Assignment 4: Facial Manipulation Detection

Computer Vision — 0510-6251 — 2021-2022a

Due Date: See in Moodle

Chapter 1

Introduction

1.1 The Problem:

You are given two datasets of real and fake images:

- Deepfake detection dataset ("fakes_dataset")
- Synthetic image detection dataset ("synthetic_dataset")

Each dataset contains a set of real images and a set of fake (/synthetic) images. The real images are images of existing identities. Fake images are created by planting one face into the context of another. Synthetic images are created by GANs trained against a pristine set of images.

1.2 Your Goal

Your goal is to build detection models which, given an image, predicts if the image is real or fake.

1.3 General Outline

Through the exercise we will :

- Implement a dataset and a dataloader of face images.
- Implement a generic trainer which will help us train a vanilla deep neural network and a deep neural network based on the Xception backbone.
- We will succeed and fail (and finally succeed) training the aforementioned networks.
- We will analyze classification results using both:

  - Numerical and graphical metrics (ROC curves, AuC, accuracy, average precision...)
  - Image analysis tools such as Class Activation Maps and Image-Specific Class Saliency Visualisation (Section 3 in link).

We will use Pytorch heavily in this exercise. We recommend that you use the Pytorch documentation whenever needed.

1.4 The datasets

The Deepfake detection dataset contains fake images generated by taking a face of one subject and planting it in another subject's context. This manipulation is applied using this github repository. The Synthetic 'fake' dataset contains images generated by PGAN where the discriminator is trained against images from the CelebA-HQ dataset.

The datasets are supplied in in **Assigment4_datasets.zip** file attached to the exercise. Extract the files. Your directory structure should look like in Figure 1.1

1.5 File Structure

- **common.py** and **utils.py** : includes constants of repositories locations and utility functions to load models and datasets.
- **faces_dataset.py** : holds the faces dataset module which upon querying returns an image and a label.
- **show_faces_dataset.py** : plots samples of the two datasets to a figure.
- **train_main.py** : the main training script. Among other arguments, it receives the model name and dataset name we wish to train.
- **trainer.py** : an abstract training class. Takes the datasets and the model as inputs and trains the model with the train dataset. Evaluate and test the models performance with the corresponding datasets.
- **plot_accuracy_and_loss.py** : the trainer abstract class logs loss and accuracy to a json file. This script prints these metrics on graphs.
- **numerical_analysis.py** : loops through the test dataset and logs the scores each model computes, and prints ROC and DET curves and logs AuC performance score.
- **saliency_map.py** : compute the gradients of the correct class with respect to the image pixels to visualize the importance of each pixel.
- **grad_cam_analysis.py** : compute the class activation map of an image with respect to some layer of the model.

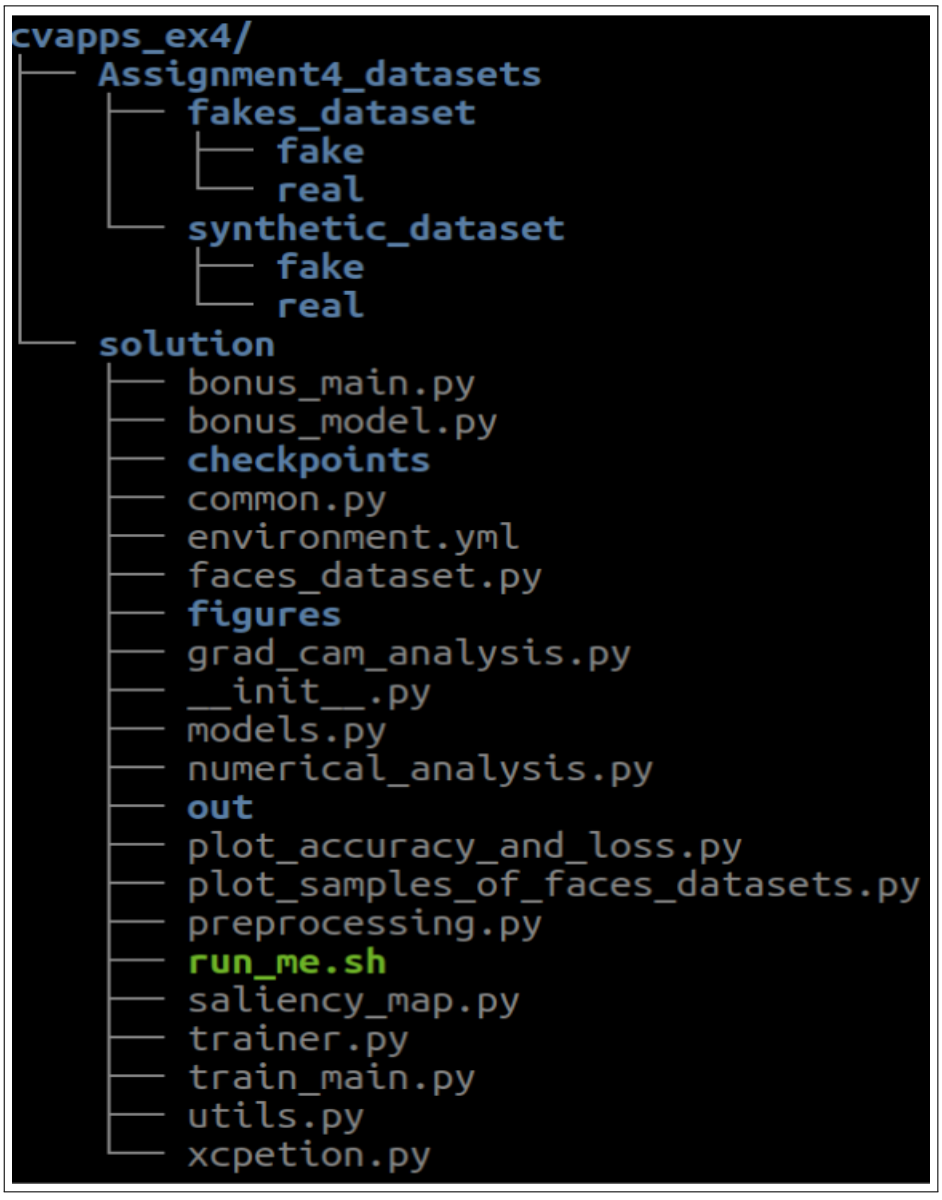

Figure 1.1: Folder hierarchy. Color code: Blue= folders, Gray = files , Green = executable bash script (consider it as a text file).

1.6 Downloading The Dataset

A link to the dataset is here. The link is restricted only to TAU students so make sure you're logged in with the Tel-Aviv University account. If you do not have a TAU account, contact the TA.

1.7 Exercise Submission

The exercise will be implemented in python 3.9.5 or higher. We recommend working with conda. We have supplied an environment.yml file so you can create the conda environment from this file. You're free to use an IDE of your choice (some people prefer Pycharm over others), but your code needs to run from the terminal. That is, we'll run:

```
./run_me.sh
```

If your system does not support bash scripts, copy and paste the python execution lines to your CLI.

This line trains all models, and runs all analysis scripts to print all graphs to the figures directory.

It is highly recommended to use Linux, and specifically Ubuntu from an LTS version. We will test your code on an Ubuntu 16.04 machine. Python packages in this assignment: pillow, numpy, scipy, random, matplotlib, time and opencv. No other additional libraries are allowed. The exercise has changed from past years and it is now a "fill your code here" exercise. That means, that in addition to following the exercise as it is specified in this document, you can use the function documentation to your assistance.

Tip: Read the entire outline of the exercise before you implement it. You will find it helpful to devise a concise solution.

Tip2: Before submitting the report make sure that you answered all questions. We've marked all questions with a gray boxes and all the scripts you need to run with a green boxes.

1.8 Frequently Asked Questions

What do I submit? Code + Report. Code should be submitted only in ".py" files. Figures and graphs should be attached to the report. All questions should be answered in the report. The report should be a ".pdf" file. The exercise is a "fill-your-code-here" kind of exercise, so no additional files are allowed. Go to Chapter 7 for all the details.

I cannot load/train the models on my PC: First, validate that it is not a bug (code issue). If you realize that you cannot load/train the models on your machine as a result of a resources issue (hardware) : use Google-Colab. You should do the following: Extract the images on your local machine, upload them

to your drive (use the university account, as it is not space limited). Upload the python files we supplied to the drive and arrange the folder hierarchy as in Figure 1.1, you may need to create empty folders etc. Finally, mount your drive to Google-Colab. Make the necessary changes in your code to be able to run it on Google-Colab. As for submitting the solutions see previous item.

I decided to use Google-Colab: Use Google-Colab smartly as use is restricted if you lock resources without using them / use them above budget. Before changing the Runtime Type to GPU, make sure that your code runs bug-free by running the train phase on 3-4 batches without a GPU. Close Colab tabs when you don't use them, avoid opting for a GPU when it is not needed for your work. If you use Google-Colab, make sure to read at least the FAQ page and concretely, the resource-limits section.

My python gets killed: Debug using linux's **dmesg**. If you get a Memory Error, you may decrease batch-size / move to an external framework such as Google-Colab. Decreasing the batch-size might not yield the expected results and can cause increase in runtime, but your grade will not be affected by this change. If you decrease the batch-size, write it explicitly in your report.

My code fails on a JSON load: Validate that your json is not corrupted by opening it by hand and use a small python script to load it to validate that it is loadable.

Chapter 2

Build Faces Dataset

2.1 Intro

During the entire exercise, we will work with a dataset which yields an image and a label each time we query it. A pytorch dataset has to implement (at least) three methods:

- **__init__** : the dataset initializer.
- **__getitem__** : a method which allows the syntax sugar of "[]". That is, getting an item from some object [a]: In our case:

```
(sample, label) = dataset[sample_index]
```

- **__len__** : a method which returns the number of samples in the dataset.

[a] When one calls: dataset[index] python actually runs: dataset.__getitem__(index). Actually, : type(dataset).__getitem__(dataset, index) is called, but it's irrelevant to the discussion.

The initialization of FacesDataset is given. It is initialized with two parameters:

- The path to the dataset.
- The transform we wish to apply to each image in the dataset.

def __len__(self)	
Input	**Description**
-	-
Output	**Description**
dataset_length	The size of the entire dataset. That is, the sum of the size of real and fake images.

Table 2.1: __len__ inputs(none) and outputs.

Question 1:

Implement two methods: **__getitem__** and **__len__** under **faces_dataset.py**.

```
def __len__(self) -> int:
```

```
def __getitem__(self, index) -> (torch.tensor, int):
```

See inputs and outputs definitions in Table 2.1 and Table 2.2.

Hint: For the getitem implementation, first load the image as a PIL image and then check if the transform is not None. If the transform is not None, apply the transform on the image and return the transformed image (and the label; we always return the label).

Use pytorch's Writing Custom Dataset as a reference.

Question 2:

Run the **plot_samples_of_faces_dataset.py** and add the result figure to your report.

def __getitem__(self, index)	
Input	**Description**
index	Index of the image.
Output	**Description**
(image, label)	a tuple containing an image and a label. Upon the transforms defined in **utils.py**, the image should be a Tensor image. The label is an integer: 0 for real images and 1 for fake / synthetic images..

Table 2.2: __getitem__ inputs and outputs.

Chapter 3

Write an Abstract Trainer

Intro

trainer.py is a module which holds two classes:

1. **LoggingParameters** : a data-class which holds information about the model and dataset we're training.
2. **Trainer** : an abstract trainer class. This class trains the model on the train dataset, evaluates the model on the validation and test sets and logs results to a json file.

The **LoggingParameters** data-class and the Trainer's logging are implemented for you.

Your goal is to implement the train and evaluation methods of the abstract trainer.

The models the **Trainer** should train all have the same output size: 2. That means that for every image, the model outputs two scores corresponding to the probability that the image belongs to the two optional classes: real & fake/synthetic (not-real). See Figure 3.2.

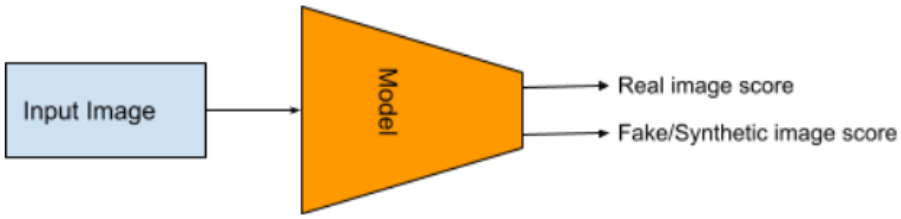

Figure 3.1: For each input image, the model outputs two scores: score for how real the image is and how Fake / Synthetic the image is. The classification is done by taking the *argmax* of the two scores.

3.1 Implement an Abstract Trainer

Question 3:

Write a function that trains a model for a single epoch under: **trainer.py:Trainer**.
Use the following API:

```
def train_one_epoch(self) -> tuple[float, float]:
```

See Table 3.1 for inputs and outputs.
The method should include a loop over all batches of the data-loader. For each batch:

1. zero the gradients
2. compute a forward pass
3. compute the loss w.r.t to the criterion
4. compute a backward pass
5. step optimizer
6. update the average loss and accuracy

Use the print line to keep track of the training procedure.
Use Pytorch's "Optimizing Model Parameters" tutorial as a reference.

def train_one_epoch(self)	
Input	**Description**
-	-
Output	**Description**
(avg_loss, accuracy)	A tuple containing the average loss over all samples, and accuracy percentage.

Table 3.1: **train_one_epoch** inputs(none) and outputs.

def evaluate_model_on_dataloader(self, dataset)	
Input	**Description**
dataset	torch.utils.Dataset object. This is the dataset we wish to evaluate (it will probably be the validation / test datasets).
Output	**Description**
(avg_loss, accuracy)	A tuple containing the average loss over all samples, and accuracy percentage of labeling the samples in the dataset.

Table 3.2: **evaluate_model_on_dataloader** inputs and outputs.

Question 4:

Write a function that evaluates the model on a given dataset under: **trainer.py:Trainer**.
Use the following API:

```
def evaluate_model_on_dataloader(self, dataset) -> tuple[
                                    float, float]:
```

See Table 3.2 for inputs and outputs.
The method should include a loop over all batches of the data-loader. For each batch:

1. compute a forward pass under a torch.no_grad() context manager.
2. compute the loss w.r.t to the criterion
3. update the average loss and accuracy

Note that the evaluation method is called in the validation and test methods.

3.2 Train a Deepfake Detection Classifier

Question 5:

Run the main training script: **train_main.py** to train the **SimpleNet** architecture on the Deepfakes dataset. Use a learning rate of $1e-3$, batch size: 32, 5 epochs and the Adam optimizer.
That is, run:

```
python train_main.py -d fakes_dataset -m SimpleNet --lr
0.001 -b 32 -e 5 -o Adam
```

3.3 Analyze The Deepfake Detection Classifier

Question 6:

Use your IDE (or even notepad) to open the json created in the *out* directory as a result of running the script.
Does it make sense?

Question 7:

Run the plot_accuracy_and_loss.py script to visualize the data held in the json.
Add the figures created to your report.
You should run:

```
python plot_accuracy_and_loss.py -m SimpleNet -j out-
/fakes_dataset_SimpleNet_Adam.json -d fakes_dataset
```

Question 8:

What is the test accuracy corresponding to the highest validation accuracy you received?

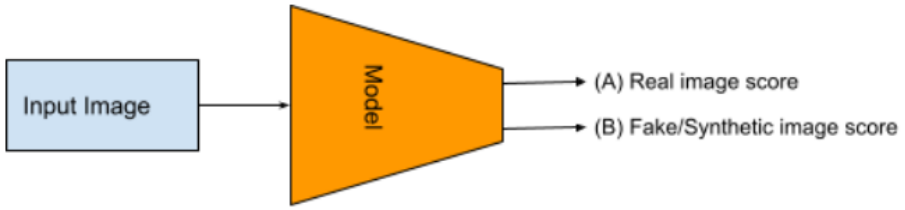

Figure 3.2: For each input image, the model outputs two scores: The first score is tells you how real the image is. The second score tells you how Fake / Synthetic the image is.

Question 9:

What is the proportion of fake images to real images in the test set?

We would like to evaluate the performance of the classifier we trained when examining tradeoffs between false positive and false negatives rates. There are two graphs we can generate to examine this: ROC [1] curve and DET [2] curve. To generate these graphs, one needs to create two iterables: the first, containing the ground truth labels and the second containing soft scores.

We will examine these curves for two soft scores groups:

A. soft scores obtained from the first output of the network corresponding to the "real" class score.

B. soft scores obtained from the second output of the network corresponding to the "Fake/Synthetic" (not-real) class score.

Question 10:

Fill in the missing code in **numerical_analysis.py** to generate ROC and DET graphs for SimpleNet trained on the Deepfakes dataset for the three types of scores. Add the graphs to your report.

Question 11:

Why do the graphs for the scores in (A) show completely different results from the graphs for the scores in (B)?

[1] Receiver Operating Characteristic
[2] Detection Error Tradeoff

3.4 Train a Synthetic Image Detection Classifier

Question 12:

Run the main training script: **train_main.py** to train **SimpleNet** on the Synthetic faces dataset. Use the same training parameters as before. Add the figures created to your report.
You should run:

```
python train_main.py -d synthetic_dataset -m SimpleNet
--lr 0.001 -b 32 -e 5 -o Adam
```

3.5 Analyze the Synthetic Image Detection Classifier

Question 13:

Run the **plot_accuracy_and_loss.py** script to visualize the data held in the json.
Add the figures created to your report.
You should run:

```
python plot_accuracy_and_loss.py -m SimpleNet -
j out/synthetic_dataset_SimpleNet_Adam.json -d syn-
thetic_dataset
```

Question 14:

What is the test accuracy corresponding to the highest validation accuracy you received?

Question 15:

What is the proportion of synthetic images to real images in the test set?

Question 16:

What kind of classifier did we get?

Question 17:

Looking at samples from the Synthetic Images dataset and the Deepfake dataset, does this result make sense?

Let's try to do better...

Chapter 4

Fine Tuning a Pre-trained Model

4.1 Intro

We would like to obtain better performance measures on the Synthetic Images dataset. To do that, we will use a different model.

We will take a pre-trained Xception backbone and change its fully-connected head to a Multi-Layered-Perceptron (MLP). Then, we will fine tune it on the task of Synthetic Image Detection.

We take the implementation of Xception as in this github repository. It is already copied to the **xception.py** supplied to you in the exercise files. Note that we renamed the last function in this file, to comply with our exercise logic.

Read the code & documentation and answer the following questions:

Question 18

What is Xception pre-trained on?

Question 19

What are the basic building blocks of Xception?

Question 20

What is Xception pre-trained on?

Question 21

What is the input feature dimension to the final classification block "**fc**" ?

Question 22

What is the number of parameters the Xception network holds by default? That is, without architectural change and default parameters. Look for the answer both in the Xception paper and use the method

```
get_nof_params
```

in **utils.py**.

4.2 Attaching a new Head to the Xception backbone

We would like to adjust Xception's head to our classification problem. Specifically, we would like to override Xception's "**fc**" block with the following MLP architecture:

Fully-Connected 2048x1000 → ReLu → Fully-Connected 1000x256 → ReLU → Fully-Connected 256x64 → ReLU → Fully-Connected 64x2

def get_xception_based_model()	
Input	**Description**
-	-
Output	**Description**
custom_network	Pre-trained Xception network with the new MLP head.

Table 4.1: **get_xception_based_model** inputs(none) and outputs.

Question 23:

Implement a function which returns the Xception backbone with the MLP head stated above.
Use the

```
get_xception_based_model
```

method under **models.py** to:

A. Build a pre-trained Xception backbone using "**build_xception_backbone**" from **xception.py**.

B. Override the network's "**fc**" block with the aforementioned MLP architecture.

Use the following API:

```
def get_xception_based_model() -> torch.nn.Module
```

See Table 4.1 for inputs and outputs.

Hint: As a debugging tool, use the method "**get_nof_params**" in the **utils.py** module to calculate the number of parameters in your model. The full Xception with the MLP head should include 23128786 parameters.

Question 24:

How many parameters did we add with the MLP on top of the original Xception's parameters count?

4.3 Train and evaluate a new architecture

Question 25:

Train the Xception-Based network we created on the Synthetic Faces dataset. Use the same learning rate, batch size and optimizer as before, but train for 2 epochs instead of 5.
You should run:

```
python train_main.py -d synthetic_dataset -m Xception-Based --lr 0.001 -b 32 -e 2 -o Adam
```

Question 26:

Run the **plot_accuracy_and_loss.py** script to visualize the data held in the json. Add the figures created to your report.
You should run:

```
python plot_accuracy_and_loss.py -m XceptionBased -j out/synthetic_dataset_XceptionBased_Adam.json -d synthetic_dataset
```

Question 27:

What is the test accuracy corresponding to the highest validation accuracy you received?

Question 28:

Run the **numerical_analysis.py** script for the Xception-Based network trained on the Synthetic Faces dataset.

4.4 Rhetorical questions

Why did we get better results than 3.4? The answer "the model has more parameters" is obscure. In what way adding more parameters helps? Let's try to answer this question with the following analysis tools.

Chapter 5

Saliency Maps and Grad-CAM analysis

5.1 Intro

In this part we want to develop tools to understand what the classifiers were trained to "see". We will use two tools: First we will ask what is the contribution of each pixel to the true class soft score, and answer this question with Saliency Maps. Second we will be Class Activation Maps, specifically Grad-CAMs.

Question 29:

> Read about Saliency Maps here and explain shortly, in your own words, what Image-Specific Class Saliency Visualisation are.

An additional source of information is here (Vanilla Gradient). Feel free to search for other sources of information.

Question 30:

> Read about Grad-CAMs and explain shortly, in your own words what Grad-CAMs are.
> Resources: Grad-CAM paper, CAM project.

def compute_gradient_saliency_maps(samples, true_labels, model)	
Input	**Description**
samples	A torch.tensor of shape: $B \times 3 \times H \times W$ of B images.
true_labels	A torch.tensor of B labels.
model	A nn.Module model.
Output	**Description**
saliency	A torch.tensor of saliency maps for each image. The tensor shape is: $B \times H \times W$.

Table 5.1: **compute_gradient_saliency_maps** inputs and outputs.

5.2 Saliency Maps

Question 31:

Implement a method which takes a model, and samples and their true label as inputs, and outputs saliency maps for each sample.
Fill in the missing code in:

```
def compute_gradient_saliency_maps(samples: torch.tensor,
                                   true_labels: torch.tensor,
                                   model: nn.Module) -> torch.
                                   tensor:
```

which is under: **saliency_maps.py**.
See Table 5.1 for inputs and outputs details.

Question 32:

Run the **saliency_map.py** script to generate saliency maps for **SimpleNet** over the Deepfakes dataset and **XceptionBased** network over the Synthetic faces dataset.
Add the figures to the report and explain the results.
You should run:

```
python saliency_map.py -m SimpleNet -cpp checkpoints/fakes_dataset_SimpleNet_Adam.pt -d fakes_dataset
```

and:

```
python saliency_map.py -m XceptionBased -cpp checkpoints/synthetic_dataset_XceptionBased_Adam.pt -d synthetic_dataset
```

5.3 Grad-CAM

To generate Grad-CAM images, we will use an external repository from Github.

Question 33:

Install the grad-cam python package.
Use the installation line from this Github repo.

def get_grad_cam_visualization(test_dataset, model):	
Input	**Description**
test_dataset	torch.utils.Dataset object. This is the dataset we wish to sample from.
model	A nn.Module model.
Output	**Description**
(visualization, true_label)	A tuple containing the visualization and the true label of the image. The visualization is a numpy.ndarray of size $256 \times 256 \times 3$. The true_label is a tensor of shape: torch.Size([1]). .

Table 5.2: **compute_gradient_saliency_maps** inputs and outputs.

Question 34:

Add minimal changes to the code snippet here to write a method which:

a. Takes a dataset and a model as input

b. Samples a single image from the dataset (use DataLoader with batch_size=1 and shuffle=True).

c. Computes a Grad-CAM for that image for the target layer: model.conv3

d. The method should return the visualization (see code snippet) and the true label.

Use the following API:

```
def get_grad_cam_visualization(test_dataset, model) -> tuple[
                                np.ndarray, torch.tensor]
```

To return the visualization of the Grad-CAM on the image and the image label.
Use Table 5.2 for inputs and outputs details.

Question 35:

Run the **grad_cam_analysis.py** script to generate Grad-CAM visualizations for:

1. SimpleNet trained on Deepfakes
2. SimpleNet trained on Synthetic Faces
3. Xception-Based trained on Synthetic faces

You should run:

```
python grad_cam_analysis.py -m SimpleNet -cpp checkpoints/fakes_dataset_SimpleNet_Adam.pt -d fakes_dataset
```

and:

```
python grad_cam_analysis.py -m SimpleNet -cpp checkpoints/synthetic_dataset_SimpleNet_Adam.pt -d synthetic_dataset
```

and:

```
python grad_cam_analysis.py -m XceptionBased -cpp checkpoints/synthetic_dataset_XceptionBased_Adam.pt -d synthetic_dataset
```

Chapter 6

Bonus Part

In this exercise, we have demonstrated to you the impact of using a pre-trained network and fine-tuning it over a different task. We would like to stress that sometimes, we need to tradeoff between the accuracy and model size. The reason is that the number of parameters is usually a bottleneck when it comes to real life solutions (think about a low resource edge device).

We would like you to suggest a model that achieves a good tradeoff between test-set AuC score on the Deepfakes dataset and the number model parameters. You're exposed only to the train and validation sets of the Deepfakes dataset. You'll be tested on the test set, but you cannot use it from training / hyper-parameter validation.

We will rank all models submitted this semester with a secret rank that will produce a single score which is calculated over your models' AuC and number of parameters:

$score = secret(AuC, nofParameters)$

Your model architecture should be contained in a single file bonus_model.py. You should be able to load and return it from the

```
def my_bonus_model()
```

method (the model should be saved to "**checkpoints/bonus.pt**"). **bonus_main.py** shows you how your model will be evaluated (apart from the secret sauce (-;). Explain how you designed and trained the architecture, attach the code.

Good luck and Enjoy !

Chapter 7

Submission instructions:

- A ".pdf" document containing reference to all sections of the exercise must be submitted, showing all the results and answering all questions (no code is required in the document).
- All API-defined python functions of the assignment must be submitted, as well as any associated functions that you wrote. The functions will be automatically run and tested.
- Check that you are able to run all lines in **run_me.sh** without making any changes to it.
- The solution with all relevant files must be submitted in the submission box in the Moodle within a zip file named:

  ```
  assignment4_ID1_<id_1>_ID2_<id_2>.zip
  ```

  If the zip file exceeds 50MB, you may email it to: *amirjevn@mail.tau.ac.il* (share it with me in Google Drive).
- Late submission will result in grade reduction.