*rgb*)

Convert an RGB tuple to it's hex counterpart.

Parameters **rgb** (*tuple*) – The (*R*, *G*, *B*) integer values to convert (e.g. *(0, 255, 255)*)

Returns The 6 digit hex code with leading # applied

Return type str

`lib.image.`**`update_existing_metadata`**(*filename*, *metadata*)

Update the png header metadata for an existing .png extracted face file on the filesystem.

Parameters

- **filename** (*str*) – The full path to the face to be updated
- **metadata** (*dict or bytes*) – The dictionary to write to the header. Can be pre-encoded as utf-8.

1.1.7 logger module

Logging Functions for Faceswap.

class `lib.logger.`**`FaceswapFormatter`**(*fmt=None*, *datefmt=None*, *style='%'*)

Bases: `logging.Formatter`

Overrides the standard `logging.Formatter`.

Strip newlines from incoming log messages.

Rewrites some upstream warning messages to debug level to avoid spamming the console.

`format`(*record*)

Strip new lines from log records and rewrite certain warning messages to debug level.

Parameters **record** (`logging.LogRecord`) – The incoming log record to be formatted for entry into the logger.

Returns The formatted log message

Return type str

class `lib.logger.`**`FaceswapLogger`**(*name*)

Bases: `logging.Logger`

A standard `logging.logger` with additional "verbose" and "trace" levels added.

`trace`(*msg*, **args*, ***kwargs*)

Create a log message at severity level 5.

Parameters

- **msg** (*str*) – The log message to be recorded at Trace level
- **args** (*tuple*) – Standard logging arguments
- **kwargs** (*dict*) – Standard logging key word arguments

verbose(*msg*, **args*, ***kwargs*)
Create a log message at severity level 15.

Parameters

- **msg** (*str*) – The log message to be recorded at Verbose level
- **args** (*tuple*) – Standard logging arguments
- **kwargs** (*dict*) – Standard logging key word arguments

class lib.logger.**RollingBuffer**
Bases: collections.deque

File-like that keeps a certain number of lines of text in memory for writing out to the crash log.

write(*buffer*)
Splits lines from the incoming buffer and writes them out to the rolling buffer.

Parameters **buffer** (*str*) – The log messages to write to the rolling buffer

class lib.logger.**TqdmHandler**(*stream=None*)
Bases: logging.StreamHandler

Overrides logging.StreamHandler to use tqdm.tqdm.write() rather than writing to sys.stderr() so that log messages do not mess up tqdm progress bars.

emit(*record*)
Format the incoming message and pass to tqdm.tqdm.write().

Parameters **record** (logging.LogRecord) – The incoming log record to be formatted for entry into the logger.

lib.logger.**crash_log**()
On a crash, write out the contents of _DEBUG_BUFFER() containing the last 100 lines of debug messages to a crash report in the root Faceswap folder.

Returns The filename of the file that contains the crash report

Return type str

lib.logger.**get_loglevel**(*loglevel*)
Check whether a valid log level has been supplied, and return the numeric log level that corresponds to the given string level.

Parameters **loglevel** (*str*) – The loglevel that has been requested

Returns The numeric representation of the given loglevel

Return type int

lib.logger.**log_setup**(*loglevel*, *log_file*, *command*, *is_gui=False*)
Set up logging for Faceswap.

Sets up the root logger, the formatting for the crash logger and the file logger, and sets up the crash, file and stream log handlers.

Parameters

- **loglevel** (*str*) – The requested log level that Faceswap should be run at.

- **log_file** (*str*) – The location of the log file to write Faceswap's log to
- **command** (*str*) – The Faceswap command that is being run. Used to dictate whether the log file should have "_gui" appended to the filename or not.
- **is_gui** (*bool, optional*) – Whether Faceswap is running in the GUI or not. Dictates where the stream handler should output messages to. Default: `False`

1.1.8 model package

The Model Package handles interfacing with the neural network backend and holds custom objects.

Contents

model.backup_restore module

Functions for backing up, restoring and creating model snapshots.

class lib.model.backup_restore.**Backup**(*model_dir*, *model_name*)
Bases: object

Performs the back up of models at each save iteration, and the restoring of models from their back up location.

Parameters

- **model_dir** (*str*) – The folder that contains the model to be backed up
- **model_name** (*str*) – The name of the model that is to be backed up

static backup_model(*full_path*)
Backup a model file.

The backed up file is saved with the original filename in the original location with *.bk* appended to the end of the name.

Parameters **full_path** (*str*) – The full path to a *.h5* model file or a *.json* state file

restore()
Restores a model from backup.

The original model files are migrated into a folder within the original model folder named *<model_name>_archived_<timestamp>*. The *.bk* backup files are then moved to the location of the previously existing model files. Logs that were generated after the the last backup was taken are removed.

snapshot_models(*iterations*)
Take a snapshot of the model at the current state and back it up.

The snapshot is a copy of the model folder located in the same root location as the original model file, with the number of iterations appended to the end of the folder name.

Parameters **iterations** (*int*) – The number of iterations that the model has trained when performing the snapshot.

model.initializers module

Module Summary

ConvolutionAware	Initializer that generates orthogonal convolution filters in the Fourier space.
ICNR	ICNR initializer for checkerboard artifact free sub pixel convolution
compute_fans	Computes the number of input and output units for a weight shape.

Custom Initializers for faceswap.py

class lib.model.initializers.**ConvolutionAware**(*eps_std=0.05*, *seed=None*, *initialized=False*)

Bases: keras.initializers.Initializer

Initializer that generates orthogonal convolution filters in the Fourier space. If this initializer is passed a shape that is not 3D or 4D, orthogonal initialization will be used.

Adapted, fixed and optimized from: https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/initializers/convaware.py

Parameters

- **eps_std** (*float, optional*) – The Standard deviation for the random normal noise used to break symmetry in the inverse Fourier transform. Default: 0.05
- **seed** (*int, optional*) – Used to seed the random generator. Default: None
- **initialized** (*bool, optional*) – This should always be set to False. To avoid Keras re-calculating the values every time the model is loaded, this parameter is internally set on first time initialization. Default:False

Returns The modified kernel weights

Return type tensor

References

Armen Aghajanyan, https://arxiv.org/abs/1702.06295

get_config()

Return the Convolutional Aware Initializer configuration.

Returns The configuration for ICNR Initialization

Return type dict

class lib.model.initializers.**ICNR**(*initializer*, *scale=2*)

Bases: keras.initializers.Initializer

ICNR initializer for checkerboard artifact free sub pixel convolution

Parameters

- **initializer** (keras.initializers.Initializer) – The initializer used for sub kernels (orthogonal, glorot uniform, etc.)
- **scale** (*int, optional*) – scaling factor of sub pixel convolution (up sampling from 8x8 to 16x16 is scale 2). Default: *2*

Returns The modified kernel weights

Return type tensor

Example

```
>>> x = conv2d(... weights_initializer=ICNR(initializer=he_uniform(), scale=2))
```

References

Andrew Aitken et al. Checkerboard artifact free sub-pixel convolution https://arxiv.org/pdf/1707.02937.pdf, https://distill.pub/2016/deconv-checkerboard/

get_config()
Return the ICNR Initializer configuration.

Returns The configuration for ICNR Initialization

Return type dict

lib.model.initializers.**compute_fans**(*shape*, *data_format='channels_last'*)
Computes the number of input and output units for a weight shape.

Ported directly from Keras as the location moves between keras and tensorflow-keras

Parameters

- **shape** (*tuple*) – shape tuple of integers
- **data_format** (*str*) – Image data format to use for convolution kernels. Note that all kernels in Keras are standardized on the *"channels_last"* ordering (even when inputs are set to *"channels_first"*).

Returns A tuple of scalars, *(fan_in, fan_out)*.

Return type tuple

Raises ValueError – In case of invalid *data_format* argument.

model.layers module

Module Summary

GlobalMinPooling2D	Global minimum pooling operation for spatial data.
GlobalStdDevPooling2D	Global standard deviation pooling operation for spatial data.
L2_normalize	Normalizes a tensor w.r.t.
PixelShuffler	PixelShuffler layer for Keras.
ReflectionPadding2D	Reflection-padding layer for 2D input (e.g.

Continued on next page

Table 13 – continued from previous page

SubPixelUpscaling	Sub-pixel convolutional up-scaling layer.

Custom Layers for faceswap.py.

class lib.model.layers.**GlobalMinPooling2D**(*data_format=None*, ***kwargs*)
Bases: lib.model.layers._GlobalPooling2D

Global minimum pooling operation for spatial data.

call(*inputs*, ***kwargs*)
This is where the layer's logic lives.

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments

Returns A tensor or list/tuple of tensors

Return type tensor

class lib.model.layers.**GlobalStdDevPooling2D**(*data_format=None*, ***kwargs*)
Bases: lib.model.layers._GlobalPooling2D

Global standard deviation pooling operation for spatial data.

call(*inputs*, ***kwargs*)
This is where the layer's logic lives.

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments

Returns A tensor or list/tuple of tensors

Return type tensor

class lib.model.layers.**KResizeImages**(*size=2*, *interpolation='nearest'*, ***kwargs*)
Bases: keras.engine.base_layer.Layer

A custom upscale function that uses keras.backend.resize_images to upsample.

Parameters

- **size** (*int or float, optional*) – The scale to upsample to. Default: *2*
- **interpolation** (*["nearest", "bilinear"], optional*) – The interpolation to use. Default: *"nearest"*
- **kwargs** (*dict*) – The standard Keras Layer keyword arguments (if any)

call(*inputs*, ***kwargs*)
Call the upsample layer

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments. Unused

Returns A tensor or list/tuple of tensors

Return type tensor

compute_output_shape(*input_shape*)
Computes the output shape of the layer.

This is the input shape with size dimensions multiplied by `size`

Parameters **input_shape** (*tuple or list of tuples*) – Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns An input shape tuple

Return type tuple

get_config()
Returns the config of the layer.

Returns A python dictionary containing the layer configuration

Return type dict

class lib.model.layers.**L2_normalize**(*axis*, ***kwargs*)
Bases: `keras.engine.base_layer.Layer`

Normalizes a tensor w.r.t. the L2 norm alongside the specified axis.

Parameters

- **axis** (*int*) – The axis to perform normalization across
- **kwargs** (*dict*) – The standard Keras Layer keyword arguments (if any)

call(*inputs*)
This is where the layer's logic lives.

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments

Returns A tensor or list/tuple of tensors

Return type tensor

get_config()
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstated later (without its trained weights) from this configuration.

The configuration of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns A python dictionary containing the layer configuration

Return type dict

class lib.model.layers.**PixelShuffler**(*size=(2, 2)*, *data_format=None*, ***kwargs*)
Bases: `keras.engine.base_layer.Layer`

PixelShuffler layer for Keras.

This layer requires a Convolution2D prior to it, having output filters computed according to the formula $filters = k * (scale_factor * scale_factor)$ where *k* is a user defined number of filters (generally larger than 32) and *scale_factor* is the up-scaling factor (generally 2).

This layer performs the depth to space operation on the convolution filters, and returns a tensor with the size as defined below.

Notes

In practice, it is useful to have a second convolution layer after the *PixelShuffler* layer to speed up the learning process. However, if you are stacking multiple *PixelShuffler* blocks, it may increase the number of parameters greatly, so the Convolution layer after *PixelShuffler* layer can be removed.

Example

```
>>> # A standard sub-pixel up-scaling block
>>> x = Convolution2D(256, 3, 3, padding="same", activation="relu")(...)
>>> u = PixelShuffler(size=(2, 2))(x)
[Optional]
>>> x = Convolution2D(256, 3, 3, padding="same", activation="relu")(u)
```

Parameters

- **size** (*tuple, optional*) – The (*h*, *w*) scaling factor for up-scaling. Default: *(2, 2)*
- **data_format** (["channels_first", "channels_last", None], optional) – The data format for the input. Default: None
- **kwargs** (*dict*) – The standard Keras Layer keyword arguments (if any)

References

https://gist.github.com/t-ae/6e1016cc188104d123676ccef3264981

call(*inputs*, ***kwargs*)
This is where the layer's logic lives.

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments. Unused

Returns A tensor or list/tuple of tensors

Return type tensor

compute_output_shape(*input_shape*)
Computes the output shape of the layer.

Assumes that the layer will be built to match that input shape provided.

Parameters **input_shape** (*tuple or list of tuples*) – Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns An input shape tuple

Return type tuple

get_config()
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstated later (without its trained weights) from this configuration.

The configuration of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns A python dictionary containing the layer configuration

Return type dict

class lib.model.layers.**ReflectionPadding2D**(*stride=2*, *kernel_size=5*, ***kwargs*)
Bases: keras.engine.base_layer.Layer

Reflection-padding layer for 2D input (e.g. picture).

This layer can add rows and columns at the top, bottom, left and right side of an image tensor.

Parameters

- **stride** (*int, optional*) – The stride of the following convolution. Default: *2*
- **kernel_size** (*int, optional*) – The kernel size of the following convolution. Default: *5*
- **kwargs** (*dict*) – The standard Keras Layer keyword arguments (if any)

build(*input_shape*)
Creates the layer weights.

Must be implemented on all layers that have weights.

Parameters **input_shape** (*tensor*) – Keras tensor (future input to layer) or list/tuple of Keras tensors to reference for weight shape computations.

call(*var_x*, *mask=None*)
This is where the layer's logic lives.

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments

Returns A tensor or list/tuple of tensors

Return type tensor

compute_output_shape(*input_shape*)
Computes the output shape of the layer.

Assumes that the layer will be built to match that input shape provided.

Parameters **input_shape** (*tuple or list of tuples*) – Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns An input shape tuple

Return type tuple

get_config()
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstated later (without its trained weights) from this configuration.

The configuration of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns A python dictionary containing the layer configuration

Return type dict

class lib.model.layers.**SubPixelUpscaling**(*scale_factor=2*, *data_format=None*, ***kwargs*)
Bases: keras.engine.base_layer.Layer

Sub-pixel convolutional up-scaling layer.

This layer requires a Convolution2D prior to it, having output filters computed according to the formula $filters = k * (scale_factor * scale_factor)$ where *k* is a user defined number of filters (generally larger than 32) and *scale_factor* is the up-scaling factor (generally 2).

This layer performs the depth to space operation on the convolution filters, and returns a tensor with the size as defined below.

Notes

This method is deprecated as it just performs the same as *PixelShuffler* using explicit Tensorflow ops. The method is kept in the repository to support legacy models that have been created with this layer.

In practice, it is useful to have a second convolution layer after the *SubPixelUpscaling* layer to speed up the learning process. However, if you are stacking multiple *SubPixelUpscaling* blocks, it may increase the number of parameters greatly, so the Convolution layer after *SubPixelUpscaling* layer can be removed.

Example

```
>>> # A standard sub-pixel up-scaling block
>>> x = Convolution2D(256, 3, 3, padding="same", activation="relu")(...)
>>> u = SubPixelUpscaling(scale_factor=2)(x)
[Optional]
>>> x = Convolution2D(256, 3, 3, padding="same", activation="relu")(u)
```

Parameters

- **size** (*int, optional*) – The up-scaling factor. Default: *2*
- **data_format** ([“channels_first”, “channels_last”, None], optional) – The data format for the input. Default: None
- **kwargs** (*dict*) – The standard Keras Layer keyword arguments (if any)

References

based on the paper “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network” (https://arxiv.org/abs/1609.05158).

build(*input_shape*)
Creates the layer weights.

Must be implemented on all layers that have weights.

Parameters **input_shape** (*tensor*) – Keras tensor (future input to layer) or list/tuple of Keras tensors to reference for weight shape computations.

call(*inputs*, ***kwargs*)
This is where the layer's logic lives.

Parameters

- **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors
- **kwargs** (*dict*) – Additional keyword arguments. Unused

Returns A tensor or list/tuple of tensors

Return type tensor

compute_output_shape(*input_shape*)
Computes the output shape of the layer.

Assumes that the layer will be built to match that input shape provided.

Parameters **input_shape** (*tuple or list of tuples*) – Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns An input shape tuple

Return type tuple

get_config()
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstated later (without its trained weights) from this configuration.

The configuration of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns A python dictionary containing the layer configuration

Return type dict

class lib.model.layers.**Swish**(*beta=1.0*, ***kwargs*)
Bases: keras.engine.base_layer.Layer

Swish Activation Layer implementation for Keras.

Parameters

- **beta** (*float, optional*) – The beta value to apply to the activation function. Default: *1.0*
- **kwargs** (*dict*) – The standard Keras Layer keyword arguments (if any)

References

Swish: a Self-Gated Activation Function: https://arxiv.org/abs/1710.05941v1

call(*inputs*)
Call the Swish Activation function.

Parameters **inputs** (*tensor*) – Input tensor, or list/tuple of input tensors

get_config()
Returns the config of the layer.

Adds the `beta` to config.

Returns A python dictionary containing the layer configuration

Return type dict

model.losses module

The losses listed here are generated from the docstrings in *lib.model.losses_tf*, however the functions are excactly the same for `lib.model.losses_plaid`. The correct loss module will be imported as `lib.model.losses` depending on the backend in use.

Module Summary

DSSIMObjective	DSSIM Loss Function
GeneralizedLoss	Generalized function used to return a large variety of mathematical loss functions.
GMSDLoss	Gradient Magnitude Similarity Deviation Loss.
GradientLoss	Gradient Loss Function.
LInfNorm	Calculate the L-inf norm as a loss function.
LossWrapper	A wrapper class for multiple keras losses to enable multiple weighted loss functions on a single output.

Custom Loss Functions for faceswap.py

class lib.model.losses_tf.**DSSIMObjective**(*k_1=0.01*, *k_2=0.03*, *kernel_size=3*, *max_value=1.0*)

Bases: `tensorflow.python.keras.losses.Loss`

DSSIM Loss Function

Difference of Structural Similarity (DSSIM loss function). Clipped between 0 and 0.5

Parameters

- **k_1** (*float, optional*) – Parameter of the SSIM. Default: *0.01*
- **k_2** (*float, optional*) – Parameter of the SSIM. Default: *0.03*
- **kernel_size** (*int, optional*) – Size of the sliding window Default: *3*
- **max_value** (*float, optional*) – Max value of the output. Default: *1.0*

Notes

You should add a regularization term like a l2 loss in addition to this one.

References

https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/losses/dssim.py

MIT License

Copyright (c) 2017 Fariz Rahman

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

call(*y_true*, *y_pred*)
Call the DSSIM Loss Function.

Parameters

- **y_true** (*tensor or variable*) – The ground truth value
- **y_pred** (*tensor or variable*) – The predicted value

Returns The DSSIM Loss value

Return type tensor

Notes

There are additional parameters for this function. some of the 'modes' for edge behavior do not yet have a gradient definition in the Theano tree and cannot be used for learning

extract_image_patches(*input_tensor*, *k_sizes*, *s_sizes*, *padding='same'*, *data_format='channels_last'*)
Extract the patches from an image.

Parameters

- **input_tensor** (*tensor*) – The input image
- **k_sizes** (*tuple*) – 2-d tuple with the kernel size
- **s_sizes** (*tuple*) – 2-d tuple with the strides size
- **padding** (*str, optional*) – *"same"* or *"valid"*. Default: *"same"*
- **data_format** (*str, optional.*) – *"channels_last"* or *"channels_first"*. Default: *"channels_last"*

Returns Tensorflow ==> (batch_size, w, h, k_w, k_h, c) Theano ==> (batch_size, w, h, c, k_w, k_h)

Return type The (k_w, k_h) patches extracted

class lib.model.losses_tf.**GMSDLoss**(*reduction='auto'*, *name=None*)
Bases: tensorflow.python.keras.losses.Loss

Gradient Magnitude Similarity Deviation Loss.

Improved image quality metric over MS-SSIM with easier calculations

References

http://www4.comp.polyu.edu.hk/~cslzhang/IQA/GMSD/GMSD.htm https://arxiv.org/ftp/arxiv/papers/1308/1308.3052.pdf

call(*y_true*, *y_pred*)
Return the Gradient Magnitude Similarity Deviation Loss.

Parameters

- **y_true** (*tensor or variable*) – The ground truth value
- **y_pred** (*tensor or variable*) – The predicted value

Returns The loss value

Return type tensor

class lib.model.losses_tf.**GeneralizedLoss**(*alpha=1.0*, *beta=0.00392156862745098*)
Bases: tensorflow.python.keras.losses.Loss

Generalized function used to return a large variety of mathematical loss functions.

The primary benefit is a smooth, differentiable version of L1 loss.

References

Barron, J. A More General Robust Loss Function - https://arxiv.org/pdf/1701.03077.pdf

Example

```
>>> a=1.0, x>>c , c=1.0/255.0  # will give a smoothly differentiable version of
↪L1 / MAE loss
>>> a=1.999999 (limit as a->2), beta=1.0/255.0 # will give L2 / RMSE loss
```

Parameters

- **alpha** (*float, optional*) – Penalty factor. Larger number give larger weight to large deviations. Default: *1.0*
- **beta** (*float, optional*) – Scale factor used to adjust to the input scale (i.e. inputs of mean *1e-4* or *256*). Default: *1.0/255.0*

call(*y_true*, *y_pred*)
Call the Generalized Loss Function

Parameters

- **y_true** (*tensor or variable*) – The ground truth value
- **y_pred** (*tensor or variable*) – The predicted value

Returns The loss value from the results of function(y_pred - y_true)

Return type tensor

class lib.model.losses_tf.**GradientLoss**
Bases: tensorflow.python.keras.losses.Loss

Gradient Loss Function.

Calculates the first and second order gradient difference between pixels of an image in the x and y dimensions. These gradients are then compared between the ground truth and the predicted image and the difference is taken. When used as a loss, its minimization will result in predicted images approaching the same level of sharpness / blurriness as the ground truth.

References

TV+TV2 Regularization with Non-Convex Sparseness-Inducing Penalty for Image Restoration, Chengwu Lu & Hua Huang, 2014 - http://downloads.hindawi.com/journals/mpe/2014/790547.pdf

call(*y_true*, *y_pred*)
Call the gradient loss function.

Parameters

- **y_true** (*tensor or variable*) – The ground truth value
- **y_pred** (*tensor or variable*) – The predicted value

Returns The loss value

Return type tensor

class lib.model.losses_tf.**LInfNorm**(*reduction='auto'*, *name=None*)
Bases: tensorflow.python.keras.losses.Loss

Calculate the L-inf norm as a loss function.

call(*y_true*, *y_pred*)
Call the L-inf norm loss function.

Parameters

- **y_true** (*tensor or variable*) – The ground truth value
- **y_pred** (*tensor or variable*) – The predicted value

Returns The loss value

Return type tensor

class lib.model.losses_tf.**LossWrapper**
Bases: tensorflow.python.keras.losses.Loss

A wrapper class for multiple keras losses to enable multiple weighted loss functions on a single output.

add_loss(*function*, *weight=1.0*, *mask_channel=-1*)
Add the given loss function with the given weight to the loss function chain.

Parameters

- **function** (keras.losses.Loss) – The loss function to add to the loss chain
- **weight** (*float, optional*) – The weighting to apply to the loss function. Default: *1.0*
- **mask_channel** (*int, optional*) – The channel in the *y_true* image that the mask exists in. Set to *-1* if there is no mask for the given loss function. Default: *-1*

call(*y_true*, *y_pred*)
Call the sub loss functions for the loss wrapper.

Weights are returned as the weighted sum of the chosen losses.

If a mask is being applied to the loss, then the appropriate mask is extracted from y_true and added as the 4th channel being passed to the penalized loss function.

Parameters

- **y_true** (*tensor or variable*) – The ground truth value
- **y_pred** (*tensor or variable*) – The predicted value

Returns The final loss value

Return type tensor

model.nn_blocks module

Module Summary

Conv2D	A standard Keras Convolution 2D layer with parameters updated to be more appropriate for Faceswap architecture.
Conv2DBlock	A standard Convolution 2D layer which applies user specified configuration to the layer.
Conv2DOutput	A Convolution 2D layer that separates out the activation layer to explicitly set the data type on the activation to float 32 to fully support mixed precision training.
ResidualBlock	Residual block from dfaker.
SeparableConv2DBlock	Seperable Convolution Block.
Upscale2xBlock	Custom hybrid upscale layer for sub-pixel up-scaling.
UpscaleBlock	An upscale layer for sub-pixel up-scaling.
set_config	Set the global configuration parameters from the user's config file.

Neural Network Blocks for faceswap.py.

class lib.model.nn_blocks.**Conv2D**(**args*, *padding='same'*, *check_icnr_init=False*, ***kwargs*)

Bases: keras.layers.convolutional.Conv2D

A standard Keras Convolution 2D layer with parameters updated to be more appropriate for Faceswap architecture.

Parameters are the same, with the same defaults, as a standard keras.layers.Conv2D except where listed below. The default initializer is updated to *he_uniform* or *convolutional aware* based on user configuration settings.

Parameters

- **padding** (*str, optional*) – One of *"valid"* or *"same"* (case-insensitive). Default: *"same"*. Note that *"same"* is slightly inconsistent across backends with *strides* != 1, as described here.
- **check_icnr_init** (*bool*, optional) – True if the user configuration options should be checked to apply ICNR initialization to the layer. This should only be passed in from *UpscaleBlock* layers. Default: False

class lib.model.nn_blocks.**Conv2DBlock**(*filters*, *kernel_size=5*, *strides=2*, *padding='same'*, *normalization=None*, *activation='leakyrelu'*, *use_depthwise=False*, ***kwargs*)

Bases: object

A standard Convolution 2D layer which applies user specified configuration to the layer.

Adds reflection padding if it has been selected by the user, and other post-processing if requested by the plugin.

Adds instance normalization if requested. Adds a LeakyReLU if a residual block follows.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)
- **kernel_size** (*int, optional*) – An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. NB: If *use_depthwise* is `True` then a value must still be provided here, but it will be ignored. Default: 5
- **strides** (*tuple or int, optional*) – An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: *2*
- **padding** (*["valid", "same"], optional*) – The padding to use. NB: If reflect padding has been selected in the user configuration options, then this argument will be ignored in favor of reflect padding. Default: *"same"*
- **normalization** (str or `None`, optional) – Normalization to apply after the Convolution Layer. Select one of "batch" or "instance". Set to `None` to not apply normalization. Default: `None`
- **activation** (str or `None`, optional) – The activation function to use. This is applied at the end of the convolution block. Select one of *"leakyrelu"*, *"prelu"* or *"swish"*. Set to `None` to not apply an activation function. Default: *"leakyrelu"*
- **use_depthwise** (*bool, optional*) – Set to `True` to use a Depthwise Convolution 2D layer rather than a standard Convolution 2D layer. Default: `False`
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Convolutional 2D layer

class lib.model.nn_blocks.**Conv2DOutput**(*filters*, *kernel_size*, *activation='sigmoid'*, *padding='same'*, ***kwargs*)

Bases: `object`

A Convolution 2D layer that separates out the activation layer to explicitly set the data type on the activation to float 32 to fully support mixed precision training.

The Convolution 2D layer uses default parameters to be more appropriate for Faceswap architecture.

Parameters are the same, with the same defaults, as a standard `keras.layers.Conv2D` except where listed below. The default initializer is updated to he_uniform or convolutional aware based on user config settings.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)
- **kernel_size** (*int or tuple/list of 2 ints*) – The height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **activation** (*str, optional*) – The activation function to apply to the output. Default: *"sigmoid"*

- **padding** (*str, optional*) – One of *"valid"* or *"same"* (case-insensitive). Default: *"same"*. Note that *"same"* is slightly inconsistent across backends with *strides* != 1, as described here.
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Convolutional 2D layer

class lib.model.nn_blocks.**DepthwiseConv2D**(**args*, *padding='same'*, *check_icnr_init=False*, ***kwargs*)

Bases: keras.layers.convolutional.DepthwiseConv2D

A standard Keras Depthwise Convolution 2D layer with parameters updated to be more appropriate for Faceswap architecture.

Parameters are the same, with the same defaults, as a standard keras.layers.DepthwiseConv2D except where listed below. The default initializer is updated to *he_uniform* or *convolutional aware* based on user configuration settings.

Parameters

- **padding** (*str, optional*) – One of *"valid"* or *"same"* (case-insensitive). Default: *"same"*. Note that *"same"* is slightly inconsistent across backends with *strides* != 1, as described here.
- **check_icnr_init** (*bool*, optional) – True if the user configuration options should be checked to apply ICNR initialization to the layer. This should only be passed in from *UpscaleBlock* layers. Default: False

class lib.model.nn_blocks.**ResidualBlock**(*filters*, *kernel_size=3*, *padding='same'*, ***kwargs*)

Bases: object

Residual block from dfaker.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)
- **kernel_size** (*int, optional*) – An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. Default: 3
- **padding** (*["valid", "same"], optional*) – The padding to use. Default: *"same"*
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Convolutional 2D layer

Returns The output tensor from the Upscale layer

Return type tensor

class lib.model.nn_blocks.**SeparableConv2DBlock**(*filters*, *kernel_size=5*, *strides=2*, ***kwargs*)

Bases: object

Seperable Convolution Block.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)

- **kernel_size** (*int, optional*) – An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. Default: 5
- **strides** (*tuple or int, optional*) – An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: *2*
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Separable Convolutional 2D layer

class lib.model.nn_blocks.**Upscale2xBlock**(*filters*, *kernel_size=3*, *padding='same'*, *activation='leakyrelu'*, *interpolation='bilinear'*, *sr_ratio=0.5*, *scale_factor=2*, *fast=False*, ***kwargs*)

Bases: object

Custom hybrid upscale layer for sub-pixel up-scaling.

Most of up-scaling is approximating lighting gradients which can be accurately achieved using linear fitting. This layer attempts to improve memory consumption by splitting with bilinear and convolutional layers so that the sub-pixel update will get details whilst the bilinear filter will get lighting.

Adds reflection padding if it has been selected by the user, and other post-processing if requested by the plugin.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)
- **kernel_size** (*int, optional*) – An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. Default: 3
- **padding** (*["valid", "same"], optional*) – The padding to use. Default: *"same"*
- **activation** (str or None, optional) – The activation function to use. This is applied at the end of the convolution block. Select one of *"leakyrelu"*, *"prelu"* or *"swish"*. Set to None to not apply an activation function. Default: *"leakyrelu"*
- **interpolation** (*["nearest", "bilinear"], optional*) – Interpolation to use for up-sampling. Default: *"bilinear"*
- **scale_factor** (*int, optional*) – The amount to upscale the image. Default: *2*
- **sr_ratio** (*float, optional*) – The proportion of super resolution (pixel shuffler) filters to use. Non-fast mode only. Default: *0.5*
- **fast** (*bool, optional*) – Use a faster up-scaling method that may appear more rugged. Default: False
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Convolutional 2D layer

class lib.model.nn_blocks.**UpscaleBlock**(*filters*, *kernel_size=3*, *padding='same'*, *scale_factor=2*, *normalization=None*, *activation='leakyrelu'*, ***kwargs*)

Bases: object

An upscale layer for sub-pixel up-scaling.

Adds reflection padding if it has been selected by the user, and other post-processing if requested by the plugin.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)
- **kernel_size** (*int, optional*) – An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. Default: 3
- **padding** (*["valid", "same"], optional*) – The padding to use. NB: If reflect padding has been selected in the user configuration options, then this argument will be ignored in favor of reflect padding. Default: *"same"*
- **scale_factor** (*int, optional*) – The amount to upscale the image. Default: *2*
- **normalization** (str or `None`, optional) – Normalization to apply after the Convolution Layer. Select one of "batch" or "instance". Set to `None` to not apply normalization. Default: `None`
- **activation** (str or `None`, optional) – The activation function to use. This is applied at the end of the convolution block. Select one of *"leakyrelu"*, *"prelu"* or *"swish"*. Set to `None` to not apply an activation function. Default: *"leakyrelu"*
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Convolutional 2D layer

class lib.model.nn_blocks.**UpscaleResizeImagesBlock**(*filters*, *kernel_size=3*, *padding='same'*, *activation='leakyrelu'*, *scale_factor=2*, *interpolation='bilinear'*)

Bases: `object`

Upscale block that uses the Keras Backend function resize_images to perform the up scaling Similar in methodology to the *Upscale2xBlock*

Adds reflection padding if it has been selected by the user, and other post-processing if requested by the plugin.

Parameters

- **filters** (*int*) – The dimensionality of the output space (i.e. the number of output filters in the convolution)
- **kernel_size** (*int, optional*) – An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions. Default: 3
- **padding** (*["valid", "same"], optional*) – The padding to use. Default: *"same"*
- **activation** (str or `None`, optional) – The activation function to use. This is applied at the end of the convolution block. Select one of *"leakyrelu"*, *"prelu"* or *"swish"*. Set to `None` to not apply an activation function. Default: *"leakyrelu"*
- **scale_factor** (*int, optional*) – The amount to upscale the image. Default: *2*
- **interpolation** (*["nearest", "bilinear"], optional*) – Interpolation to use for up-sampling. Default: *"bilinear"*
- **kwargs** (*dict*) – Any additional Keras standard layer keyword arguments to pass to the Convolutional 2D layer

lib.model.nn_blocks.**set_config**(*configuration*)

Set the global configuration parameters from the user's config file.

These options are used when creating layers for new models.

Parameters **configuration** (*dict*) – The configuration options that exist in the training configuration files that pertain specifically to Custom Faceswap Layers. The keys should be: *icnr_init*, *conv_aware_init* and 'reflect_padding'

model.normalization module

Module Summary

InstanceNormalization	Instance normalization layer (Lei Ba et al, 2016, Ulyanov et al., 2016).

Conditional imports depending on whether the AMD version is installed or not

model.session module

Settings manager for Keras Backend

class lib.model.session.**KSession**(*name*, *model_path*, *model_kwargs=None*, *allow_growth=False*, *exclude_gpus=None*)

Bases: object

Handles the settings of backend sessions for inference models.

This class acts as a wrapper for various keras.Model() functions, ensuring that actions performed on a model are handled consistently and can be performed in parallel in separate threads.

This is an early implementation of this class, and should be expanded out over time with relevant *AMD*, *CPU* and *NVIDIA* backend methods.

Notes

The documentation refers to keras. This is a pseudonym for either keras or tensorflow.keras depending on the backend in use.

Parameters

- **name** (*str*) – The name of the model that is to be loaded
- **model_path** (*str*) – The path to the keras model file
- **model_kwargs** (*dict, optional*) – Any kwargs that need to be passed to keras.models.load_models(). Default: None
- **allow_growth** (*bool, optional*) – Enable the Tensorflow GPU allow_growth configuration option. This option prevents Tensorflow from allocating all of the GPU VRAM, but can lead to higher fragmentation and slower performance. Default: False
- **exclude_gpus** (*list, optional*) – A list of indices correlating to connected GPUs that Tensorflow should not use. Pass None to not exclude any GPUs. Default: None

append_softmax_activation(*layer_index=-1*)

Append a softmax activation layer to a model

Occasionally a softmax activation layer needs to be added to a model's output. This is a convenience function to append this layer to the loaded model.

Parameters **layer_index** (*int, optional*) – The layer index of the model to select the output from to use as an input to the softmax activation layer. Default: *-1* (The final layer of the model)

define_model (*function*)
Defines a model from the given function.

This method acts as a wrapper for `keras.models.Model()`.

Parameters **function** (*function*) – A function that defines a `keras.Model` and returns it's `inputs` and `outputs`. The function that generates these results should be passed in, NOT the results themselves, as the function needs to be executed within the correct context.

load_model()
Loads a model.

This method is a wrapper for `keras.models.load_model()`. Loads a model and its weights from `model_path` defined during initialization of this class. Any additional `kwargs` to be passed to `keras.models.load_model()` should also be defined during initialization of the class.

For Tensorflow backends, the *make_predict_function* method is called on the model to make it thread safe.

load_model_weights()
Load model weights for a defined model inside the correct session.

This method is a wrapper for `keras.load_weights()`. Once a model has been defined in *define_model()* this method can be called to load its weights from the `model_path` defined during initialization of this class.

For Tensorflow backends, the *make_predict_function* method is called on the model to make it thread safe.

predict (*feed*, *batch_size=None*)
Get predictions from the model.

This method is a wrapper for `keras.predict()` function. For Tensorflow backends this is a straight call to the predict function. For PlaidML backends, this attempts to optimize the inference batch sizes to reduce the number of kernels that need to be compiled.

Parameters **feed** (*numpy.ndarray or list*) – The feed to be provided to the model as input. This should be a `numpy.ndarray` for single inputs or a *list* of `numpy.ndarray` objects for multiple inputs.

1.1.9 plaidml_tools module

PlaidML tools.

Statistics and setup for PlaidML on AMD devices.

This module must be kept separate from Keras, and be called prior to any Keras import, as the plaidML Keras backend is set from this module.

class lib.plaidml_tools.**PlaidMLStats**(*log_level='INFO'*, *log=True*)
Bases: `object`

Handles the initialization of PlaidML and the returning of GPU information for connected cards from the PlaidML library.

This class is initialized early in Faceswap's Launch process from *setup_plaidml()*, with statistics made available from *GPUStats*

Parameters

- **log_level** (*str, optional*) – The requested Faceswap log level. Also dictates the level that PlaidML logging is set at. Default:*"INFO"*
- **log** (*bool, optional*) – Whether this class should output to the logger. If statistics are being accessed during a crash, then the logger may not be available, so this gives the option to turn logging off in those kinds of situations. Default:`True`

active_devices
List of device indices for active GPU devices.

Type list

device_count
The total number of GPU Devices discovered.

Type int

devices
The `pladml._DeviceConfig` objects for GPUs that PlaidML has discovered.

Type list

drivers
The driver versions for each GPU device that PlaidML has discovered.

Type list

names
The name of each GPU device that PlaidML has discovered.

Type list

vram
The VRAM of each GPU device that PlaidML has discovered.

Type list

`lib.plaidml_tools.`**setup_plaidml**(*log_level*, *exclude_devices*)
Setup PlaidML for AMD Cards.

Sets the Keras backend to PlaidML, loads the plaidML backend and makes GPU Device information from PlaidML available to *GPUStats*.

Parameters

- **log_level** (*str*) – Faceswap's log level. Used for setting the log level inside PlaidML
- **exclude_devices** (*list*) – A list of integers of device IDs that should not be used by Faceswap

1.1.10 serializer module

Module Summary

Serializer	A convenience class for various serializers.
get_serializer	Obtain a serializer object
get_serializer_from_filename	Obtain a serializer object from a filename

Module

Library for serializing python objects to and from various different serializer formats

class lib.serializer.**Serializer**
Bases: object

A convenience class for various serializers.

This class should not be called directly as it acts as the parent for various serializers. All serializers should be called from *get_serializer()* or *get_serializer_from_filename()*

Example

```
>>> from lib.serializer import get_serializer
>>> serializer = get_serializer('json')
>>> json_file = '/path/to/json/file.json'
>>> data = serializer.load(json_file)
>>> serializer.save(json_file, data)
```

file_extension
The file extension of the serializer

Type str

load(*filename*)
Load data from an existing serialized file

Parameters **filename** (*str*) – The path to the serialized file

Returns **data** – The data in a python object format

Return type varies

Example

```
>>> serializer = get_serializer('json')
>>> json_file = '/path/to/json/file.json'
>>> data = serializer.load(json_file)
```

marshal(*data*)
Serialize an object

Parameters **data** (*varies*) – The data that is to be serialized

Returns **data** – The data in a the serialized data format

Return type varies

Example

```
>>> serializer = get_serializer('json')
>>> data ['foo', 'bar']
>>> json_data = serializer.marshal(data)
```

save(*filename*, *data*)
Serialize data and save to a file

Parameters

- **filename** (*str*) – The path to where the serialized file should be saved
- **data** (*varies*) – The data that is to be serialized to file

Example

```
>>> serializer = get_serializer('json')
>>> data ['foo', 'bar']
>>> json_file = '/path/to/json/file.json'
>>> serializer.save(json_file, data)
```

unmarshal(*serialized_data*)
Unserialize data to its original object type

Parameters **serialized_data** (*varies*) – Data in serializer format that is to be unmarshalled to its original object

Returns **data** – The data in a python object format

Return type varies

Example

```
>>> serializer = get_serializer('json')
>>> json_data = <json object>
>>> data = serializer.unmarshal(json_data)
```

lib.serializer.**get_serializer**(*serializer*)
Obtain a serializer object

Parameters **serializer** (*{'json', 'pickle', yaml', 'npy', 'compressed'}*) – The required serializer format

Returns **serializer** – A serializer object for handling the requested data format

Return type *Serializer*

Example

```
>>> serializer = get_serializer('json')
```

lib.serializer.**get_serializer_from_filename**(*filename*)
Obtain a serializer object from a filename

Parameters **filename** (*str*) – Filename to determine the serializer type from

Returns **serializer** – A serializer object for handling the requested data format

Return type *Serializer*

Example

```
>>> filename = '/path/to/json/file.json'
>>> serializer = get_serializer_from_filename(filename)
```

1.1.11 sysinfo module

Obtain information about the running system, environment and GPU.

lib.sysinfo.**get_sysinfo**()
Obtain extensive system information stats, formatted into a human readable format. If an error occurs obtaining the system information, then the error message is returned instead.

Returns The system information for the currently running system, formatted for output to console or a log file.

Return type str

1.1.12 training package

The training Package handles the processing of faces for feeding into a Faceswap model.

Contents

augmentation module

Processes the augmentation of images for feeding into a Faceswap model.

class lib.training.augmentation.**ImageAugmentation**(*batchsize*, *is_display*, *input_size*, *output_shapes*, *coverage_ratio*, *config*)

Bases: object

Performs augmentation on batches of training images.

Parameters

- **batchsize** (*int*) – The number of images that will be fed through the augmentation functions at once.
- **is_display** (*bool*) – Whether the images being fed through will be used for Preview or Time-lapse. Disables the "warp" augmentation for these images.
- **input_size** (*int*) – The expected input size for the model. It is assumed that the input to the model is always a square image. This is the size, in pixels, of the *width* and the *height* of the input to the model.
- **output_shapes** (*list*) – A list of tuples defining the output shapes from the model, in the order that the outputs are returned. The tuples should be in (*height*, *width*, *channels*) format.
- **coverage_ratio** (*float*) – The ratio of the training image to be trained on. Dictates how much of the image will be cropped out. E.G: a coverage ratio of 0.625 will result in cropping a 160px box from a 256px image ($256 * 0.625 = 160$)
- **config** (*dict*) – The configuration *dict* generated from config.train.ini containing the trainer plugin configuration options.

initialized
Flag to indicate whether *ImageAugmentation* has been initialized with the training image size in order to cache certain augmentation operations (see *initialize()*)

Type bool

is_display
Flag to indicate whether these augmentations are for time-lapses/preview images (`True`) or standard training data (`False`)

Type bool

color_adjust(*batch*)
Perform color augmentation on the passed in batch.

The color adjustment parameters are set in `config.train.ini`

Parameters **batch** (`numpy.ndarray`) – The batch should be a 4-dimensional array of shape (*batchsize*, *height*, *width*, *3*) and in *BGR* format.

Returns A 4-dimensional array of the same shape as `batch` with color augmentation applied.

Return type `numpy.ndarray`

get_targets(*batch*)
Returns the target images, and masks, if required.

Parameters **batch** (`numpy.ndarray`) – This should be a 4+-dimensional array of training images in the format (*batchsize*, *height*, *width*, *channels*). Targets should be requested after performing image transformations but prior to performing warps.

The 4th channel should be the mask. Any channels above the 4th should be any additional masks that are requested.

Returns

The following keys will be within the returned dictionary:

- **targets** (*list*) - A list of 4-dimensional `numpy.ndarray` s in the order and size of each output of the model as defined in `output_shapes`. The format of these arrays will be (*batchsize*, *height*, *width*, *3*). **NB:** masks are not included in the *targets* list. If masks are to be included in the output they will be returned as their own item from the *masks* key.
- **masks** (`numpy.ndarray`) - A 4-dimensional array containing the target masks in the format (*batchsize*, *height*, *width*, *1*).

Return type dict

initialize(*training_size*)
Initializes the caching of constants for use in various image augmentations.

The training image size is not known prior to loading the images from disk and commencing training, so it cannot be set in the `__init__()` method. When the first training batch is loaded this function should be called to initialize the class and perform various calculations based on this input size to cache certain constants for image augmentation calculations.

Parameters **training_size** (*int*) – The size of the training images stored on disk that are to be fed into *ImageAugmentation*. The training images should always be square and of the same size. This is the size, in pixels, of the *width* and the *height* of the training images.

random_flip(*batch*)
Perform random horizontal flipping on the passed in batch.

The probability of flipping an image is set in `config.train.ini`

Parameters **batch** (`numpy.ndarray`) – The batch should be a 4-dimensional array of shape (*batchsize*, *height*, *width*, *channels*) and in *BGR* format.

Returns A 4-dimensional array of the same shape as `batch` with transformation applied.

Return type `numpy.ndarray`

skip_warp(*batch*)
Returns the images resized and cropped for feeding the model, if warping has been disabled.

Parameters **batch** (numpy.ndarray) – The batch should be a 4-dimensional array of shape (*batchsize*, *height*, *width*, *3*) and in *BGR* format.

Returns The given batch cropped and resized for feeding the model

Return type numpy.ndarray

transform(*batch*)
Perform random transformation on the passed in batch.

The transformation parameters are set in config.train.ini

Parameters **batch** (numpy.ndarray) – The batch should be a 4-dimensional array of shape (*batchsize*, *height*, *width*, *channels*) and in *BGR* format.

Returns A 4-dimensional array of the same shape as batch with transformation applied.

Return type numpy.ndarray

warp(*batch*, *to_landmarks=False*, ***kwargs*)
Perform random warping on the passed in batch by one of two methods.

Parameters

- **batch** (numpy.ndarray) – The batch should be a 4-dimensional array of shape (*batchsize*, *height*, *width*, *3*) and in *BGR* format.
- **to_landmarks** (*bool, optional*) – If False perform standard random warping of the input image. If True perform warping to semi-random similar corresponding landmarks from the other side. Default: False
- **kwargs** (*dict*) – If to_landmarks is True the following additional kwargs must be passed in:
  - **batch_src_points** (numpy.ndarray) - A batch of 68 point landmarks for the source faces. This is a 3-dimensional array in the shape (*batchsize*, *68*, *2*).
  - **batch_dst_points** (numpy.ndarray) - A batch of randomly chosen closest match destination faces landmarks. This is a 3-dimensional array in the shape (*batchsize*, *68*, *2*).

Returns A 4-dimensional array of the same shape as batch with warping applied.

Return type numpy.ndarray

generator module

Handles Data Augmentation for feeding Faceswap Models

class lib.training.generator.**TrainingDataGenerator**(*model_input_size*, *model_output_shapes*, *coverage_ratio*, *color_order*, *augment_color*, *no_flip*, *no_warp*, *warp_to_landmarks*, *config*)

Bases: object

A Training Data Generator for compiling data for feeding to a model.

This class is called from *plugins.train.trainer._base* and launches a background iterator that compiles augmented data, target data and sample data.

Parameters

- **model_input_size** (*int*) – The expected input size for the model. It is assumed that the input to the model is always a square image. This is the size, in pixels, of the *width* and the *height* of the input to the model.
- **model_output_shapes** (*list*) – A list of tuples defining the output shapes from the model, in the order that the outputs are returned. The tuples should be in (*height*, *width*, *channels*) format.
- **coverage_ratio** (*float*) – The ratio of the training image to be trained on. Dictates how much of the image will be cropped out. E.G: a coverage ratio of 0.625 will result in cropping a 160px box from a 256px image ($256 * 0.625 = 160$).
- **color_order** (*["rgb", "bgr"]*) – The color order that the model expects as input
- **augment_color** (*bool*) – `True` if color is to be augmented, otherwise `False`
- **no_flip** (*bool*) – `True` if the image shouldn't be randomly flipped as part of augmentation, otherwise `False`
- **no_warp** (*bool*) – `True` if the image shouldn't be warped as part of augmentation, otherwise `False`
- **warp_to_landmarks** (*bool*) – `True` if the random warp method should warp to similar landmarks from the other side, `False` if the standard random warp method should be used.
- **face_cache** (*dict*) – A thread safe dictionary containing a cache of information relating to all faces being trained on
- **config** (*dict*) – The configuration *dict* generated from `config.train.ini` containing the trainer plugin configuration options.

minibatch_ab(*images*, *batchsize*, *side*, *do_shuffle=True*, *is_preview=False*, *is_timelapse=False*)

A Background iterator to return augmented images, samples and targets.

The exit point from this class and the sole attribute that should be referenced. Called from *plugins.train.trainer._base*. Returns an iterator that yields images for training, preview and time-lapses.

Parameters

- **images** (*list*) – A list of image paths that will be used to compile the final augmented data from.
- **batchsize** (*int*) – The batchsize for this iterator. Images will be returned in `numpy.ndarray` objects of this size from the iterator.
- **side** (*{'a' or 'b'}*) – The side of the model that this iterator is for.
- **do_shuffle** (*bool, optional*) – Whether data should be shuffled prior to loading from disk. If true, each time the full list of filenames are processed, the data will be reshuffled to make sure they are not returned in the same order. Default: `True`
- **is_preview** (*bool, optional*) – Indicates whether this iterator is generating preview images. If `True` then certain augmentations will not be performed. Default: `False`
- **is_timelapse** (*bool optional*) – Indicates whether this iterator is generating time-lapse images. If `True`, then certain augmentations will not be performed. Default: `False`

Yields *dict* – The following items are contained in each *dict* yielded from this iterator:

- **feed** (`numpy.ndarray`) - The feed for the model. The array returned is in the format (*batchsize*, *height*, *width*, *channels*). This is the `x` parameter for `keras.models.model.train_on_batch()`.
- **targets** (*list*) - A list of 4-dimensional `numpy.ndarray` objects in the order and size of each output of the model as defined in `model_output_shapes`. the format of these arrays will be (*batchsize*, *height*, *width*, *3*). This is the `y` parameter for `keras.models.model.train_on_batch()` **NB:** masks are not included in the *targets* list. If required for feeding into the Keras model, they will need to be added to this list in *plugins.train.trainer._base* from the *masks* key.
- **masks** (`numpy.ndarray`) - A 4-dimensional array containing the target masks in the format (*batchsize*, *height*, *width*, *1*).
- **samples** (`numpy.ndarray`) - A 4-dimensional array containing the samples for feeding to the model's predict function for generating preview and time-lapse samples. The array will be in the format (*batchsize*, *height*, *width*, *channels*). **NB:** This item will only exist in the *dict* if `is_preview` or `is_timelapse` is `True`

1.1.13 utils module

Utilities available across all scripts

exception lib.utils.**FaceswapError**
Bases: `Exception`

Faceswap Error for handling specific errors with useful information.

Raises *FaceswapError* – on a captured error

class lib.utils.**GetModel**(*model_filename*, *cache_dir*, *git_model_id*)
Bases: `object`

Check for models in their cache path.

If available, return the path, if not available, get, unzip and install model

Parameters

- **model_filename** (*str or list*) – The name of the model to be loaded (see notes below)
- **cache_dir** (*str*) – The model cache folder of the current plugin calling this class. IE: The folder that holds the model to be loaded.
- **git_model_id** (*int*) – The second digit in the github tag that identifies this model. See https://github.com/deepfakes-models/faceswap-models for more information

Notes

Models must have a certain naming convention: *<model_name>_v<version_number>.<extension>* (eg: *s3fd_v1.pb*).

Multiple models can exist within the model_filename. They should be passed as a list and follow the same naming convention as above. Any differences in filename should occur AFTER the version number: *<model_name>_v<version_number><differentiating_information>.<extension>* (eg: *["mtcnn_det_v1.1.py", "mtcnn_det_v1.2.py", "mtcnn_det_v1.3.py"], ["resnet_ssd_v1.caffemodel" ,"resnet_ssd_v1.prototext"]*

model_path
The model path(s) in the cache folder.

Type str

class lib.utils.**KerasFinder**
Bases: importlib.abc.MetaPathFinder

Importlib Abstract Base Class for intercepting the import of Keras and returning either Keras (AMD backend) or tensorflow.keras (any other backend).

The Importlib documentation is sparse at best, and real world examples are pretty much non-existent. Coupled with this, the import `tensorflow.keras` does not resolve so we need to split out to the actual location of Keras within `tensorflow_core`. This method works, but it relies on hard coded paths, and is likely to not be the most robust.

A custom loader is not used, as we can use the standard loader once we have returned the correct spec.

find_spec(*fullname*, *path*, *target=None*)
Obtain the spec for either keras or tensorflow.keras depending on the backend in use.

If keras is not passed in as part of the `fullname` or the path is not `None` (i.e this is a dependency import) then this returns `None` to use the standard import library.

Parameters

- **fullname** (*str*) – The absolute name of the module to be imported
- **path** (*str*) – The search path for the module
- **target** (*module object, optional*) – Inherited from parent but unused

Returns The spec for the Keras module to be imported

Return type importlib.ModuleSpec

lib.utils.**camel_case_split**(*identifier*)
Split a camel case name

Parameters **identifier** (*str*) – The camel case text to be split

Returns A list of the given identifier split into it's constituent parts

Return type list

References

https://stackoverflow.com/questions/29916065

lib.utils.**convert_to_secs**(**args*)
Convert a time to seconds.

Parameters **args** (*tuple*) – 2 or 3 ints. If 2 ints are supplied, then (*minutes*, *seconds*) is implied. If 3 ints are supplied then (*hours*, *minutes*, *seconds*) is implied.

Returns The given time converted to seconds

Return type int

lib.utils.**deprecation_warning**(*function*, *additional_info=None*)
Log at warning level that a function will be removed in a future update.

Parameters

- **function** (*str*) – The function that will be deprecated.
- **additional_info** (*str, optional*) – Any additional information to display with the deprecation message. Default: `None`

lib.utils.**full_path_split**(*path*)
Split a full path to a location into all of it's separate components.

Parameters **path** (*str*) – The full path to be split

Returns The full path split into a separate item for each part

Return type list

Example

```
>>> path = "/foo/baz/bar"
>>> full_path_split(path)
>>> ["foo", "baz", "bar"]
```

lib.utils.**get_backend**()
Get the backend that Faceswap is currently configured to use.

Returns The backend configuration in use by Faceswap

Return type str

lib.utils.**get_folder**(*path*, *make_folder=True*)
Return a path to a folder, creating it if it doesn't exist

Parameters

- **path** (*str*) – The path to the folder to obtain
- **make_folder** (*bool, optional*) – `True` if the folder should be created if it does not already exist, `False` if the folder should not be created

Returns The path to the requested folder. If *make_folder* is set to `False` and the requested path does not exist, then `None` is returned

Return type `pathlib.Path` or *None*

lib.utils.**get_image_paths**(*directory*, *extension=None*)
Obtain a list of full paths that reside within a folder.

Parameters

- **directory** (*str*) – The folder that contains the images to be returned
- **extension** (*str*) – The specific image extensions that should be returned

Returns The list of full paths to the images contained within the given folder

Return type list

lib.utils.**safe_shutdown**(*got_error=False*)
Close all tracked queues and threads in event of crash or on shut down.

Parameters **got_error** (*bool, optional*) – `True` if this function is being called as the result of raised error, otherwise `False`. Default: `False`

lib.utils.**set_backend**(*backend*)
Override the configured backend with the given backend.

Parameters **backend** (*["amd", "cpu", "nvidia"]*) – The backend to set faceswap to

lib.utils.**set_system_verbosity**(*log_level*)
Set the verbosity level of tensorflow and suppresses future and deprecation warnings from any modules

Parameters **log_level** (*str*) – The requested Faceswap log level

References

https://stackoverflow.com/questions/35911252/disable-tensorflow-debugging-information Can be set to: 0: all logs shown. 1: filter out INFO logs. 2: filter out WARNING logs. 3: filter out ERROR logs.

1.2 plugins package

The plugins package holds Extraction, Training and Conversion plugins for Faceswap.

1.2.1 convert package

The Convert Package handles the various plugins available for performing conversion in Faceswap

Contents

mask package

mask._base module

Base class for Faceswap `mask` Plugins

class plugins.convert.mask._base.**Adjustment**(*mask_type*, *output_size*, *configfile=None*, *config=None*)

Bases: `object`

Parent class for Mask Adjustment Plugins.

All mask plugins must inherit from this class.

Parameters

- **mask_type** (*str*) – The type of mask that this plugin is being used for
- **output_size** (*int*) – The size, in pixels, of the output from the Faceswap model.
- **configfile** (*str, Optional*) – Optional location of custom configuration `ini` file. If `None` then use the default config location. Default: `None`
- **config** (`lib.config.FaceswapConfig`, Optional) – Optional pre-loaded `lib.config.FaceswapConfig`. If passed, then this will be used over any configuration on disk. If `None` then it is ignored. Default: `None`

config

The configuration dictionary for this plugin.

Type dict

mask_type
The type of mask that this plugin is being used for.

Type str

dummy
(`output_size`, `output_size`, *3*)

Type `numpy.ndarray`

Type A dummy mask of all zeros of the shape

process(**args*, ***kwargs*)
Override for specific mask adjustment plugin processes.

Input parameters will vary from plugin to plugin.

Should return a `numpy.ndarray` mask with the plugin's actions applied

run(**args*, ***kwargs*)
Perform selected adjustment on face

skip
`True` if the blur type config attribute is `None` otherwise `False`

Type bool

mask.box_blend module

Plugin to blend the edges of the face box that comes out of the Faceswap Model into the final frame.

class plugins.convert.mask.box_blend.**Mask**(*output_size*, ***kwargs*)
Bases: *plugins.convert.mask._base.Adjustment*

Manipulations to perform on the edges of the box that is received from the Faceswap model.

As the size of the box coming out of the model is identical for every face, the mask to be applied is just calculated once (at launch).

Parameters

- **output_size** (*int*) – The size of the output from the Faceswap model.
- ****kwargs** (*dict, optional*) – See the parent *_base* for additional keyword arguments.

process(*new_face*)
Apply the box mask to the swapped face.

Parameters **new_face** (`numpy.ndarray`) – The swapped face that has been output from the Faceswap model

Returns The input face is returned with the box mask added to the alpha channel if a blur type has been specified in the plugin configuration. If this configuration is set to `None` then the input face is returned with no mask applied.

Return type `numpy.ndarray`

mask.mask_blend module

Plugin to blend the edges of the face between the swap and the original face.

class plugins.convert.mask.mask_blend.**Mask**(*mask_type*, *output_size*, *coverage_ratio*, ***kwargs*)

Bases: *plugins.convert.mask._base.Adjustment*

Manipulations to perform to the mask that is to be applied to the output of the Faceswap model.

Parameters

- **mask_type** (*str*) – The mask type to use for this plugin
- **output_size** (*int*) – The size of the output from the Faceswap model.
- **coverage_ratio** (*float*) – The coverage ratio that the Faceswap model was trained at.
- ****kwargs** (*dict, optional*) – See the parent *_base* for additional keyword arguments.

process(*detected_face*, *sub_crop_offset*, *predicted_mask=None*)

Obtain the requested mask type and perform any defined mask manipulations.

Parameters

- **detected_face** (lib.align.DetectedFace) – The DetectedFace object as returned from scripts.convert.Predictor.
- **sub_crop_offset** (numpy.ndarray, optional) – The (x, y) offset to crop the mask from the center point.
- **predicted_mask** (numpy.ndarray, optional) – The predicted mask as output from the Faceswap Model, if the model was trained with a mask, otherwise None. Default: None.

Returns

- **mask** (numpy.ndarray) – The mask with all requested manipulations applied
- **raw_mask** (numpy.ndarray) – The mask with no erosion/dilation applied

1.2.2 extract package

The Extract Package handles the various plugins available for extracting face sets in Faceswap.

Contents

pipeline module

Module Summary

ExtractMedia	An object that passes through the *Extractor* pipeline.
Extractor	Creates a `detect`/`align`` / `mask` pipeline and yields results frame by frame from the `detected_faces` generator

Module

Return a requested detector/aligner/masker pipeline

Tensorflow does not like to release GPU VRAM, so parallel plugins need to be managed to work together.

This module sets up a pipeline for the extraction workflow, loading detect, align and mask plugins either in parallel or in series, giving easy access to input and output.

class plugins.extract.pipeline.**ExtractMedia**(*filename*, *image*, *detected_faces=None*)
Bases: object

An object that passes through the *Extractor* pipeline.

Parameters

- **filename** (*str*) – The base name of the original frame's filename
- **image** (numpy.ndarray) – The original frame
- **detected_faces** (*list, optional*) – A list of DetectedFace objects. Detected faces can be added later with *add_detected_faces()*. Default: None

add_detected_faces(*faces*)
Add detected faces to the object. Called at the end of each extraction phase.

Parameters **faces** (*list*) – A list of DetectedFace objects

detected_faces
A list of DetectedFace objects in the *image*.

Type list

filename
The base name of the *image* filename.

Type str

get_image_copy(*color_format*)
Get a copy of the image in the requested color format.

Parameters **color_format** (*['BGR', 'RGB', 'GRAY']*) – The requested color format of *image*

Returns A copy of *image* in the requested color_format

Return type numpy.ndarray

image
The source frame for this object.

Type numpy.ndarray

image_shape
The shape of the stored *image*.

Type tuple

image_size
The (*height*, *width*) of the stored *image*.

Type tuple

remove_image()
Delete the image and reset *image* to None.

Required for multi-phase extraction to avoid the frames stacking RAM.

set_image(*image*)
Add the image back into *image*

Required for multi-phase extraction adds the image back to this object.

Parameters **image** (numpy.ndarry) – The original frame to be re-applied to for this *filename*

class plugins.extract.pipeline.**Extractor**(*detector*, *aligner*, *masker*, *configfile=None*, *multiprocess=False*, *exclude_gpus=None*, *rotate_images=None*, *min_size=20*, *normalize_method=None*, *re_feed=0*, *image_is_aligned=False*)

Bases: object

Creates a detect/align`/ mask pipeline and yields results frame by frame from the *detected_faces* generator

input_queue is dynamically set depending on the current *phase* of extraction

Parameters

- **detector** (*str*) – The name of a detector plugin as exists in plugins.extract.detect
- **aligner** (*str*) – The name of an aligner plugin as exists in plugins.extract.align
- **masker** (*str or list*) – The name of a masker plugin(s) as exists in plugins.extract.mask. This can be a single masker or a list of multiple maskers
- **configfile** (*str, optional*) – The path to a custom extract.ini configfile. If None then the system config/extract.ini file will be used.
- **multiprocess** (*bool, optional*) – Whether to attempt processing the plugins in parallel. This may get overridden internally depending on the plugin combination. Default: False
- **exclude_gpus** (*list, optional*) – A list of indices correlating to connected GPUs that Tensorflow should not use. Pass None to not exclude any GPUs. Default: None
- **rotate_images** (*str, optional*) – Used to set the plugins.extract.detect.rotation attribute. Pass in a single number to use increments of that size up to 360, or pass in a list of ints to enumerate exactly what angles to check. Can also pass in 'on' to increment at 90 degree intervals. Default: None
- **min_size** (*int, optional*) – Used to set the plugins.extract.detect.min_size attribute Filters out faces detected below this size. Length, in pixels across the diagonal of the bounding box. Set to 0 for off. Default: 0

- **normalize_method** ({*None*, 'clahe', 'hist', 'mean'}, optional) – Used to set the `plugins.extract.align.normalize_method` attribute. Normalize the images fed to the aligner.Default: `None`
- **re_feed** (*int*) – The number of times to re-feed a slightly adjusted bounding box into the aligner. Default: *0*
- **image_is_aligned** (*bool, optional*) – Used to set the `plugins.extract.mask.image_is_aligned` attribute. Indicates to the masker that the fed in image is an aligned face rather than a frame. Default: `False`

phase
The current phase that the pipeline is running. Used in conjunction with *passes* and *final_pass* to indicate to the caller which phase is being processed

Type str

detected_faces()
Generator that returns results, frame by frame from the extraction pipeline

This is the exit point for the extraction pipeline and is used to obtain the output of any pipeline *phase*

Yields **faces** (*ExtractMedia*) – The populated extracted media object.

Example

```
>>> for extract_media in extractor.detected_faces():
>>>     filename = extract_media.filename
>>>     image = extract_media.image
>>>     detected_faces = extract_media.detected_faces
```

final_pass
bool, Return `True` if this is the final extractor pass otherwise `False`

Useful for iterating over the pipeline *passes* or *detected_faces()* and handling accordingly.

Example

```
>>> for face in extractor.detected_faces():
>>>     if extractor.final_pass:
>>>         <do final processing>
>>>     else:
>>>         extract_media.set_image(image)
>>>         <do intermediate processing>
>>>         extractor.input_queue.put(extract_media)
```

input_queue
Return the correct input queue depending on the current phase

The input queue is the entry point into the extraction pipeline. An *ExtractMedia* object should be put to the queue.

For detect/single phase operations the *ExtractMedia.filename* and *image* attributes should be populated.

For align/mask (2nd/3rd pass operations) the *ExtractMedia.detected_faces* should also be populated by calling `ExtractMedia.set_detected_faces()`.

Type queue

launch()
Launches the plugin(s)

This launches the plugins held in the pipeline, and should be called at the beginning of each *phase*. To ensure VRAM is conserved, It will only launch the plugin(s) required for the currently running phase

Example

```
>>> for phase in extractor.passes:
>>>     extractor.launch():
>>>         <do processing>
```

passes
Returns the total number of passes the extractor needs to make.

This is calculated on several factors (vram available, plugin choice, `multiprocess` etc.). It is useful for iterating over the pipeline and handling accordingly.

Example

```
>>> for phase in extractor.passes:
>>>     if phase == 1:
>>>         extract_media = ExtractMedia("path/to/image/file", image)
>>>         extractor.input_queue.put(extract_media)
>>>     else:
>>>         extract_media.set_image(image)
>>>         extractor.input_queue.put(extract_media)
```

Type int

phase_text
The plugins that are running in the current phase, formatted for info text output.

Type str

set_aligner_normalization_method(*method*)
Change the normalization method for faces fed into the aligner.

Parameters **method** (*{"none", "clahe", "hist", "mean"}*) – The normalization method to apply to faces prior to feeding into the aligner's model

set_batchsize(*plugin_type*, *batchsize*)
Set the batch size of a given `plugin_type` to the given `batchsize`.

This should be set prior to *launch()* if the batch size is to be manually overridden

Parameters

- **plugin_type** (*{'aligner', 'detector'}*) – The plugin_type to be overridden
- **batchsize** (*int*) – The batch size to use for this plugin type

extract plugins package

Contents

_base module

Base class for Faceswap `detect`, `align` and `mask` Plugins

class plugins.extract._base.**Extractor**(*git_model_id=None*, *model_filename=None*, *exclude_gpus=None*, *configfile=None*, *instance=0*)

Bases: `object`

Extractor Plugin Object

All `_base` classes for Aligners, Detectors and Maskers inherit from this class.

This class sets up a pipeline for working with ML plugins.

Plugins are split into 3 threads, to utilize Numpy and CV2s parallel processing, as well as allow the predict function of the model to sit in a dedicated thread. A plugin is expected to have 3 core functions, each in their own thread: - *`process_input()`* - Prepare the data for feeding into a model - *`predict()`* - Feed the data through the model - *`process_output()`* - Perform any data post-processing

Parameters

- **git_model_id** (*int*) – The second digit in the github tag that identifies this model. See https://github.com/deepfakes-models/faceswap-models for more information
- **model_filename** (*str*) – The name of the model file to be loaded
- **exclude_gpus** (*list, optional*) – A list of indices correlating to connected GPUs that Tensorflow should not use. Pass `None` to not exclude any GPUs. Default: `None`
- **configfile** (*str, optional*) – Path to a custom configuration `ini` file. Default: Use system configfile
- **instance** (*int, optional*) – If this plugin is being executed multiple times (i.e. multiple pipelines have been launched), the instance of the plugin must be passed in for naming convention reasons. Default: 0

The following attributes should be set in the plugin's `__init__()` method after initializing the parent.

name
Name of this plugin. Used for display purposes.

Type str

input_size
The input size to the model in pixels across one edge. The input size should always be square.

Type int

color_format
Color format for model. Must be `'BGR'`, `'RGB'` or `'GRAY'`. Defaults to `'BGR'` if not explicitly set.

Type str

vram
Approximate VRAM used by the model at *input_size*. Used to calculate the *batchsize*. Be conservative to avoid OOM.

Type int

vram_warnings
Approximate VRAM used by the model at *input_size* that will still run, but generates warnings. Used to calculate the *batchsize*. Be conservative to avoid OOM.

Type int

vram_per_batch
Approximate additional VRAM used by the model for each additional batch. Used to calculate the *batchsize*. Be conservative to avoid OOM.

Type int

See also:

plugins.extract.detect._base Detector parent class for extraction plugins.

plugins.extract.align._base Aligner parent class for extraction plugins.

plugins.extract.mask._base Masker parent class for extraction plugins.

plugins.extract.pipeline The extract pipeline that configures and calls all plugins

batchsize = None
Batchsize for feeding this model. The number of images the model should feed through at once.

Type int

check_and_raise_error()
Check all threads for errors

Exposed for *pipeline* to check plugin's threads for errors

config = None
Config for this plugin, loaded from `extract.ini` configfile

Type dict

finalize(*batch*)
Override method (at <*plugin_type*> level)

This method should be overridden at the <*plugin_type*> level (IE. *plugins.extract.detect._base*, *plugins.extract.align._base* or *plugins.extract.mask._base*) and should not be overridden within plugins themselves.

Handles consistent finalization for all plugins that exist within that plugin type. Its input is always the output from *process_output()*

Parameters **batch** (*dict*) – Contains the batch that is currently being passed through the plugin process

get_batch(*queue*)
Override method (at <*plugin_type*> level)

This method should be overridden at the <*plugin_type*> level (IE. *plugins.extract.detect._base*, *plugins.extract.align._base* or *plugins.extract.mask._base*) and should not be overridden within plugins themselves.

Get *ExtractMedia* items from the queue in batches of *batchsize*

Parameters **queue** (*queue.Queue()*) – The queue that the batch will be fed from. This will be the input to the plugin.

init_model()
Override method

Override this method to execute the specific model initialization method

initialize(**args*, ***kwargs*)
Initialize the extractor plugin

Should be called from *pipeline*

join()
Join all threads

Exposed for *pipeline* to join plugin's threads

model = None
The model for this plugin. Set in the plugin's *init_model()* method

Type varies

model_path = None
Path to the model file(s) (if required). Multiple model files should be a list of strings

Type str or list

predict(*batch*)
Override method

Override this method for specific extractor model prediction function

Parameters **batch** (*dict*) – Contains the batch that is currently being passed through the plugin process

Notes

Input for *predict()* should have been set in *process_input()* with the addition of a `feed` key to the `batch` `dict`.

Output from the model should add the key `prediction` to the `batch` `dict`.

For Detect: the expected output for the `prediction` key of the `batch` dict should be a `list` of *batchsize* of detected face points. These points should be either a `list`, `tuple` or `numpy.ndarray` with the first 4 items being the *left*, *top*, *right*, *bottom* points, in that order

process_input(*batch*)
Override method

Override this method for specific extractor pre-processing of image

Parameters **batch** (*dict*) – Contains the batch that is currently being passed through the plugin process

Notes

When preparing an input to the model a key `feed` must be added to the `batch` `dict` which contains this input.

process_output(*batch*)
Override method

Override this method for specific extractor model post predict function

Parameters **batch** (*dict*) – Contains the batch that is currently being passed through the plugin process

Notes

For Align: The key `landmarks` must be returned in the `batch` `dict` from this method. This should be a `list` or `numpy.ndarray` of *batchsize* containing a `list`, `tuple` or `numpy.ndarray` of *(x, y)* coordinates of the 68 point landmarks as calculated from the *model*.

queue_size = None
Queue size for all internal queues. Set in *initialize()*

Type int

start()
Start all threads

Exposed for *pipeline* to start plugin's threads

detect._base module

Base class for Face Detector plugins

All Detector Plugins should inherit from this class. See the override methods for which methods are required.

The plugin will receive a *ExtractMedia* object.

For each source frame, the plugin must pass a dict to finalize containing:

```
>>> {'filename': <filename of source frame>,
>>>  'detected_faces': <list of DetectedFace objects containing bounding box points}}
```

To get a `DetectedFace` object use the function:

```
>>> face = self.to_detected_face(<face left>, <face top>, <face right>, <face bottom>)
```

class plugins.extract.detect._base.**Detector**(*git_model_id=None*, *model_filename=None*, *configfile=None*, *instance=0*, *rotation=None*, *min_size=0*, ***kwargs*)

Bases: *plugins.extract._base.Extractor*

Detector Object

Parent class for all Detector plugins

Parameters

- **git_model_id** (*int*) – The second digit in the github tag that identifies this model. See https://github.com/deepfakes-models/faceswap-models for more information
- **model_filename** (*str*) – The name of the model file to be loaded
- **rotation** (*str, optional*) – Pass in a single number to use increments of that size up to 360, or pass in a `list` of `ints` to enumerate exactly what angles to check. Can also pass in `'on'` to increment at 90 degree intervals. Default: `None`

- **min_size** (*int, optional*) – Filters out faces detected below this size. Length, in pixels across the diagonal of the bounding box. Set to `0` for off. Default: `0`

Other Parameters **configfile** (*str, optional*) – Path to a custom configuration `ini` file. Default: Use system configfile

See also:

plugins.extract.pipeline The extraction pipeline for calling plugins

plugins.extract.detect Detector plugins

plugins.extract._base Parent class for all extraction plugins

plugins.extract.align._base Aligner parent class for extraction plugins.

plugins.extract.mask._base Masker parent class for extraction plugins.

finalize(*batch*)

Finalize the output from Detector

This should be called as the final task of each `plugin`.

Parameters **batch** (*dict*) – The final `dict` from the *plugin* process. It must contain the keys `filename`, `faces`

Yields *ExtractMedia* – The `DetectedFaces` list will be populated for this class with the bounding boxes for the detected faces found in the frame.

get_batch(*queue*)

Get items for inputting to the detector plugin in batches

Items are received as *ExtractMedia* objects and converted to `dict` for internal processing.

Items are returned from the `queue` in batches of *batchsize*

Remember to put `'EOF'` to the out queue after processing the final batch

Outputs items in the following format. All lists are of length *batchsize*:

```
>>> {'filename': [<filenames of source frames>],
>>>  'image': <numpy.ndarray of images standardized for prediction>,
>>>  'scale': [<scaling factors for each image>],
>>>  'pad': [<padding for each image>],
>>>  'detected_faces': [[<lib.align.DetectedFace objects]]}
```

Parameters **queue** (*queue.Queue()*) – The `queue` that the batch will be fed from. This will be a queue that loads images.

Returns

- *exhausted, bool* – `True` if queue is exhausted, `False` if not.
- *batch, dict* – A dictionary of lists of *batchsize*.

static to_detected_face(*left*, *top*, *right*, *bottom*)

Return a `DetectedFace` object for the bounding box

align._base module

Base class for Face Aligner plugins

All Aligner Plugins should inherit from this class. See the override methods for which methods are required.

The plugin will receive a *ExtractMedia* object.

For each source item, the plugin must pass a dict to finalize containing:

```
>>> {"filename": [<filename of source frame>],
>>>  "landmarks": [list of 68 point face landmarks]
>>>  "detected_faces": [<list of DetectedFace objects>]}
```

class plugins.extract.align._base.**Aligner**(*git_model_id=None*, *model_filename=None*, *configfile=None*, *instance=0*, *normalize_method=None*, *re_feed=0*, ***kwargs*)

Bases: *plugins.extract._base.Extractor*

Aligner plugin _base Object

All Aligner plugins must inherit from this class

Parameters

- **git_model_id** (*int*) – The second digit in the github tag that identifies this model. See https://github.com/deepfakes-models/faceswap-models for more information
- **model_filename** (*str*) – The name of the model file to be loaded
- **normalize_method** ({*None*, 'clahe', 'hist', 'mean'}, optional) – Normalize the images fed to the aligner. Default: `None`
- **re_feed** (*int*) – The number of times to re-feed a slightly adjusted bounding box into the aligner. Default: *0*

Other Parameters **configfile** (*str, optional*) – Path to a custom configuration `ini` file. Default: Use system configfile

See also:

plugins.extract.pipeline The extraction pipeline for calling plugins

plugins.extract.align Aligner plugins

plugins.extract._base Parent class for all extraction plugins

plugins.extract.detect._base Detector parent class for extraction plugins.

plugins.extract.mask._base Masker parent class for extraction plugins.

finalize(*batch*)

Finalize the output from Aligner

This should be called as the final task of each *plugin*.

Pairs the detected faces back up with their original frame before yielding each frame.

Parameters **batch** (*dict*) – The final `dict` from the *plugin* process. It must contain the *keys*: `detected_faces`, `landmarks`, `filename`

Yields *ExtractMedia* – The `DetectedFaces` list will be populated for this class with the bounding boxes and landmarks for the detected faces found in the frame.

get_batch(*queue*)
Get items for inputting into the aligner from the queue in batches

Items are returned from the `queue` in batches of *batchsize*

Items are received as *ExtractMedia* objects and converted to `dict` for internal processing.

To ensure consistent batch sizes for aligner the items are split into separate items for each `DetectedFace` object.

Remember to put `'EOF'` to the out queue after processing the final batch

Outputs items in the following format. All lists are of length *batchsize*:

```
>>> {'filename': [<filenames of source frames>],
>>>  'image': [<source images>],
>>>  'detected_faces': [[<lib.align.DetectedFace objects]]}
```

Parameters **queue** (*queue.Queue()*) – The `queue` that the plugin will be fed from.

Returns

- *exhausted, bool* – `True` if queue is exhausted, `False` if not
- *batch, dict* – A dictionary of lists of *batchsize*:

set_normalize_method(*method*)
Set the normalization method for feeding faces into the aligner.

Parameters **method** (*{"none", "clahe", "hist", "mean"}*) – The normalization method to apply to faces prior to feeding into the model

mask._base module

Base class for Face Masker plugins

Plugins should inherit from this class

See the override methods for which methods are required.

The plugin will receive a *ExtractMedia* object.

For each source item, the plugin must pass a dict to finalize containing:

```
>>> {"filename": <filename of source frame>,
>>>  "detected_faces": <list of bounding box dicts from lib/plugins/extract/detect/_
↪base>}
```

class plugins.extract.mask._base.**Masker**(*git_model_id=None*, *model_filename=None*, *configfile=None*, *instance=0*, *image_is_aligned=False*, ***kwargs*)
Bases: *plugins.extract._base.Extractor*

Masker plugin _base Object

All Masker plugins must inherit from this class

Parameters

- **git_model_id** (*int*) – The second digit in the github tag that identifies this model. See https://github.com/deepfakes-models/faceswap-models for more information
- **model_filename** (*str*) – The name of the model file to be loaded

- **image_is_aligned** (*bool, optional*) – Indicates that the passed in image is an aligned face rather than a frame. Default: `False`

Other Parameters **configfile** (*str, optional*) – Path to a custom configuration `ini` file. Default: Use system configfile

See also:

plugins.extract.pipeline The extraction pipeline for calling plugins

plugins.extract.align Aligner plugins

plugins.extract._base Parent class for all extraction plugins

plugins.extract.detect._base Detector parent class for extraction plugins.

plugins.extract.align._base Aligner parent class for extraction plugins.

finalize(*batch*)

Finalize the output from Masker

This should be called as the final task of each *plugin*.

Pairs the detected faces back up with their original frame before yielding each frame.

Parameters **batch** (*dict*) – The final `dict` from the *plugin* process. It must contain the *keys*: `detected_faces`, `filename`, `feed_faces`

Yields *ExtractMedia* – The `DetectedFaces` list will be populated for this class with the bounding boxes, landmarks and masks for the detected faces found in the frame.

get_batch(*queue*)

Get items for inputting into the masker from the queue in batches

Items are returned from the `queue` in batches of *batchsize*

Items are received as *ExtractMedia* objects and converted to `dict` for internal processing.

To ensure consistent batch sizes for masker the items are split into separate items for each `DetectedFace` object.

Remember to put `'EOF'` to the out queue after processing the final batch

Outputs items in the following format. All lists are of length *batchsize*:

```
>>> {'filename': [<filenames of source frames>],
>>>  'detected_faces': [[<lib.align.DetectedFace objects]]}
```

Parameters **queue** (*queue.Queue()*) – The `queue` that the plugin will be fed from.

Returns

- *exhausted, bool* – `True` if queue is exhausted, `False` if not
- *batch, dict* – A dictionary of lists of *batchsize*:

vgg_face2_keras module

VGG_Face2 inference and sorting

class plugins.extract.recognition.vgg_face2_keras.**VGGFace2**(**args*, ***kwargs*)
Bases: *plugins.extract._base.Extractor*

VGG Face feature extraction.

Extracts feature vectors from faces in order to compare similarity.

Notes

Input images should be in BGR Order

Model exported from: https://github.com/WeidiXie/Keras-VGGFace2-ResNet50 which is based on: https://www.robots.ox.ac.uk/~vgg/software/vgg_face/

Licensed under Creative Commons Attribution License. https://creativecommons.org/licenses/by-nc/4.0/

static find_cosine_similiarity(*source_face*, *test_face*)
Find the cosine similarity between two faces.

Parameters

- **source_face** (*numpy.ndarray*) – The first face to test against `test_face`
- **test_face** (*numpy.ndarray*) – The second face to test against `source_face`

Returns The cosine similarity between the two faces

Return type float

init_model()
Initialize VGG Face 2 Model.

predict(*batch*)
Return encodings for given image from vgg_face2.

Parameters **batch** (*numpy.ndarray*) – The face to be fed through the predictor. Should be in BGR channel order

Returns The encodings for the face

Return type numpy.ndarray

sorted_similarity(*predictions*, *method='ward'*)
Sort a matrix of predictions by similarity.

Transforms a distance matrix into a sorted distance matrix according to the order implied by the hierarchical tree (dendrogram).

Parameters

- **predictions** (*numpy.ndarray*) – A stacked matrix of vgg_face2 predictions of the shape (*N*, *D*) where *N* is the number of observations and *D* are the number of dimensions. NB: The given `predictions` will be overwritten to save memory. If you still require the original values you should take a copy prior to running this method
- **method** (*['single','centroid','median','ward']*) – The clustering method to use.

Returns List of indices with the order implied by the hierarchical tree

Return type list

1.2.3 plugin_loader module

Plugin loader for Faceswap extract, training and convert tasks

class plugins.plugin_loader.**PluginLoader**
Bases: object

Retrieve, or get information on, Faceswap plugins

Return a specific plugin, list available plugins, or get the default plugin for a task.

Example

```
>>> from plugins.plugin_loader import PluginLoader
>>> align_plugins = PluginLoader.get_available_extractors('align')
>>> aligner = PluginLoader.get_aligner('cv2-dnn')
```

static get_aligner(*name*, *disable_logging=False*)
Return requested aligner plugin

Parameters

- **name** (*str*) – The name of the requested aligner plugin
- **disable_logging** (*bool, optional*) – Whether to disable the INFO log message that the plugin is being imported. Default: *False*

Returns An extraction aligner plugin

Return type plugins.extract.align object

static get_available_convert_plugins(*convert_category*, *add_none=True*)
Return a list of available converter plugins in the given category

Parameters

- **convert_category** (*{'color', 'mask', 'scaling', 'writer'}*) – The category of converter plugin to return the plugins for
- **add_none** (*bool, optional*) – Append "none" to the list of returned plugins. Default: True

Returns A list of the available converter plugin names in the given category

Return type list

static get_available_extractors(*extractor_type*, *add_none=False*)
Return a list of available extractors of the given type

Parameters

- **extractor_type** (*{'aligner', 'detector', 'masker'}*) – The type of extractor to return the plugins for
- **add_none** (*bool, optional*) – Append "none" to the list of returned plugins. Default: False

Returns A list of the available extractor plugin names for the given type

Return type list

static get_available_models()
Return a list of available training models

Returns A list of the available training model plugin names

Return type list

static get_converter(*category*, *name*, *disable_logging=False*)
Return requested converter plugin

Converters work slightly differently to other faceswap plugins. They are created to do a specific task (e.g. color adjustment, mask blending etc.), so multiple plugins will be loaded in the convert phase, rather than just one plugin for the other phases.

Parameters

- **name** (*str*) – The name of the requested converter plugin
- **disable_logging** (*bool, optional*) – Whether to disable the INFO log message that the plugin is being imported. Default: *False*

Returns A converter sub plugin

Return type plugins.convert object

static get_default_model()
Return the default training model plugin name

Returns The default faceswap training model

Return type str

static get_detector(*name*, *disable_logging=False*)
Return requested detector plugin

Parameters

- **name** (*str*) – The name of the requested detector plugin
- **disable_logging** (*bool, optional*) – Whether to disable the INFO log message that the plugin is being imported. Default: *False*

Returns An extraction detector plugin

Return type plugins.extract.detect object

static get_masker(*name*, *disable_logging=False*)
Return requested masker plugin

Parameters

- **name** (*str*) – The name of the requested masker plugin
- **disable_logging** (*bool, optional*) – Whether to disable the INFO log message that the plugin is being imported. Default: *False*

Returns An extraction masker plugin

Return type plugins.extract.mask object

static get_model(*name*, *disable_logging=False*)
Return requested training model plugin

Parameters

- **name** (*str*) – The name of the requested training model plugin
- **disable_logging** (*bool, optional*) – Whether to disable the INFO log message that the plugin is being imported. Default: *False*

Returns A training model plugin

Return type `plugins.train.model` object

static get_trainer(*name*, *disable_logging=False*)
Return requested training trainer plugin

Parameters

- **name** (*str*) – The name of the requested training trainer plugin
- **disable_logging** (*bool, optional*) – Whether to disable the INFO log message that the plugin is being imported. Default: *False*

Returns A training trainer plugin

Return type `plugins.train.trainer` object

1.2.4 train package

The Train Package handles the Model and Trainer plugins for training models in Faceswap.

Contents

model._base module

Module Summary

KerasModel	wrapper for `keras.models.Model`.
ModelBase	Base class that all model plugins should inherit from.
State	Holds state information relating to the plugin's saved model.

Module

Base class for Models. ALL Models should at least inherit from this class.

See *original* for an annotated example for how to create model plugins.

`plugins.train.model._base.`**KerasModel**(*inputs*, *outputs*, *name*)
wrapper for `keras.models.Model`.

There are some minor foibles between Keras 2.2 and the Tensorflow version of Keras, so this catches potential issues and fixes prior to returning the requested model.

All models created within plugins should use this method, and should not call keras directly for a model.

Parameters

- **inputs** (*a keras.Input object or list of keras.Input objects.*) – The input(s) of the model
- **outputs** (*keras objects*) – The output(s) of the model.

- **name** (*str*) – The name of the model.

Returns A Keras Model

Return type keras.models.Model

class plugins.train.model._base.**ModelBase**(*model_dir*, *arguments*, *predict=False*)
Bases: object

Base class that all model plugins should inherit from.

Parameters

- **model_dir** (*str*) – The full path to the model save location
- **arguments** (argparse.Namespace) – The arguments that were passed to the train or convert process as generated from Faceswap's command line arguments
- **predict** (*bool, optional*) – True if the model is being loaded for inference, False if the model is being loaded for training. Default: False

input_shape
A *tuple* of *ints* defining the shape of the faces that the model takes as input. This should be overridden by model plugins in their __init__() function. If the input size is the same for both sides of the model, then this can be a single 3 dimensional *tuple*. If the inputs have different sizes for *"A"* and *"B"* this should be a *list* of 2 3 dimensional shape *tuples*, 1 for each side respectively.

Type tuple or list

trainer
Currently there is only one trainer available (*"original"*), so at present this attribute can be ignored. If/when more trainers are added, then this attribute should be overridden with the trainer name that a model requires in the model plugin's __init__() function.

Type str

add_history(*loss*)
Add the current iteration's loss history to _io.history.

Called from the trainer after each iteration, for tracking loss drop over time between save iterations.

Parameters **loss** (*list*) – The loss values for the A and B side for the current iteration. This should be the collated loss values for each side.

build()
Build the model and assign to *model*.

Within the defined strategy scope, either builds the model from scratch or loads an existing model if one exists.

If running inference, then the model is built only for the required side to perform the swap function, otherwise the model is then compiled with the optimizer and chosen loss function(s).

Finally, a model summary is outputted to the logger at verbose level.

build_model(*inputs*)
Override for Model Specific autoencoder builds.

Parameters **inputs** (*list*) – A list of keras.layers.Input tensors. This will be a list of 2 tensors (one for each side) each of shapes *input_shape*.

command_line_arguments
The command line arguments passed to the model plugin from either the train or convert script

Type argparse.Namespace

config
The configuration dictionary for current plugin, as set by the user's configuration settings.

Type dict

coverage_ratio
The ratio of the training image to crop out and train on as defined in user configuration options.

NB: The coverage ratio is a raw float, but will be applied to integer pixel images.

To ensure consistent rounding and guaranteed even image size, the calculation for coverage should always be: $(original_size * coverage_ratio//2) * 2$

Type float

iterations
The total number of iterations that the model has trained.

Type int

model
The compiled model for this plugin.

Type `Keras.models.Model`

model_dir
The full path to the model folder location.

Type str

name
The name of this model based on the plugin name.

Type str

output_shapes
A list of list of shape tuples for the outputs of the model with the batch dimension removed. The outer list contains 2 sub-lists (one for each side "a" and "b"). The inner sub-lists contain the output shapes for that side.

Type list

save()
Save the model to disk.

Saves the serialized model, with weights, to the folder location specified when initializing the plugin. If loss has dropped on both sides of the model, then a backup is taken.

snapshot()
Creates a snapshot of the model folder to the models parent folder, with the number of iterations completed appended to the end of the model name.

state
The state settings for the current plugin.

Type *State*

class plugins.train.model._base.**State**(*model_dir*, *model_name*, *config_changeable_items*, *no_logs*)

Bases: `object`

Holds state information relating to the plugin's saved model.

Parameters

- **model_dir** (*str*) – The full path to the model save location

- **model_name** (*str*) – The name of the model plugin
- **config_changeable_items** (*dict*) – Configuration options that can be altered when resuming a model, and their current values
- **no_logs** (*bool*) – `True` if Tensorboard logs should not be generated, otherwise `False`

add_session_batchsize(*batch_size*)
Add the session batch size to the sessions dictionary.

Parameters **batch_size** (*int*) – The batch size for the current training session

add_session_loss_names(*loss_names*)
Add the session loss names to the sessions dictionary.

The loss names are used for Tensorboard logging

Parameters **loss_names** (*list*) – The list of loss names for this session.

current_session
The state dictionary for the current *session_id*.

Type dict

increment_iterations()
Increment *iterations* and session iterations by 1.

iterations
The total number of iterations that the model has trained.

Type int

loss_names
The loss names for the current session

Type list

lowest_avg_loss
The lowest average save interval loss seen for each side.

Type dict

save()
Save the state values to the serialized state file.

session_id
The current training session id.

Type int

model.original module

Original Model Based on the original https://www.reddit.com/r/deepfakes/ code sample + contributions.

This model is heavily documented as it acts as a template that other model plugins can be developed from.

class plugins.train.model.original.**Model**(**args*, ***kwargs*)
Bases: *plugins.train.model._base.ModelBase*

Original Faceswap Model.

This is the original faceswap model and acts as a template for plugin development.

All plugins must define the following attribute override after calling the parent's `__init__()` method:

- `input_shape` (*tuple* or *list*): a tuple of ints defining the shape of the faces that the model takes as input. If the input size is the same for both sides, this can be a single 3 dimensional tuple. If the inputs have different sizes for "A" and "B" this should be a list of 2 3 dimensional shape tuples, 1 for each side.

Any additional attributes used exclusively by this model should be defined here, but make sure that you are not accidentally overriding any existing *ModelBase* attributes.

Parameters

- **args** (*varies*) – The default command line arguments passed in from *Train* or `Convert`
- **kwargs** (*varies*) – The default keyword arguments passed in from *Train* or `Convert`

build_model(*inputs*)

Create the model's structure.

This function is automatically called immediately after `__init__()` has been called if a new model is being created. It is ignored if an existing model is being loaded from disk as the model structure will be defined in the saved model file.

The model's final structure is defined here.

For the original model, An encoder instance is defined, then the same instance is referenced twice, one for each input "A" and "B" so that the same model is used for both inputs.

2 Decoders are then defined (one for each side) with the encoder instances passed in as input to the corresponding decoders.

It is important to note that any models and sub-models should not call `keras.models.Model` directly, but rather call *plugins.train.model._base.KerasModel*. This acts as a wrapper for Keras' Model class, but handles some minor differences which need to be handled between Nvidia and AMD backends.

The final output of the model should always call *lib.model.nn_blocks.Conv2DOutput* so that the correct data type is set for the final activation, to support Mixed Precision Training. Failure to do so is likely to lead to issues when Mixed Precision is enabled.

Parameters **inputs** (*list*) – A list of input tensors for the model. This will be a list of 2 tensors of shape `input_shape`, the first for side "a", the second for side "b".

Returns The output of this function must be a keras model generated from *plugins.train.model._base.KerasModel*. See Keras documentation for the correct structure, but note that parameter `name` is a required rather than an optional argument in Faceswap. You should assign this to the attribute `self.name` that is automatically generated from the plugin's filename.

Return type `keras.models.Model`

decoder(*side*)

The original Faceswap Decoder Network.

The decoders for the original model have separate weights for each side "A" and "B", so two instances are created in *build_model()*, one for each side.

Parameters **side** (*str*) – Either *"a* or *"b"*. This is used for naming the decoder model.

Returns The Keras decoder model. This will be called twice, once for each side.

Return type `keras.models.Model`

encoder()

The original Faceswap Encoder Network.

The encoder for the original model has it's weights shared between both the "A" and "B" side of the model, so only one instance is created *build_model()*. However this same instance is then used twice (once for A and once for B) meaning that the weights get shared.

Returns The Keras encoder model, for sharing between inputs from both sides.

Return type keras.models.Model

trainer._base module

Base Class for Faceswap Trainer plugins. All Trainer plugins should be inherited from this class.

At present there is only the original plugin, so that entirely inherits from this class. If further plugins are developed, then common code should be kept here, with "original" unique code split out to the original plugin.

class plugins.train.trainer._base.**TrainerBase**(*model*, *images*, *batch_size*, *configfile*)
Bases: object

Handles the feeding of training images to Faceswap models, the generation of Tensorboard logs and the creation of sample/time-lapse preview images.

All Trainer plugins must inherit from this class.

Parameters

- **model** (plugin from plugins.train.model) – The model that will be running this trainer
- **images** (*dict*) – The file paths for the images to be trained on for each side. The dictionary should contain 2 keys ("a" and "b") with the values being a list of full paths corresponding to each side.
- **batch_size** (*int*) – The requested batch size for iteration to be trained through the model.
- **configfile** (*str*) – The path to a custom configuration file. If None is passed then configuration is loaded from the default .config.train.ini file.

clear_tensorboard()
Stop Tensorboard logging.

Tensorboard logging needs to be explicitly shutdown on training termination. Called from *scripts.train.Train* when training is stopped.

train_one_step(*viewer*, *timelapse_kwargs*)
Running training on a batch of images for each side.

Triggered from the training cycle in *scripts.train.Train*.

- Runs a training batch through the model.
- Outputs the iteration's loss values to the console
- Logs loss to Tensorboard, if logging is requested.
- If a preview or time-lapse has been requested, then pushes sample images through the model to generate the previews
- Creates a snapshot if the total iterations trained so far meet the requested snapshot criteria

Notes

As every iteration is called explicitly, the Parameters defined should always be `None` except on save iterations.

Parameters

- **viewer** (`scripts.train.Train._show()`) – The function that will display the preview image
- **timelapse_kwargs** (*dict*) – The keyword arguments for generating time-lapse previews. If a time-lapse preview is not required then this should be `None`. Otherwise all values should be full paths the keys being *input_a*, *input_b*, *output*.

1.3 scripts package

The Scripts Package is the entry point into Faceswap.

Contents

1.3.1 extract module

Main entry point to the extract process of FaceSwap

class `scripts.extract.`**Extract**(*arguments*)

Bases: `object`

The Faceswap Face Extraction Process.

The extraction process is responsible for detecting faces in a series of images/video, aligning these faces and then generating a mask.

It leverages a series of user selected plugins, chained together using *plugins.extract.pipeline*.

The extract process is self contained and should not be referenced by any other scripts, so it contains no public properties.

Parameters **arguments** (*argparse.Namespace*) – The arguments to be passed to the extraction process as generated from Faceswap's command line arguments

process()

The entry point for triggering the Extraction Process.

Should only be called from *lib.cli.launcher.ScriptExecutor*

1.3.2 train module

Main entry point to the training process of FaceSwap

class scripts.train.**Train**(*arguments*)
Bases: object

The Faceswap Training Process.

The training process is responsible for training a model on a set of source faces and a set of destination faces.

The training process is self contained and should not be referenced by any other scripts, so it contains no public properties.

Parameters **arguments** (*argparse.Namespace*) – The arguments to be passed to the training process as generated from Faceswap's command line arguments

process()
The entry point for triggering the Training Process.

Should only be called from *lib.cli.launcher.ScriptExecutor*

1.3.3 convert module

Module Summary

Convert	The Faceswap Face Conversion Process.
DiskIO	Disk Input/Output for the converter process.
OptionalActions	Process specific optional actions for Convert.
Predict	Obtains the output from the Faceswap model.

Module

Main entry point to the convert process of FaceSwap

class scripts.convert.**Convert**(*arguments*)
Bases: object

The Faceswap Face Conversion Process.

The conversion process is responsible for swapping the faces on source frames with the output from a trained model.

It leverages a series of user selected post-processing plugins, executed from *lib.convert.Converter*.

The convert process is self contained and should not be referenced by any other scripts, so it contains no public properties.

Parameters **arguments** (argparse.Namespace) – The arguments to be passed to the convert process as generated from Faceswap's command line arguments

process()
The entry point for triggering the Conversion Process.

Should only be called from *lib.cli.launcher.ScriptExecutor*

class scripts.convert.**DiskIO**(*alignments*, *images*, *arguments*)
Bases: object

Disk Input/Output for the converter process.

Background threads to:

- Load images from disk and get the detected faces

- Save images back to disk

Parameters

- **alignments** (lib.alignmnents.Alignments) – The alignments for the input video
- **images** (*lib.image.ImagesLoader*) – The input images
- **arguments** (argparse.Namespace) – The arguments that were passed to the convert process as generated from Faceswap's command line arguments

completion_event
Event is set when the DiskIO Save task is complete

Type event.Event

draw_transparent
True if the selected writer's Draw_transparent configuration item is set otherwise False

Type bool

load_queue
The queue that images and detected faces are loaded into.

Type queue.Queue()

load_thread
The thread that is running the image loading operation.

Type lib.multithreading.MultiThread

pre_encode
Selected writer's pre-encode function, if it has one, otherwise None

Type python function

save_thread
The thread that is running the image writing operation.

Type lib.multithreading.MultiThread

class scripts.convert.**OptionalActions**(*arguments*, *input_images*, *alignments*)
Bases: object

Process specific optional actions for Convert.

Currently only handles skip faces. This class should probably be (re)moved.

Parameters

- **arguments** (argparse.Namespace) – The arguments that were passed to the convert process as generated from Faceswap's command line arguments
- **input_images** (*list*) – List of input image files
- **alignments** (lib.align.Alignments) – The alignments file for this conversion

class scripts.convert.**Predict**(*in_queue*, *queue_size*, *arguments*)
Bases: object

Obtains the output from the Faceswap model.

Parameters

- **in_queue** (queue.Queue) – The queue that contains images and detected faces for feeding the model
- **queue_size** (*int*) – The maximum size of the input queue
- **arguments** (argparse.Namespace) – The arguments that were passed to the convert process as generated from Faceswap's command line arguments

centering
The centering that the model was trained on (*"face"* or *"legacy"*)

Type str

coverage_ratio
The coverage ratio that the model was trained at.

Type float

faces_count
The total number of faces seen by the Predictor.

Type int

has_predicted_mask
True if the model was trained to learn a mask, otherwise False.

Type bool

in_queue
The input queue to the predictor.

Type queue.Queue

load_aligned(*item*)
Load the model's feed faces and the reference output faces.

For each detected face in the incoming item, load the feed face and reference face images, correctly sized for input and output respectively.

Parameters **item** (*dict*) – The incoming image, list of DetectedFace objects and list of AlignedFace objects for the feed face(s) and list of AlignedFace objects for the reference face(s)

out_queue
The output queue from the predictor.

Type queue.Queue

output_size
The size in pixels of the Faceswap model output.

Type int

thread
The thread that is running the prediction function from the Faceswap model.

Type MultiThread

verify_output
True if multiple faces have been found in frames, otherwise False.

Type bool

1.3.4 fsmedia module

Module Summary

Alignments	Override `lib.align.Alignments` to add custom loading based on command line arguments.
DebugLandmarks	Draw debug landmarks on face output.
FaceFilter	Filter in or out faces based on input image(s).
Images	Handles the loading of frames from a folder of images or a video file for extract and convert processes.
PostProcess	Optional pre/post processing tasks for convert and extract.
finalize	Output summary statistics at the end of the extract or convert processes.

Module

Helper functions for *extract* and *convert*.

Holds the classes for the 2 main Faceswap 'media' objects: Images and Alignments.

Holds optional pre/post processing functions for convert and extract.

class scripts.fsmedia.**Alignments**(*arguments*, *is_extract*, *input_is_video=False*)

Bases: *lib.align.alignments.Alignments*

Override `lib.align.Alignments` to add custom loading based on command line arguments.

Parameters

- **arguments** (`argparse.Namespace`) – The command line arguments that were passed to Faceswap
- **is_extract** (*bool*) – `True` if the process calling this class is extraction otherwise `False`
- **input_is_video** (*bool, optional*) – `True` if the input to the process is a video, `False` if it is a folder of images. Default: False

class scripts.fsmedia.**DebugLandmarks**(**args*, ***kwargs*)

Bases: *scripts.fsmedia.PostProcessAction*

Draw debug landmarks on face output. Extract Only

process(*extract_media*)

Draw landmarks on a face.

Parameters **extract_media** (*ExtractMedia*) – The *ExtractMedia* object that contains the faces to draw the landmarks on to

Returns The original *ExtractMedia* with landmarks drawn onto the face

Return type *ExtractMedia*

class scripts.fsmedia.**FaceFilter**(**args*, ***kwargs*)

Bases: *scripts.fsmedia.PostProcessAction*

Filter in or out faces based on input image(s). Extract or Convert

Parameters

- **args** (*tuple*) – Unused
- **kwargs** (*dict*) – Keyword arguments for face filter:
  - **detector** (*str*) - The detector to use
  - **aligner** (*str*) - The aligner to use
  - **multiprocess** (*bool*) - Whether to run the extraction pipeline in single process mode or not
  - **ref_threshold** (*float*) - The reference threshold for a positive match
  - **filter_lists** (*dict*) - The filter and nfilter image paths

process(*extract_media*)

Filters in or out any wanted or unwanted faces based on command line arguments.

Parameters **extract_media** (*ExtractMedia*) – The *ExtractMedia* object to perform the face filtering on.

Returns The original *ExtractMedia* with any requested filters applied

Return type *ExtractMedia*

class scripts.fsmedia.**Images**(*arguments*)

Bases: object

Handles the loading of frames from a folder of images or a video file for extract and convert processes.

Parameters **arguments** (argparse.Namespace) – The command line arguments that were passed to Faceswap

images_found

The number of frames that exist in the video file, or the folder of images.

Type int

input_images

Path to the video file if the input is a video otherwise list of image paths.

Type str or list

is_video

True if the input is a video file otherwise False.

Type bool

load()

Generator to load frames from a folder of images or from a video file.

Yields

- **filename** (*str*) – The filename of the current frame
- **image** (numpy.ndarray) – A single frame

load_one_image(*filename*)

Obtain a single image for the given filename.

Parameters **filename** (*str*) – The filename to return the image for

Returns The image for the requested filename,

Return type numpy.ndarray

class scripts.fsmedia.**PostProcess**(*arguments*)
Bases: object

Optional pre/post processing tasks for convert and extract.

Builds a pipeline of actions that have optionally been requested to be performed in this session.

Parameters **arguments** (argparse.Namespace) – The command line arguments that were passed to Faceswap

do_actions(*extract_media*)
Perform the requested optional post-processing actions on the given image.

Parameters **extract_media** (*ExtractMedia*) – The *ExtractMedia* object to perform the action on.

Returns The original *ExtractMedia* with any actions applied

Return type *ExtractMedia*

class scripts.fsmedia.**PostProcessAction**(**args*, ***kwargs*)
Bases: object

Parent class for Post Processing Actions.

Usable in Extract or Convert or both depending on context. Any post-processing actions should inherit from this class.

Parameters

- **args** (*tuple*) – Varies for specific post process action
- **kwargs** (*dict*) – Varies for specific post process action

process(*extract_media*)
Override for specific post processing action

Parameters **extract_media** (*ExtractMedia*) – The *ExtractMedia* object to perform the action on.

valid
True if the action if the parameters passed in for this action are valid, otherwise False

Type bool

scripts.fsmedia.**finalize**(*images_found*, *num_faces_detected*, *verify_output*)
Output summary statistics at the end of the extract or convert processes.

Parameters

- **images_found** (*int*) – The number of images/frames that were processed
- **num_faces_detected** (*int*) – The number of faces that have been detected
- **verify_output** (*bool*) – True if multiple faces were detected in frames otherwise False.

1.4 tools package

The Tools Package provides various tools for working with Faceswap outside of the core functionality.

Contents

1.4.1 Subpackages

manual package

Contents

Subpackages

The following subpackages handle the main two display areas of the Manual Tool's GUI.

faceviewer package

Handles the display of faces in the Face Viewer section of Faceswap's Manual Tool.

Contents

frame module

Module Summary

`ContextMenu`	Enables a right click context menu for the `FacesViewer`.
`FacesActionsFrame`	The left hand action frame holding the optional annotation buttons.
`FacesFrame`	The faces display frame (bottom section of GUI).

Continued on next page

Table 22 – continued from previous page

FacesViewer	The `tkinter.Canvas` that holds the faces viewer section of the Manual Tool.
Grid	Holds information on the current filtered grid layout.

Module

The Faces Viewer Frame and Canvas for Faceswap's Manual Tool.

class `tools.manual.faceviewer.frame.`**ContextMenu**(*canvas*, *detected_faces*)
Bases: `object`

Enables a right click context menu for the *FacesViewer*.

Parameters

- **canvas** (`tkinter.Canvas`) – The *FacesViewer* canvas
- **detected_faces** (*detected_faces*) – The manual tool's detected faces class

class `tools.manual.faceviewer.frame.`**FacesActionsFrame**(*parent*)
Bases: `tkinter.ttk.Frame`

The left hand action frame holding the optional annotation buttons.

Parameters **parent** (*FacesFrame*) – The Faces frame that this actions frame reside in

key_bindings
The mapping of key presses to optional annotations to display. Keyboard shortcuts utilize the function keys.

Type dict

on_click(*display*)
Click event for the optional annotation buttons. Loads and unloads the annotations from the faces viewer.

Parameters **display** (*str*) – The display name for the button that has called this event as exists in `_buttons`

class `tools.manual.faceviewer.frame.`**FacesFrame**(*parent*, *tk_globals*, *detected_faces*, *display_frame*)
Bases: `tkinter.ttk.Frame`

The faces display frame (bottom section of GUI). This frame holds the faces viewport and the tkinter objects.

Parameters

- **parent** (`ttk.PanedWindow`) – The paned window that the faces frame resides in
- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **detected_faces** (*DetectedFaces*) – The `DetectedFace` objects for this video
- **display_frame** (*DisplayFrame*) – The section of the Manual Tool that holds the frames viewer

canvas_scroll(*direction*)
Scroll the canvas on an up/down or page-up/page-down key press.

Notes

To protect against a held down key press stacking tasks and locking up the GUI a background thread is launched and discards subsequent key presses whilst the previous update occurs.

Parameters **direction** (*["up", "down", "page-up", "page-down"]*) – The request page scroll direction and amount.

set_annotation_display(*key*)
Set the optional annotation overlay based on keyboard shortcut.

Parameters **key** (*str*) – The pressed key

class tools.manual.faceviewer.frame.**FacesViewer**(*parent*, *tk_globals*, *tk_action_vars*, *detected_faces*, *display_frame*, *event*)
Bases: tkinter.Canvas

The tkinter.Canvas that holds the faces viewer section of the Manual Tool.

Parameters

- **parent** (tkinter.ttk.Frame) – The parent frame for the canvas
- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **tk_action_vars** (*dict*) – The tkinter.BooleanVar objects for selectable optional annotations as set by the buttons in the *FacesActionsFrame*
- **detected_faces** (*DetectedFaces*) – The DetectedFace objects for this video
- **display_frame** (*DisplayFrame*) – The section of the Manual Tool that holds the frames viewer
- **event** (threading.Event) – The threading event object for repeated key press protection

canvas_scroll(*amount*, *units*, *event*)
Scroll the canvas on an up/down or page-up/page-down key press.

Parameters

- **amount** (*int*) – The number of units to scroll the canvas
- **units** (*["page", "units"]*) – The unit type to scroll by
- **event** (threading.Event) – event to indicate to the calling process whether the scroll is still updating

control_colors
The frame Editor name as key with the current user selected hex code as value.

Type dict

face_size
The currently selected thumbnail size in pixels

Type int

get_muted_color(*color_key*)
Creates a muted version of the given annotation color for non-active faces.

Parameters **color_key** (*str*) – The annotation key to obtain the color for from *control_colors*

grid
The grid for the current *FacesViewer*.

Type *Grid*

optional_annotations
The values currently set for the selectable optional annotations.

Type dict

refresh_grid(*trigger_var*, *retain_position=False*)
Recalculate the full grid and redraw. Used when the active filter pull down is used, a face has been added or removed, or the face thumbnail size has changed.

Parameters

- **trigger_var** (tkinter.BooleanVar) – The tkinter variable that has triggered the grid update. Will either be the variable indicating that the face size have been changed, or the variable indicating that the selected filter mode has been changed.
- **retain_position** (*bool, optional*) – `True` if the grid should be set back to the position it was at after the update has been processed, otherwise `False`. Default: `False`.

selected_mask
The currently selected mask from the display frame control panel.

Type str

viewport
The viewport area of the faces viewer.

Type *Viewport*

class tools.manual.faceviewer.frame.**Grid**(*canvas*, *detected_faces*)
Bases: object

Holds information on the current filtered grid layout.

The grid keeps information on frame indices, face indices, x and y positions and detected face objects laid out in a numpy array to reflect the current full layout of faces within the face viewer based on the currently selected filter and face thumbnail size.

Parameters

- **canvas** (tkinter.Canvas) – The *FacesViewer* canvas
- **detected_faces** (*DetectedFaces*) – The DetectedFace objects for this video

columns_rows
the (*columns*, *rows*) required to hold all display images.

Type tuple

dimensions
The (*width*, *height*) required to hold all display images.

Type tuple

face_size
The pixel size of each thumbnail within the face viewer.

Type int

frame_has_faces(*frame_index*)
Check whether the given frame index contains any faces.

Parameters **frame_index** (*int*) – The frame index to locate in the grid

Returns `True` if there are faces in the given frame otherwise `False`

Return type bool

is_valid
`True` if the current filter means that the grid holds faces. `False` if there are no faces displayed in the grid.

Type bool

transport_index_from_frame (*frame_index*)
Return the main frame's transport index for the given frame index based on the current filter criteria.

Parameters **frame_index** (*int*) – The absolute index for the frame within the full frames list

Returns The index of the requested frame within the filtered frames view.

Return type int

update()
Update the underlying grid.

Called on initialization, on a filter change or on add/remove faces. Recalculates the underlying grid for the current filter view and updates the attributes `_grid`, `_display_faces`, `_raw_indices`, `_frames_list` and *is_valid*

visible_area
A numpy array of shape (*4*, *rows*, *columns*) corresponding to the viewable area of the display grid. 1st dimension contains frame indices, 2nd dimension face indices. The 3rd and 4th dimension contain the x and y position of the top left corner of the face respectively.

Any locations that are not populated by a face will have a frame and face index of -1

Type `numpy.ndarray`

y_coord_from_frame (*frame_index*)
Return the y coordinate for the first face that appears in the given frame.

Parameters **frame_index** (*int*) – The frame index to locate in the grid

Returns The y coordinate of the first face for the given frame

Return type int

viewport module

Module Summary

ActiveFrame	Handles the display of faces and annotations for the currently active frame.
HoverBox	Handle the current mouse location when over the `Viewport`.
TKFace	An object that holds a single `tkinter.PhotoImage` face, ready for placement in the `Viewport`, Handles the placement of and removal of masks for the face as well as updates on any edits.
Viewport	Handles the display of faces and annotations in the currently viewable area of the canvas.
VisibleObjects	Holds the objects from the *Grid* that appear in the viewable area of the `Viewport`.

Module

Handles the visible area of the *FacesViewer* canvas.

class tools.manual.faceviewer.viewport.**ActiveFrame**(*viewport*, *tk_edited_variable*)
Bases: object

Handles the display of faces and annotations for the currently active frame.

Parameters

- **canvas** (tkinter.Canvas) – The *FacesViewer* canvas
- **tk_edited_variable** (tkinter.BooleanVar) – The tkinter callback variable indicating that a face has been edited

current_frame
A BGR version of the frame currently being displayed.

Type numpy.ndarray

frame_index
The frame index of the currently displayed frame.

Type int

move_to_top()
Move the currently selected frame's faces to the top of the viewport if they are moving off the bottom of the viewer.

reload_annotations()
Handles the reloading of annotations for the currently active faces.

Highlights the faces within the viewport of those faces that exist in the currently displaying frame. Applies annotations based on the optional annotations and current editor selections.

class tools.manual.faceviewer.viewport.**HoverBox**(*viewport*)
Bases: object

Handle the current mouse location when over the *Viewport*.

Highlights the face currently underneath the cursor and handles actions when clicking on a face.

Parameters **viewport** (*Viewport*) – The viewport object for the *FacesViewer* canvas

on_hover(*event*)
Highlight the face and set the mouse cursor for the mouse's current location.

Parameters **event** (tkinter.Event or None) – The tkinter mouse event. Provides the current location of the mouse cursor. If None is passed as the event (for example when this function is being called outside of a mouse event) then the location of the cursor will be calculated

class tools.manual.faceviewer.viewport.**TKFace**(*face*, *size=128*, *mask=None*)
Bases: object

An object that holds a single tkinter.PhotoImage face, ready for placement in the *Viewport*, Handles the placement of and removal of masks for the face as well as updates on any edits.

Parameters

- **face** (numpy.ndarray) – The face, sized correctly as a 3 channel BGR image or an encoded jpg to create a tkinter.PhotoImage from
- **size** (*int, optional*) – The pixel size of the face image. Default: *128*

- **mask** (numpy.ndarray or None, optional) – The mask to be applied to the face image. Pass None if no mask is to be used. Default None

photo
The face in a format that can be placed on the *FacesViewer* canvas.

Type tkinter.PhotoImage

update(*face*, *mask*)
Update the *photo* with the given face and mask.

Parameters

- **face** (numpy.ndarray) – The face, sized correctly as a 3 channel BGR image
- **mask** (numpy.ndarray or None) – The mask to be applied to the face image. Pass None if no mask is to be used

update_mask(*mask*)
Update the mask in the 4th channel of *photo* to the given mask.

Parameters **mask** (numpy.ndarray or None) – The mask to be applied to the face image. Pass None if no mask is to be used

class tools.manual.faceviewer.viewport.**Viewport**(*canvas*, *tk_edited_variable*)
Bases: object

Handles the display of faces and annotations in the currently viewable area of the canvas.

Parameters

- **canvas** (tkinter.Canvas) – The *FacesViewer* canvas
- **tk_edited_variable** (tkinter.BooleanVar) – The variable that indicates that a face has been edited

face_from_point(*point_x*, *point_y*)
Given an (x, y) point on the *Viewport*, obtain the face information at that location.

Parameters

- **point_x** (*int*) – The x position on the canvas of the point to retrieve the face for
- **point_y** (*int*) – The y position on the canvas of the point to retrieve the face for

Returns

Array of shape (4,) containing the (*frame index*, *face index*, *x_point of top left corner*, *y point of top left corner*) of the face at the given coordinates.

If the given coordinates are not over a face, then the frame and face indices will be -1

Return type numpy.ndarray

face_size
The pixel size of each thumbnail

Type int

get_landmarks(*frame_index*, *face_index*, *face*, *top_left*, *refresh=False*)
Obtain the landmark points for each mesh annotation.

First tries to obtain the aligned landmarks from the cache. If the landmarks do not exist in the cache, or a refresh has been requested, then the landmarks are calculated from the detected face object.

Parameters

- **frame_index** (*int*) – The frame index to obtain the face for
- **face_index** (*int*) – The face index of the face within the requested frame
- **face** (lib.align.DetectedFace) – The detected face object to obtain landmarks for
- **top_left** (*tuple*) – The top left (x, y) points of the face's bounding box within the viewport
- **refresh** (*bool, optional*) – Whether to force a reload of the face's aligned landmarks, even if they already exist within the cache. Default: False

Returns The key is the tkinter canvas object type for each part of the mesh annotation (*polygon*, *line*). The value is a list containing the (x, y) coordinates of each part of the mesh annotation, from the top left corner location.

Return type dict

get_tk_face(*frame_index*, *face_index*, *face*)

Obtain the *TKFace* object for the given face from the cache. If the face does not exist in the cache, then it is generated and added prior to returning.

Parameters

- **frame_index** (*int*) – The frame index to obtain the face for
- **face_index** (*int*) – The face index of the face within the requested frame
- **face** (DetectedFace) – The detected face object, containing the thumbnail jpg

Returns An object for displaying in the faces viewer canvas populated with the aligned mesh landmarks and face thumbnail

Return type *TKFace*

hover_box

The hover box for the viewport.

Type *HoverBox*

mesh_kwargs

The color and state keyword arguments for the objects that make up a single face's mesh annotation based on the current user selected options. Key is the object type (*polygon* or *line*), value are the keyword arguments for that type.

Type dict

move_active_to_top()

Check whether the active frame is going off the bottom of the viewport, if so: move it to the top of the viewport.

reset()

Reset all the cached objects on a face size change.

selected_editor

The currently selected editor.

Type str

toggle_mask(*state*, *mask_type*)

Toggles the mask optional annotation on and off.

Parameters

- **state** (*["hidden", "normal"]*) – Whether the mask should be displayed or hidden
- **mask_type** (*str*) – The type of mask to overlay onto the face

toggle_mesh(*state*)
Toggles the mesh optional annotations on and off.

Parameters **state** (*["hidden", "normal"]*) – The state to set the mesh annotations to

update(*refresh_annotations=False*)
Update the viewport.

Parameters

- **refresh_annotations** (*bool, optional*) – `True` if mesh annotations should be re-calculated otherwise `False`. Default: `False`
- **the objects that are currently visible. Updates the visible area of the canvas** (*Obtains*) –
- **reloads the active frame's annotations.** (*and*) –

class tools.manual.faceviewer.viewport.**VisibleObjects**(*viewport*)
Bases: `object`

Holds the objects from the *Grid* that appear in the viewable area of the *Viewport*.

Parameters **viewport** (*Viewport*) – The viewport object for the *FacesViewer* canvas

images
The viewport's tkinter canvas image objects.

A numpy array of shape (*rows*, *columns*) corresponding to the viewable area of the display grid and containing the tkinter canvas image object for the face at the corresponding location.

Type `numpy.ndarray`

meshes
The viewport's tkinter canvas mesh annotation objects.

A numpy array of shape (*rows*, *columns*) corresponding to the viewable area of the display grid and containing a dictionary of the corresponding tkinter polygon and line objects required to build a face's mesh annotation for the face at the corresponding location.

Type `numpy.ndarray`

update()
Load and unload thumbnails in the visible area of the faces viewer.

visible_faces
The currently visible `DetectedFace` objects.

A numpy array of shape (*rows*, *columns*) corresponding to the viewable area of the display grid and containing the detected faces at their currently viewable position.

Any locations that are not populated by a face will have `None` in it's place.

Type `numpy.ndarray`

visible_grid
The currently visible section of the *Grid*

A numpy array of shape (*4*, *rows*, *columns*) corresponding to the viewable area of the display grid. 1st dimension contains frame indices, 2nd dimension face indices. The 3rd and 4th dimension contain the x and y position of the top left corner of the face respectively.

Any locations that are not populated by a face will have a frame and face index of -1.

Type numpy.ndarray

frameviewer module

Handles the display of frames in the Frame Viewer section of Faceswap's Manual Tool.

Contents

frame module

Module Summary

ActionsFrame	The left hand action frame holding the action buttons.
BackgroundImage	The background image of the canvas
DisplayFrame	The main video display frame (top left section of GUI).
FrameViewer	Annotation onto tkInter Canvas.
Navigation	Handles playback and frame navigation for the Frame Viewer Window.

Module

The frame viewer section of the manual tool GUI

class tools.manual.frameviewer.frame.**ActionsFrame**(*parent*)
Bases: tkinter.ttk.Frame

The left hand action frame holding the action buttons.

Parameters **parent** (*DisplayFrame*) – The Display frame that the Actions reside in

actions
The available action names as a tuple of strings.

Type tuple

add_optional_buttons(*editors*)
Add the optional editor specific action buttons

key_bindings
action}. The mapping of key presses to actions. Keyboard shortcut is the first letter of each action.

Type dict

Type {*key*

on_click(*action*)
Click event for all of the main buttons.

Parameters **action** (*str*) – The action name for the button that has called this event as exists in _buttons

tk_selected_action
The variable holding the currently selected action

Type tkinter.StringVar

class tools.manual.frameviewer.frame.**DisplayFrame**(*parent*, *tk_globals*, *detected_faces*)
Bases: tkinter.ttk.Frame

The main video display frame (top left section of GUI).

Parameters

- **parent** (ttk.PanedWindow) – The paned window that the display frame resides in
- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **detected_faces** (*tools.manual.detected_faces.DetectedFaces*) – The detected faces stored in the alignments file

active_editor
The current editor in use based on selected_action.

Type Editor

cycle_filter_mode()
Cycle the navigation mode combo entry

editors
All of the Editor that the canvas holds

Type dict

navigation
Class that handles frame Navigation and transport.

Type *Navigation*

set_action(*key*)
Set the current action based on keyboard shortcut

Parameters **key** (*str*) – The pressed key

tk_control_colors
Editor key with tkinter.StringVar containing the selected color hex code for each annotation

Type dict

tk_selected_action
The variable holding the currently selected action

Type tkinter.StringVar

tk_selected_mask
Editor key with tkinter.StringVar containing the selected color hex code for each annotation

Type dict

class tools.manual.frameviewer.frame.**FrameViewer**(*parent*, *tk_globals*, *detected_faces*, *actions*, *tk_action_var*)

Bases: tkinter.Canvas

Annotation onto tkInter Canvas.

Parameters

- **parent** (tkinter.ttk.Frame) – The parent frame for the canvas
- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **detected_faces** (AlignmentsData) – The alignments data for this manual session
- **actions** (*tuple*) – The available actions from ActionFrame.actions
- **tk_action_var** (tkinter.StringVar) – The variable holding the currently selected action

active_editor
The current editor in use based on *selected_action*.

Type Editor

annotation_formats
The selected formatting options for each annotation

Type dict

control_tk_vars
dictionary of tkinter variables as populated by the right hand control panel. Tracking for all control panel variables, for access from all editors.

Type dict

editor_display
List of editors and any additional annotations they should display.

Type dict

editors
All of the Editor objects that exist

Type dict

key_bindings
dictionary of key bindings for each editor for access from all editors.

Type dict

offset
The (*width*, *height*) offset of the canvas based on the size of the currently displayed image

Type tuple

selected_action
The name of the currently selected Editor action

Type str

control module

Module Summary

BackgroundImage	The background image of the canvas
Navigation	Handles playback and frame navigation for the Frame Viewer Window.

Module

Handles Navigation and Background Image for the Frame Viewer section of the manual tool GUI.

class tools.manual.frameviewer.control.**BackgroundImage**(*canvas*)
Bases: object

The background image of the canvas

refresh(*view_mode*)
Update the displayed frame.

Parameters **view_mode** (*["frame", "face"]*) – The currently active editor's selected view mode.

class tools.manual.frameviewer.control.**Navigation**(*display_frame*)
Bases: object

Handles playback and frame navigation for the Frame Viewer Window.

Parameters **display_frame** (DisplayFrame) – The parent frame viewer window

decrement_frame()
Update The frame navigation position to the previous frame based on filter.

goto_first_frame()
Go to the first frame that meets the filter criteria.

goto_last_frame()
Go to the last frame that meets the filter criteria.

handle_play_button()
Handle the play button.

Switches the *tk_is_playing* variable.

increment_frame(*frame_count=None*, *is_playing=False*)
Update The frame navigation position to the next frame based on filter.

nav_scale_callback(**args*, *reset_progress=True*)
Adjust transport slider scale for different filters.

Returns True if the navigation scale has been updated otherwise False

Return type bool

stop_playback()
Stop play back if playing

tk_is_playing
Whether the stream is currently playing.

Type tkinter.BooleanVar

editor package

Contents

_base module

Module Summary

Editor	Parent Class for Object Editors.
View	The view Editor.

Module

Editor objects for the manual adjustments tool

class tools.manual.frameviewer.editor._base.**Editor**(*canvas*, *detected_faces*, *control_text=''*, *key_bindings=None*)

Bases: object

Parent Class for Object Editors.

Editors allow the user to use a variety of tools to manipulate alignments from the main display frame.

Parameters

- **canvas** (tkinter.Canvas) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session
- **control_text** (*str*) – The text that is to be displayed at the top of the Editor's control panel.

actions
The optional action buttons for the actions frame in the GUI for the current editor

Type list

bind_mouse_motion()
Binds the mouse motion for the current editor's mouse <Motion> event to the editor's _update_cursor() function.

Called on initialization and active editor update.

controls
The control panel options and header text for the current editor

Type dict

hide_annotation(*tag=None*)
Hide annotations for this editor.

Parameters **tag** (*str, optional*) – The specific tag to hide annotations for. If `None` then all annotations for this editor are hidden, otherwise only the annotations specified by the given tag are hidden. Default: `None`

scale_from_display(*points*, *do_offset=True*)
Scale and offset the given points from the current display to the correct original values.

Parameters

- **points** (`numpy.ndarray`) – Array of x, y co-ordinates to adjust
- **offset** (*bool, optional*) – `True` if the offset should be calculated otherwise `False`. Default: `True`

Returns The adjusted x, y co-ordinates to the original frame location rounded to the nearest integer

Return type `numpy.ndarray`

set_mouse_click_actions()
Add the bindings for left mouse button click and drag actions.

This binds the mouse to the `_drag_start()`, `_drag()` and `_drag_stop()` methods.

By default these methods do nothing (except for `_drag_stop()` which resets `_drag_data`.

This bindings should be added for all editors. To add additional bindings, *super().set_mouse_click_actions* should be called prior to adding them..

update_annotation()
Update the display annotations for the current objects.

Override for specific editors.

view_mode
The view mode for the currently selected editor. If the editor does not have a view mode that can be updated, then *"frame"* will be returned.

Type ["frame", "face"]

class tools.manual.frameviewer.editor._base.**View**(*canvas*, *detected_faces*)
Bases: *tools.manual.frameviewer.editor._base.Editor*

The view Editor.

Does not allow any editing, just used for previewing annotations.

This is the default start-up editor.

Parameters

- **canvas** (`tkinter.Canvas`) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session

bounding_box module

Bounding Box Editor for the manual adjustments tool

class tools.manual.frameviewer.editor.bounding_box.**BoundingBox**(*canvas*, *detected_faces*)

Bases: *tools.manual.frameviewer.editor._base.Editor*

The Bounding Box Editor.

Adjusting the bounding box feeds the aligner to generate new 68 point landmarks.

Parameters

- **canvas** (tkinter.Canvas) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session

set_mouse_click_actions()
Add context menu to OS specific right click action.

update_annotation()
Get the latest bounding box data from alignments and update.

extract_box module

Extract Box Editor for the manual adjustments tool

class tools.manual.frameviewer.editor.extract_box.**ExtractBox**(*canvas*, *detected_faces*)

Bases: *tools.manual.frameviewer.editor._base.Editor*

The Extract Box Editor.

Adjust the calculated Extract Box to shift all of the 68 point landmarks in place.

Parameters

- **canvas** (tkinter.Canvas) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session

set_mouse_click_actions()
Add context menu to OS specific right click action.

update_annotation()
Draw the latest Extract Boxes around the faces.

landmarks module

Module Summary

Landmarks	The Landmarks Editor.
Mesh	The Landmarks Mesh Display.

Module

Landmarks Editor and Landmarks Mesh viewer for the manual adjustments tool

class tools.manual.frameviewer.editor.landmarks.**Landmarks**(*canvas*, *detected_faces*)

Bases: *tools.manual.frameviewer.editor._base.Editor*

The Landmarks Editor.

Adjust individual landmark points and re-generate Extract Box.

Parameters

- **canvas** (tkinter.Canvas) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session

update_annotation()
Get the latest Landmarks points and update.

class tools.manual.frameviewer.editor.landmarks.**Mesh**(*canvas*, *detected_faces*)
Bases: *tools.manual.frameviewer.editor._base.Editor*

The Landmarks Mesh Display.

There are no editing options for Mesh editor. It is purely aesthetic and updated when other editors are used.

Parameters

- **canvas** (tkinter.Canvas) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session

update_annotation()
Get the latest Landmarks and update the mesh.

mask module

Mask Editor for the manual adjustments tool

class tools.manual.frameviewer.editor.mask.**Mask**(*canvas*, *detected_faces*)
Bases: *tools.manual.frameviewer.editor._base.Editor*

The mask Editor.

Edit a mask in the alignments file.

Parameters

- **canvas** (tkinter.Canvas) – The canvas that holds the image and annotations
- **detected_faces** (*DetectedFaces*) – The _detected_faces data for this manual session

hide_annotation(*tag=None*)
Clear the mask _meta dict when hiding the annotation.

update_annotation()
Update the mask annotation with the latest mask.

manual module

The Manual Module is the main entry point into the Manual Editor Tool.

Module Summary

Aligner	The `Aligner` class sets up an extraction pipeline for each of the current Faceswap Aligners, along with the Landmarks based Maskers.
FrameLoader	Loads the frames, sets the frame count to `TkGlobals.frame_count` and handles the return of the correct frame for the GUI.
Manual	The main entry point for Faceswap's Manual Editor Tool.
TkGlobals	Holds Tkinter Variables and other frame information that need to be accessible from all areas of the GUI.

Module

The Manual Tool is a tkinter driven GUI app for editing alignments files with visual tools. This module is the main entry point into the Manual Tool.

class `tools.manual.manual.`**Aligner**(*tk_globals*, *exclude_gpus*)

Bases: `object`

The *Aligner* class sets up an extraction pipeline for each of the current Faceswap Aligners, along with the Landmarks based Maskers. When new landmarks are required, the bounding boxes from the GUI are passed to this class for pushing through the pipeline. The resulting Landmarks and Masks are then returned.

Parameters

- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **exclude_gpus** (list or `None`) – A list of indices correlating to connected GPUs that Tensorflow should not use. Pass `None` to not exclude any GPUs.

get_landmarks(*frame_index*, *face_index*, *aligner*)

Feed the detected face into the alignment pipeline and retrieve the landmarks.

The face to feed into the aligner is generated from the given frame and face indices.

Parameters

- **frame_index** (*int*) – The frame index to extract the aligned face for
- **face_index** (*int*) – The face index within the current frame to extract the face for
- **aligner** (*["FAN", "cv2-dnn"]*) – The aligner to use to extract the face

Returns The 68 point landmark alignments

Return type `numpy.ndarray`

get_masks(*frame_index*, *face_index*)

Feed the aligned face into the mask pipeline and retrieve the updated masks.

The face to feed into the aligner is generated from the given frame and face indices. This is to be called when a manual update is done on the landmarks, and new masks need generating

Parameters

- **frame_index** (*int*) – The frame index to extract the aligned face for
- **face_index** (*int*) – The face index within the current frame to extract the face for

Returns The updated masks

Return type dict

is_initialized
The Aligners are initialized in a background thread so that other tasks can be performed whilst we wait for initialization. `True` is returned if the aligner has completed initialization otherwise `False`.

Type bool

link_faces(*detected_faces*)
As the Aligner has the potential to take the longest to initialize, it is kicked off as early as possible. At this time *DetectedFaces* is not yet available.

Once the Aligner has initialized, this function is called to add the *DetectedFaces* class as a property of the Aligner.

Parameters **detected_faces** (*DetectedFaces*) – The class that holds the `DetectedFace` objects for the current Manual session

set_normalization_method(*method*)
Change the normalization method for faces fed into the aligner. The normalization method is user adjustable from the GUI. When this method is triggered the method is updated for all aligner pipelines.

Parameters **method** (*str*) – The normalization method to use

class tools.manual.manual.**FrameLoader**(*tk_globals*, *frames_location*, *video_meta_data*)
Bases: `object`

Loads the frames, sets the frame count to *TkGlobals.frame_count* and handles the return of the correct frame for the GUI.

Parameters

- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **frames_location** (*str*) – The path to the input frames
- **video_meta_data** (*dict*) – The meta data held within the alignments file, if it exists and the input is a video

is_initialized
`True` if the Frame Loader has completed initialization otherwise `False`.

Type bool

video_meta_data
The pts_time and key frames for the loader.

Type dict

class tools.manual.manual.**Manual**(*arguments*)
Bases: `tkinter.Tk`

The main entry point for Faceswap's Manual Editor Tool. This tool is part of the Faceswap Tools suite and should be called from `python tools.py manual` command.

Allows for visual interaction with frames, faces and alignments file to perform various adjustments to the alignments file.

Parameters **arguments** (`argparse.Namespace`) – The `argparse` arguments as passed in from `tools.py`

process()
The entry point for the Visual Alignments tool from `lib.tools.manual.cli`.

Launch the tkinter Visual Alignments Window and run main loop.

class tools.manual.manual.**TkGlobals**(*input_location*)
Bases: object

Holds Tkinter Variables and other frame information that need to be accessible from all areas of the GUI.

Parameters **input_location** (*str*) – The location of the input folder of frames or video file

current_frame
The currently displayed frame in the frame viewer with it's meta information. Key and Values are as follows:

image (numpy.ndarry): The currently displayed frame in original dimensions

scale (*float*): The scaling factor to use to resize the image to the display window

interpolation (*int*): The opencv interpolator ID to use for resizing the image to the display window

display_dims (*tuple*): The size of the currently displayed frame, sized for the display window

filename (*str*): The filename of the currently displayed frame

Type dict

face_index
The currently displayed face index when in zoomed mode.

Type int

filter_mode
The currently selected navigation mode.

Type str

frame_count
The total number of frames for the input location

Type int

frame_display_dims
The (*width*, *height*) of the video display frame in pixels.

Type tuple

frame_index
The currently displayed frame index. NB This returns -1 if there are no frames that meet the currently selected filter criteria.

Type int

is_video
True if the input is a video file, False if it is a folder of images.

Type bool

is_zoomed
True if the frame viewer is zoomed into a face, False if the frame viewer is displaying a full frame.

Type bool

set_current_frame(*image*, *filename*)
Set the frame and meta information for the currently displayed frame. Populates the attribute *current_frame*

Parameters

- **image** (numpy.ndarray) – The image used to display in the Frame Viewer
- **filename** (*str*) – The filename of the current frame

set_frame_count(*count*)
Set the count of total number of frames to *frame_count* when the FramesLoader has completed loading.

Parameters **count** (*int*) – The number of frames that exist for this session

set_frame_display_dims(*width*, *height*)
Set the size, in pixels, of the video frame display window and resize the displayed frame.

Used on a frame resize callback, sets the :attr:frame_display_dims‘.

Parameters

- **width** (*int*) – The width of the frame holding the video canvas in pixels
- **height** (*int*) – The height of the frame holding the video canvas in pixels

tk_face_index
The variable that holds the face index of the selected face within the current frame when in zoomed mode.

Type tkinter.IntVar

tk_faces_size
The variable holding the currently selected Faces Viewer thumbnail size.

Type tkinter.StringVar

tk_filter_mode
The variable holding the currently selected navigation filter mode.

Type tkinter.StringVar

tk_frame_index
The variable holding the current frame index.

Type tkinter.IntVar

tk_is_zoomed
The variable holding the value indicating whether the frame viewer is zoomed into a face or zoomed out to the full frame.

Type tkinter.BooleanVar

tk_transport_index
The current index of the display frame's transport slider.

Type tkinter.IntVar

tk_update
The variable holding the trigger that indicates that a full update needs to occur.

Type tkinter.BooleanVar

tk_update_active_viewport
Boolean Variable that is traced by the viewport's active frame to update..

Type tkinter.BooleanVar

detected_faces module

Module Summary

DetectedFaces	Handles the manipulation of DetectedFace objects stored in the alignments file.
FaceUpdate	Perform updates on DetectedFace objects stored in DetectedFaces when changes are made within the GUI.
Filter	Returns stats and frames for filtered frames based on the user selected navigation mode filter.
ThumbsCreator	Background loader to generate thumbnails for the alignments file.

Module

Alignments handling for Faceswap's Manual Adjustments tool. Handles the conversion of alignments data to DetectedFace objects, and the update of these faces when edits are made in the GUI.

class tools.manual.detected_faces.**DetectedFaces**(*tk_globals*, *alignments_path*, *input_location*, *extractor*)

Bases: object

Handles the manipulation of DetectedFace objects stored in the alignments file. Acts as a parent class for the IO operations (saving and loading from an alignments file), the face update operations (when changes are made to alignments in the GUI) and the face filters (when a user changes the filter navigation mode.)

Parameters

- **tk_globals** (*TkGlobals*) – The tkinter variables that apply to the whole of the GUI
- **alignments_path** (*str*) – The full path to the alignments file
- **input_location** (*str*) – The location of the input folder of frames or video file
- **extractor** (*Aligner*) – The pipeline for passing faces through the aligner and retrieving results

available_masks
The mask type names stored in the alignments; type as key with the number of faces which possess the mask type as value.

Type dict

current_faces
The most up to date full list of DetectedFace objects.

Type list

extract()
Extract the faces in the current video to a user supplied folder.

extractor
The pipeline for passing faces through the aligner and retrieving results.

Type *Aligner*

face_count_per_index
Count of faces for each frame. List is in frame index order.

The list needs to be calculated on the fly as the number of faces in a frame can change based on user actions.

Type list

filter
Handles returning of faces and stats based on the current user set navigation mode filter.

Type *Filter*

is_frame_updated(*frame_index*)
bool: `True` if the given frame index has updated faces within it otherwise `False`

load_faces()
Load the faces as `DetectedFace` objects from the alignments file.

revert_to_saved(*frame_index*)
Revert the frame's alignments to their saved version for the given frame index.

Parameters **frame_index** (*int*) – The frame that should have their faces reverted to their saved version

save()
Save the alignments file with the latest edits.

save_video_meta_data(*pts_time*, *keyframes*)
Save video meta data to the alignments file. This is executed if the video meta data does not already exist in the alignments file, so the video does not need to be scanned on every use of the Manual Tool.

Parameters

- **pts_time** (*list*) – A list of presentation timestamps (*float*) in frame index order for every frame in the input video
- **keyframes** (*list*) – A list of frame indices corresponding to the key frames in the input video.

tk_edited
The variable indicating whether an edit has occurred meaning a GUI redraw needs to be triggered.

Type `tkinter.BooleanVar`

tk_face_count_changed
The variable indicating whether a face has been added or removed meaning the `FaceViewer` grid redraw needs to be triggered.

Type `tkinter.BooleanVar`

tk_unsaved
The variable indicating whether the alignments have been updated since the last save.

Type `tkinter.BooleanVar`

update
Handles the adding, removing and updating of `DetectedFace` stored within the alignments file.

Type *FaceUpdate*

video_meta_data
The frame meta data stored in the alignments file. If data does not exist in the alignments file then `None` is returned for each Key

Type dict

class tools.manual.detected_faces.**FaceUpdate**(*detected_faces*)
Bases: object

Perform updates on DetectedFace objects stored in *DetectedFaces* when changes are made within the GUI.

Parameters **detected_faces** (*DetectedFaces*) – The parent *DetectedFaces* object

add(*frame_index*, *pnt_x*, *width*, *pnt_y*, *height*)
Add a DetectedFace object to the current frame with the given dimensions.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **pnt_x** (*int*) – The left point of the bounding box
- **width** (*int*) – The width of the bounding box
- **pnt_y** (*int*) – The top point of the bounding box
- **height** (*int*) – The height of the bounding box

bounding_box(*frame_index*, *face_index*, *pnt_x*, *width*, *pnt_y*, *height*, *aligner='FAN'*)
Update the bounding box for the DetectedFace object at the given frame and face indices, with the given dimensions and update the 68 point landmarks from the *Aligner* for the updated bounding box.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame
- **pnt_x** (*int*) – The left point of the bounding box
- **width** (*int*) – The width of the bounding box
- **pnt_y** (*int*) – The top point of the bounding box
- **height** (*int*) – The height of the bounding box
- **aligner** (*["cv2-dnn", "FAN"], optional*) – The aligner to use to generate the landmarks. Default: "FAN"

copy(*frame_index*, *direction*)
Copy the alignments from the previous or next frame that has alignments to the current frame.

Parameters

- **frame_index** (*int*) – The frame that the needs to have alignments copied to it
- **direction** (*["prev", "next"]*) – Whether to copy alignments from the previous frame with alignments, or the next frame with alignments

delete(*frame_index*, *face_index*)
Delete the DetectedFace object for the given frame and face indices.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame

landmark(*frame_index*, *face_index*, *landmark_index*, *shift_x*, *shift_y*, *is_zoomed*)
Shift a single landmark point for the DetectedFace object at the given frame and face indices by the given x and y values.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame
- **landmark_index** (*int or list*) – The landmark index to shift. If a list is provided, this should be a list of landmark indices to be shifted
- **shift_x** (*int*) – The amount to shift the landmark by along the x axis
- **shift_y** (*int*) – The amount to shift the landmark by along the y axis
- **is_zoomed** (*bool*) – `True` if landmarks are being adjusted on a zoomed image otherwise `False`

landmarks (*frame_index*, *face_index*, *shift_x*, *shift_y*)
Shift all of the landmarks and bounding box for the `DetectedFace` object at the given frame and face indices by the given x and y values and update the masks.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame
- **shift_x** (*int*) – The amount to shift the landmarks by along the x axis
- **shift_y** (*int*) – The amount to shift the landmarks by along the y axis

Notes

Whilst the bounding box does not need to be shifted, it is anyway, to ensure that it is aligned with the newly adjusted landmarks.

landmarks_rotate (*frame_index*, *face_index*, *angle*, *center*)
Rotate the landmarks on an Extract Box rotate for the `DetectedFace` object at the given frame and face indices for the given angle from the given center point.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame
- **angle** (`numpy.ndarray`) – The angle, in radians to rotate the points by
- **center** (`numpy.ndarray`) – The center point of the Landmark's Extract Box

landmarks_scale (*frame_index*, *face_index*, *scale*, *center*)
Scale the landmarks on an Extract Box resize for the `DetectedFace` object at the given frame and face indices from the given center point.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame
- **scale** (*float*) – The amount to scale the landmarks by
- **center** (`numpy.ndarray`) – The center point of the Landmark's Extract Box

mask (*frame_index*, *face_index*, *mask*, *mask_type*)
Update the mask on an edit for the `DetectedFace` object at the given frame and face indices, for the given mask and mask type.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame
- **mask** (class:*numpy.ndarray*:) – The mask to replace
- **mask_type** (*str*) – The name of the mask that is to be replaced

post_edit_trigger(*frame_index*, *face_index*)
Update the jpg thumbnail and the viewport thumbnail on a face edit.

Parameters

- **frame_index** (*int*) – The frame that the face is being set for
- **face_index** (*int*) – The face index within the frame

class tools.manual.detected_faces.**Filter**(*detected_faces*)
Bases: object

Returns stats and frames for filtered frames based on the user selected navigation mode filter.

Parameters **detected_faces** (*DetectedFaces*) – The parent *DetectedFaces* object

count
The number of frames that meet the filter criteria returned by *filter_mode*.

Type int

frames_list
The list of frame indices that meet the filter criteria returned by *filter_mode*.

Type list

raw_indices
The frame and face indices that meet the current filter criteria for each displayed face.

Type dict

class tools.manual.detected_faces.**ThumbsCreator**(*detected_faces*, *input_location*, *single_process*)
Bases: object

Background loader to generate thumbnails for the alignments file. Generates low resolution thumbnails in parallel threads for faster processing.

Parameters

- **detected_faces** (DetectedFaces) – The DetectedFace objects for this video
- **input_location** (*str*) – The location of the input folder of frames or video file

generate_cache()
Extract the face thumbnails from a video or folder of images into the alignments file.

has_thumbs
True if the underlying alignments file holds thumbnail images otherwise False.

Type bool

1.4.2 alignments module

Tools for manipulating the alignments serialized file

class tools.alignments.alignments.**Alignments**(*arguments*)
Bases: object

The main entry point for Faceswap's Alignments Tool. This tool is part of the Faceswap Tools suite and should be called from the `python tools.py alignments` command.

The tool allows for manipulation, and working with Faceswap alignments files.

Parameters **arguments** (argparse.Namespace) – The argparse arguments as passed in from tools.py

process()
The entry point for the Alignments tool from lib.tools.alignments.cli.

Launches the selected alignments job.

1.4.3 mask module

Tool to generate masks and previews of masks for existing alignments file

class tools.mask.mask.**Mask**(*arguments*)
Bases: object

This tool is part of the Faceswap Tools suite and should be called from `python tools.py mask` command.

Faceswap Masks tool. Generate masks from existing alignments files, and output masks for preview.

Parameters **arguments** (argparse.Namespace) – The argparse arguments as passed in from tools.py

process()
The entry point for the Mask tool from lib.tools.cli. Runs the Mask process

1.4.4 preview module

Module Summary

ActionFrame	Frame that holds the left hand side options panel containing the command line options.
ConfigFrame	Holds the configuration options for a convert plugin inside the OptionsBook.
ConfigTools	Tools for loading, saving, setting and retrieving configuration file values.
FacesDisplay	Compiles the 2 rows of sample faces (original and swapped) into a single image
ImagesCanvas	tkinter Canvas that holds the preview images.
OptionsBook	The notebook that holds the Convert configuration options.
Patch	The Patch pipeline
Preview	This tool is part of the Faceswap Tools suite and should be called from `python tools.py preview` command.
Samples	The display samples.

Module

Tool to preview swaps and tweak configuration prior to running a convert

class tools.preview.preview.**ActionFrame**(*parent*, *available_masks*, *has_predicted_mask*, *selected_color*, *selected_mask_type*, *config_tools*, *patch_callback*, *refresh_callback*, *tk_vars*)

Bases: tkinter.ttk.Frame

Frame that holds the left hand side options panel containing the command line options.

Parameters

- **parent** (*tkinter object*) – The parent tkinter object that holds the Action Frame
- **available_masks** (*list*) – The available masks that exist within the alignments file
- **has_predicted_mask** (*bool*) – Whether the model was trained with a mask
- **selected_color** (*str*) – The selected color adjustment type
- **selected_mask_type** (*str*) – The selected mask type
- **config_tools** (*ConfigTools*) – Tools for loading and saving configuration files
- **patch_callback** (*python function*) – The function to execute when a patch callback is received
- **refresh_callback** (*python function*) – The function to execute when a refresh callback is received
- **tk_vars** (*dict*) – Global tkinter variables. *Refresh* and *Busy* tkinter.BooleanVar

convert_args

Currently selected Command line arguments from the *ActionFrame*.

Type dict

class tools.preview.preview.**ConfigFrame**(*parent*, *config_key*, *options*)

Bases: tkinter.ttk.Frame

Holds the configuration options for a convert plugin inside the *OptionsBook*.

Parameters

- **parent** (*tkinter object*) – The tkinter object that will hold this configuration frame
- **config_key** (*str*) – The section/plugin key for these configuration options
- **options** (*dict*) – The options for this section/plugin

class tools.preview.preview.**ConfigTools**

Bases: object

Tools for loading, saving, setting and retrieving configuration file values.

tk_vars

Global tkinter variables. *Refresh* and *Busy* tkinter.BooleanVar

Type dict

config

plugins.convert._config.Config The convert configuration

config_dicts

The convert configuration options in dictionary form.

Type dict

plugins_dict
Dictionary of configuration option sections as key with a list of containing plugins as the value

Type dict

reset_config_to_default(*section=None*)
Reset the GUI parameters to their default configuration values.

Parameters **section** (*str, optional*) – The configuration section to reset the values for, If None provided then all sections are reset. Default: None

reset_config_to_saved(*section=None*)
Reset the GUI parameters to their saved values within the configuration file.

Parameters **section** (*str, optional*) – The configuration section to reset the values for, If None provided then all sections are reset. Default: None

save_config(*section=None*)
Save the configuration .ini file with the currently stored values.

Parameters **section** (*str, optional*) – The configuration section to save, If None provided then all sections are saved. Default: None

sections
The sorted section names that exist within the convert Configuration options.

Type list

update_config()
Update *config* with the currently selected values from the GUI.

class tools.preview.preview.**FacesDisplay**(*size*, *padding*, *tk_vars*)
Bases: object

Compiles the 2 rows of sample faces (original and swapped) into a single image

Parameters

- **size** (*int*) – The size of each individual face sample in pixels
- **padding** (*int*) – The amount of extra padding to apply to the outside of the face
- **tk_vars** (*dict*) – Global tkinter variables. *Refresh* and *Busy* tkinter.BooleanVar

update_source
Flag to indicate that the source images for the preview have been updated, so the preview should be recompiled.

Type bool

source
The list of numpy.ndarray source preview images for top row of display

Type list

destination
The list of numpy.ndarray swapped and patched preview images for bottom row of display

Type list

set_centering(*centering*)
The centering that the model uses is not known at initialization time. Set _centering when the model has been loaded.

Parameters **centering** (*str*) – The centering that the model was trained on

set_display_dimensions(*dimensions*)
Adjust the size of the frame that will hold the preview samples.

Parameters **dimensions** (*tuple*) – The (*width*, *height*) of the frame that holds the preview

tk_image
The compiled preview display in tkinter display format

Type PIL.ImageTk.PhotoImage

update_tk_image()
Build the full preview images and compile *tk_image* for display.

class tools.preview.preview.**ImagesCanvas**(*parent*, *tk_vars*)
Bases: tkinter.ttk.Frame

tkinter Canvas that holds the preview images.

Parameters

- **parent** (*tkinter object*) – The parent tkinter object that holds the canvas
- **tk_vars** (*dict*) – Global tkinter variables. *Refresh* and *Busy* tkinter.BooleanVar

class tools.preview.preview.**OptionsBook**(*parent*, *config_tools*, *patch_callback*)
Bases: tkinter.ttk.Notebook

The notebook that holds the Convert configuration options.

Parameters

- **parent** (*tkinter object*) – The parent tkinter object that holds the Options book
- **config_tools** (*ConfigTools*) – Tools for loading and saving configuration files
- **patch_callback** (*python function*) – The function to execute when a patch callback is received

config_tools
Tools for loading and saving configuration files

Type *ConfigTools*

class tools.preview.preview.**Patch**(*arguments*, *available_masks*, *samples*, *display*, *lock*, *trigger*, *config_tools*, *tk_vars*)
Bases: object

The Patch pipeline

Runs in it's own thread. Takes the output from the Faceswap model predictor and runs the faces through the convert pipeline using the currently selected options.

Parameters

- **arguments** (argparse.Namespace) – The argparse arguments as passed in from tools.py
- **available_masks** (*list*) – The masks that are available for convert
- **samples** (*Samples*) – The Samples for display.
- **display** (*FacesDisplay*) – The display section of the Preview GUI.
- **lock** (threading.Lock) – A threading lock to prevent multiple GUI updates at the same time.
- **trigger** (threading.Event) – An event to indicate that a converter patch should be run

- **config_tools** (*ConfigTools*) – Tools for loading and saving configuration files
- **tk_vars** (*dict*) – Global tkinter variables. *Refresh* and *Busy* `tkinter.BooleanVar`

converter_arguments
The currently selected converter command line arguments for the patch queue

Type dict

current_config
The currently set configuration for the patch queue

Type `lib.config.FaceswapConfig`

converter
The converter to use for patching the images.

Type *lib.convert.Converter*

trigger
The trigger to indicate that a patching run should commence.

Type `threading.Event`

class `tools.preview.preview.`**Preview**(*arguments*)
Bases: `tkinter.Tk`

This tool is part of the Faceswap Tools suite and should be called from `python tools.py preview` command.

Loads up 5 semi-random face swaps and displays them, cropped, in place in the final frame. Allows user to live tweak settings, before saving the final config to `./config/convert.ini`

Parameters **arguments** (`argparse.Namespace`) – The `argparse` arguments as passed in from `tools.py`

process()
The entry point for the Preview tool from `lib.tools.cli`.

Launch the tkinter preview Window and run main loop.

class `tools.preview.preview.`**Samples**(*arguments*, *sample_size*, *display*, *lock*, *trigger_patch*)
Bases: `object`

The display samples.

Obtains and holds *sample_size* semi random test faces for displaying in the preview GUI.

The file list is split into evenly sized groups of *sample_size*. When a display set is generated, a random image from each of the groups is selected to provide an array of images across the length of the video.

Parameters

- **arguments** (`argparse.Namespace`) – The `argparse` arguments as passed in from `tools.py`
- **sample_size** (*int*) – The number of samples to take from the input video/images
- **display** (*FacesDisplay*) – The display section of the Preview GUI.
- **lock** (`threading.Lock`) – A threading lock to prevent multiple GUI updates at the same time.
- **trigger_patch** (`threading.Event`) – An event to indicate that a converter patch should be run

alignments
The alignments for the preview faces

Type Alignments

generate()
Generate a sample set.

Selects *sample_size* random faces. Runs them through prediction to obtain the swap, then trigger the patch event to run the faces through patching.

predicted_images
The predicted faces output from the Faceswap model

Type list

predictor
The Predictor for the Faceswap model

Type *Predict*

sample_size
The number of samples to take from the input video/images

Type int

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

p

s

t

Index

C

D

E

F

G

H

I

J

K

L

Q

R

S

T

U

V

W

X

Y