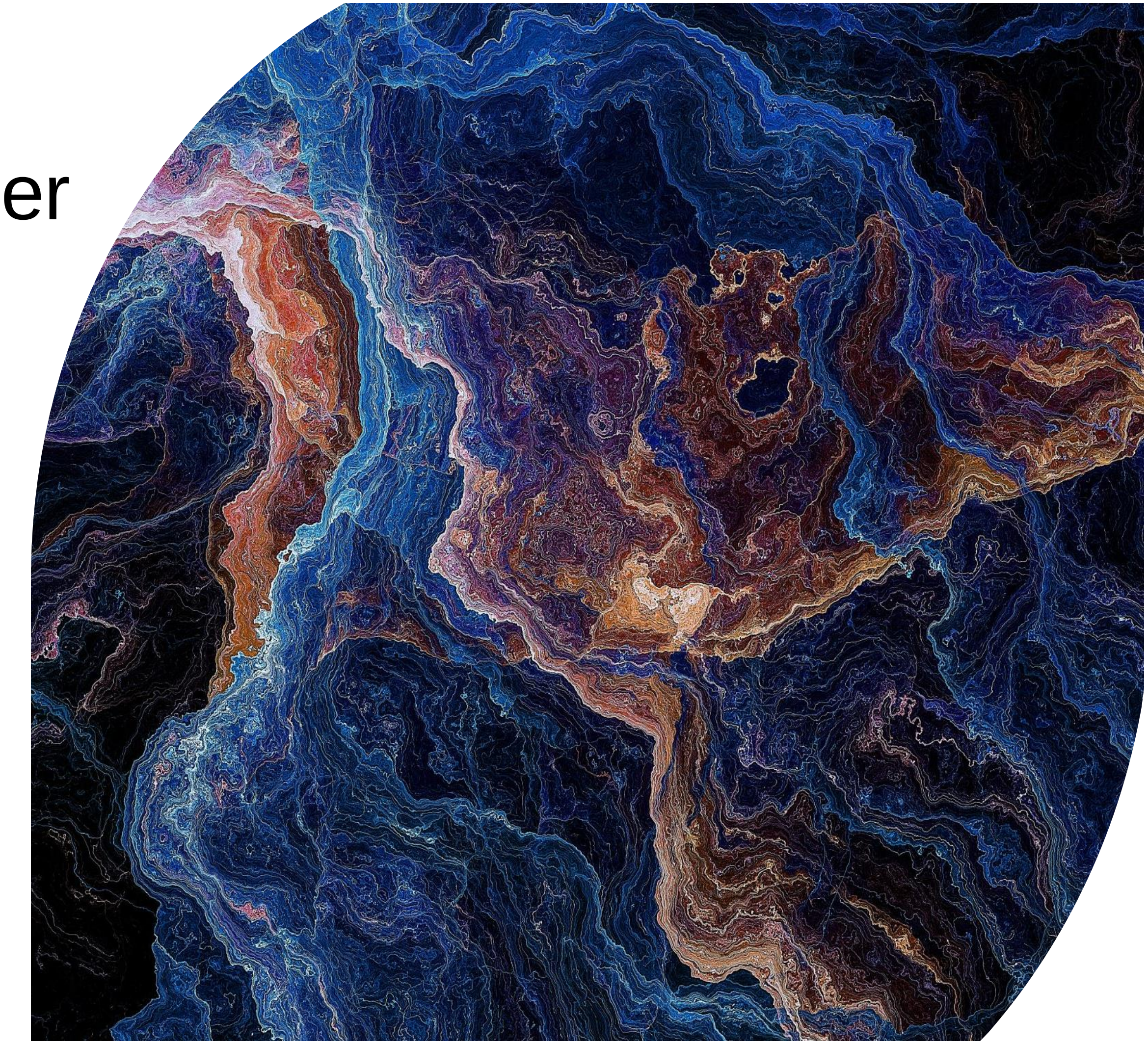
Kaggle DFDC Winner selimsef Code Walkthrough



Catherine Bernaciak Ph.D.

Aug 30, 2022

SEI-CMU Deepfakes Day

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM22-0758

Introduction

These slides constitute a walkthrough of setting up and running selimsef's winning model for the DFDC.

I follow the code as it appears in it's github repo as of July 2022:

- https://github.com/selimsef/dfdc_deepfake_challenge
- Code was downloaded to the /selimsef branch of a local SEI-CMU repo, pulled to a specific GPU machine (Cage) and run there.

Each step involved in running the model are reviewed:

1. Codebase Acquisition
2. Building & Running Docker Image
3. Data Acquisition & Processing
4. EfficientNet Models & Training (in progress)
5. selimsef Public Results

1. Codebase Acquistion

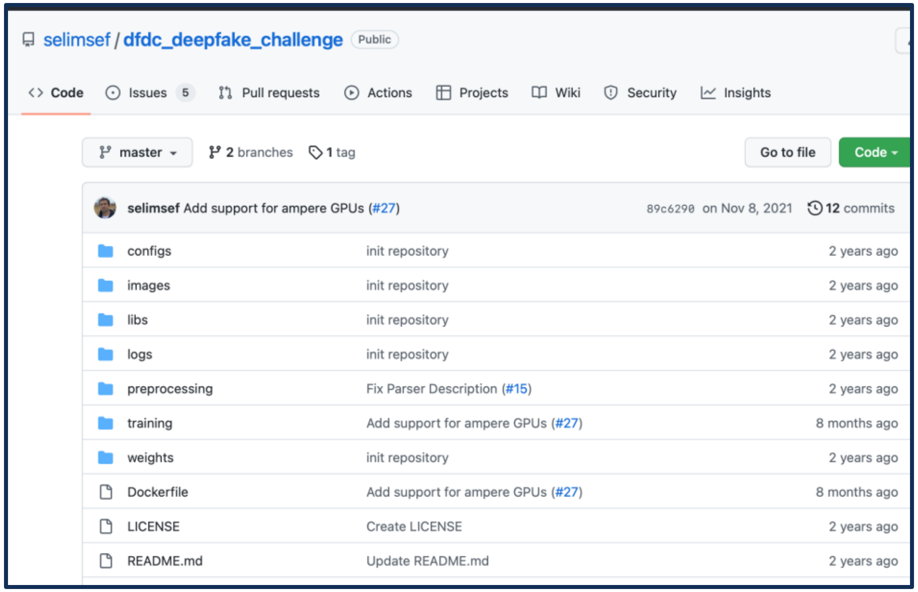
The original repo is selimsef/dfdc_deepfake_challenge:

- https://github.com/selimsef/dfdc_deepfake_challenge

It was transferred to our shared local deepfake bitbucket and placed on it's own branch (selimsef):

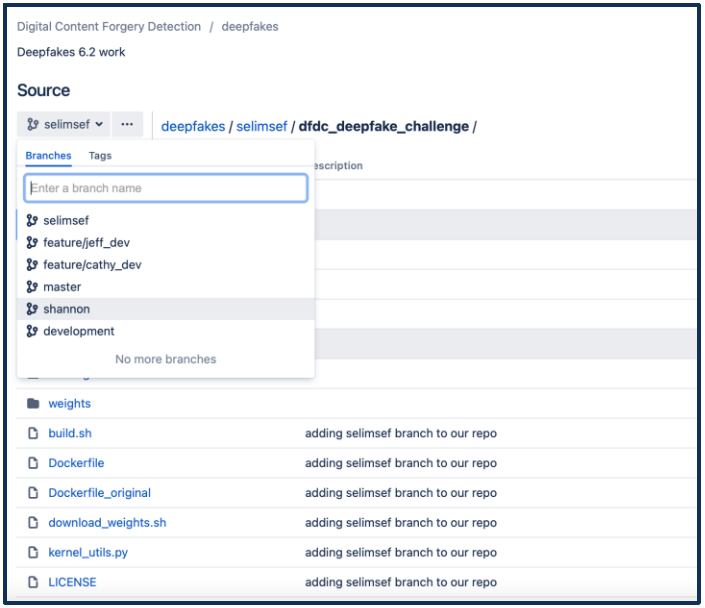
- https://code.sei.cmu.edu/bitbucket/projects/DCFD/repos/deepfakes/browse/selimsef/dfdc_deepfake_challenge?at=selimsef

It was pulled to a local machine (Cage) and run there.



selimsef Add support for ampere GPUs (#27) 89c6290 on Nov 8, 2021 12 commits

configs	init repository	2 years ago
images	init repository	2 years ago
libs	init repository	2 years ago
logs	init repository	2 years ago
preprocessing	Fix Parser Description (#15)	2 years ago
training	Add support for ampere GPUs (#27)	8 months ago
weights	init repository	2 years ago
Dockerfile	Add support for ampere GPUs (#27)	8 months ago
LICENSE	Create LICENSE	2 years ago
README.md	Update README.md	2 years ago

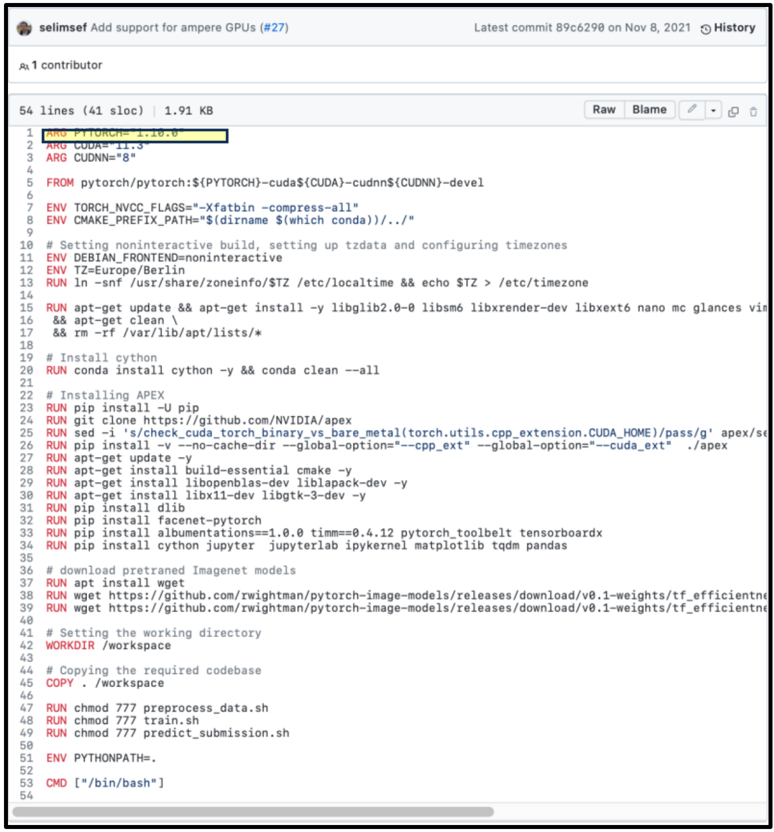


weights	
build.sh	adding selimsef branch to our repo
Dockerfile	adding selimsef branch to our repo
Dockerfile_original	adding selimsef branch to our repo
download_weights.sh	adding selimsef branch to our repo
kernel_utils.py	adding selimsef branch to our repo
LICENSE	adding selimsef branch to our repo

2. Building and Running Docker Image – updating Dockerfile

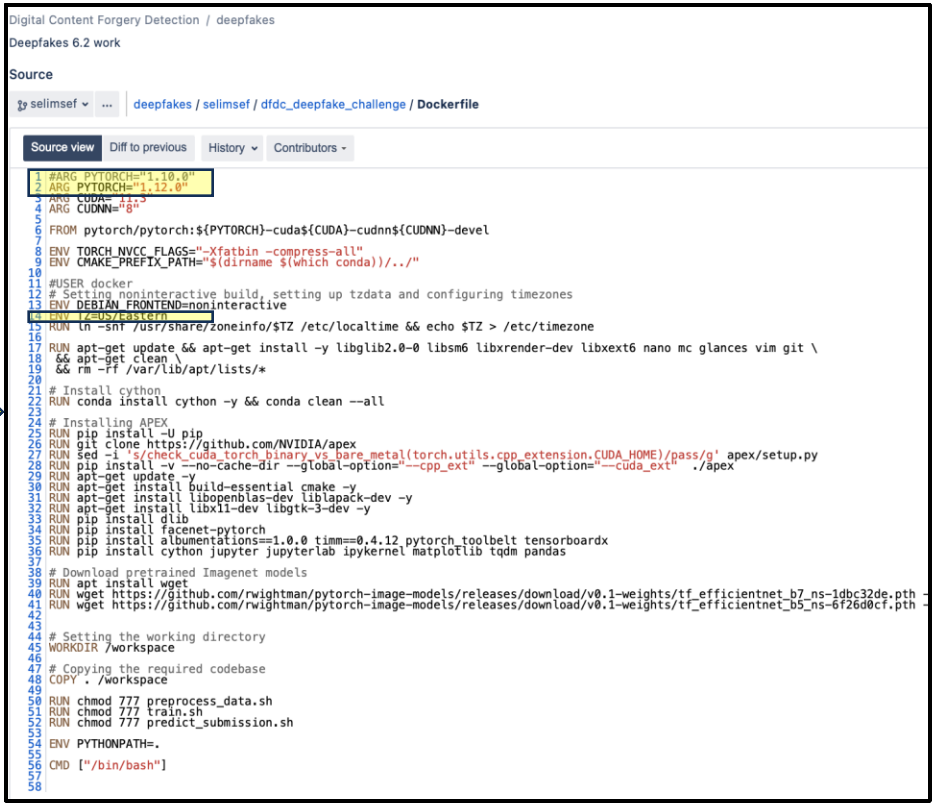
Two Changes:

1. Updated PyTorch Version from 1.10.0 to 1.12.0
2. Updated timezone



selimsef Add support for ampere GPUs (#27)
Latest commit 89c6290 on Nov 8, 2021
History
1 contributor
54 lines (41 sloc) | 1.91 KB
Raw
Blame

```
ARG PYTORCH="1.10.0"
ARG CUDA="11.3"
ARG CUDNN="8"

FROM pytorch/pytorch:${PYTORCH}-cuda${CUDA}-cudnn${CUDNN}-devel

ENV TORCH_NVCC_FLAGS="-Xfatbin -compress-all"
ENV CMAKE_PREFIX_PATH="$(dirname $(which conda))/../"

# Setting noninteractive build, setting up tzdata and configuring timezones
ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Berlin
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get update && apt-get install -y libglib2.0-0 libsm6 libxrender-dev libxext6 nano mc glances vim
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

# Install cython
RUN conda install cython -y && conda clean --all

# Installing APEX
RUN pip install -U pip
RUN git clone https://github.com/NVIDIA/apex
RUN sed -i 's/check_cuda_torch_binary_vs_bare_metal(torch.utils.cpp_extension.CUDA_HOME)/pass/g' apex/se
RUN pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext"  ./apex
RUN apt-get update -y
RUN apt-get install build-essential cmake -y
RUN apt-get install libopenblas-dev liblapack-dev -y
RUN apt-get install libx11-dev libgtk-3-dev -y
RUN pip install dlib
RUN pip install facenet-pytorch
RUN pip install albumentations==1.0.0 timm==0.4.12 pytorch_toolbelt tensorboardx
RUN pip install cython jupyter  jupyterlab ipykernel matplotlib tqdm pandas

# download pretraned Imagenet models
RUN apt install wget
RUN wget https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientne
RUN wget https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientne

# Setting the working directory
WORKDIR /workspace

# Copying the required codebase
COPY . /workspace

RUN chmod 777 preprocess_data.sh
RUN chmod 777 train.sh
RUN chmod 777 predict_submission.sh

ENV PYTHONPATH=.

CMD ["/bin/bash"]

```



Digital Content Forgery Detection / deepfakes
Deepfakes 6.2 work
Source
selimsef
...
deepfakes / selimsef / dfdc_deepfake_challenge / Dockerfile
Source view
Diff to previous
History
Contributors

```
#ARG PYTORCH="1.10.0"
ARG PYTORCH="1.12.0"
ARG CUDA="11.3"
ARG CUDNN="8"

FROM pytorch/pytorch:${PYTORCH}-cuda${CUDA}-cudnn${CUDNN}-devel

ENV TORCH_NVCC_FLAGS="-Xfatbin -compress-all"
ENV CMAKE_PREFIX_PATH="$(dirname $(which conda))/../"

#USER docker
# Setting noninteractive build, setting up tzdata and configuring timezones
ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=US/Eastern
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get update && apt-get install -y libglib2.0-0 libsm6 libxrender-dev libxext6 nano mc glances vim git \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

# Install cython
RUN conda install cython -y && conda clean --all

# Installing APEX
RUN pip install -U pip
RUN git clone https://github.com/NVIDIA/apex
RUN sed -i 's/check_cuda_torch_binary_vs_bare_metal(torch.utils.cpp_extension.CUDA_HOME)/pass/g' apex/setup.py
RUN pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext"  ./apex
RUN apt-get update -y
RUN apt-get install build-essential cmake -y
RUN apt-get install libopenblas-dev liblapack-dev -y
RUN apt-get install libx11-dev libgtk-3-dev -y
RUN pip install dlib
RUN pip install facenet-pytorch
RUN pip install albumentations==1.0.0 timm==0.4.12 pytorch_toolbelt tensorboardx
RUN pip install cython jupyter jupyterlab ipykernel matplotlib tqdm pandas

# Download pretrained Imagenet models
RUN apt install wget
RUN wget https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientnet_b7_ns-1dbc32de.pth -
RUN wget https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientnet_b5_ns-6f26d0cf.pth -


# Setting the working directory
WORKDIR /workspace

# Copying the required codebase
COPY . /workspace

RUN chmod 777 preprocess_data.sh
RUN chmod 777 train.sh
RUN chmod 777 predict_submission.sh

ENV PYTHONPATH=.

CMD ["/bin/bash"]


```

2. Building and Running Docker Image – building Dockerfile

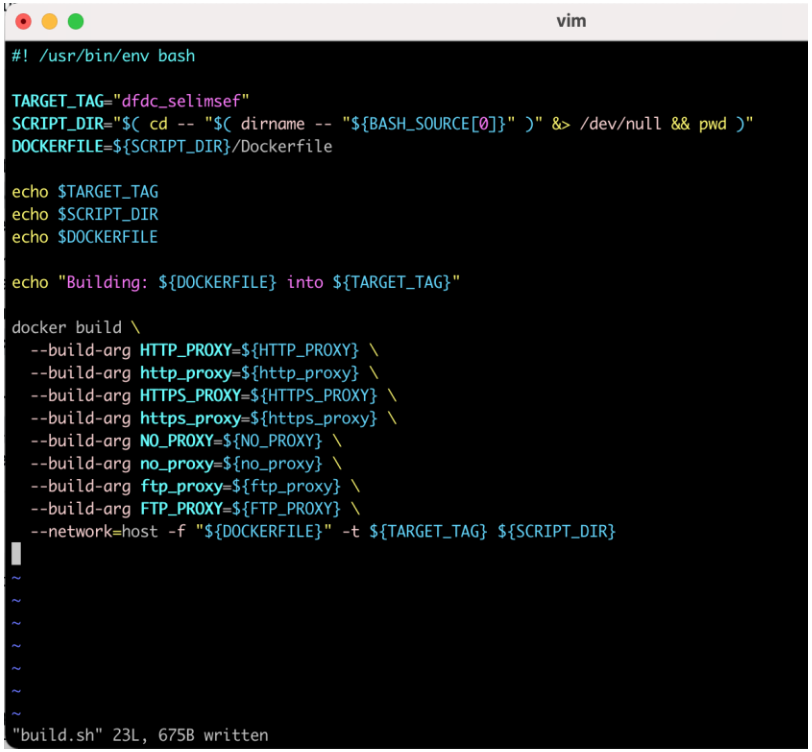
`build.sh` script was added to the `/dfdc_deepfake_challenge` directory

`build.sh` handles building of Dockerfile.

It has all the proxies necessary to run from any of our machines (Cage/Buscemi/Weaving).

As shown, it will build an image with the name ‘dfdc_selimsef’ as denoted by the TARGET_TAG variable.

Run at command line as `> ./build.sh`

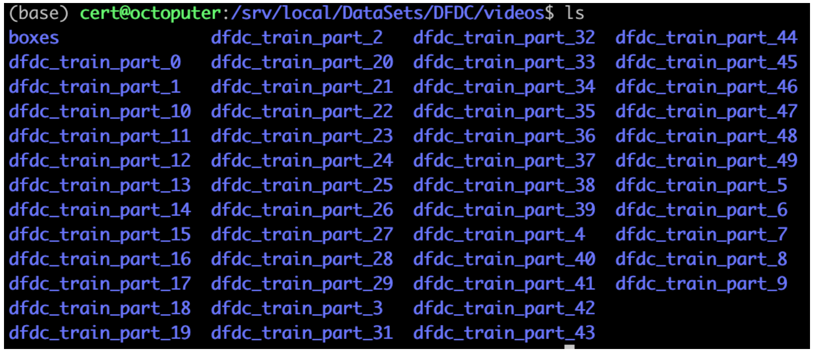


```
#! /usr/bin/env bash

TARGET_TAG="dfdc_selimsef"
SCRIPT_DIR="$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &> /dev/null && pwd )"
DOCKERFILE=${SCRIPT_DIR}/Dockerfile

echo $TARGET_TAG
echo $SCRIPT_DIR
echo $DOCKERFILE

echo "Building: ${DOCKERFILE} into ${TARGET_TAG}"

docker build \
  --build-arg HTTP_PROXY=${HTTP_PROXY} \
  --build-arg http_proxy=${http_proxy} \
  --build-arg HTTPS_PROXY=${HTTPS_PROXY} \
  --build-arg https_proxy=${https_proxy} \
  --build-arg NO_PROXY=${NO_PROXY} \
  --build-arg no_proxy=${no_proxy} \
  --build-arg ftp_proxy=${ftp_proxy} \
  --build-arg FTP_PROXY=${FTP_PROXY} \
  --network=host -f "${DOCKERFILE}" -t ${TARGET_TAG} ${SCRIPT_DIR}

~
~
~
~
~
~
~
"build.sh" 23L, 675B written
```

2. Building and Running Docker Image – running Dockerfile

Run the container with the run_cage.sh script

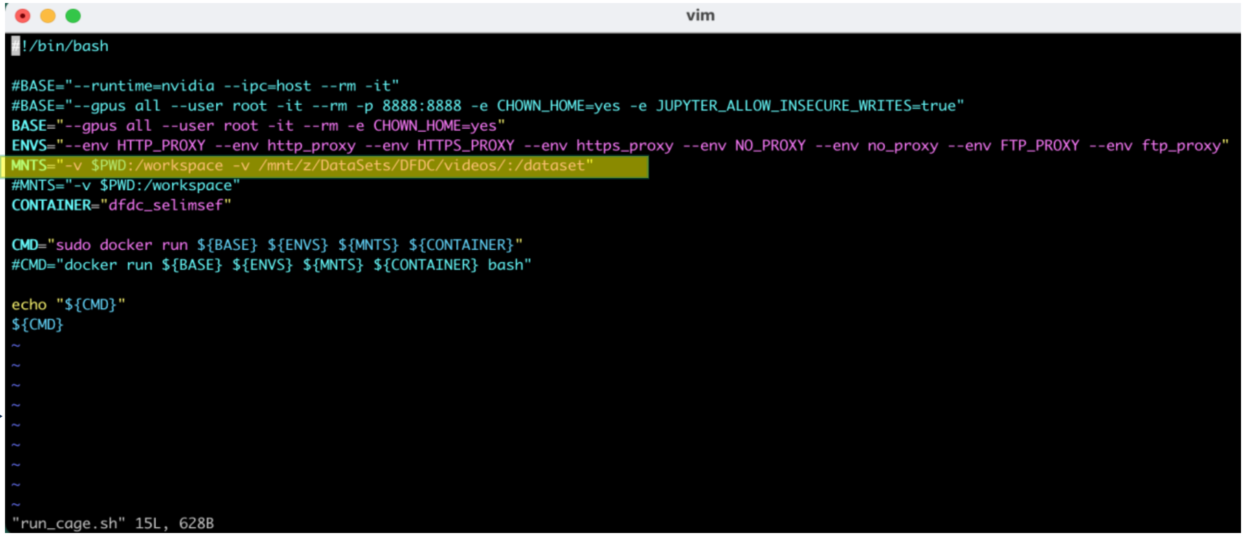
Note there are two mount points:



```
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos$ ls
boxes               dfdc_train_part_2   dfdc_train_part_32  dfdc_train_part_44
dfdc_train_part_0   dfdc_train_part_20  dfdc_train_part_33  dfdc_train_part_45
dfdc_train_part_1   dfdc_train_part_21  dfdc_train_part_34  dfdc_train_part_46
dfdc_train_part_10  dfdc_train_part_22  dfdc_train_part_35  dfdc_train_part_47
dfdc_train_part_11  dfdc_train_part_23  dfdc_train_part_36  dfdc_train_part_48
dfdc_train_part_12  dfdc_train_part_24  dfdc_train_part_37  dfdc_train_part_49
dfdc_train_part_13  dfdc_train_part_25  dfdc_train_part_38  dfdc_train_part_5
dfdc_train_part_14  dfdc_train_part_26  dfdc_train_part_39  dfdc_train_part_6
dfdc_train_part_15  dfdc_train_part_27  dfdc_train_part_4   dfdc_train_part_7
dfdc_train_part_16  dfdc_train_part_28  dfdc_train_part_40  dfdc_train_part_8
dfdc_train_part_17  dfdc_train_part_29  dfdc_train_part_41  dfdc_train_part_9
dfdc_train_part_18  dfdc_train_part_3   dfdc_train_part_42
dfdc_train_part_19  dfdc_train_part_31  dfdc_train_part_43
```

- `/mnt/z/DataSets/DFDC/videos` is a local mountpoint within WSL that is connected to the Octoputer via samba. Here are located the dfdc_train_part_XX videos.

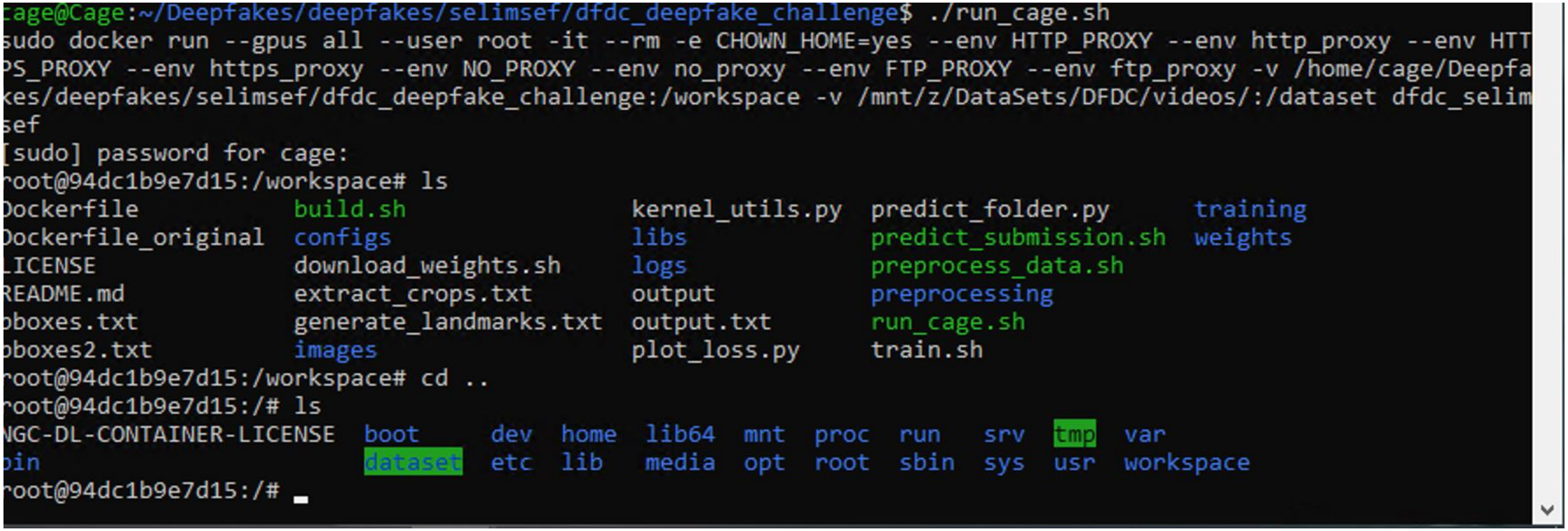
- both `/dataset` and `/workspace` are directories within the containers filesystem.



```
#!/bin/bash

#BASE="--runtime=nvidia --ipc=host --rm -it"
#BASE="--gpus all --user root -it --rm -p 8888:8888 -e CHOWN_HOME=yes -e JUPYTER_ALLOW_INSECURE_WRITES=true"
BASE="--gpus all --user root -it --rm -e CHOWN_HOME=yes"
ENVS="--env HTTP_PROXY --env http_proxy --env HTTPS_PROXY --env https_proxy --env NO_PROXY --env no_proxy --env FTP_PROXY --env ftp_proxy"
MNTS="-v $PWD:/workspace -v /mnt/z/DataSets/DFDC/videos/:/dataset"
#MNTS="-v $PWD:/workspace"
CONTAINER="dfdc_selimsef"

CMD="sudo docker run ${BASE} ${ENVS} ${MNTS} ${CONTAINER}"
#CMD="docker run ${BASE} ${ENVS} ${MNTS} ${CONTAINER} bash"

echo "${CMD}"
${CMD}
~
~
~
~
~
~
~
~
~
"run_cage.sh" 15L, 628B
```

2. Building and Running Docker Image – running Docker Image

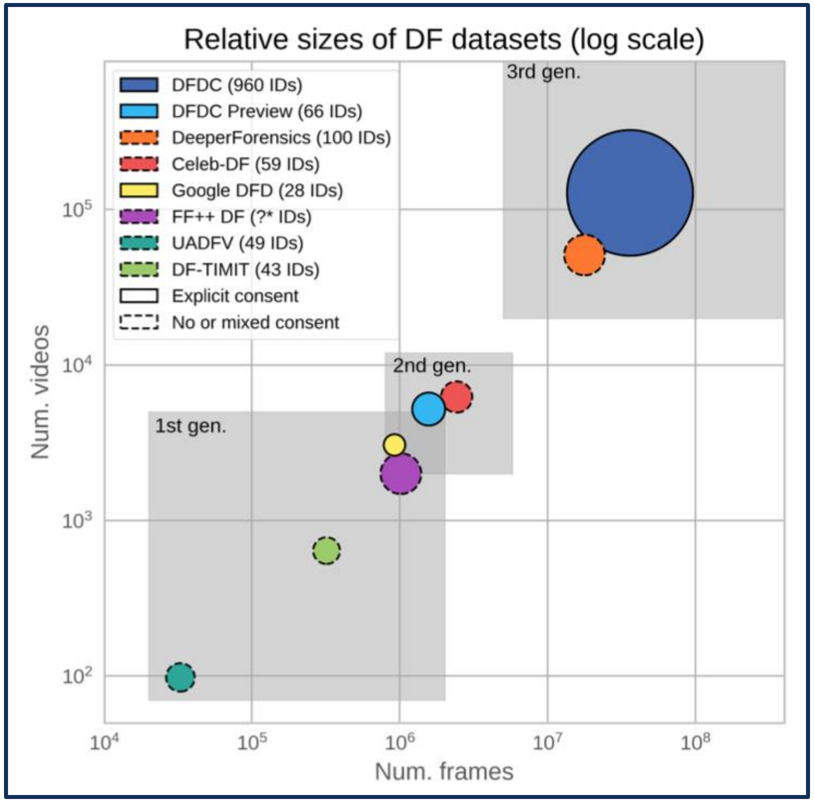
Here's what happens when it's run. Filesystem of the container is shown



```
cage@Cage:~/Deepfakes/deepfakes/selimsef/dfdc_deepfake_challenge$ ./run_cage.sh
sudo docker run --gpus all --user root -it --rm -e CHOWN_HOME=yes --env HTTP_PROXY --env http_proxy --env HTT
PS_PROXY --env https_proxy --env NO_PROXY --env no_proxy --env FTP_PROXY --env ftp_proxy -v /home/cage/Deepfa
kes/deepfakes/selimsef/dfdc_deepfake_challenge:/workspace -v /mnt/z/DataSets/DFDC/videos/:/dataset dfdc_selim
sef
[sudo] password for cage:
root@94dc1b9e7d15:/workspace# ls
Dockerfile           build.sh               kernel_utils.py  predict_folder.py      training
Dockerfile_original  configs                libs             predict_submission.sh  weights
LICENSE              download_weights.sh    logs             preprocess_data.sh
README.md            extract_crops.txt      output           preprocessing
bboxes.txt           generate_landmarks.txt output.txt       run_cage.sh
bboxes2.txt          images                 plot_loss.py     train.sh
root@94dc1b9e7d15:/workspace# cd ..
root@94dc1b9e7d15:/# ls
NGC-DL-CONTAINER-LICENSE  boot     dev  home  lib64  mnt  proc  run   srv  tmp  var
bin                       dataset  etc  lib   media  opt  root  sbin  sys  usr  workspace
root@94dc1b9e7d15:/# _
```

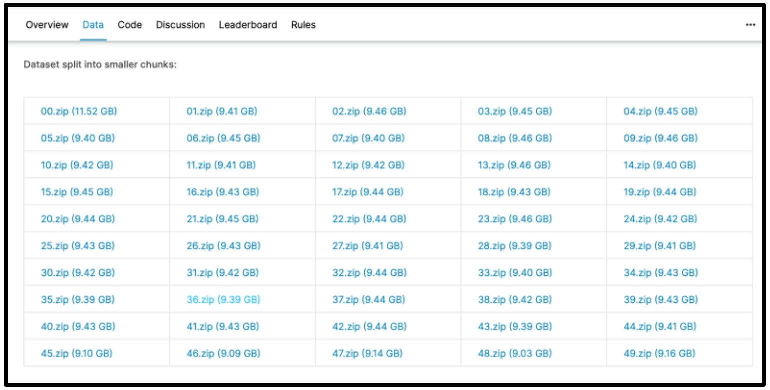
3. Data Acquisition

- Data is located here: https://www.kaggle.com/competitions/deepfake-detection-challenge/data
- There are 50 datasets totaling ~ 500 GB, ~ 130K real & fake videos
- Each datasets contains ~2000 videos, a metadata.json file which has labels
- Each video is 10s
- Technical details on the dataset: https://arxiv.org/pdf/2006.07397.pdf



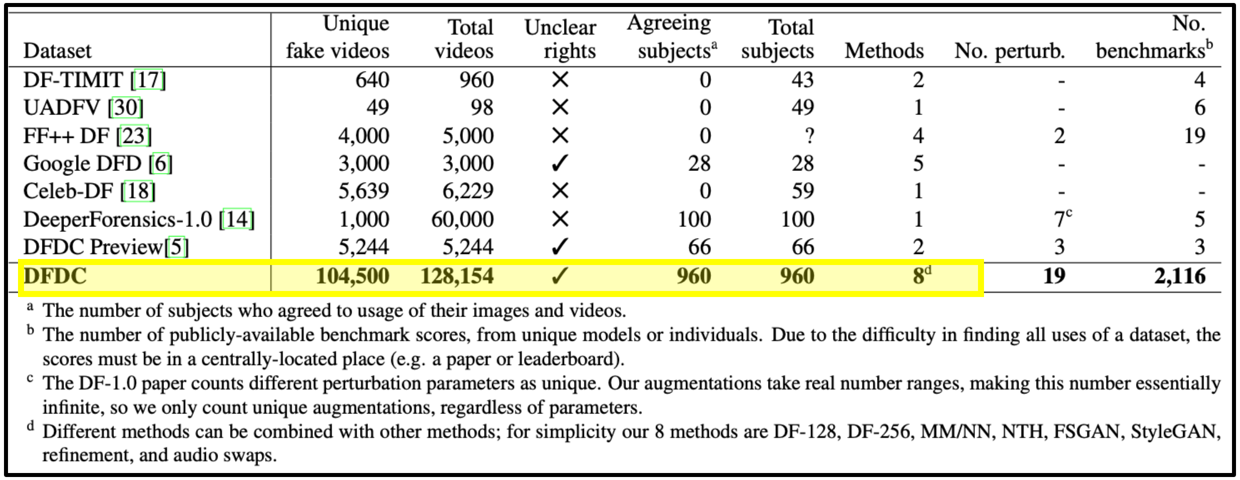
3. Data Acquisition

- Data is located here: https://www.kaggle.com/competitions/deepfake-detection-challenge/data
- There are 50 datasets totaling ~ 500 GB, ~ 130K real & fake videos
- Each datasets contains ~2000 videos, a metadata.json file which has labels
- Each video is 10s
- Technical details on the dataset:
  - https://arxiv.org/pdf/2006.07397.pdf



Dataset split into smaller chunks:

00.zip (11.52 GB)	01.zip (9.41 GB)	02.zip (9.46 GB)	03.zip (9.45 GB)	04.zip (9.45 GB)
05.zip (9.40 GB)	06.zip (9.45 GB)	07.zip (9.40 GB)	08.zip (9.46 GB)	09.zip (9.46 GB)
10.zip (9.42 GB)	11.zip (9.41 GB)	12.zip (9.42 GB)	13.zip (9.46 GB)	14.zip (9.40 GB)
15.zip (9.45 GB)	16.zip (9.43 GB)	17.zip (9.44 GB)	18.zip (9.43 GB)	19.zip (9.44 GB)
20.zip (9.44 GB)	21.zip (9.45 GB)	22.zip (9.44 GB)	23.zip (9.46 GB)	24.zip (9.42 GB)
25.zip (9.43 GB)	26.zip (9.43 GB)	27.zip (9.41 GB)	28.zip (9.39 GB)	29.zip (9.41 GB)
30.zip (9.42 GB)	31.zip (9.42 GB)	32.zip (9.44 GB)	33.zip (9.40 GB)	34.zip (9.43 GB)
35.zip (9.39 GB)	36.zip (9.39 GB)	37.zip (9.44 GB)	38.zip (9.42 GB)	39.zip (9.43 GB)
40.zip (9.43 GB)	41.zip (9.43 GB)	42.zip (9.44 GB)	43.zip (9.39 GB)	44.zip (9.41 GB)
45.zip (9.10 GB)	46.zip (9.09 GB)	47.zip (9.14 GB)	48.zip (9.03 GB)	49.zip (9.16 GB)



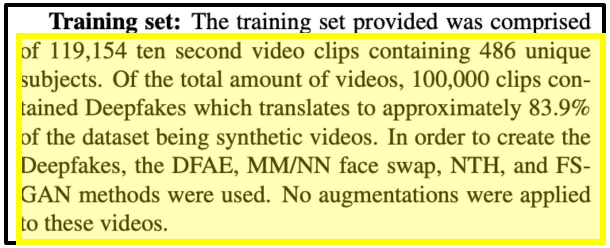
Dataset	Unique fake videos	Total videos	Unclear rights	Agreeing subjects[a]	Total subjects	Methods	No. perturb.	No. benchmarks[b]
DF-TIMIT [17]	640	960	✗	0	43	2	-	4
UADFV [30]	49	98	✗	0	49	1	-	6
FF++ DF [23]	4,000	5,000	✗	0	?	4	2	19
Google DFD [6]	3,000	3,000	✓	28	28	5	-	-
Celeb-DF [18]	5,639	6,229	✗	0	59	1	-	-
DeeperForensics-1.0 [14]	1,000	60,000	✗	100	100	1	7[c]	5
DFDC Preview[5]	5,244	5,244	✓	66	66	2	3	3
DFDC	**104,500**	**128,154**	✓	**960**	**960**	**8[d]**	**19**	**2,116**

[a] The number of subjects who agreed to usage of their images and videos.

[b] The number of publicly-available benchmark scores, from unique models or individuals. Due to the difficulty in finding all uses of a dataset, the scores must be in a centrally-located place (e.g. a paper or leaderboard).

[c] The DF-1.0 paper counts different perturbation parameters as unique. Our augmentations take real number ranges, making this number essentially infinite, so we only count unique augmentations, regardless of parameters.

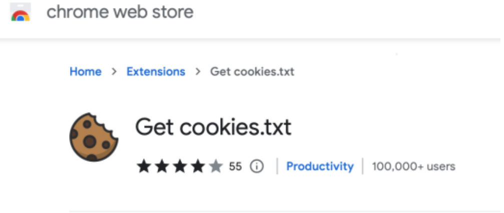
[d] Different methods can be combined with other methods; for simplicity our 8 methods are DF-128, DF-256, MM/NN, NTH, FSGAN, StyleGAN, refinement, and audio swaps.

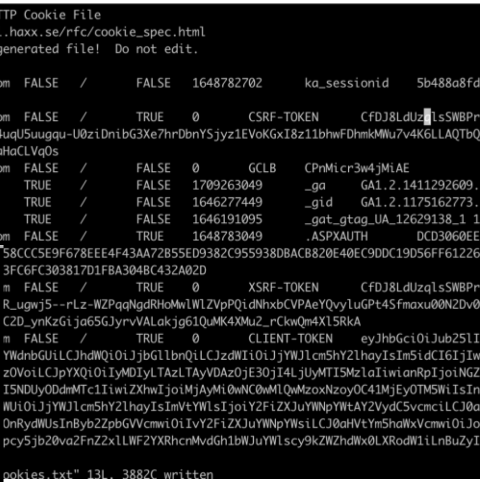


Training set: The training set provided was comprised of 119,154 ten second video clips containing 486 unique subjects. Of the total amount of videos, 100,000 clips contained Deepfakes which translates to approximately 83.9% of the dataset being synthetic videos. In order to create the Deepfakes, the DFAE, MM/NN face swap, NTH, and FS-GAN methods were used. No augmentations were applied to these videos.

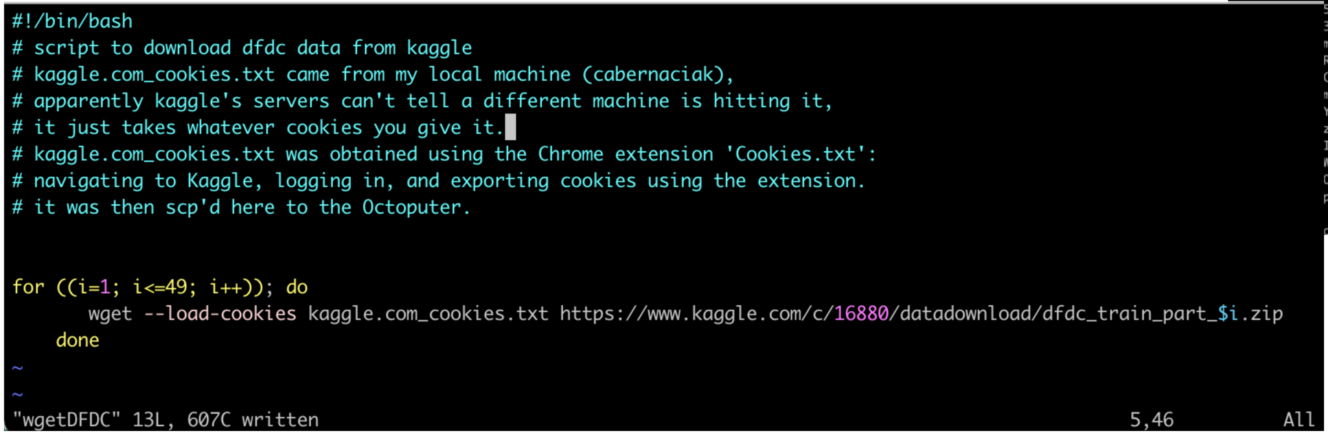
3. Data Acquisition

- Kaggle doesn’t make it easy to grab all at once, but there is still a way…
  1. Make a Kaggle account & login (needed to download any amount)
  2. Grab your cookies once logged in (cookies.txt worked)
  3. Use ‘wget’ and –load-cookies, feed in cookies file, grab link to zip file by inspecting html on page


chrome web store
Home > Extensions > Get cookies.txt
Get cookies.txt
★★★★★ 55 ⓘ | Productivity | 100,000+ users



```
# Netscape HTTP Cookie File
# http://curl.haxx.se/rfc/cookie_spec.html
# This is a generated file!  Do not edit.

www.kaggle.com  FALSE   /       FALSE   1648782702      ka_sessionid    5b488a8fd
28832
www.kaggle.com  FALSE   /       TRUE    0       CSRF-TOKEN      CfDJ8LdUzqlsSWBPr
SLYRfXbNxLUQ4uqU5uugqu-U0ziDnibG3Xe7hrDbnYSjyz1EVoKGxI8z11bhwFDhmkMWu7v4K6LLAQTbQ
dXGcGRKduSFvaHaCLVqOs
www.kaggle.com  FALSE   /       FALSE   0       GCLB    CPnMicr3w4jMiAE
.kaggle.com     TRUE    /       FALSE   1709263049      _ga     GA1.2.1411292609.
.kaggle.com     TRUE    /       FALSE   1646277449      _gid    GA1.2.1175162773.
.kaggle.com     TRUE    /       FALSE   1646191095      _gat_gtag_UA_12629138_1 1
www.kaggle.com  FALSE   /       TRUE    1648783049      .ASPXAUTH       DCD3060EE
58CCC5E9F678EEE4F43AA72B55ED9382C955938DBACB820E40EC9DDC19D56FF61226
3FC6FC303817D1FBA304BC432A02D
m  FALSE   /       TRUE    0       XSRF-TOKEN      CfDJ8LdUzqlsSWBPr
R_ugwj5--rLz-WZPqqNgdRHoMwlWlZVpPQidNhxbCVPAeYQvyluGPt4Sfmaxu00N2Dv0
C2D_ynKzGija65GJyrvVALakjg61QuMK4XMu2_rCkwQm4Xl5RkA
m  FALSE   /       TRUE    0       CLIENT-TOKEN    eyJhbGciOiJub25lI
YWdnbGUiLCJhdWQiOiJjbGllbnQiLCJzdWIiOiJjYWJlcm5hY2lhayIsIm5idCI6IjIw
zOVoiLCJpYXQiOiIyMDIyLTAzLTAyVDAzOjE3OjI4LjUyMTI5MzlaIiwianRpIjoiNGZ
I5NDUyODdmMTc1IiwiZXhwIjoiMjAyMi0wNC0wMlQwMzoxNzoyOC41MjEyOTM5WiIsIn
WUiOiJjYWJlcm5hY2lhayIsImVtYWlsIjoiY2FiZXJuYWNpYWtAY2VydC5vcmciLCJ0a
OnRydWUsInByb2ZpbGVVcmwiOiIvY2FiZXJuYWNpYWsiLCJ0aHVtYm5haWxVcmwiOiJo
pcy5jb20va2FnZ2xlLWF2YXRhcnMvdGh1bWJuYWlscy9kZWZhdWx0LXRodW1iLnBuZyI

ookies.txt" 13L, 3882C written
```

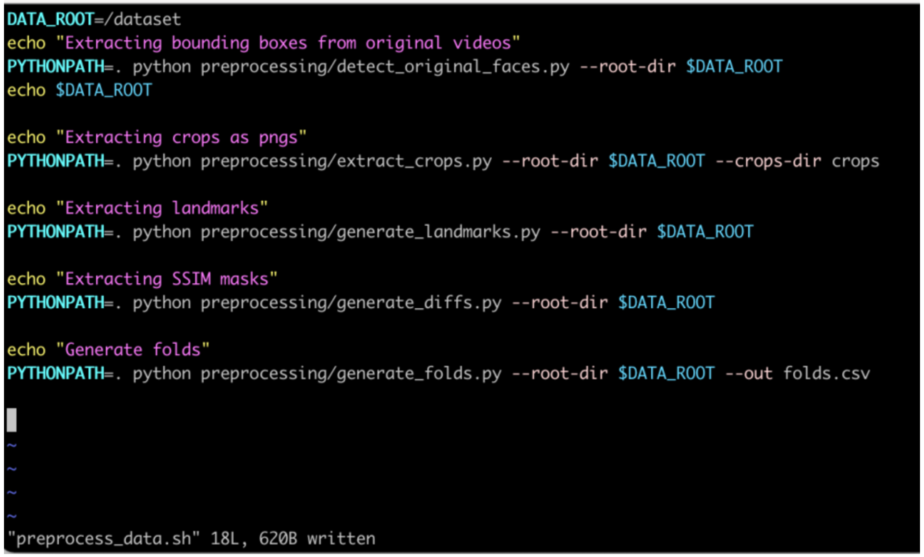


```
#!/bin/bash
# script to download dfdc data from kaggle
# kaggle.com_cookies.txt came from my local machine (cabernaciak),
# apparently kaggle's servers can't tell a different machine is hitting it,
# it just takes whatever cookies you give it.
# kaggle.com_cookies.txt was obtained using the Chrome extension 'Cookies.txt':
# navigating to Kaggle, logging in, and exporting cookies using the extension.
# it was then scp'd here to the Octoputer.


for ((i=1; i<=49; i++)); do
       wget --load-cookies kaggle.com_cookies.txt https://www.kaggle.com/c/16880/datadownload/dfdc_train_part_$i.zip
    done
~
~
"wgetDFDC" 13L, 607C written                                                                  5,46          All
```

3. Data Preparation

There are 5 steps involved with preparing the videos for training:

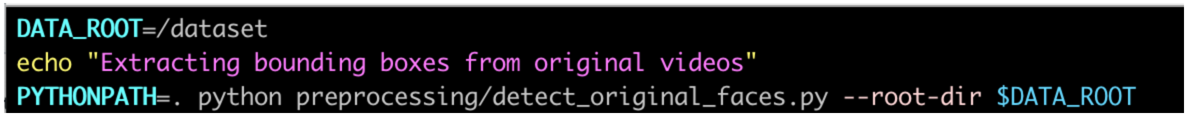
1. Extracting bounding boxes from original videos
2. Crop faces from frames
3. Extracting landmarks
4. Extracting SSIM masks
5. Generate folds



```
DATA_ROOT=/dataset
echo "Extracting bounding boxes from original videos"
PYTHONPATH=. python preprocessing/detect_original_faces.py --root-dir $DATA_ROOT
echo $DATA_ROOT

echo "Extracting crops as pngs"
PYTHONPATH=. python preprocessing/extract_crops.py --root-dir $DATA_ROOT --crops-dir crops

echo "Extracting landmarks"
PYTHONPATH=. python preprocessing/generate_landmarks.py --root-dir $DATA_ROOT

echo "Extracting SSIM masks"
PYTHONPATH=. python preprocessing/generate_diffs.py --root-dir $DATA_ROOT

echo "Generate folds"
PYTHONPATH=. python preprocessing/generate_folds.py --root-dir $DATA_ROOT --out folds.csv

~
~
~
~
"preprocess_data.sh" 18L, 620B written
```

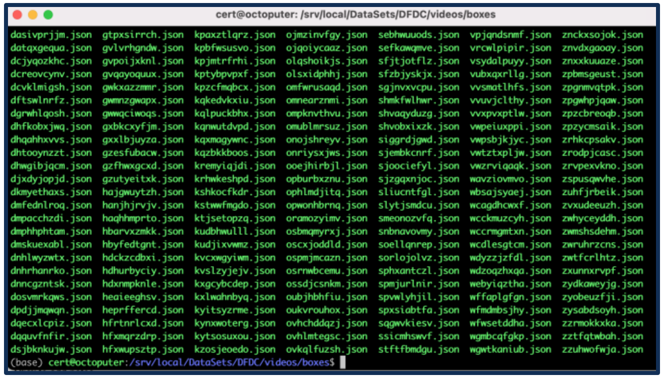
All of these steps are performed sequentially in /dfdc_deepfake_challenge/preprocess_data.sh

3.1 Extract Bounding Boxes



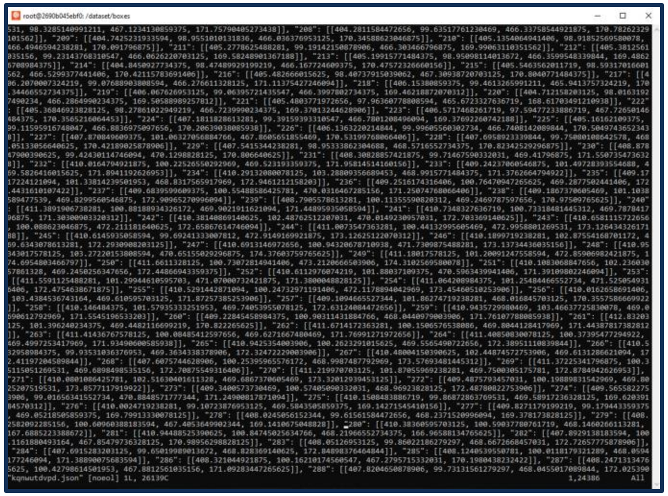
```
DATA_ROOT=/dataset
echo "Extracting bounding boxes from original videos"
PYTHONPATH=. python preprocessing/detect_original_faces.py --root-dir $DATA_ROOT
```

- For all real videos (18657), bounding boxes are generated for faces in each frame and stored in json files
  - stored in the container filesystem in /dataset/boxes and
  - Through samba share on a 2nd computer (Octoputer) in /srv/local/DataSets/DFDC/videos/boxes)
- Make sure DATA_ROOT=/dataset
- As you can see, file contains a list of pairs of points, for lower left and upper right corners of each box for each frame
- Each video takes about 15s to process through samba share, and about 1s if data is local.
- A random sampling of files had approximately 300 bounding box coordinates which implies a sampling of 30 fps (each video is 10s).



cert@octoputer: /srv/local/DataSets/DFDC/videos/boxes

```
dasivprjjm.json  gtpxsirrch.json  kpaxztlqrz.json  ojmzinvfgy.json  sebhwuuods.json  vpjqndsnmf.json  znckxsojok.json
datqxgequa.json  gvlvrhgndw.json  kpbfwsusvo.json  ojqoiycaaz.json  sefkawqmve.json  vrcwlpipir.json  znvdxgaoay.json
dcjyqozkhc.json  gvpoijxknl.json  kpjmtrfrhi.json  olqshoikjs.json  sfjtjotflz.json  vsydalpuyy.json  znxxkuuaze.json
dcreovcynv.json  gvqayoquux.json  kptybpvpxf.json  olsxidphhj.json  sfzbjyskjx.json  vubxqxrllg.json  zpbmsgeust.json
dcvklmigsh.json  gwkxazzmmr.json  kpzcfmqbcx.json  omfwrusaqd.json  sgjnvxvcpu.json  vvsmatlhfs.json  zpgnmvqtpk.json
dftswlnrfz.json  gwmnzgwapx.json  kqkedvkxiu.json  omnearznmi.json  shmkfwlhwr.json  vvuvjclthy.json  zpgwhpjqaw.json
dgrwhlqosh.json  gwwqciwoqs.json  kqlpuckbhx.json  ompknvthvu.json  shvaqyduzg.json  vvxpvxptlw.json  zpzcbreoqb.json
dhfkobxjwq.json  gxbkcxyfjm.json  kqnwutdvpd.json  omublmrsuz.json  shvobxixzk.json  vwpeiuxppi.json  zpzycmsaik.json
dhqahhxvvs.json  gxxlbjuyza.json  kqxmagywnc.json  onojshreyv.json  siggrdjgwd.json  vwpsbjkjyc.json  zrhkcpsakv.json
dhtooynzzt.json  gzesfubacw.json  kqzbkkboos.json  onriysxjws.json  sjembkcnrf.json  vwtztxpljw.json  zrodpjcasc.json
dhwgibjqcm.json  gzfhwxgcxd.json  kremyiqjdi.json  ooejhirbjl.json  sjoociefyl.json  vwzrviqaqk.json  zrvpexvkno.json
djxdyjopjd.json  gzutyeitxk.json  krhwkeshpd.json  opburbxznu.json  sjzgqxnjoc.json  wavziovmvo.json  zspusqwvhe.json
dkmyethaxs.json  hajgwuytzh.json  kshkocfkdr.json  ophlmdjitq.json  sliucntfgl.json  wbsajsyaej.json  zuhfjrbeik.json
dmfednlroq.json  hanjhjrvjv.json  kstwwfmgdo.json  opwonhbrnq.json  slytjsmdcu.json  wcagdhcwxf.json  zvxudeeuzh.json
dmpacchzdi.json  haqhhmprto.json  ktjsetopzq.json  oramozyimv.json  smeonozvfq.json  wcckmuzcyh.json  zwhyceyddh.json
dmphhphtam.json  hbarvxzmkk.json  kudbhwulll.json  osbmqmyrxj.json  snbnavovmy.json  wccrmgmtxn.json  zwmshsdehm.json
dmskuexabl.json  hbyfedtgnt.json  kudjixvwmz.json  oscxjoddld.json  soellqnrep.json  wcdlesgtcm.json  zwruhrzcns.json
dnhlwyzwtx.json  hdckzcdbxi.json  kvcxwgyiwm.json  ospmjmcazn.json  sorlojolvz.json  wdyzzjzfdl.json  zwtfcrlhtz.json
dnhrhanrko.json  hdhurbyciy.json  kvslzyjejv.json  osrnwbcemu.json  sphxantczl.json  wdzoqzhxqa.json  zxunnxrvpf.json
dnncgzntsk.json  hdxnmpknle.json  kxgcybcdep.json  ossdjcsnkm.json  spmjurlnir.json  webyiqztha.json  zydkaweyjg.json
dosvmrkqws.json  heaieeghsv.json  kxlwahnbyq.json  oubjhbhfiu.json  spvwlyhjil.json  wffaplgfgn.json  zyobeuzfji.json
dpdjjmqwqn.json  heprffercd.json  kyitsyzrme.json  oukvrouhox.json  spxsiabtfa.json  wfmdmbsjhy.json  zysabdsoyh.json
dqecxlcpiz.json  hfrtnrlcxd.json  kynxwoterg.json  ovhchddqzj.json  sqgwvkiesv.json  wfwsetddha.json  zzrmokkxka.json
dqquvfnfir.json  hfxmqrzdrp.json  kytsosuxou.json  ovhlmtegsc.json  ssicmhswvf.json  wgmbcqfgkp.json  zztfqtwbah.json
dsjbknkujw.json  hfxwupsztp.json  kzosjeoedo.json  ovkqlfuzsh.json  stftfbmdgu.json  wgwtkaniub.json  zzuhwofwja.json
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/boxes$
```

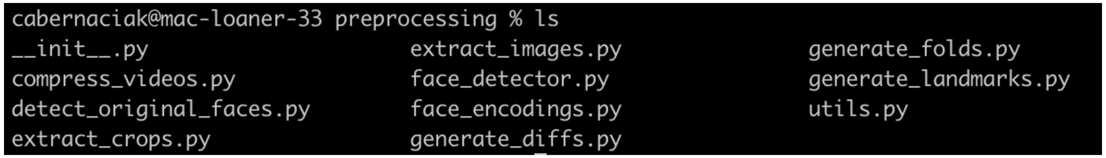


root@2690b045ebf0: /dataset/boxes

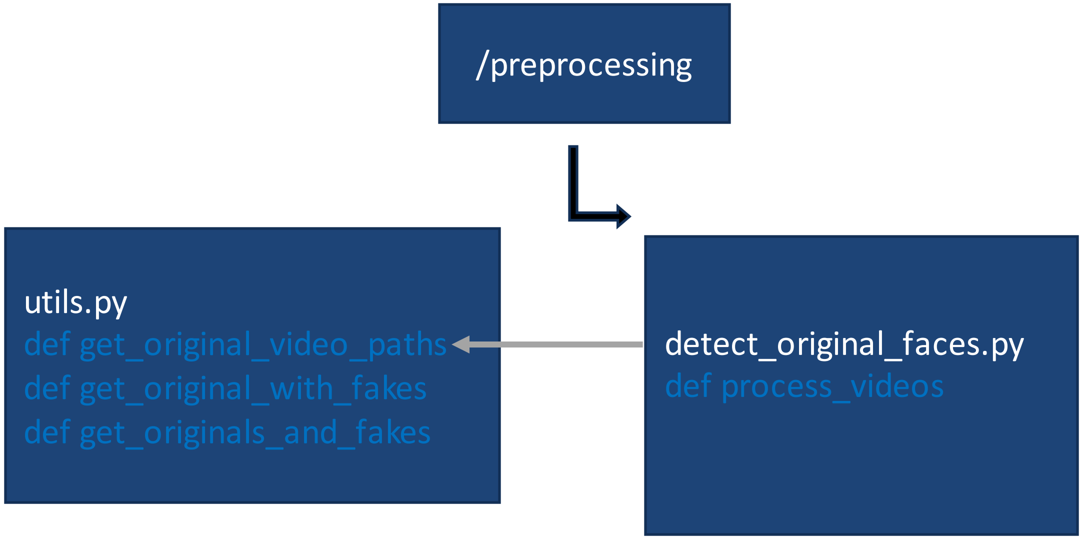
```
531, 98.3285140991211, 467.1234130859375, 171.75790405273438]], "208": [[404.2811584472656, 99.63517761230469, 466.33758544921875, 170.78262329
101562]], "209": [[404.7425231933594, 98.9551010131836, 466.036376953125, 170.34588623046875]], "210": [[405.1354064941406, 98.9185256958008,
466.4946594238281, 170.091796875]], "211": [[405.2778625488281, 99.19142150878906, 466.303466796875, 169.99063110351562]], "212": [[405.3812561
035156, 99.23143768310547, 466.0626220703125, 169.58248901367188]], "213": [[405.19915771484375, 98.95098114013672, 466.3599548339844, 169.4862
6708984375]], "214": [[404.8450927734375, 98.47489929199219, 466.167724609375, 170.47572326660156]], "215": [[405.5463562011719, 98.59317016601
562, 466.5299377441406, 170.4211578369140 6]], "216": [[405.482666015625, 98.40737915039062, 467.30938720703125, 170.8040771484375]], "217": [[4
06.2070007324219, 99.07688903808594, 466.276611328125, 171.11375427246094]], "218": [[406.15380859375, 99.4613265991211, 465.9413757324219, 170
.34466552734375]], "219": [[406.067626953125, 99.0639572143554 7, 466.3997802734375, 169.46218872070312]], "220": [[404.712158203125, 98.0163192
7490234, 466.2864990234375, 169.50588989257812]], "221": [[405.4803771972656, 97.96360778808594, 465.6723327636719, 168.61703491210938]], "222"
: [[405.36846923828125, 98.27861022949219, 466.7239990234375, 169.37013244628906]], "223": [[406.5717468261719, 97.59477233886719, 467.72650146
484375, 170.3565216064453]], "224": [[407.1811828613281, 99.39159393310547, 466.7801208496094, 169.37692260742188]], "225": [[405.16162109375,
99.11595916748047, 466.8836975097656, 170.20639038085938]], "226": [[406.1363220214844, 99.99605560302734, 466.7408142089844, 170.5049743652343
8]], "227": [[407.870849609375, 101.06327056884766, 467.8605651855469, 170.53199768066406]], "228": [[407.6958923339844, 99.75080108642578, 468
.05133056640625, 170.42189025878906]], "229": [[407.5415344238281, 98.95333862304688, 468.5716552734375, 170.82342529296875]], "230": [[408.878
47900390625, 99.42430114746094, 470.1298828125, 170.806640625]], "231": [[408.30828857421875, 99.71467590332031, 469.41796875, 171.550735473632
8]], "232": [[410.0164794921875, 100.22526550292969, 469.523193359375, 171.95814514160156]], "233": [[409.24237060546875, 101.49728393554688, 4
69.5826416015625, 171.8941192626953]], "234": [[410.29132080078125, 103.28809356689453, 468.9915771484375, 171.3762664794922]], "235": [[409.17
17224121094, 101.33814239501953, 468.8317565917969, 172.94612121582031]], "236": [[409.2516174316406, 100.76470947265625, 469.2877502441406, 172
.4431610107422]], "237": [[409.683959960937 5, 100.55488586425781, 470.0316467285156, 171.25074768066406]], "238": [[409.1867370605469, 101.1038
589477539, 469.8299560546875, 172.90965270996094]], "239": [[408.7905578613281, 100.11355590820312, 469.2469787597656, 170.97509765625]], "240"
: [[411.3891906738281, 100.88188934326172, 469.9021911621094, 171.44895935058594]], "241": [[410.7348327636719, 100.73318481445312, 469.7878417
96875, 171.30300903320312]], "242": [[410.38140869140625, 102.48762512207031, 470.0149230957031, 172.703369140625]], "243": [[410.6581115722656
, 100.088623046875, 472.211181640625, 172.65867614746094]], "244": [[411.0073547363281, 100.44132995605469, 472.9958801269531, 173.126434326171
88]], "245": [[410.6145935058594, 99.69241333007812, 472.9149169921875, 173.12625122070312]], "246": [[410.1899719238281, 102.87554168701172, 4
69.6343078613281, 172.2930908203125]], "247": [[410.6913146972656, 100.94320678710938, 471.7309875488281, 173.13734436035156]], "248": [[410.95
34301757812 5, 103.27220153808594, 470.65155029296875, 174.37603759765625]], "249": [[411.18017578125, 101.20091247558594, 472.8590698242187 5, 1
74.6954803466797]], "250": [[411.6611328125, 100.73072814941406, 473.2120666503906, 174.3102569580078]], "251": [[410.1083068847656, 102.236030
57861328, 469.2450256347656, 172.44866943359375]], "252": [[410.6112976074219, 101.88037109375, 470.5963439941406, 171.39109802246094]], "253":
 [[411.5591125488281, 101.29944610595703, 471.07000732421875, 171.3800048828125]], "254": [[411.064208984375, 101.2548446655273 4, 471.525054931
6406, 172.4754638671875]], "255": [[410.5291442871094, 100.24732971191406, 472.1178894042969, 173.45460510253906]], "256": [[410.0162658691406,
 103.4384536743164, 469.610595703125, 171.87257385253906]], "257": [[409.1094665527344, 101.86274719238281, 468.016845703125, 170.3557586669922
]], "258": [[410.146484375, 101.57935333251953, 469.74053955078125, 172.63124084472656]], "259": [[410.9435729980469, 103.46637725830078, 469.0
690612792969, 171.5545196533203]], "260": [[409.22845458984375, 100.90311431884766, 468.0440979003906, 171.76107788085938]], "261": [[412.83203
125, 101.396240234375, 469.4482116699219, 170.822265625]], "262": [[411.6714172363281, 100.1506576538086, 469.8044128417969, 171.44387817382812
]], "263": [[411.41436767578125, 100.08485412597656, 469.6271667480469, 171.76991271972656]], "264": [[411.40850830078125, 100.37395477294922,
469.4997253417969, 171.93490600585938]], "265": [[410.9425354003906, 100.2623291015625, 469.5565490722656, 172.38951110839844]], "266": [[410.5
32958984375, 99.93531036376953, 469.3634338378906, 172.32472229003906]], "267": [[410.48004150390625, 102.44874572753906, 469.6131286621094, 17
2.41197204589844]], "268": [[407.6075744628906, 100.25395965576172, 468.9987487792969, 173.57693481445312]], "269": [[411.37225341796875, 100.3
5115051269531, 469.6898498535156, 172.70875549316406]], "270": [[411.21997070703125, 101.87055969238281, 469.7500305175781, 172.8784942626953]],
"271": [[410.0801086425781, 102.51630401611328, 469.6867370605469, 173.32012939453125]], "272": [[409.4875793457031, 100.19889831542969, 469.80
25207519531, 173.8577117919922]], "273": [[409.3400573730469, 100.57405090332031, 468.96923828125, 172.48780822753906]], "274": [[409.565582275
3906, 99.01656341552734, 470.8848571777344, 171.24900817871094]], "275": [[410.1508483886719, 99.86872863769531, 469.58917236328125, 169.620391
84570312]], "276": [[410.0024719238281, 99.10723876953125, 469.5843505859375, 169.14271545410156]], "277": [[409.8271179199219, 99.179443359375
, 469.05218505859375, 169.79913330078125]], "278": [[408.0245056152344, 99.61561584472656, 468.2371520996094, 169.37817382812 5]], "279": [[408.
2582092285156, 100.60960388183594, 467.4053649902344, 169.1410675048828]], "280": [[410.38360595703125, 100.59037780761719, 468.1460266113281,
167.68852233886672]], "281": [[410.94488525390625, 100.84745025634766, 468.21966552734375, 166.96588134765625]], "282": [[407.8929138183594, 100
.1161880493164, 467.85479736328125, 170.98956298828125]], "283": [[408.05126953125, 99.86022186279297, 468.6672668457031, 172.72657775878906]],
 "284": [[407.6915283203125, 99.65019989013672, 468.828369140625, 172.84898376464844]], "285": [[408.1240539550781, 100.0118179321289, 468.0594
177246094, 171.38890075683594]], "286": [[408.321044921875, 100.16210174560547, 467.2795715332031, 170.1980438232422]], "287": [[408.2471313476
5625, 100.42798614501953, 467.8812561035156, 171.09283447265625]], "288": [[407.8204650878906, 99.73131561279297, 468.0455017089844, 172.025390
"kqnwutdvpd.json" [noeol] 1L, 26139C                                                                        1,24386        All
```

3.1 Extract Bounding Boxes - Files & Functions

/preprocessing/detect_original_faces.py



```
cabernaciak@mac-loaner-33 preprocessing % ls
__init__.py                     extract_images.py               generate_folds.py
compress_videos.py              face_detector.py                generate_landmarks.py
detect_original_faces.py        face_encodings.py               utils.py
extract_crops.py                generate_diffs.py
```



/preprocessing

utils.py
def get_original_video_paths
def get_original_with_fakes
def get_originals_and_fakes

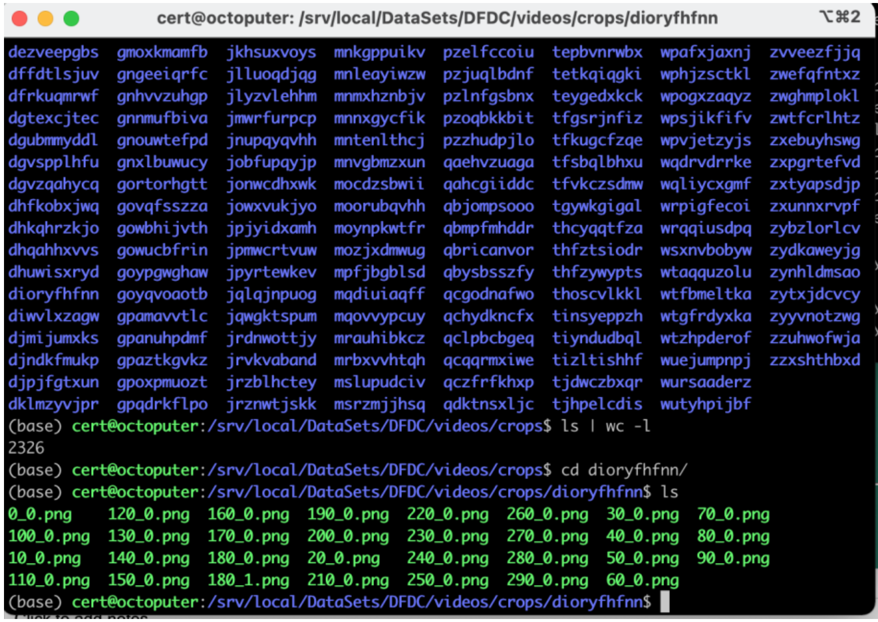
detect_original_faces.py
def process_videos

3.2 Crop Faces From Frames

```
echo "Extracting crops as pngs"
PYTHONPATH=. python preprocessing/extract_crops.py --root-dir $DATA_ROOT --crops-dir crops
```

The bounding boxes for real videos are used to extract crops for real and the fakes made from them.

- In each folder in /datasets/videos, the metadata.json file is used to check if there is a matching json file in /boxes
- A directory for each video is created, inside are the png crops

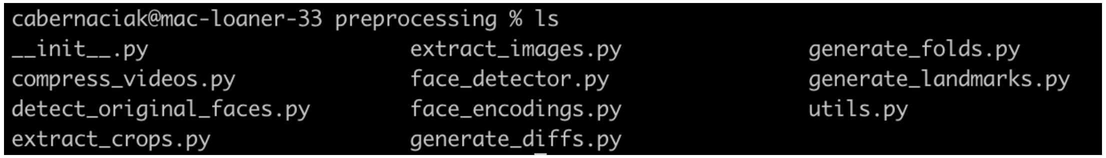


cert@octoputer: /srv/local/DataSets/DFDC/videos/crops/dioryfhfnn ⌥⌘2

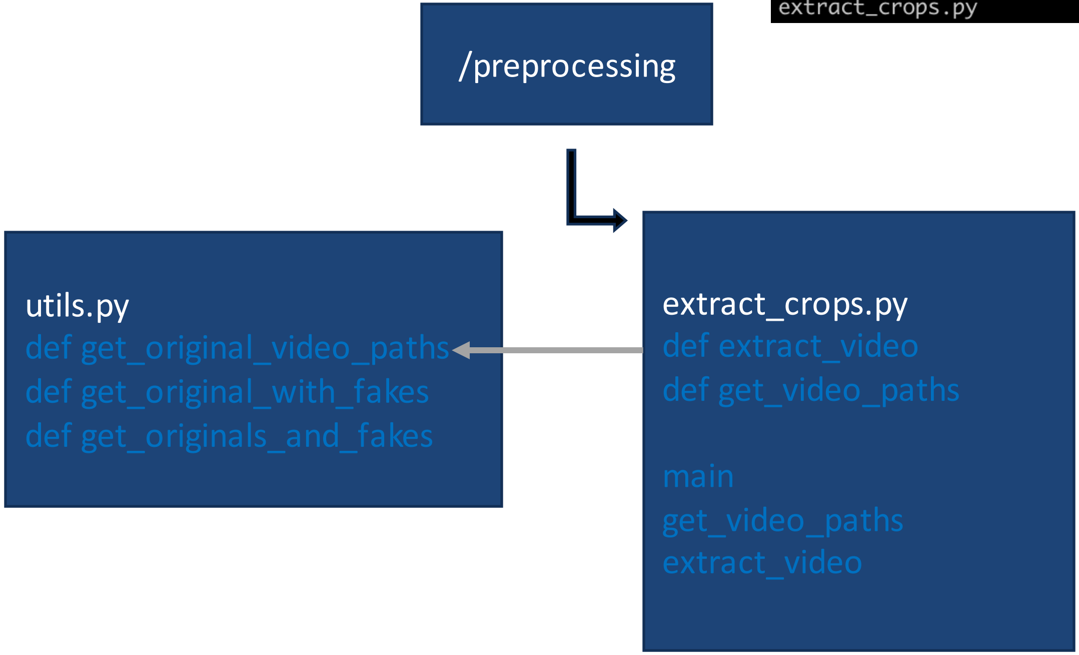
```
dezveepgbs  gmoxkmamfb  jkhsuxvoys  mnkgppuikv  pzelfccoiu  tepbvnrwbx  wpafxjaxnj  zvveezfjjq
dffdtlsjuv  gngeeiqrfc  jlluoqdjqg  mnleayiwzw  pzjuqlbdnf  tetkqiqgki  wphjzsctkl  zwefqfntxz
dfrkuqmrwf  gnhvvzuhgp  jlyzvlehhm  mnmxhznbjv  pzlnfgsbnx  teygedxkck  wpogxzaqyz  zwghmplokl
dgtexcjtec  gnnmufbiva  jmwrfurpcp  mnnxgycfik  pzoqbkkbit  tfgsrjnfiz  wpsjikfifv  zwtfcrlhtz
dgubmmyddl  gnouwtefpd  jnupqyqvhh  mntenlthcj  pzzhudpjlo  tfkugcfzqe  wpvjetzyjs  zxebuyhswg
dgvspplhfu  gnxlbuwucy  jobfupqyjp  mnvgbmzxun  qaehvzuaga  tfsbqlbhxu  wqdrvdrrke  zxpgrtefvd
dgvzqahycq  gortorhgtt  jonwcdhxwk  mocdzsbwii  qahcgiiddc  tfvkczsdmw  wqliycxgmf  zxtyapsdjp
dhfkobxjwq  govqfsszza  jowxvukjyo  moorubqvhh  qbjompsooo  tgywkgigal  wrpigfecoi  zxunnxrvpf
dhkqhrzkjo  gowbhijvth  jpjyidxamh  moynpkwtfr  qbmpfmhddr  thcyqqtfza  wrqqiusdpq  zybzlorlcv
dhqahhxvvs  gowucbfrin  jpmwcrtvuw  mozjxdmwug  qbricanvor  thfztsiodr  wsxnvbobyw  zydkaweyjg
dhuwisxryd  goypgwghaw  jpyrtewkev  mpfjbgblsd  qbysbsszfy  thfzywypts  wtaqquzolu  zynhldmsao
dioryfhfnn  goyqvoaotb  jqlqjnpuog  mqdiuiaqff  qcgodnafwo  thoscvlkkl  wtfbmeltka  zytxjdcvcy
diwvlxzagw  gpamavvtlc  jqwgktspum  mqovvypcuy  qchydkncfx  tinsyeppzh  wtgfrdyxka  zyyvnotzwg
djmijumxks  gpanuhpdmf  jrdnwottjy  mrauhibkcz  qclpbcbgeq  tiyndudbql  wtzhpderof  zzuhwofwja
djndkfmukp  gpaztkgvkz  jrvkvaband  mrbxvvhtqh  qcqqrmxiwe  tizltishhf  wuejumpnpj  zzxshthbxd
djpjfgtxun  gpoxpmuozt  jrzblhctey  mslupudciv  qczfrfkhxp  tjdwczbxqr  wursaaderz
dklmzyvjpr  gpqdrkflpo  jrznwtjskk  msrzmjjhsq  qdktnsxljc  tjhpelcdis  wutyhpijbf
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/crops$ ls | wc -l
2326
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/crops$ cd dioryfhfnn/
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/crops/dioryfhfnn$ ls
0_0.png    120_0.png  160_0.png  190_0.png  220_0.png  260_0.png  30_0.png  70_0.png
100_0.png  130_0.png  170_0.png  200_0.png  230_0.png  270_0.png  40_0.png  80_0.png
10_0.png   140_0.png  180_0.png  20_0.png   240_0.png  280_0.png  50_0.png  90_0.png
110_0.png  150_0.png  180_1.png  210_0.png  250_0.png  290_0.png  60_0.png
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/crops/dioryfhfnn$
```

3.2 Crop Faces From Frames - Files & Functions

/preprocessing/extract_crops.py



```
cabernaciak@mac-loaner-33 preprocessing % ls
__init__.py                     extract_images.py               generate_folds.py
compress_videos.py              face_detector.py                generate_landmarks.py
detect_original_faces.py        face_encodings.py               utils.py
extract_crops.py                generate_diffs.py
```



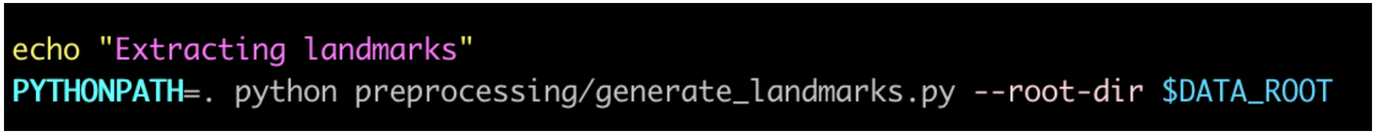
/preprocessing

utils.py
def get_original_video_paths
def get_original_with_fakes
def get_originals_and_fakes

extract_crops.py
def extract_video
def get_video_paths

main
get_video_paths
extract_video

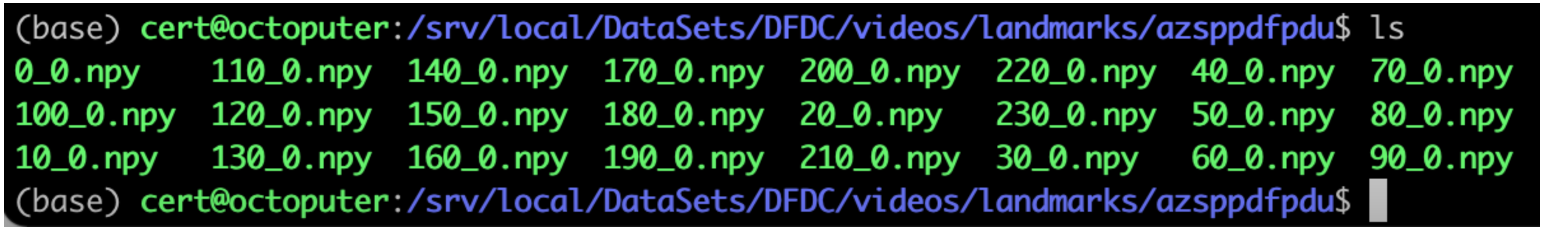
3.3 Extract Landmarks



```
echo "Extracting landmarks"
PYTHONPATH=. python preprocessing/generate_landmarks.py --root-dir $DATA_ROOT
```

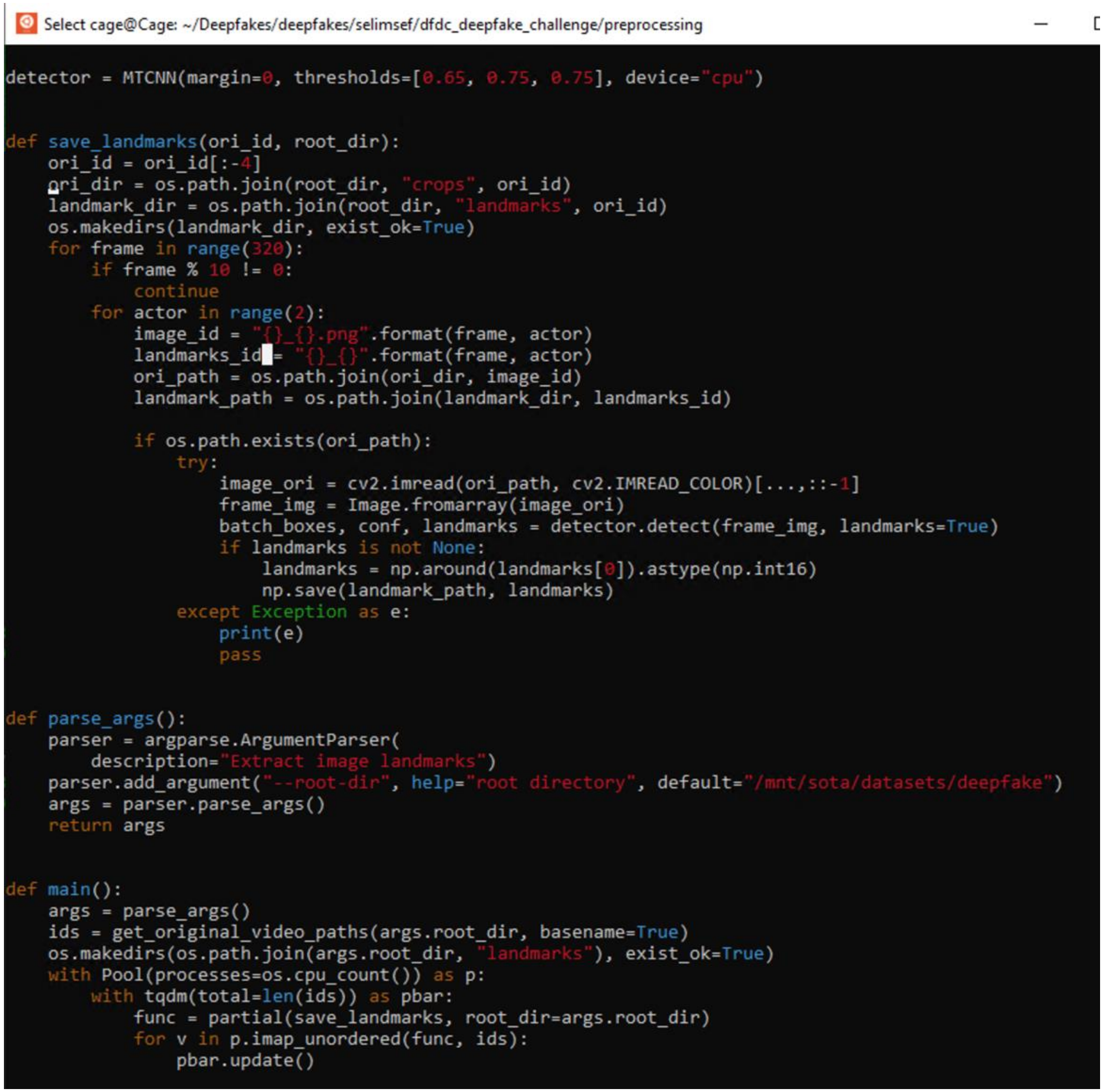
This generates landmarks as a numpy binary file (.npy) for each real video and places them in the /dataset/landmarks directory

- Landmarks are saved in a numpy binary file with extension .npy
- This process was very fast, took ~ 30 mins.



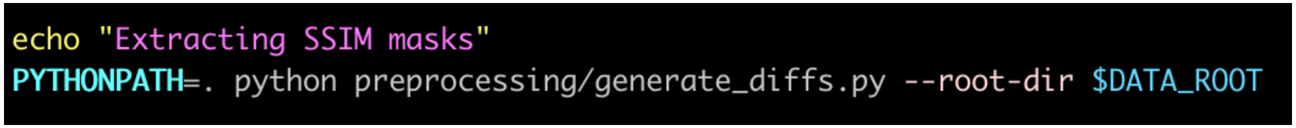
```
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/landmarks/azsppdfpdu$ ls
0_0.npy    110_0.npy  140_0.npy  170_0.npy  200_0.npy  220_0.npy  40_0.npy  70_0.npy
100_0.npy  120_0.npy  150_0.npy  180_0.npy  20_0.npy   230_0.npy  50_0.npy  80_0.npy
10_0.npy   130_0.npy  160_0.npy  190_0.npy  210_0.npy  30_0.npy   60_0.npy  90_0.npy
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/landmarks/azsppdfpdu$ 
```

3.3 Extract Landmarks


Select cage@Cage: ~/Deepfakes/deepfakes/selimsef/dfdc_deepfake_challenge/preprocessing

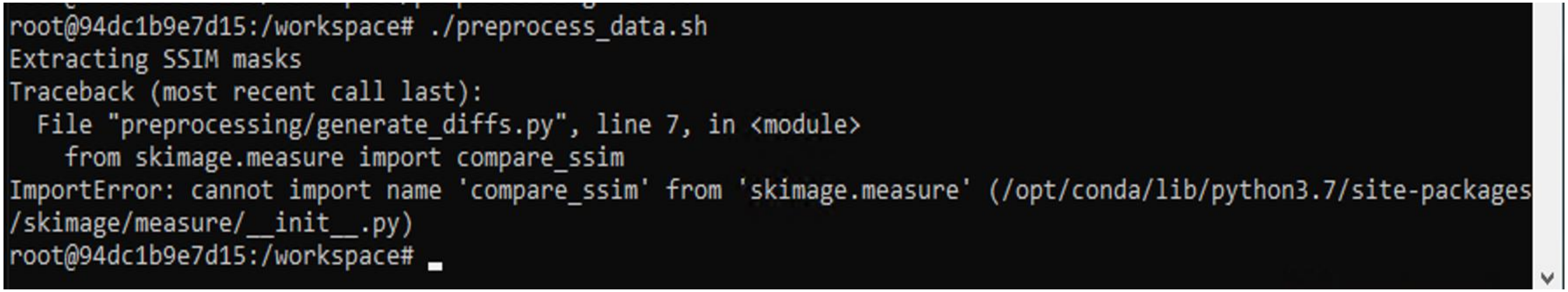
```
detector = MTCNN(margin=0, thresholds=[0.65, 0.75, 0.75], device="cpu")


def save_landmarks(ori_id, root_dir):
    ori_id = ori_id[:-4]
    ori_dir = os.path.join(root_dir, "crops", ori_id)
    landmark_dir = os.path.join(root_dir, "landmarks", ori_id)
    os.makedirs(landmark_dir, exist_ok=True)
    for frame in range(320):
        if frame % 10 != 0:
            continue
        for actor in range(2):
            image_id = "{}_{}.png".format(frame, actor)
            landmarks_id = "{}_{}".format(frame, actor)
            ori_path = os.path.join(ori_dir, image_id)
            landmark_path = os.path.join(landmark_dir, landmarks_id)

            if os.path.exists(ori_path):
                try:
                    image_ori = cv2.imread(ori_path, cv2.IMREAD_COLOR)[...,::-1]
                    frame_img = Image.fromarray(image_ori)
                    batch_boxes, conf, landmarks = detector.detect(frame_img, landmarks=True)
                    if landmarks is not None:
                        landmarks = np.around(landmarks[0]).astype(np.int16)
                        np.save(landmark_path, landmarks)
                except Exception as e:
                    print(e)
                    pass


def parse_args():
    parser = argparse.ArgumentParser(
        description="Extract image landmarks")
    parser.add_argument("--root-dir", help="root directory", default="/mnt/sota/datasets/deepfake")
    args = parser.parse_args()
    return args


def main():
    args = parse_args()
    ids = get_original_video_paths(args.root_dir, basename=True)
    os.makedirs(os.path.join(args.root_dir, "landmarks"), exist_ok=True)
    with Pool(processes=os.cpu_count()) as p:
        with tqdm(total=len(ids)) as pbar:
            func = partial(save_landmarks, root_dir=args.root_dir)
            for v in p.imap_unordered(func, ids):
                pbar.update()
```

3.4 Extract SSIM Masks

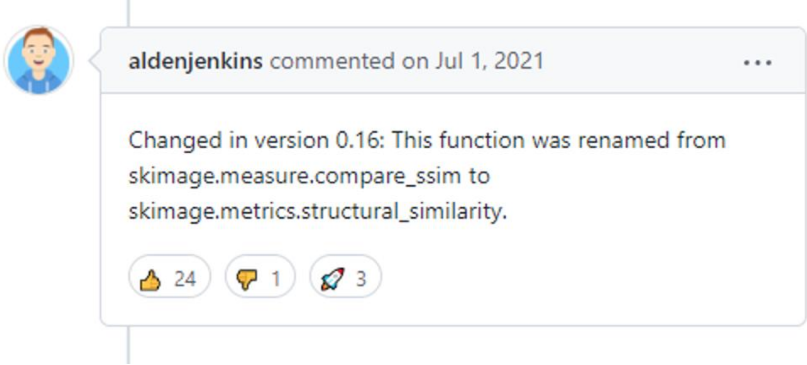


```
echo "Extracting SSIM masks"
PYTHONPATH=. python preprocessing/generate_diffs.py --root-dir $DATA_ROOT
```

Uh oh – this step throws an error



```
root@94dc1b9e7d15:/workspace# ./preprocess_data.sh
Extracting SSIM masks
Traceback (most recent call last):
  File "preprocessing/generate_diffs.py", line 7, in <module>
    from skimage.measure import compare_ssim
ImportError: cannot import name 'compare_ssim' from 'skimage.measure' (/opt/conda/lib/python3.7/site-packages
/skimage/measure/__init__.py)
root@94dc1b9e7d15:/workspace# _
```

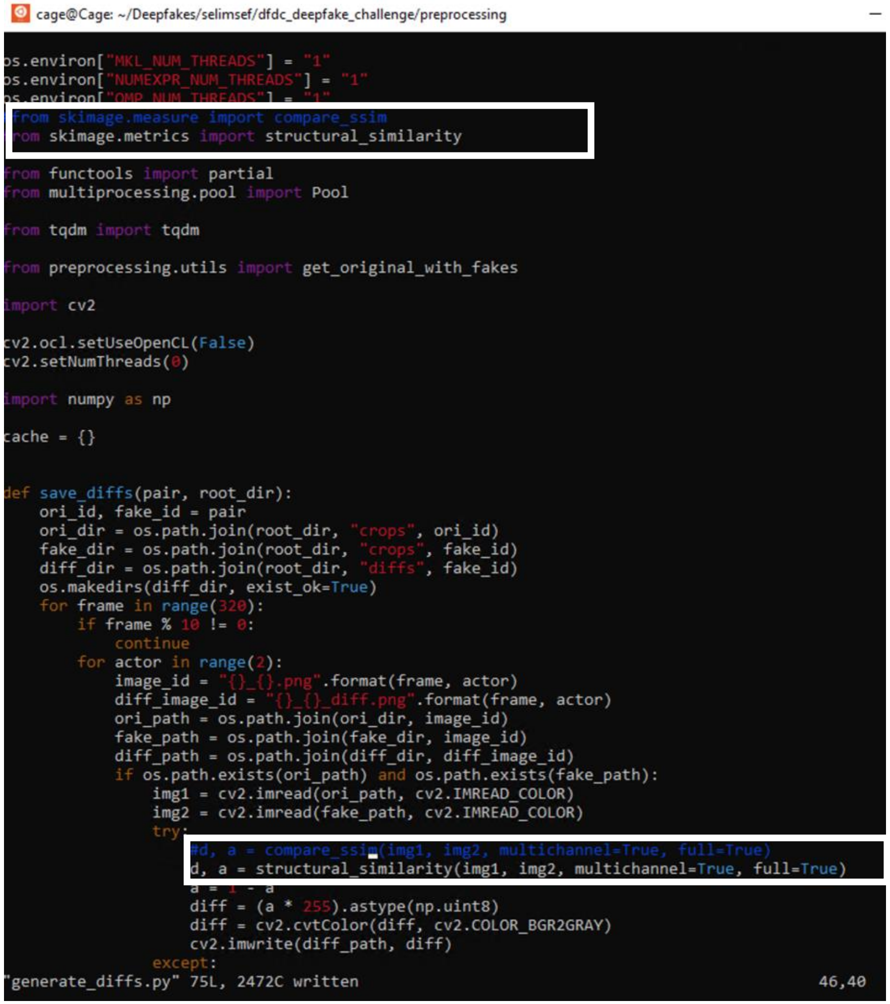


aldenjenkins commented on Jul 1, 2021

Changed in version 0.16: This function was renamed from skimage.measure.compare_ssim to skimage.metrics.structural_similarity.

3.4 Extract SSIM Masks

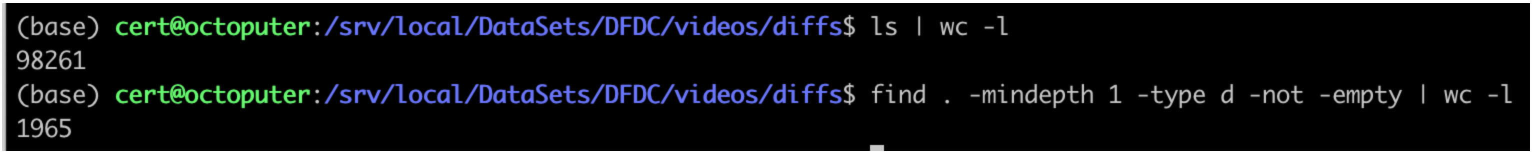
- We simply need to update function and module names in `generate_diffs.py`.
- `skimage.metrics.structural_similarity` returns the ‘mean structural similarity index over the image’.
- SSIM is a measure that quantifies similarity of two images – can be computed on RGB or greyscale images



cage@Cage: ~/Deepfakes/selimsef/dfdc_deepfake_challenge/preprocessing

```
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["NUMEXPR_NUM_THREADS"] = "1"
os.environ["OMP_NUM_THREADS"] = "1"
#from skimage.measure import compare_ssim
from skimage.metrics import structural_similarity

from functools import partial
from multiprocessing.pool import Pool

from tqdm import tqdm

from preprocessing.utils import get_original_with_fakes

import cv2

cv2.ocl.setUseOpenCL(False)
cv2.setNumThreads(0)

import numpy as np

cache = {}


def save_diffs(pair, root_dir):
    ori_id, fake_id = pair
    ori_dir = os.path.join(root_dir, "crops", ori_id)
    fake_dir = os.path.join(root_dir, "crops", fake_id)
    diff_dir = os.path.join(root_dir, "diffs", fake_id)
    os.makedirs(diff_dir, exist_ok=True)
    for frame in range(320):
        if frame % 10 != 0:
            continue
        for actor in range(2):
            image_id = "{}_{}.png".format(frame, actor)
            diff_image_id = "{}_{}_diff.png".format(frame, actor)
            ori_path = os.path.join(ori_dir, image_id)
            fake_path = os.path.join(fake_dir, image_id)
            diff_path = os.path.join(diff_dir, diff_image_id)
            if os.path.exists(ori_path) and os.path.exists(fake_path):
                img1 = cv2.imread(ori_path, cv2.IMREAD_COLOR)
                img2 = cv2.imread(fake_path, cv2.IMREAD_COLOR)
                try:
                    #d, a = compare_ssim(img1, img2, multichannel=True, full=True)
                    d, a = structural_similarity(img1, img2, multichannel=True, full=True)
                    a = 1 - a
                    diff = (a * 255).astype(np.uint8)
                    diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
                    cv2.imwrite(diff_path, diff)
                except:
"generate_diffs.py" 75L, 2472C written                                        46,40
```

3.4 Extract SSIM Masks

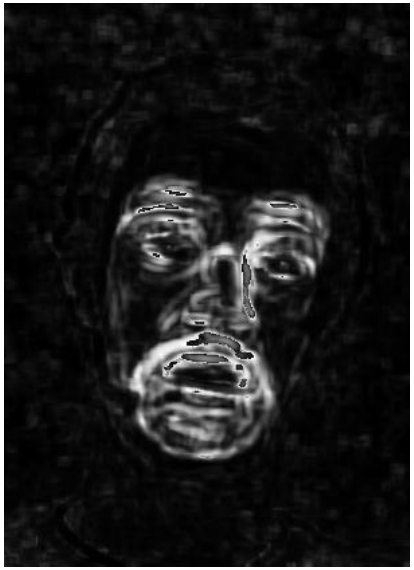


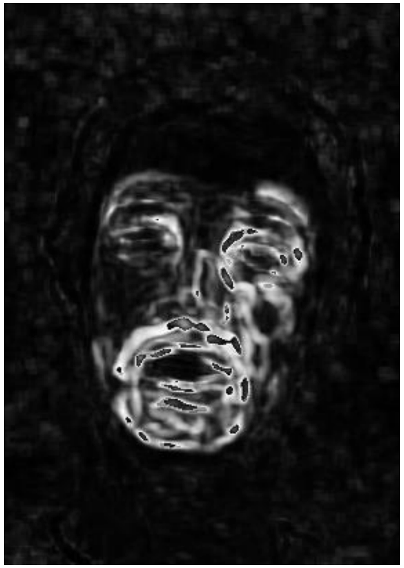
```
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/diffs$ ls | wc -l
98261
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/diffs$ find . -mindepth 1 -type d -not -empty | wc -l
1965
```

SSIM masks were extracted for 1965 fake videos using the ‘structural_similarity’ function of skimage (see previous slide)

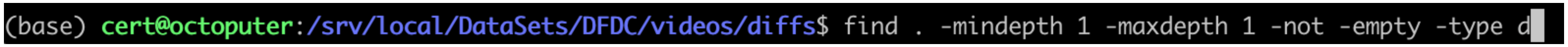
Shown are

- (left) kvfkkcctax/0_0_diff.png
- (right) kvfkkcctax/10_0_diff.png



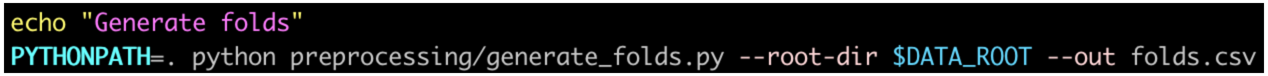


- Useful command for listing all non-empty directories:

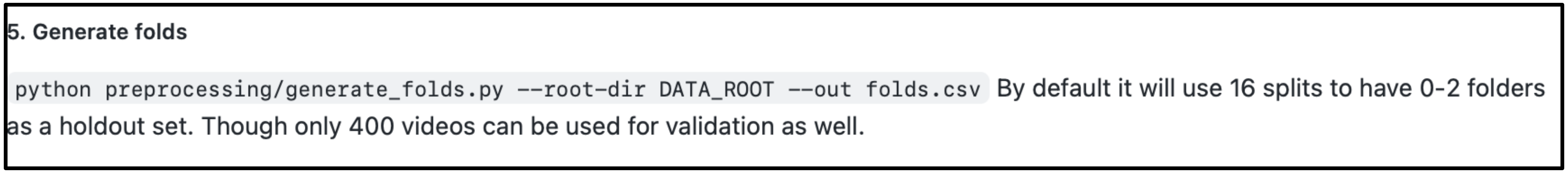


```
(base) cert@octoputer:/srv/local/DataSets/DFDC/videos/diffs$ find . -mindepth 1 -maxdepth 1 -not -empty -type d
```

3.5 Generate Folds

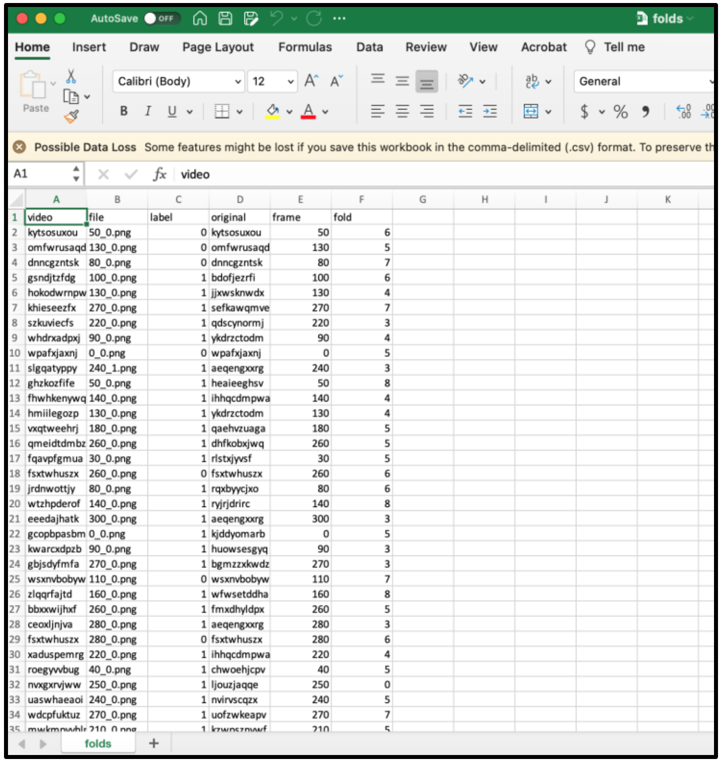


```
echo "Generate folds"
PYTHONPATH=. python preprocessing/generate_folds.py --root-dir $DATA_ROOT --out folds.csv
```



5. Generate folds

`python preprocessing/generate_folds.py --root-dir DATA_ROOT —out folds.csv` By default it will use 16 splits to have 0-2 folders as a holdout set. Though only 400 videos can be used for validation as well.



Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve th

A1 video

	A	B	C	D	E	F
1	video	file	label	original	frame	fold
2	kytsosuxou	50_0.png	0	kytsosuxou	50	6
3	omfwrusaqd	130_0.png	0	omfwrusaqd	130	5
4	dnncgzntsk	80_0.png	0	dnncgzntsk	80	7
5	gsndjtzfdg	100_0.png	1	bdofjezrfi	100	6
6	hokodwrnpw	130_0.png	1	jjxwsknwdx	130	4
7	khieseezfx	270_0.png	1	sefkawqmve	270	7
8	szkuviecfs	220_0.png	1	qdscynormj	220	3
9	whdrxadpxj	90_0.png	1	ykdrzctodm	90	4
10	wpafxjaxnj	0_0.png	0	wpafxjaxnj	0	5
11	slgqatyppy	240_1.png	1	aeqengxxrg	240	3
12	ghzkozfife	50_0.png	1	heaieeghsv	50	8
13	fhwhkenywq	140_0.png	1	ihhqcdmpwa	140	4
14	hmiilegozp	130_0.png	1	ykdrzctodm	130	4
15	vxqtweehrj	180_0.png	1	qaehvzuaga	180	5
16	qmeidtdmbz	260_0.png	1	dhfkobxjwq	260	5
17	fqavpfgmua	30_0.png	1	rlstxjyvsf	30	5
18	fsxtwhuszx	260_0.png	0	fsxtwhuszx	260	6
19	jrdnwottjy	80_0.png	1	rqxbyycjxo	80	6
20	wtzhpderof	140_0.png	1	ryjrjdrirc	140	8
21	eeedajhatk	300_0.png	1	aeqengxxrg	300	3
22	gcopbpasbm	0_0.png	1	kjddyomarb	0	5
23	kwarcxdpzb	90_0.png	1	huowsesgyq	90	3
24	gbjsdyfmfa	270_0.png	1	bgmzzxkwdz	270	3
25	wsxnvbobyw	110_0.png	0	wsxnvbobyw	110	7
26	zlqqrfajtd	160_0.png	1	wfwsetddha	160	8
27	bbxxwijhxf	260_0.png	1	fmxdhyldpx	260	5
28	ceoxljnjva	280_0.png	1	aeqengxxrg	280	3
29	fsxtwhuszx	280_0.png	0	fsxtwhuszx	280	6
30	xaduspemrg	220_0.png	1	ihhqcdmpwa	220	4
31	roegyvvbug	40_0.png	1	chwoehjcpv	40	5
32	nvxgxrvjww	250_0.png	1	ljouzjaqqe	250	0
33	uaswhaeaoi	240_0.png	1	nvirvscqzx	240	5
34	wdcpfuktuz	270_0.png	1	uofzwkeapv	270	7
35	mwkmpwhlr	210_0.png	1	kzwpsznvwf	210	5

- A folds.csv file is created in /workspace
- I think ‘label’ denotes 0=real video, 1 = fake video
- ‘frame’ is self evident, it is the frame (time step in ms) number of the video
- ‘fold’ denotes a partition of the dataset

3.6 Example of Processed Data

Taking a closer look at a deepfake, it's original, it's crops, and masks

1. Play videos, real and fake

2. Draw bounding box on first frame of real and fake video example

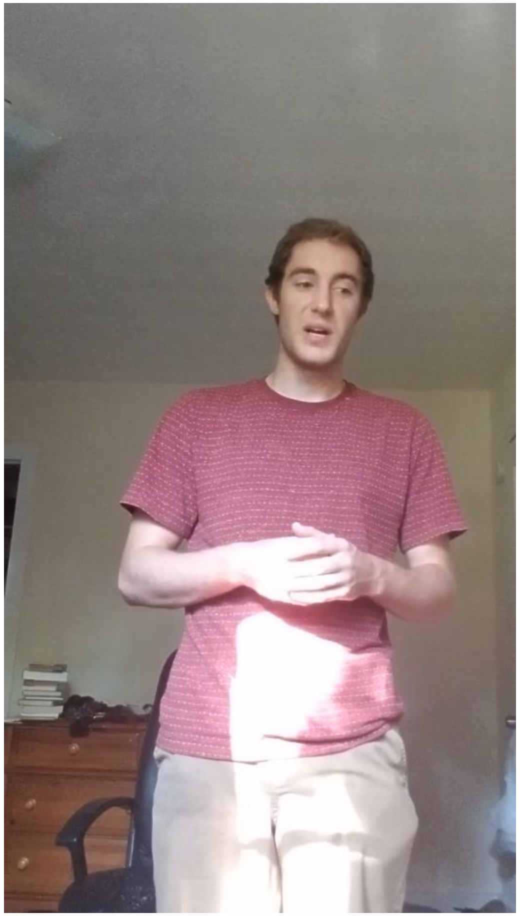
3. Show crops for real and fake

4. Draw landmarks on real and fake frame

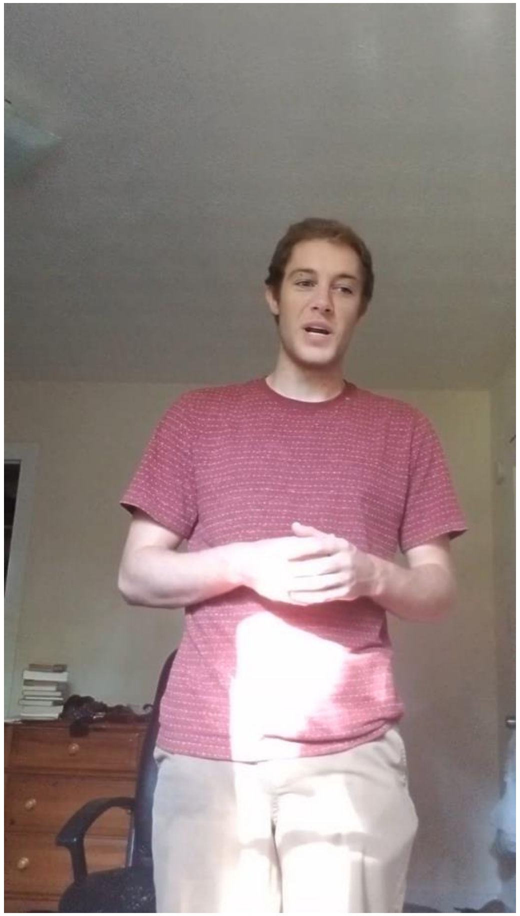
5. Display structural similarity masks

3.1 Example Real and Fake Videos

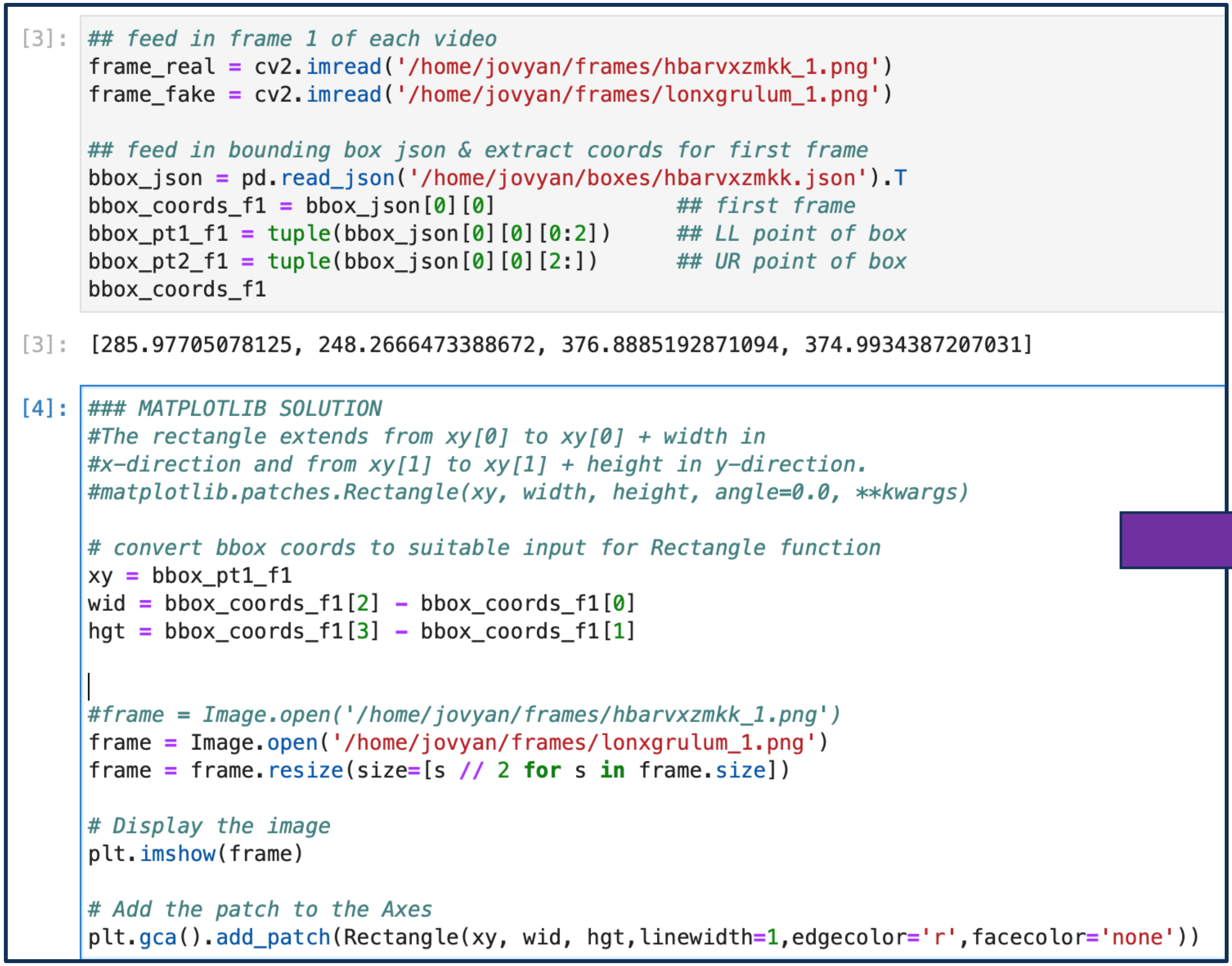
/dfdc_train_part_25/hbarvxzmkk.mp4



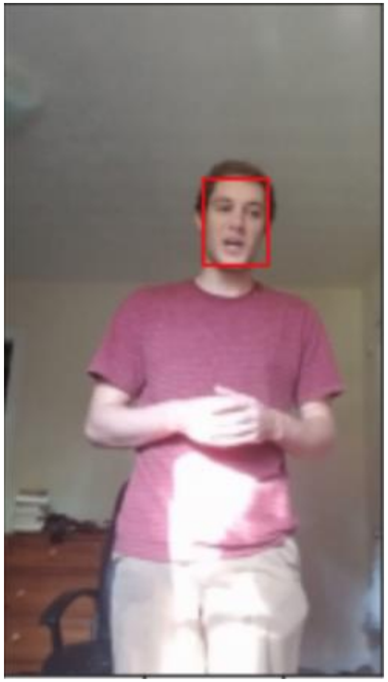
/dfdc_train_part_25/lonxgrulum.mp4



3.6.2 Drawing Bounding Boxes – sanity check

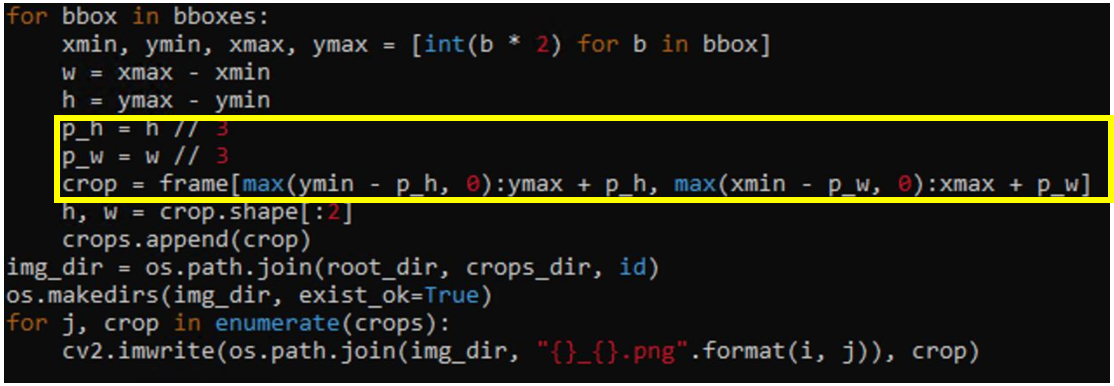


```
[3]: ## feed in frame 1 of each video
     frame_real = cv2.imread('/home/jovyan/frames/hbarvxzmkk_1.png')
     frame_fake = cv2.imread('/home/jovyan/frames/lonxgrulum_1.png')

     ## feed in bounding box json & extract coords for first frame
     bbox_json = pd.read_json('/home/jovyan/boxes/hbarvxzmkk.json').T
     bbox_coords_f1 = bbox_json[0][0]              ## first frame
     bbox_pt1_f1 = tuple(bbox_json[0][0][0:2])     ## LL point of box
     bbox_pt2_f1 = tuple(bbox_json[0][0][2:])      ## UR point of box
     bbox_coords_f1

[3]: [285.97705078125, 248.2666473388672, 376.8885192871094, 374.9934387207031]

[4]: ### MATPLOTLIB SOLUTION
     #The rectangle extends from xy[0] to xy[0] + width in
     #x-direction and from xy[1] to xy[1] + height in y-direction.
     #matplotlib.patches.Rectangle(xy, width, height, angle=0.0, **kwargs)

     # convert bbox coords to suitable input for Rectangle function
     xy = bbox_pt1_f1
     wid = bbox_coords_f1[2] - bbox_coords_f1[0]
     hgt = bbox_coords_f1[3] - bbox_coords_f1[1]


     #frame = Image.open('/home/jovyan/frames/hbarvxzmkk_1.png')
     frame = Image.open('/home/jovyan/frames/lonxgrulum_1.png')
     frame = frame.resize(size=[s // 2 for s in frame.size])

     # Display the image
     plt.imshow(frame)

     # Add the patch to the Axes
     plt.gca().add_patch(Rectangle(xy, wid, hgt,linewidth=1,edgecolor='r',facecolor='none'))
```

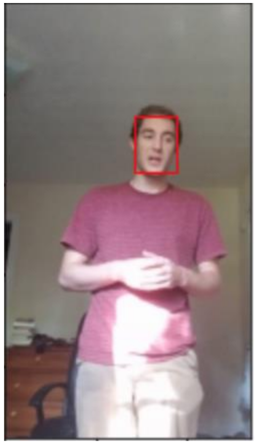


3.6.3 Generating Crops

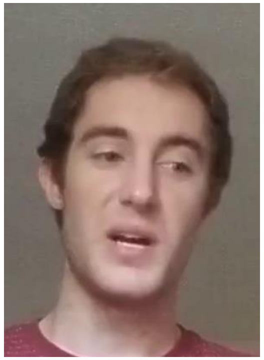
- Let’s observe the bounding box for the 1st frame of each video and the crops that are generated from them
- Crop has more pixels than bounding box:

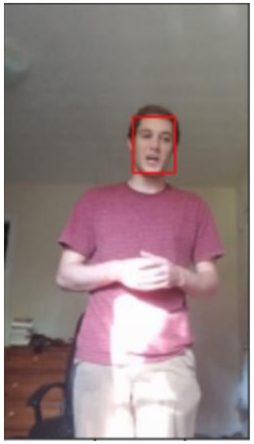


```
for bbox in bboxes:
    xmin, ymin, xmax, ymax = [int(b * 2) for b in bbox]
    w = xmax - xmin
    h = ymax - ymin
    p_h = h // 3
    p_w = w // 3
    crop = frame[max(ymin - p_h, 0):ymax + p_h, max(xmin - p_w, 0):xmax + p_w]
    h, w = crop.shape[:2]
    crops.append(crop)
img_dir = os.path.join(root_dir, crops_dir, id)
os.makedirs(img_dir, exist_ok=True)
for j, crop in enumerate(crops):
    cv2.imwrite(os.path.join(img_dir, "{}_{}.png".format(i, j)), crop)
```

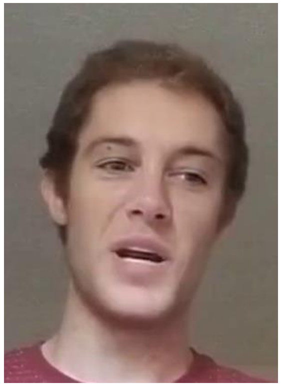


/hbarvxzmkk.mp4



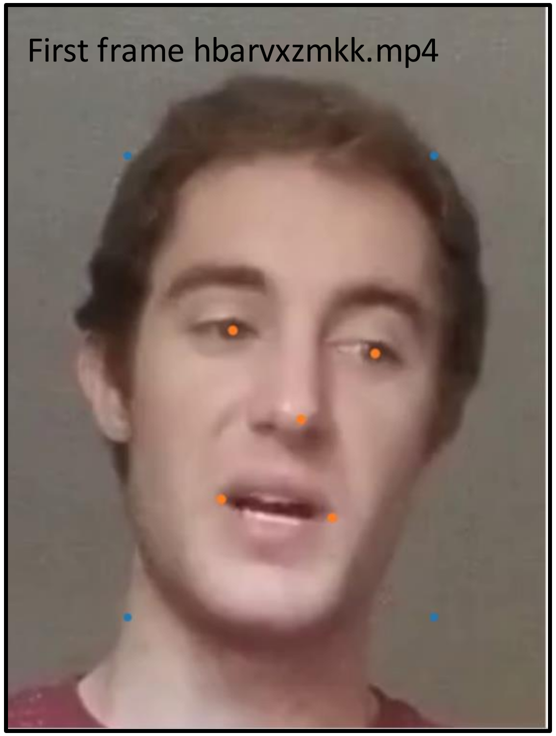


lonxgrulum.mp4

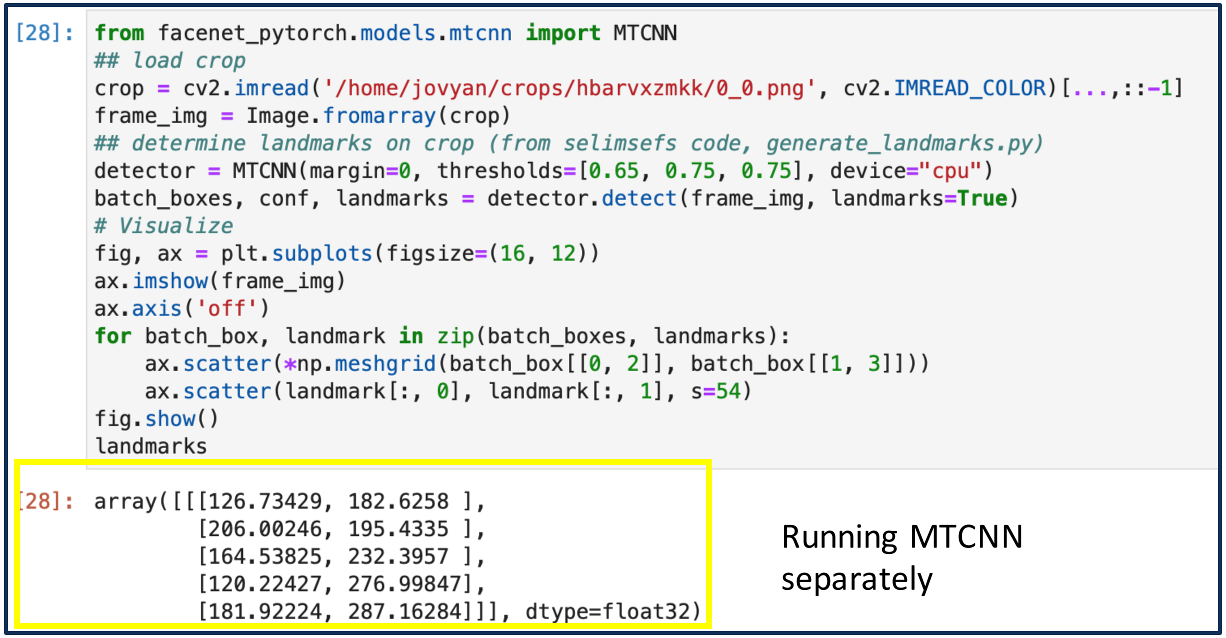


3.6.4 Landmarks – sanity check

Landmarks generated from MTCNN and selimsef match



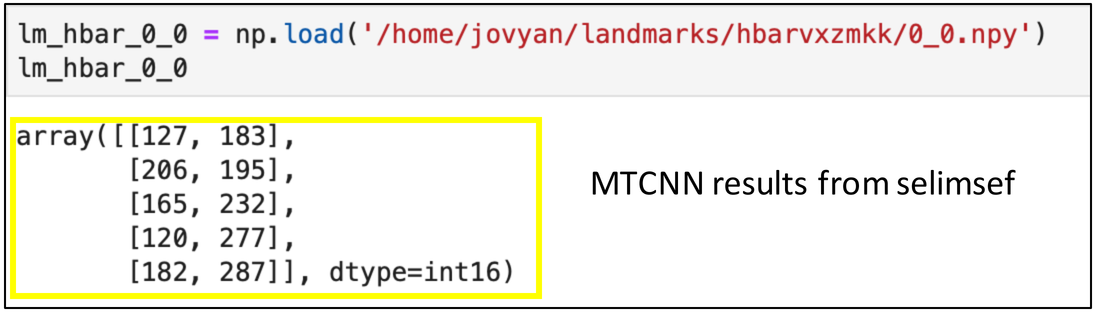
First frame hbarvxzmkk.mp4



```
[28]: from facenet_pytorch.models.mtcnn import MTCNN
      ## load crop
      crop = cv2.imread('/home/jovyan/crops/hbarvxzmkk/0_0.png', cv2.IMREAD_COLOR)[...,::-1]
      frame_img = Image.fromarray(crop)
      ## determine landmarks on crop (from selimsefs code, generate_landmarks.py)
      detector = MTCNN(margin=0, thresholds=[0.65, 0.75, 0.75], device="cpu")
      batch_boxes, conf, landmarks = detector.detect(frame_img, landmarks=True)
      # Visualize
      fig, ax = plt.subplots(figsize=(16, 12))
      ax.imshow(frame_img)
      ax.axis('off')
      for batch_box, landmark in zip(batch_boxes, landmarks):
          ax.scatter(*np.meshgrid(batch_box[[0, 2]], batch_box[[1, 3]]))
          ax.scatter(landmark[:, 0], landmark[:, 1], s=54)
      fig.show()
      landmarks
```

```
[28]: array([[[126.73429, 182.6258 ],
              [206.00246, 195.4335 ],
              [164.53825, 232.3957 ],
              [120.22427, 276.99847],
              [181.92224, 287.16284]]], dtype=float32)
```

Running MTCNN separately



```
lm_hbar_0_0 = np.load('/home/jovyan/landmarks/hbarvxzmkk/0_0.npy')
lm_hbar_0_0
```

```
array([[127, 183],
       [206, 195],
       [165, 232],
       [120, 277],
       [182, 287]], dtype=int16)
```

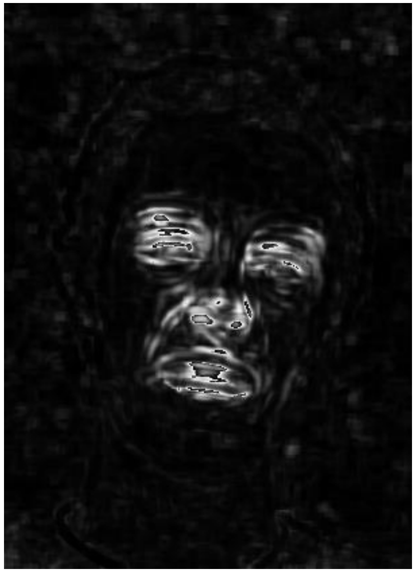
MTCNN results from selimsef

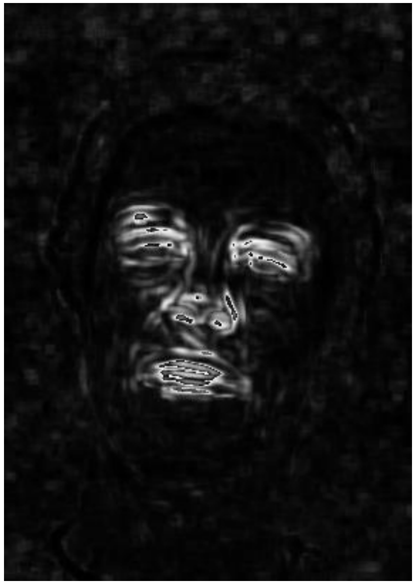
Some code from: https://www.kaggle.com/code/timesler/guide-to-mtcnn-in-facenet-pytorch

3.6.5 Structural Similarity Masks

Shown are

- (left) kvfkkcctax/0_0_diff.png
- (right) kvfkkcctax/10_0_diff.png

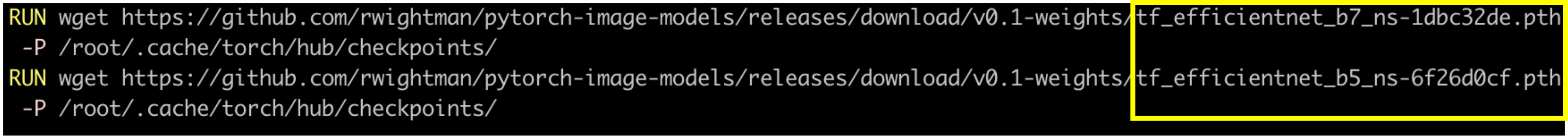




4. Model Training

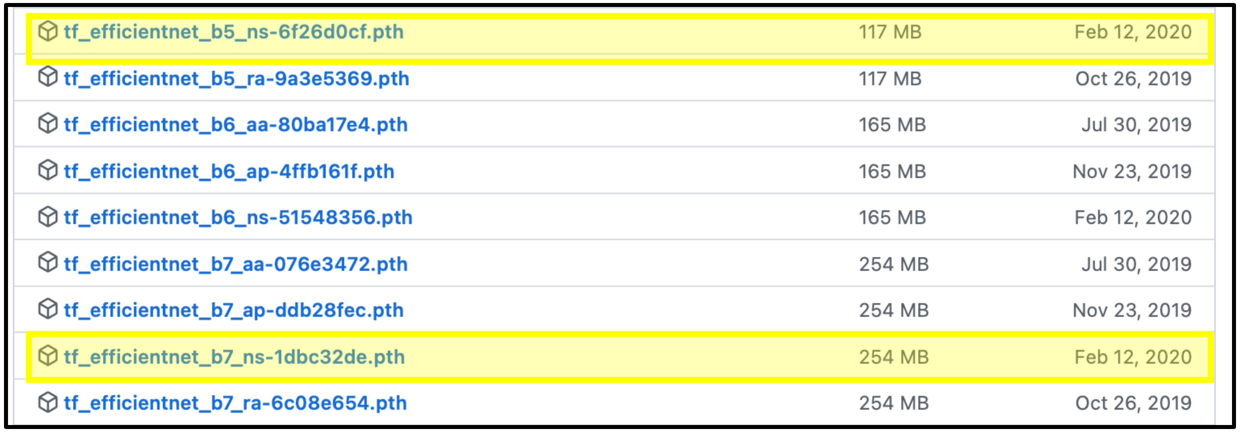
Which models are used by @selimsef? .. Look in the Dockerfile

https://github.com/rwightman/pytorch-image-models



```
RUN wget https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientnet_b7_ns-1dbc32de.pth
 -P /root/.cache/torch/hub/checkpoints/
RUN wget https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientnet_b5_ns-6f26d0cf.pth
 -P /root/.cache/torch/hub/checkpoints/
```

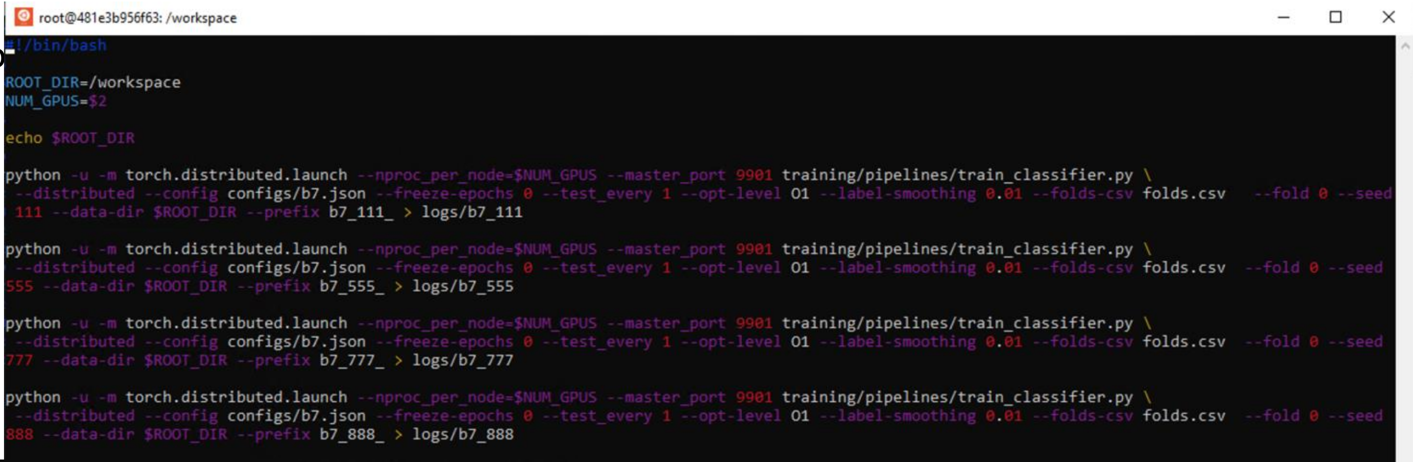
- There are many, MANY models to choose from in this repo, many of which are current as of July 2022,
- Rerunning with updated models would be a good idea



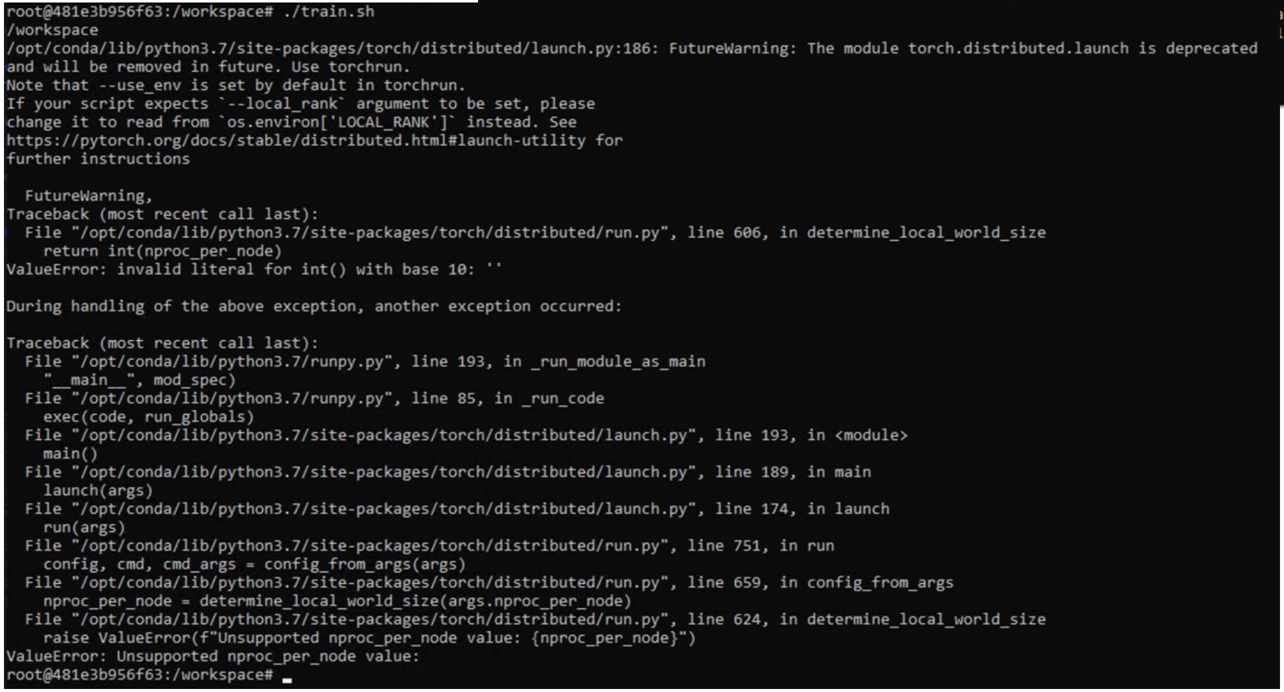
tf_efficientnet_b5_ns-6f26d0cf.pth	117 MB	Feb 12, 2020
tf_efficientnet_b5_ra-9a3e5369.pth	117 MB	Oct 26, 2019
tf_efficientnet_b6_aa-80ba17e4.pth	165 MB	Jul 30, 2019
tf_efficientnet_b6_ap-4ffb161f.pth	165 MB	Nov 23, 2019
tf_efficientnet_b6_ns-51548356.pth	165 MB	Feb 12, 2020
tf_efficientnet_b7_aa-076e3472.pth	254 MB	Jul 30, 2019
tf_efficientnet_b7_ap-ddb28fec.pth	254 MB	Nov 23, 2019
tf_efficientnet_b7_ns-1dbc32de.pth	254 MB	Feb 12, 2020
tf_efficientnet_b7_ra-6c08e654.pth	254 MB	Oct 26, 2019

4. Model Training

- Encountering some erro
- Debugging is underway



```
#!/bin/bash

ROOT_DIR=/workspace
NUM_GPUS=$2

echo $ROOT_DIR

python -u -m torch.distributed.launch --nproc_per_node=$NUM_GPUS --master_port 9901 training/pipelines/train_classifier.py \
 --distributed --config configs/b7.json --freeze-epochs 0 --test_every 1 --opt-level O1 --label-smoothing 0.01 --folds-csv folds.csv   --fold 0 --seed
 111 --data-dir $ROOT_DIR --prefix b7_111_ > logs/b7_111

python -u -m torch.distributed.launch --nproc_per_node=$NUM_GPUS --master_port 9901 training/pipelines/train_classifier.py \
 --distributed --config configs/b7.json --freeze-epochs 0 --test_every 1 --opt-level O1 --label-smoothing 0.01 --folds-csv folds.csv  --fold 0 --seed
555 --data-dir $ROOT_DIR --prefix b7_555_ > logs/b7_555

python -u -m torch.distributed.launch --nproc_per_node=$NUM_GPUS --master_port 9901 training/pipelines/train_classifier.py \
 --distributed --config configs/b7.json --freeze-epochs 0 --test_every 1 --opt-level O1 --label-smoothing 0.01 --folds-csv folds.csv  --fold 0 --seed
777 --data-dir $ROOT_DIR --prefix b7_777_ > logs/b7_777

python -u -m torch.distributed.launch --nproc_per_node=$NUM_GPUS --master_port 9901 training/pipelines/train_classifier.py \
 --distributed --config configs/b7.json --freeze-epochs 0 --test_every 1 --opt-level O1 --label-smoothing 0.01 --folds-csv folds.csv  --fold 0 --seed
888 --data-dir $ROOT_DIR --prefix b7_888_ > logs/b7_888

                                                                          ining/pipelines/train_classifier.py \
                                                                          l --label-smoothing 0.01 --folds-csv folds.csv  --fold 0 --seed

                                                                                                                   1,1           All
```



```
root@481e3b956f63:/workspace# ./train.sh
/workspace
/opt/conda/lib/python3.7/site-packages/torch/distributed/launch.py:186: FutureWarning: The module torch.distributed.launch is deprecated
and will be removed in future. Use torchrun.
Note that --use_env is set by default in torchrun.
If your script expects `--local_rank` argument to be set, please
change it to read from `os.environ['LOCAL_RANK']` instead. See
https://pytorch.org/docs/stable/distributed.html#launch-utility for
further instructions

  FutureWarning,
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/run.py", line 606, in determine_local_world_size
    return int(nproc_per_node)
ValueError: invalid literal for int() with base 10: ''

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/opt/conda/lib/python3.7/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/launch.py", line 193, in <module>
    main()
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/launch.py", line 189, in main
    launch(args)
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/launch.py", line 174, in launch
    run(args)
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/run.py", line 751, in run
    config, cmd, cmd_args = config_from_args(args)
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/run.py", line 659, in config_from_args
    nproc_per_node = determine_local_world_size(args.nproc_per_node)
  File "/opt/conda/lib/python3.7/site-packages/torch/distributed/run.py", line 624, in determine_local_world_size
    raise ValueError(f"Unsupported nproc_per_node value: {nproc_per_node}")
ValueError: Unsupported nproc_per_node value:
root@481e3b956f63:/workspace# _
```

4. EfficientNet Architecture

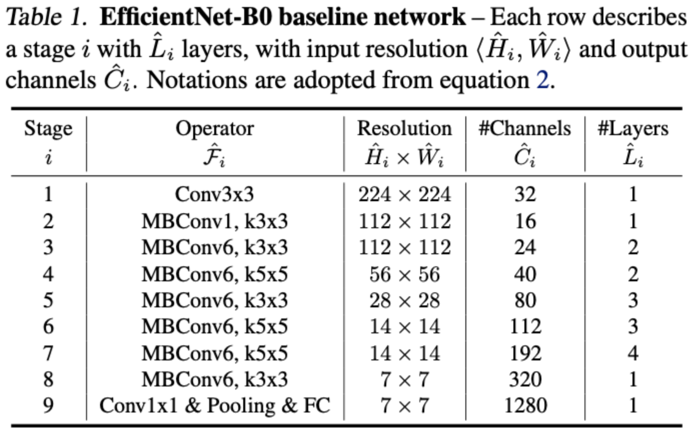
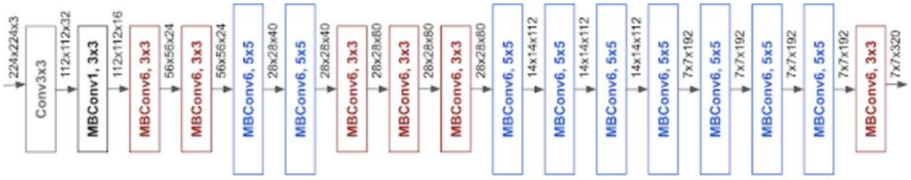


Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

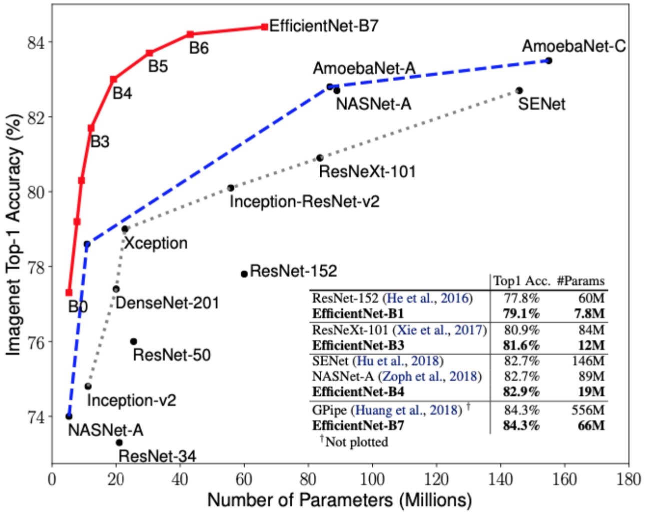
https://arxiv.org/pdf/1905.11946.pdf



The architecture for our baseline network EfficientNet-B0 is simple and clean, making it easier to scale and generalize.

https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

- First published in 2019 by Tan & Le, researchers at Google Research's Brain Team
- Up until this point, ConvNet's were scaled in ad-hoc ways
- EfficientNet is a ConvNet architecture that allows for proportional scaling of all three main NN dimensions: width of NN, depth of NN, image resolut

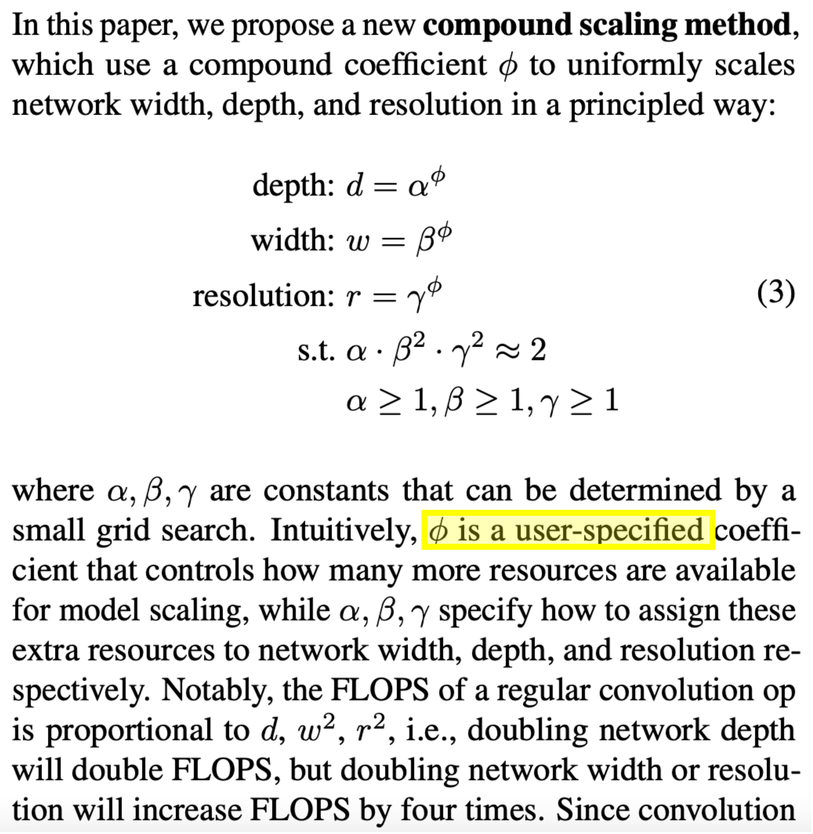


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

https://arxiv.org/pdf/1905.11946.pdf

4. EfficientNet Scaling Method



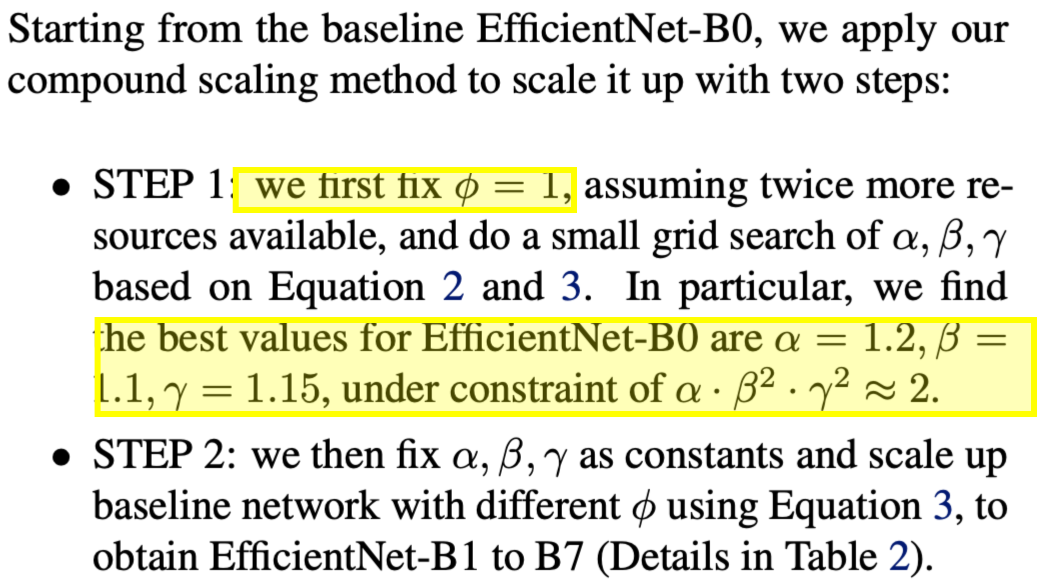
In this paper, we propose a new **compound scaling method**, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way:

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

where α, β, γ are constants that can be determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively. Notably, the FLOPS of a regular convolution op is proportional to d, w^2, r^2, i.e., doubling network depth will double FLOPS, but doubling network width or resolution will increase FLOPS by four times. Since convolution

https://arxiv.org/pdf/1905.11946.pdf

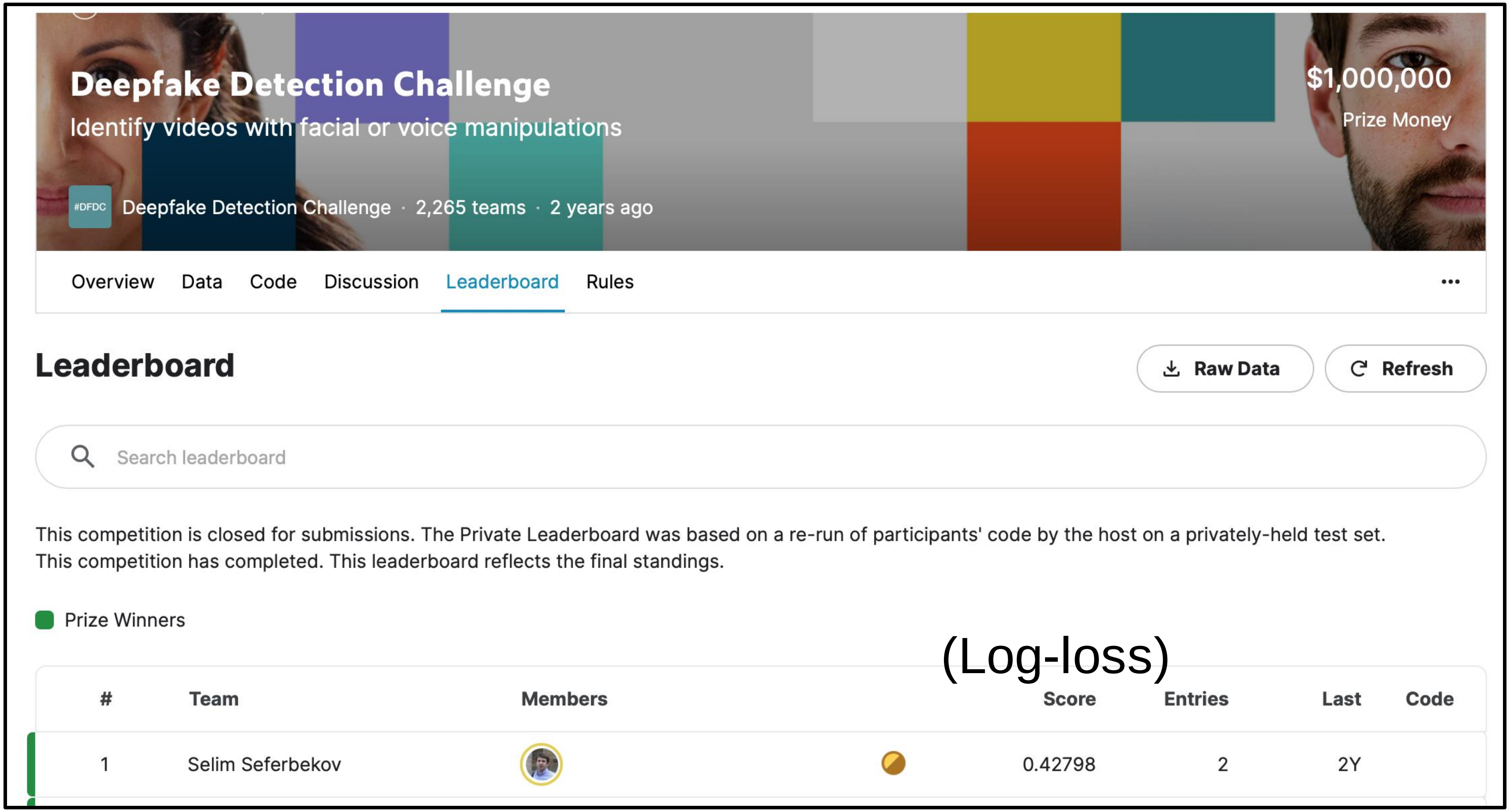
- What are B5 & B7? How do they relate to B0?



Starting from the baseline EfficientNet-B0, we apply our compound scaling method to scale it up with two steps:

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

4. Selimsef Public Results



Deepfake Detection Challenge
Identify videos with facial or voice manipulations
$1,000,000
Prize Money
#DFDC
Deepfake Detection Challenge · 2,265 teams · 2 years ago
Overview
Data
Code
Discussion
Leaderboard
Rules
•••
Raw Data
Refresh
Search leaderboard

Leaderboard

This competition is closed for submissions. The Private Leaderboard was based on a re-run of participants' code by the host on a privately-held test set. This competition has completed. This leaderboard reflects the final standings.

🟩 Prize Winners

(Log-loss)

#	Team	Members	Score	Entries	Last	Code
1	Selim Seferbekov		0.42798	2	2Y	

https://www.kaggle.com/competitions/deepfake-detection-challenge/leaderboard

Thank you!

For any code/questions please contact me at cabernaciak@cert.org