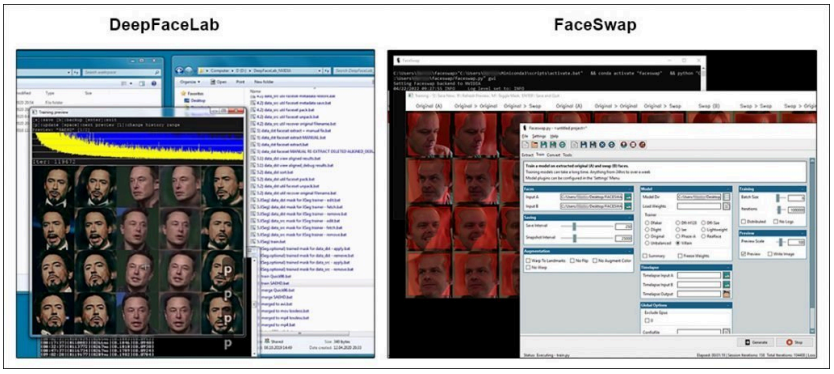
Deepfakes

Originally published at https://blog.metaphysic.ai/deepfakes/

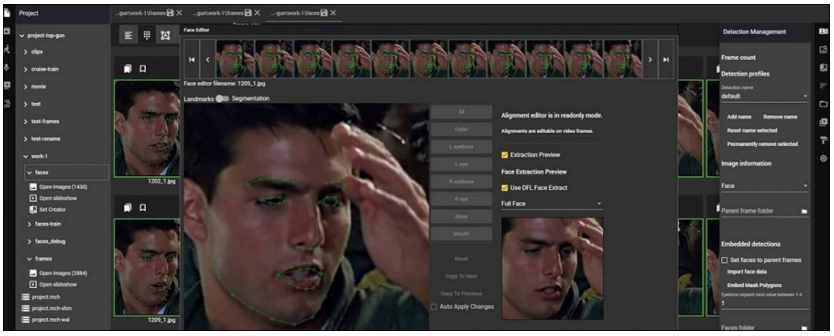
December 23, 2022

Though the term has grown in scope to take in a variety of AI-driven generative methods (such as Stable Diffusion and Generative Adversarial Networks), the term *deepfakes* came into popular usage in late 2017, after an anonymous user posted a fully-functional and relatively user-friendly implementation of autoencoder-based faceswapping to a Reddit group.
The original, anonymously-donated code was lost when its associated sub-Reddit was banned; but by this point, another user had already copied it over to the open source code platform GitHub.
Though the original code was never developed further on GitHub, In the following months, two forks of it began to gain momentum: *DeepFaceLab* (DFL) and *FaceSwap*.



DeepFaceLab and FaceSwap have become popular implementations of the original deepfakes autoencoder-based code.

Since then, both projects have developed adherents and stable communities, with a core of dedicated coders donating their time to their adopted projects, and more casual development talent occasionally contributing innovations.
Additionally, a higher-level framework called *Machine Video Editor* (MVE) has been developed as an effective GUI for the DeepFaceLab workflow, also adding diverse time-saving innovations and optimizations.



Machine Video Editor adds orchestrational functionality to DeepFaceLab, and enables project-based workflows, rather than just the linear workflows inherent in DFL. Source: https://github.com/MachineEditor/MachineVideoEditor/blob/master/images/preview-face-editor.jpg

Further, DFL has, since 2021, also branched into a live-streaming fork titled *DeepFaceLive* (usually abbreviated to *DFLive*, to avoid confusion with the source project).

<YOUTUBE LINK OMITTED>

Autoencoder Deepfake Process

Although these popular repositories are rather different in usage (DFL is largely driven on the Windows platform by batch files, while FaceSwap has a dedicated GUI written in Tk, and works on Windows or Linux), the central principles behind the original code have not been substantially altered in either project.

Extraction and Curation

'Source' and 'Target' Faces

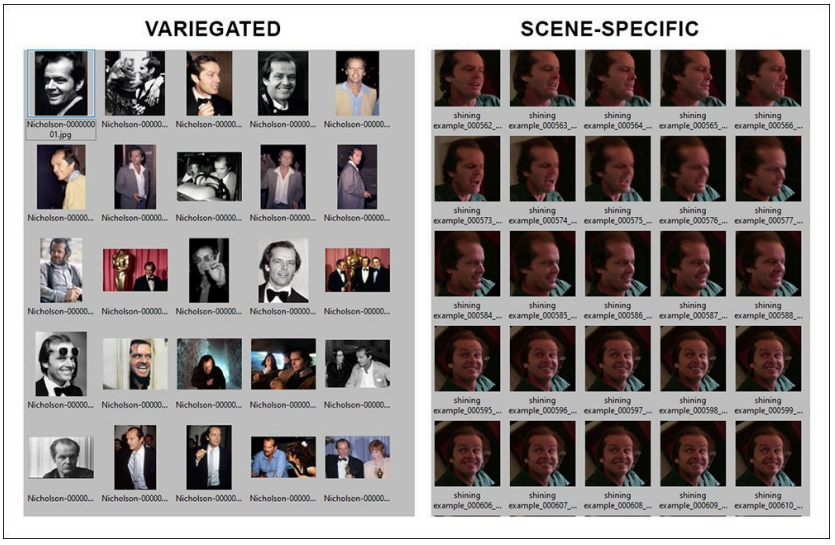
Though the terms invite an ambiguous interpretation, the *'source'* and *'target'* faces in a traditional deepfakes face-swap represent, respectively, the face that will be overwritten in a final video, and the face that will be superimposed into the video.
Therefore, in a deepfake video-clip where Brad Pitt is replaced in the film *Se7en* by Tom Cruise, Pitt is the 'source' and Cruise will be the 'target'.

Here, Judy Garland (playing in the 1944 film 'Meet Me In St. Louis'), is the 'source' identity, and singer Billie Eilish is the 'target' identity. Source: https://www.tiktok.com/@nextface/video/6936881472313806085

Facial Data Approaches

In order to train the deepfakes system, it's necessary for the user to begin by curating two image datasets – one each, for the *source* and *target* identity.

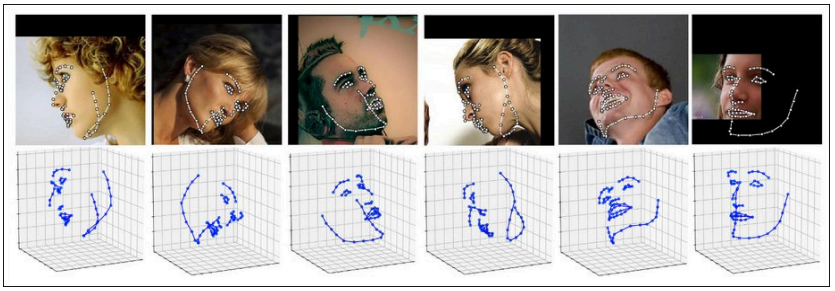
How this task is approached depends on what kind of model the deepfaker wants to create.
If they wish to create a model that can generically swap between the two identities (i.e., that will work quite well on *any* video clip), then they are likely to use a variety of sources for face images, such as frames extracted from movies and YouTube interviews, social media posts, and any usable material from Stock image companies such as Getty Images (though this is likely to be against the terms of use, even for preview images).
If, on the other hand, the intent is to train a model to perform a swap on a *specific* video clip, then only the photos for the *source* identity will be 'randomly drawn' in this way, while the *target* data images will be extracted entirely from the video clip.



Two ways that data can be curated. If, in this case, the intention is to replace Jack Nicholson in 'The Shining', it makes sense to train the model only on faces extracted from the target clip for that movie (images on right).

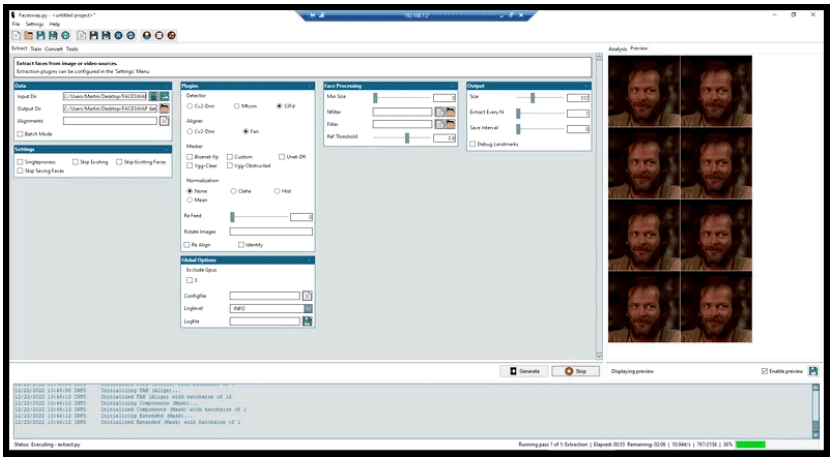
Obtaining and Sorting Face Images

Both DFL and FaceSwap use the *Facial Alignment Network* (FAN) system for extracting face images, either from video clips or static images.



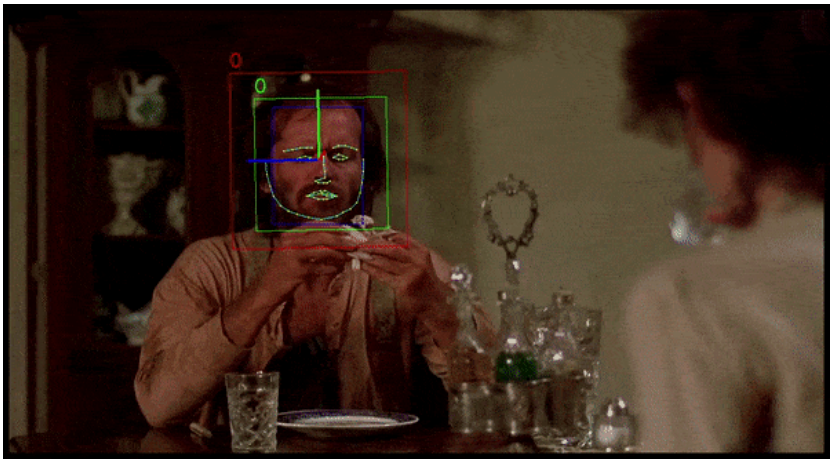
The Facial Alignment Network (FAN) assigns landmarks based on the estimated pose derived from the source image. Source: https://github.com/1adrianb/face-alignment

FAN is trained to recognize basic faces, and scans each individual extracted image (or video-frame, if extracting from a video), assigning 69 landmarks to each face that it's able to individuate.



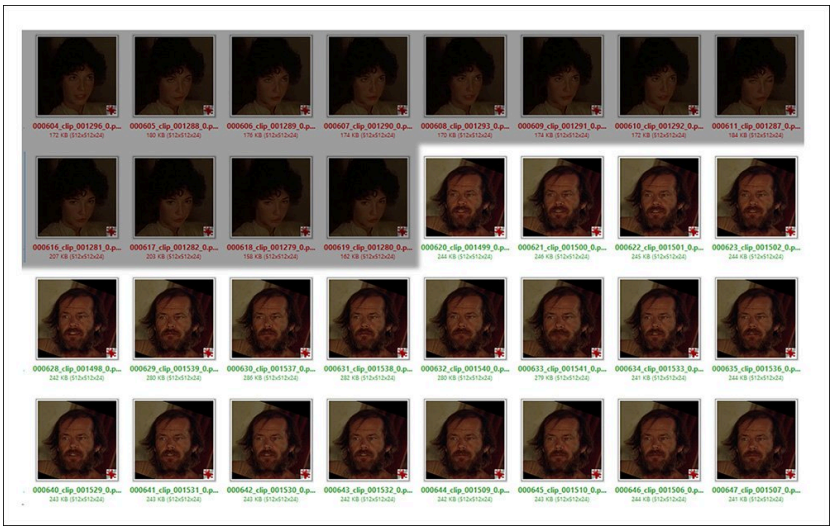
Here, FaceSwap is extracting ad hoc faces from a 1-minute video clip (DFL has almost identical functionality). The resulting extracted faces will contain all and any faces that were found, meaning that 'unwanted' recognitions will have to be removed later. The ability to recognize desired faces and ignore others has never been implemented well either in DFL or FaceSwap.

If necessary, FAN will assign multiple faces within a single frame. The *Single Shot Scale-invariant Face Detector* (S3FD) system can be chosen as the face recognition algorithm, allowing for a more flexible '3D' interpretation of facial poses.
As we can see in the image below, FAN has some difficulty in recognizing that faces may be obscured (i.e., by hands, glasses, or other obstructions), and tends to create 'scrunched up' faces. In such cases, the alignments (the vector data that defines the faces) will either have to be edited and masked properly later, or the faulty extractions discarded.



FAN is not a perfect system.

In order to filter out unwanted face extractions (i.e., not the person you wanted to extract), the two systems allow for sorting by identity, so that you can select the surplus faces contiguously and delete them.

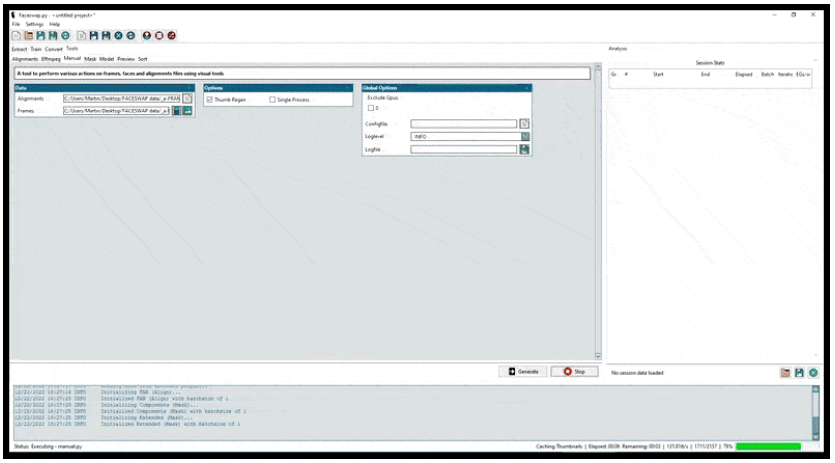


After sorting the extracted images by face, it's possible to select and delete all the unwanted face extractions.

The added benefit of sorting by face is that any completely mistaken extractions (such as the algorithm inadvertently finding faces in lace patterns, and other type of paredoilia) is that these wrong guesses are now also grouped together, and easy to delete.

The FAN algorithm can be set to aggressively seek out faces. Though this ensures that few or no real faces are missed, there are likely to be some comedic 'stowaways' among the initial extractions.

At this point, it may be necessary to adjust the alignments. Facial landmark detection systems tend to struggle with occluded angles or extreme poses, and all the popular deepfakes forks allow the user to fine-tune the alignment poses and re-do the extraction process, so that training will be more accurate to the identity.



Poor alignments can be fixed, and masks altered, in all the popular deepfakes distributions.

Automated Masking

The masking process differs somewhat between the various distributions of deepfakes packages. FaceSwap features a manual masker, which allows the user to erase and draw over the AI-estimated mask (at the same time as adjusting many other facets of the alignments, all in the 'Manual' tool).

The FaceSwap mask editor in action.

Conversely, DeepFaceLab offers a dedicated program called *XSeg*, where the user draws periodical ‘key-frame’ masks (or mask adjustments). These are then *trained*, so that the non-adjusted masks in-between those ‘manual’ frames (which represent the exemplary training data) are conformed to the user-edits, obviating the need to manually adjust each problematic mask frame.

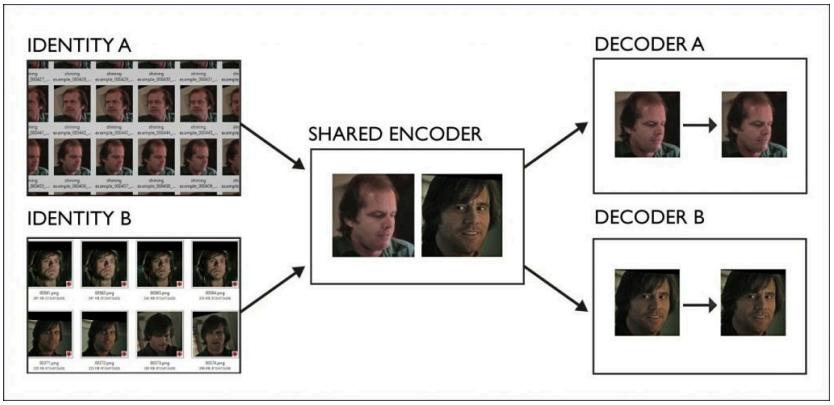
<YOUTUBE LINK OMITTED>

Though masking is useful as an aide to extraction, so that non-facial data does not get mistaken for facial data at training time, this kind of effort is primarily aimed at the conversion (or, in DFL terms, *merge*) process – the point at which the trained model actually performs the conversion – which we’ll come to later.

Training

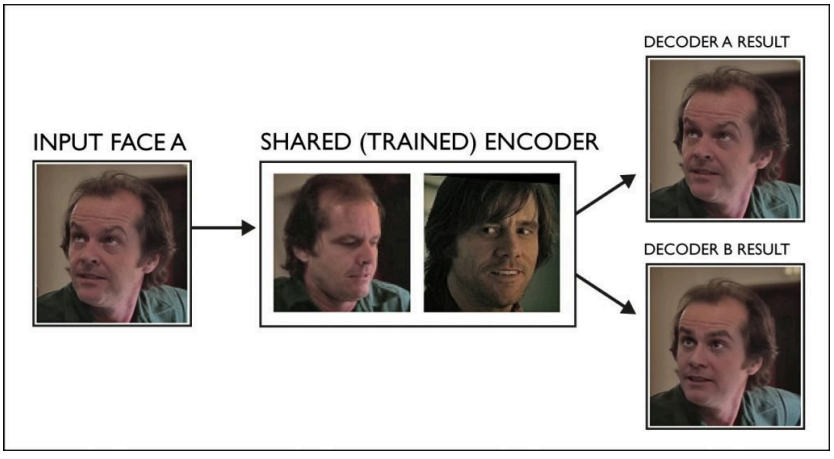
Autoencoder Principles

With the two face-sets curated, the next step is to train them in the autoencoder architecture.



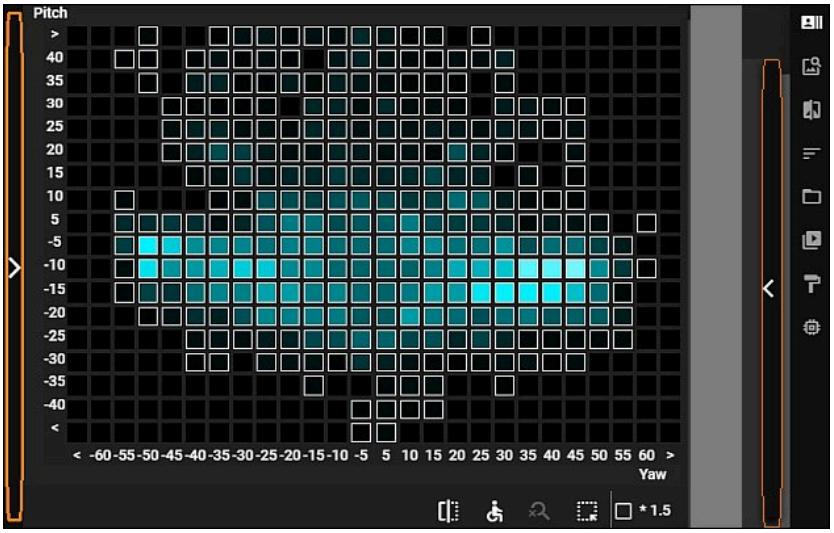
No swap yet – this figurative schema represents the way that autoencoders usually work. In this case, the trained encoder has learned to reproduce Jack Nicholson and Jim Carrey. Note that their learned embeddings are stored in a common or 'shared' encoder (center of image).

An autoencoder neural network learns to reproduce common features in the unlabeled data on which it is trained. In our case, that data is the two sets of extracted faces. In the schema above, we see that the network has learned to reproduce the two identities accurately, and also that the embeddings extracted from the data during training were stored in a shared encoder, instead of in separate ‘silos’.
By using a shared encoder, it’s possible to then simply switch the routing of the recreation process, so that identity A is transformed into identity B.



By changing the routing from A>A or B>B to A>B or B>A, we can now swap the identities trained into the encoder.

It's at this stage that we begin to understand why the two datasets need some commonality between them, in terms of lighting, pose and expression. If the dataset for identity A features a particular expression, such as smiling, but identity B has no pictures of the subject smiling, the 'smiling' data has no mapping route, since the extracted features have no corresponding feature in the other set.
Likewise for poses – if one dataset features many profile images, and the other dataset has none, again, the profile data is effectively thrown away, and the final model will not be able to accurately reproduce side-views across the two identities, defaulting to what it knows about the A>A profile characteristics, and causing 'identity bleed'.
The Machine Video Editor (MVE) system, mentioned at the start, has a handy tool called Face Graph, which allows the user to see at a glance the dispersal and coverage of poses in a face dataset, and to quickly identify areas where one set may be lacking poses:



MVE's Face Graph tool can demonstrate shortfalls in pose coverage. Source: https://discord.com/channels/730623345288151060/1001080591879524372/1001217467244351540

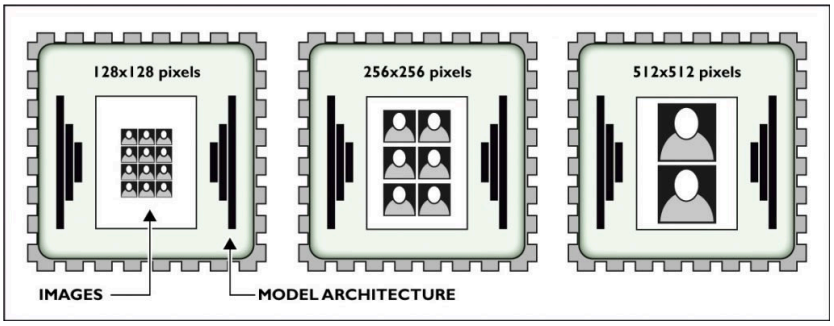
Unfortunately no such functionality exists to classify emotional range in a dataset, which could potentially be evaluated by the Facial Action Encoding System (FACS) or a similar emotion/affect recognition framework, and here the user needs to simply be aware of the equivalence (or lack) in each set.

Training Times

Training times for popular deepfake distributions have two bottlenecks: the potency of the available GPU, and the architectural limitations of the process itself. In general, training a dataset from scratch is still considered to be a lengthy process, with usable results usually requiring 1-7 days of training. If the target dataset is dedicated (i.e., the dataset uses only material from the target clip), it's possible to obtain some improvement in speed vs. quality, though the resulting model may not be flexible enough to perform similarly good swaps on other clips than the one it was trained for.

Batch Sizes

No GPU is capable of loading large datasets in a single shot, and the systematic comparison and evaluation of the images is therefore accomplished in batches. Since at least some of the architecture of the system also occupies a percentage of the available VRAM, the user is faced with making one of several possible trade-offs, in this respect.



Varying batch sizes, based on chosen dimensions of image data.

One compromise that can be made is to set a lower default image dimension, such as 64x64px, or 128x128px. Thus a larger batch can be imposed, which can help the system to compare and evaluate more of the material at any one time. This is particularly useful in the early stages of training, when the deepfakes framework is still 'sketching out' the vague likenesses of the two identities being trained.
On the negative side, high batch sizes are far less useful in the later stages of training, by which time it is too late to increase the dimensions of the resized training images, which are fixed, once training has begun.
It's even quite common for users to reduce batch sizes deliberately towards the latter part of training, so that the model, which now broadly understands the identities, can begin to acquire greater detail.
Another compromise is to set a higher default dimension, and accept a lower batch size. Smaller batches, as mentioned, may impede the model from developing accurate initial likenesses, and by the time training is advanced enough that detail (rather than general similarity) is desired, the model's ability to create a general resemblance may be quite poor.

Data Dimensions

The deepfakes system will automatically resize the training data down to a size that will fit into the available resources, and will fail to begin training if the settings, in this respect, are expecting better resources than are at hand.
Though some incremental optimizations and improvements, together with developments in GPU and CUDA implementations (among many other of the standard libraries which affect image synthesis and computer vision) have, since 2017, allowed for rational increases in the size of images passing through consumer-level hardware, the images passing into the latent space of the architecture are still very small, in a typical setup.

Improvements in usable training image sizes since 2017, according to the DeepFaceLab project. In effect, the larger sizes visualized are still not within reach of the casual user, or else require lower batch sizes (which can adversely affect the result) or extended training times (sometimes extending to months). Source: https://github.com/iperov/DeepFaceLab

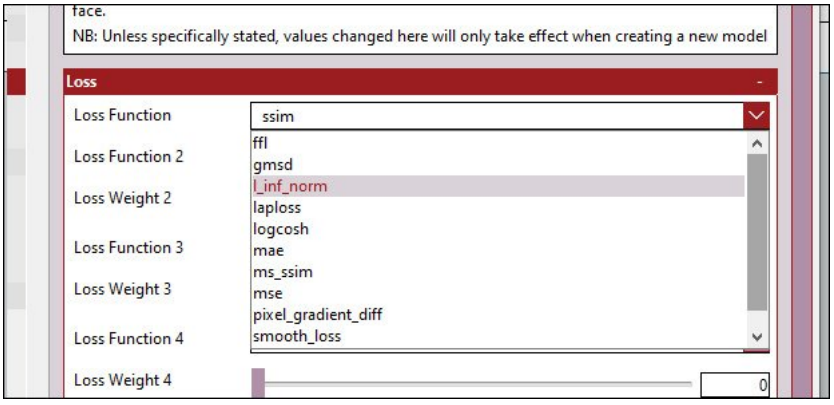
Size is not everything – the deepfakes process extracts features from any size of image passing through the training space and routinely upscales it, iteratively. However, it is likely to be able to discern a wider range of facial traits and features proportional to the size of the image data – and, in the later stages, to develop improved granular detail if the image sizes are larger, and the choices around batch size were apposite, and represented the best compromise for the available hardware.

Diversifying the Data During Training

In addition to resizing the images, the data will also be randomly flipped and rotated (the user can specify how much, or leave this to defaults) while the data is running through the latent space, and the two datasets are being evaluated and compared.
This helps the system to avoid memorizing the rote poses of the images (which would make for an inflexible or even unusable model), by making the existing data artificially more diverse, in the hope of a flexible and practicable generalization of the datasets into a performant model.

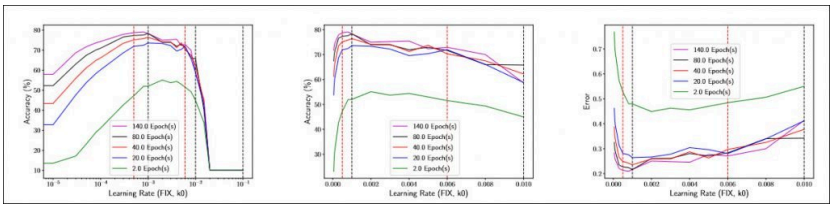
Loss Functions, Learning Rates, and Convergence

In popular usage, convergence is considered to be the point at which the model is sufficiently trained to provide a result that's acceptable to the user. Strictly speaking, though, it's actually the point where the loss values for the model have stabilized, and are unlikely to improve further with additional training
Loss functions and learning rates are essential to a successful convergence, and each may be switched out or altered throughout training.
A loss function is a prediction of how well the model is recreating the training data. There are many ways to calculate this kind of prediction, and many views, in the deepfake community, about which loss functions may be the most performant or adapted to a task.



Some of the available loss functions in the FaceSwap deepfakes distribution.

Popular choices, at least for the start of the deepfake training process, are *Mean Absolute Error* (MAE) and *Mean Squared Error* (MSE).
More recent and advanced loss functions such as LPIPS and NVIDIA's FLIP were developed from studies of human perception, and are of particular interest to the FaceSwap community. These two loss functions can be quite difficult to control, in terms of producing unwanted background patterns. While they're not suited to the early stages of training, they can produce superior detail once the model is well-established.
Meanwhile, the learning rate (LR) controls the speed at which the system iterates through the data during training. Typically the LR is adjusted downward periodically throughout training, in much the same way that a sculptor's initial broad strokes against the stone devolve into careful and crafted attention. This can also be implemented systematically as a learning rate schedule.



Accuracy and error rates across three diverse learning schedules. Source: https://arxiv.org/pdf/1908.06477.pdf

Pretraining

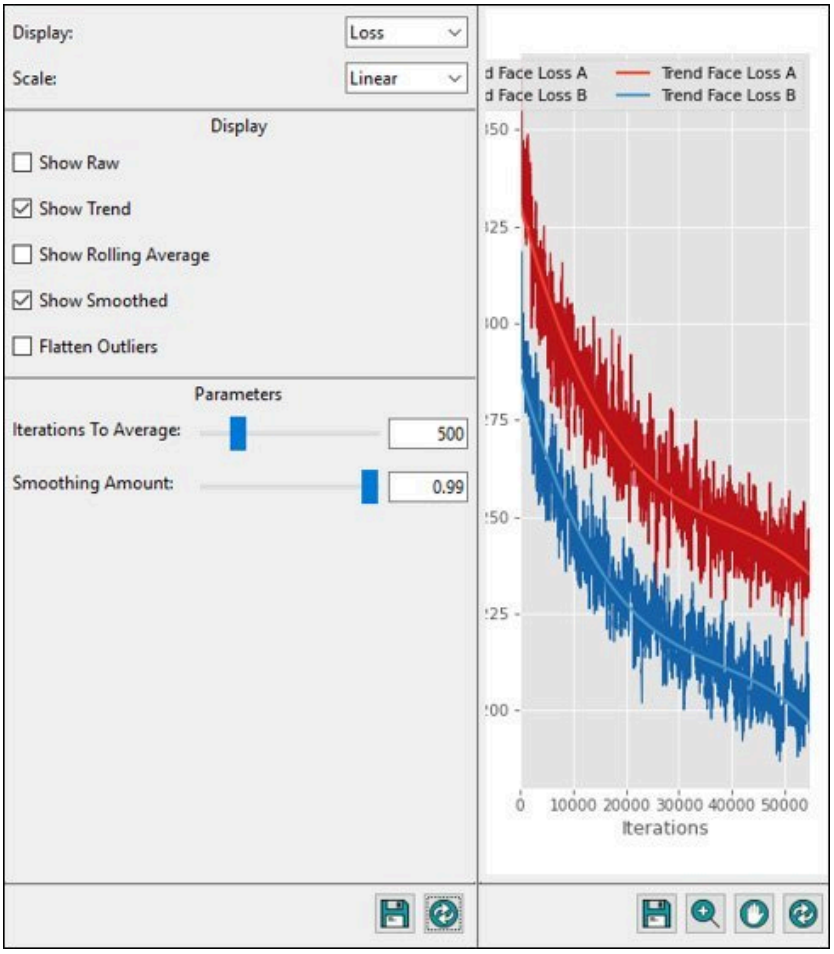
One of the most controversial topics in the deepfake developer communities is the value of pretraining in deepfake workflows. Since training from scratch is so arduous and time-consuming, it's a common practice (especially with the DFL community, since this feature is innate to the culture of DeepFaceLab) to start a new model with a partially-trained model, where core facial features have already been developed.

<YOUTUBE LINK OMITTED>

Though the FaceSwap cadre tends to disagree with this faith in pretrained models, there is no definitive answer as to whether the results obtained by 'hopping identities' in this way are any worse (or perceptually worse) than would have been obtained by training from zero.
Certainly a pretrained model that has been exposed to a variety of faces (rather than a single identity) may be adequately generalized that it could be developed effectively into a specific pair of identities; on the other hand, certain foundational features may by this point have become so entrenched that they are now challenging to overwrite with new data – similar to hoping to change the foundations of a building when you're already doing the dry-wall on the first floor.

Enacting the Facial Transfer

Finally, once this huge array of choices is successfully navigated, and the lengthy training completed, the deepfakes model should reach successful convergence, and be ready to perform its function.



A successful training session, representing nearly a week of training, in FaceSwap.

The swap is accomplished on a per-frame basis in a process called *conversion* in FaceSwap, and *Merging*, in DeepFaceLab and DeepFaceLive. Prior to this process, the clip will usually go through the same extraction process used to obtain the training data, as described earlier. This time, however, a higher level of curation and attention to masking may be necessary, as the human viewer will be less forgiving of small glitches than the generalization capacity of the autoencoder.
Many advanced deepfakers do not overwrite the faces directly to complete video frames, but choose to export the swaps as image sequences with an alpha (or transparency) channel.

An 'isolated' faceswap, suitable for import into a post-processing composition.

This allows the user to import the swaps into post-processing software such as After Effects, as a floating layer that can be adjusted to better match the target source material. After this, the composition is then rendered out, and the soundtrack (if any) is then reattached if necessary.