TEL AVIV UNIVERSITY

Computer Vision

2022-2023

Assignment #4

Project

Students

ID1: 301613501

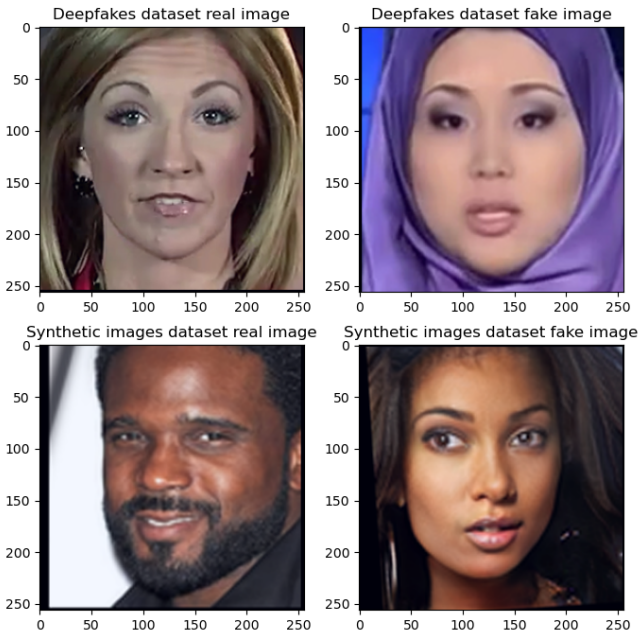
ID2: 305207946

Question 1:

See implementation of **__getitem__** and **__len__** in **faces_dataset.py**

Question 2:

Result figure of **plot_samples_of_faces_dataset.py**



Question 3:

See implementation of **train_one_epoch** in **trainer.py**

Question 4:

See implementation of **evaluate_model_on_dataloader** in **trainer.py**

Question 5:

We used the following cmd in the terminal:

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

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

Question 6:

Extracting the *.json file, we can see the following conclusions.

	Accuracy	Loss
Train	Monotonic Increased	Monotonic decreased
Validation	Partial decreasing and partial increasing	Start in decreasing and then increased
Test	Monotonic Increased	Totally increased (with fluctuations)

Since we are training the network on the Train Set of data, it is making sense that the performance will be best on this part of data, so the constant decrease of the loss and increasing for the accuracy are the results that we expected to see for the train set. On the other hand, we can see that the results on the validation set and test set are not stable. Moreover, the loss of the Test set per epoch was increased w.r.t. the previous one. A possible explanation of these results may be found in overfitting, where the resulting model describes random error or noise instead of the underlying relationship, so that the network learns patterns only relevant to the training set, and therefore might harm its ability to generalize. Indeed, the loss shows the trend of increase along epochs when evaluating the model for samples not in the Training set. Maybe if we ran more iterations, we would get better performance.

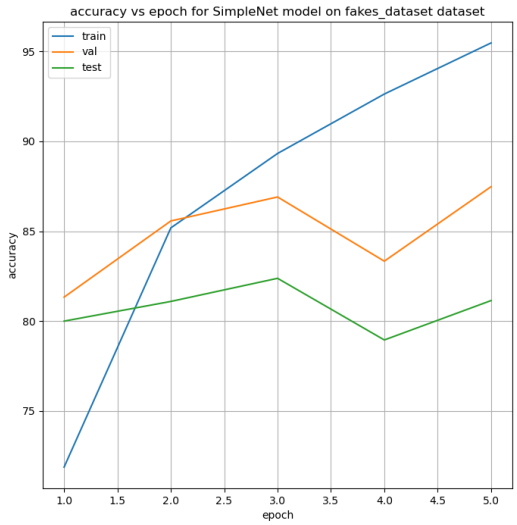
Question 7:

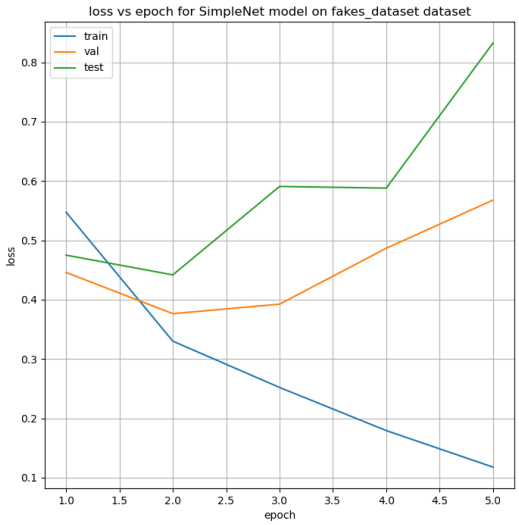
We used the following cmd in the terminal:

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

We got the 2 graphs below and it is possible to see that they show similar behavior as we described in the table above, as expected.

Accuracy **Loss**


accuracy vs epoch for SimpleNet model on fakes_dataset dataset
train
val
test
accuracy
epoch
95
90
85
80
75
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


loss vs epoch for SimpleNet model on fakes_dataset dataset
train
val
test
loss
epoch
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Question 8:

Test accuracy corresponding to the highest validation accuracy

	Accuracy	At Epoch Number
Highest Validation	87.5%	5
Corresponding Test	81.1%	5

Question 9:

Under the fakes_dataset\test we can see that we have 700 fake images and 1400 real images.
Therefore, the proportion of fake images to real images is 0.5. i.e., for every two real images in the set, there exists one fake image in the set.

Question 10:

Background

Describing the ROC

A Receiver Operating Characteristic (ROC) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. As we can see, the ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds.
We also learned that the true positive rate is also known as sensitivity, recall, or probability of detection in the machine learning field. Both of those rates are calculated by the following forms.

$$TPR = \frac{True\ Positives}{True\ Positives + False\ Negative}, \qquad FPR = \frac{False\ Positives}{True\ Negative + False\ Positives}$$

The area under the ROC curve (AUC) is a measure of the classifier's performance. A classifier with a higher AUC is considered to be better at distinguishing between positive and negative instances. It may be interpreted as the probability that the system ranks a random positive example more highly than a random negative example: the closer the AUC is to 1, the better the classifier is at distinguishing between the two classes. An AUC of 0.5 indicates that the classifier is no better than random guessing.

Describing the DET

Similar to ROC, the Detection Error Tradeoff (DET) graph is a graphical representation of the performance of a classifier, specifically for binary classification tasks. It plots the false positive rate on the x-axis and the false negative rate on the y-axis. The false positive rate is the fraction of negative examples that are incorrectly classified as positive, while the false negative rate is the fraction of positive examples that are incorrectly classified as negative. The DET curve is a useful tool for comparing the performance of different classifiers and for choosing the best operating point (i.e., the threshold at which to make a decision) for a given application.

True / False Positive and Negative

For easier understanding, let us show an auxiliary table that represent the meaning of false positive and true positive in our case.

	Actual Classes	
	1	**0**

Predicted Classes	1	True Positive	False Positive
	0	False Negative	True Negative

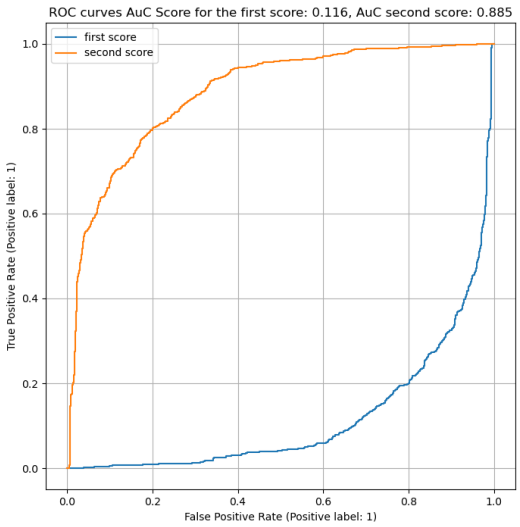
Implementation

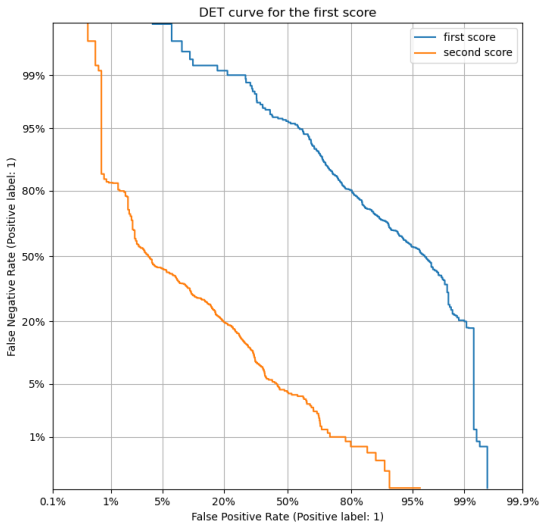
See implementation of **get_soft_scores_and_true_labels** in **numerical_analysis.py**

Running the following command plotted the ROC and DET curves.

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

ROC **DET**





It can clearly be observed that the model would assign a higher score to a random true-positive event (fake image labeled '1') than to a false-negative event (real image labeled 0) for the second score, while the exact opposite behavior is concluded for the first score. Meaning the model would assign a higher score to a random false-negative event than to a false-negative event. Due to the fact that the first score is the soft score obtained for how real the image is, the second score is the soft score obtained for how fake the image is.

Question 11:

First, we will remind that the notation in our project is (see **__getitem__** method):

- 0 → for real images
- 1 → for fake / synthetic images

In addition, as described in Figure 3.1 in the assignment document, the model returns 2 prediction values:

- 1st Score – how real the image is
- 2nd Score – how fake the image is

Such that those two scores expected to be different and even opposite as we can describe it the table below which represent the probability the model gives for images to be a real image or fake image.

	Probability to be Real	Probability to be Fake
1st Score	High	Low
2nd Score	Low	High

The reason that the graphs for the first score is completely different from the graphs for the second score is that each score represent the opposite case. It can also be seen by think of it as probabilities which the first score and the second score are pdf functions that describes the probability for how an image is a real or fake (respectively), such that the total area under the graph is sum to ~1 (see in the ROC graph title: AuC 1st + AuC 2nd = 0.116 + 0.885 = ~1). In the DET curve, the reason is similar – the opposite definitions of the parameters yield opposite results.

Question 12:

We used the following cmd in the terminal which doesn't plot any figures but plot the *.json and *.pt files that will be used and presented in the next sections.

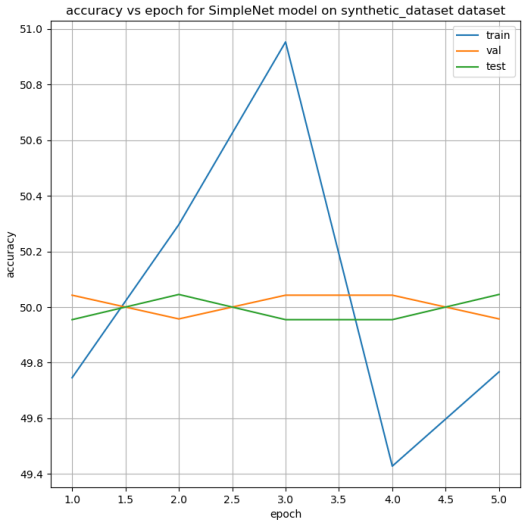
```
python train_main.py -d synthetic_dataset -m SimpleNet --lr 0.001 -b 32 -e 5 -o Adam
```

Question 13:

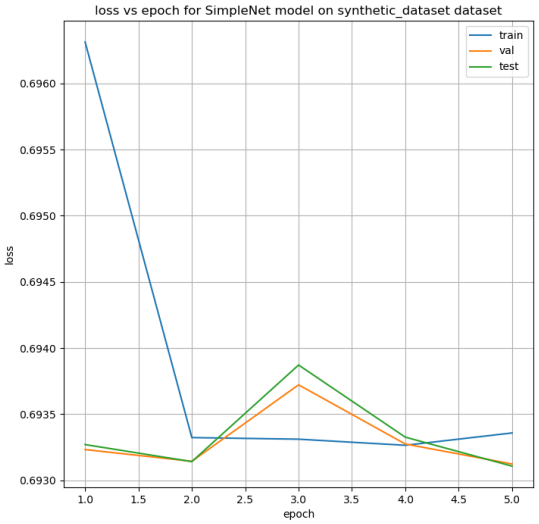
We used the following cmd in the terminal:

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

Accuracy



Loss



This time the accuracy is roughly 50% throughout all epochs for all sets. Same observation can be claimed for the loss Vs epoch for all sets – its trend is quite stable.

Question 14:

	Accuracy	At Epoch Number
Highest Validation	50.04%	3
Corresponding Test	49.95%	3

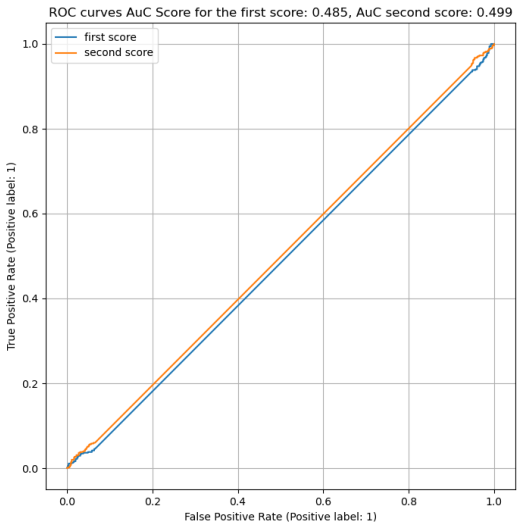
Question 15:

Under the synthetic_dataset\test we can see that we have 552 fake images and 551 real images.
Therefore, the proportion of fake images to real images is ~1.

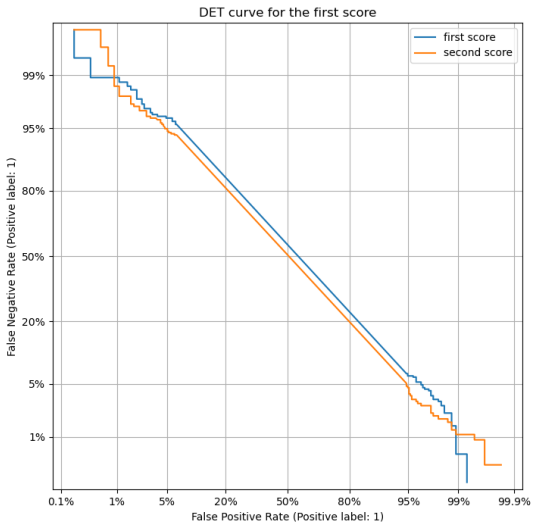
Question 16:

We got a terrible classifier model that can't separate negative classes from positive classes at all. In other words, this classifier has the same performance as a random classifier. We can emphasis this claim by looking on the ROC and DET curves of the new model which trained on the synthetic images and see that the probability to get a classification of fake image is almost equal to real image. Rather than, in the DET curves we can see that the curves too close to each other.

ROC

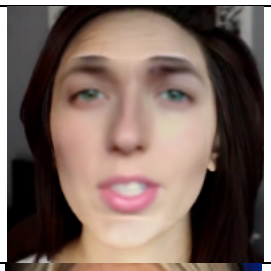
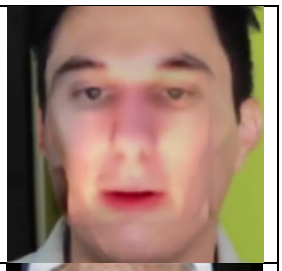


DET



Question 17:

In the following table we sampled deepfake and synthetic images from the given dataset. It is stands out that the synthetic faces are much more realistic then the deepfake faces. So, it is making sense that our classifier doesn't "know" to so separate synthetic and real images.

Deepfake			
Synthetic			

Question 18:

Following the paper (link), Xception is a deep convolutional neural network architecture that is pre-trained on the ImageNet dataset and JFT Google's internal dataset.

The ImageNet dataset is a large collection of images that is commonly used to train and evaluate image classification models. It contains more than 14 million images across more than 20,000 different object classes. While the JFT is larger dataset with over 350 million images.

Question 19:

The basic building blocks of Xception include:

1. **Depthwise Separable Convolutions:** Xception uses depthwise separable convolutions, which separates the convolutional operation into two parts: a depthwise convolution and a pointwise convolution. This allows for greater efficiency and reduces the number of parameters required.
2. **Inverted Residual Blocks:** Xception uses inverted residual blocks, which are a variation of residual blocks where the input and output are concatenated instead of added. This helps to reduce the number of parameters and improve the accuracy of the model.
3. **Skip Connections:** Xception also incorporates skip connections, which allow information to flow directly from one layer to another, bypassing intermediate layers. This helps to improve the flow of information throughout the model and improves the accuracy of the model.
4. **Global Average Pooling:** Xception also uses global average pooling, which replaces the traditional fully connected layers with a global average pooling layer. This helps to reduce the number of parameters and improve the accuracy of the model.
5. **Batch Normalization:** Xception also includes batch normalization, which helps to reduce the internal covariate shift and improve the training of the model.
6. **Activation function:** Xception uses ReLU as the activation function.

Question 20:

The same question as Q18 so the answers are identical ☺

Question 21:

First, in figure 5 from the paper, we can see that the input feature dimension to the final classification block is 2048-dimensional vectors.

Question 22:

In table 3 from the paper, we can say that the Xception holds over than 22 million parameters. The exact number represented from the paper in the table below.

Table 3. Size and training speed comparison.

	Parameter count	Steps/second
Inception V3	23,626,728	31
Xception	22,855,952	28

Implementing a short script to create the model and use the method represent exactly the same number of parameters:

```
The number of Xception params is: 22855952
```

See the implementation in **nof_params.py**.

Question 23:

See implementation of **get_xception_based_model** in **models.py**

Question 24:

Totally we added 272834 parameters $[23{,}128{,}786 - 22{,}855{,}952 = 272{,}834]$

```
The number of Xception params is:  23128786
```

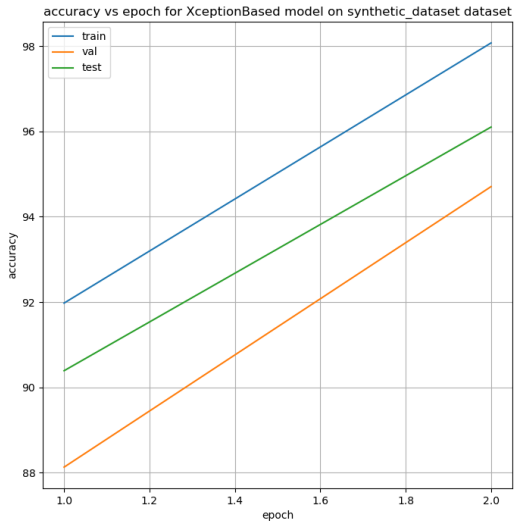
Question 25:

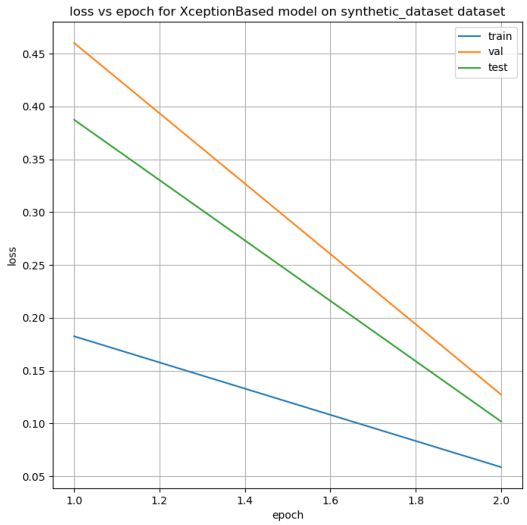
python train_main.py -d synthetic_dataset -m XceptionBased --lr 0.001 -b 32 -e2 -o Adam

Question 26:

python plot_accuracy_and_loss.py -m XceptionBased -j out/synthetic_dataset_ XceptionBased_Adam.json -d synthetic_dataset

Accuracy **Loss**





Contrary to the results we got in the previous sections, here it is noticeable that the accuracy and the loss are monotonic, increasing and decreasing (respectively), and manage to converge after 2 epochs. But we received unexpected phenomena that are manifested when the accuracy of the test set is higher than the accuracy of validation set. Moreover, we can the accuracy of the training set is too close to 100% which hint that our model is over fitting and will not be reliable for new, unseen data.

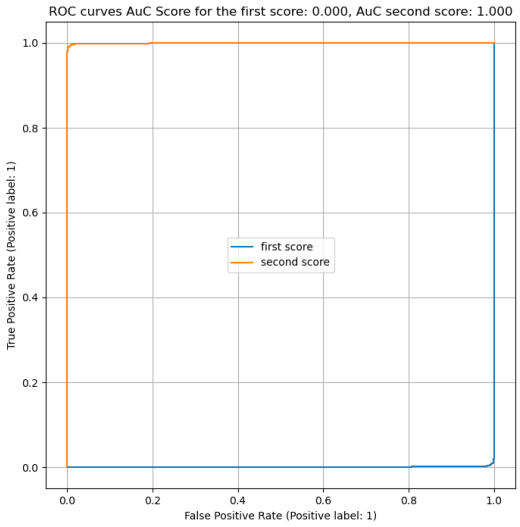
Question 27:

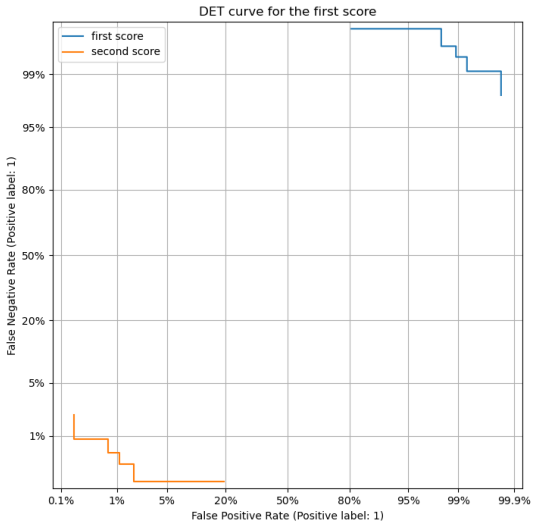
	Accuracy	At Epoch Number
Highest Validation	94.7%	2
Corresponding Test	96.1%	2

Question 28:

Both of the graphs, the ROC and the DET seems very close to perfect classifier. But since we've got hint that our model is overfitting, probably those ideal result that we see here will be relevant mostly on our data and not on new, unseen data.

ROC **DET**





Rhetorical Questions:

The first intuition about the reason we got a better result is that the XceptionBased model we used here is a model that already pre-trained and specifically designed for image classification tasks. Moreover, when we are looking on the results of the Saliency Maps below, in question 32 we can see how the pre-training and the specifically design are reflected on the features that the model is affected from. That is, we can see that the XceptionBased model is affected by much more pixels compared to the SimpleNet that affected by very specific areas in the pictures.

Adding more parameters to a neural network can improve the results because it allows the network to learn more complex and nuanced features from the data. Each parameter represents a degree of freedom in the model, and more parameters give the network more capacity to fit the data. This increased capacity allows the network to capture more subtle patterns and relationships in the data, which can lead to better performance on the classification task.

However, it is important to note that adding too many parameters can lead the model to overfitting, where the model becomes too specialized to the training data and performs poorly on new, unseen data. To avoid overfitting, it is necessary to use regularization techniques such as dropout as we will implement the competition section.

Question 29:

Image-specific class saliency visualization is a visualization technique used to highlight the regions of an image that are most important for a neural network's classification decision. The technique displays the contribution of each pixel of a particular input image to the network prediction. These maps are created by computing the gradient of the output of the neural network with respect to the input image. The resulting gradient is then overlaid on the original image, with the intensity of the overlay indicating the degree to which each pixel of the image contributed to the final classification. This helps to identify which areas of the image the network focused on when making its decision, and can be used to gain insight into how the network is processing the image.

Question 30:

Gradient-weighted Class Activation Maps (Grad-CAMs) are a technique used to visualize which parts of an image a convolutional neural network (CNN) is using to make its predictions. The technique works by using the gradients of the output of a particular layer of the CNN with respect to the input image to highlight the regions of the input image that have the greatest impact on the CNN's output. This allows researchers and engineers to gain a better understanding of how the CNN is making its predictions, and can be used to improve the interpretability of the model or to identify areas where the model is failing.

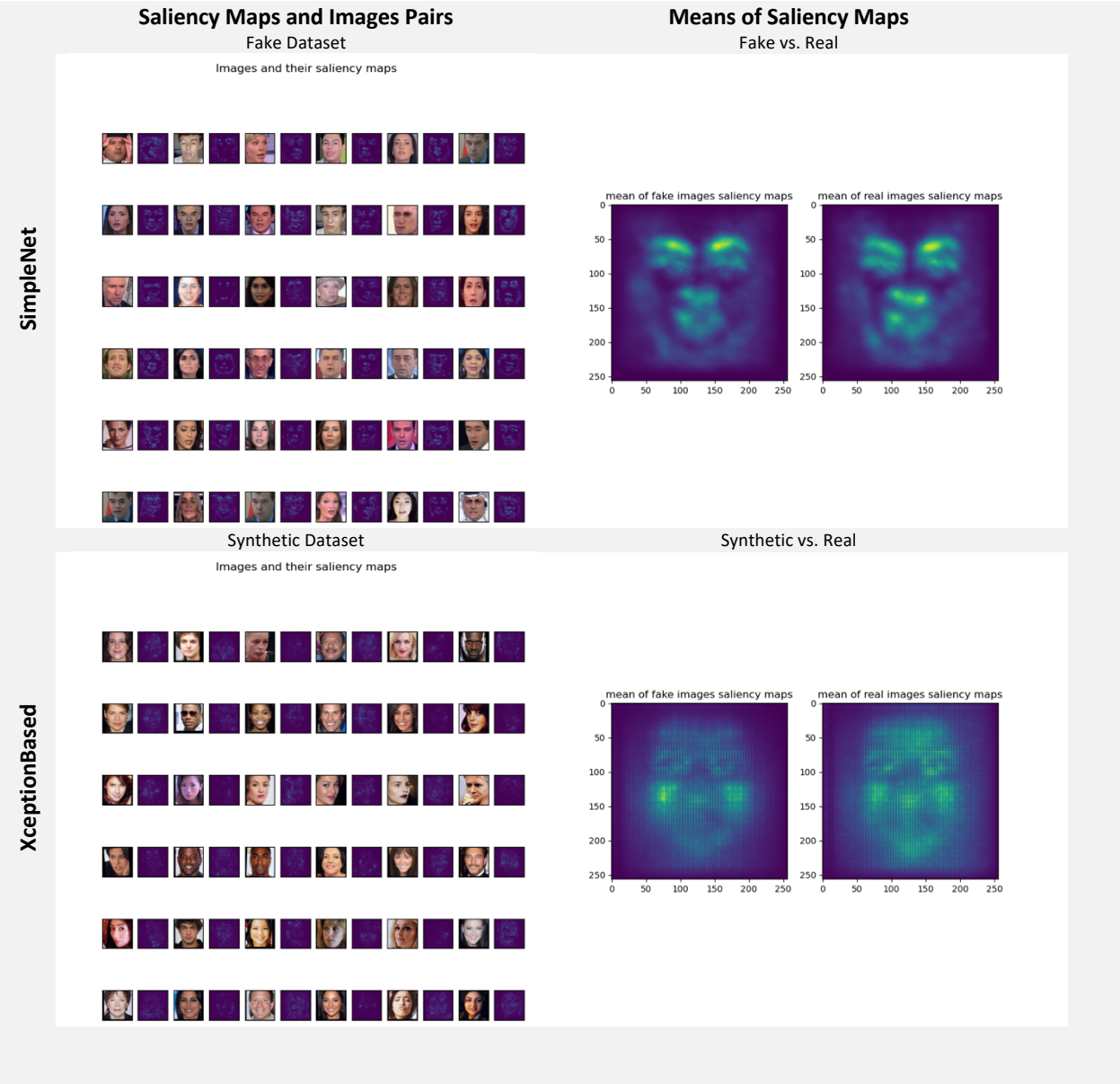
Question 31:

See implementation of **compute_gradient_saliency_maps** in **saliency_maps.py**

Question 32:

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

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



It is easier to explain the result when we look at the mean images. The ***SimpleNet*** which trained on the fake faces affected by specific areas of the human faces while the ***XceptionBased*** model affected by the whole face. We can also see that the ***SimpleNet*** mean map of real images is very similar to the map of fake images, while in the other case of ***XceptionBased*** model the difference is more significant. Which means that the model probably gives less false predictions. One more interesting conclusion from the result is that the ***XceptionBased*** network is strongly affected by the whole face and not just by small parts of it.

Question 33:

Installed by using the following *pip* command.

```
pip install grad-cam
```

Question 34:

See implementation of **get_grad_cam_visualization** in **grad_cam_analysis.py**

Question 35:

Running the following 3 commands generated the required ***Grad CAM*** images.

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

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

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

In the following table we represent the grad cam analysis on 3 different cases.
There are 3 major phenomena that correlate with the results that we've gotten previously in the work.

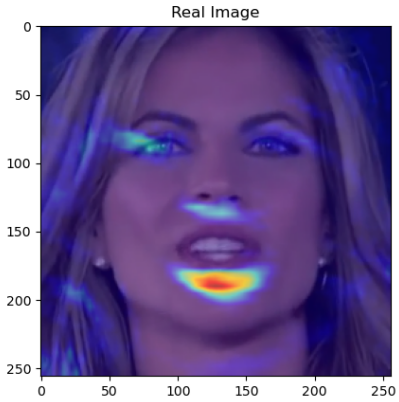
1. The SimpleNet model is impacted by only some facial features of the object in the image in order to determine its prediction, more than the others: for the real image it considers more the eyes area and lips more than the other facial features, while for the fake image the forehead and cheeks affect the classification more significantly. Which means that there are many elements and data that were not taken into account when making decision regarding their authenticity.

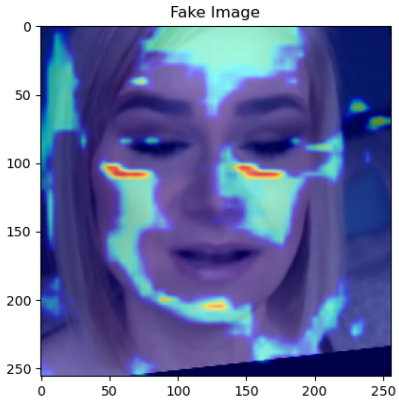
2. The SimpleNet cannot classify images as real or fake when it is based on synthetic dataset, as we already saw before. This is due to the fact that the intensity does not vary along the heatmap obtained for such configuration, which has the meaning of the performance of random classification.

3. The Xception-Based model is affected by almost by all the image in order to make the prediction of classification.

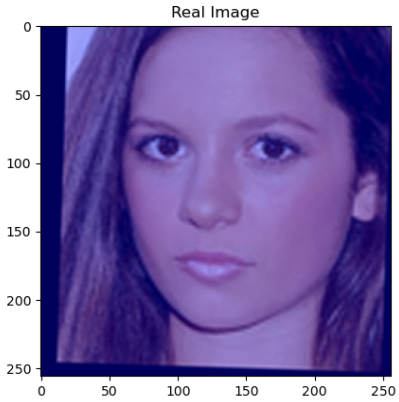
TEL AVIV UNIVERSITY

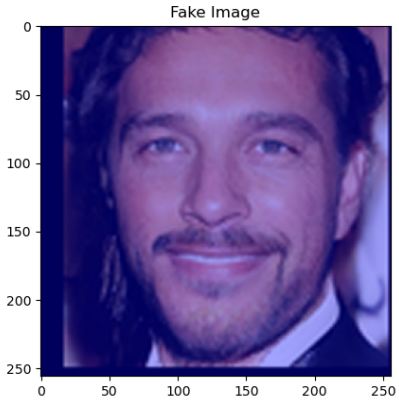
SimpleNet trained on Deepfakes



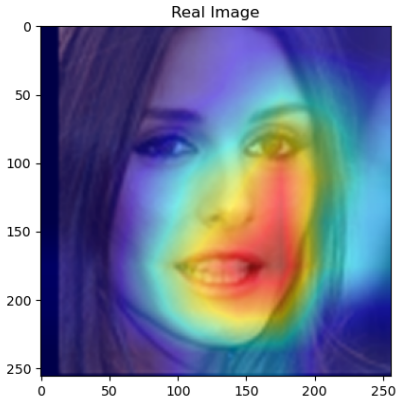


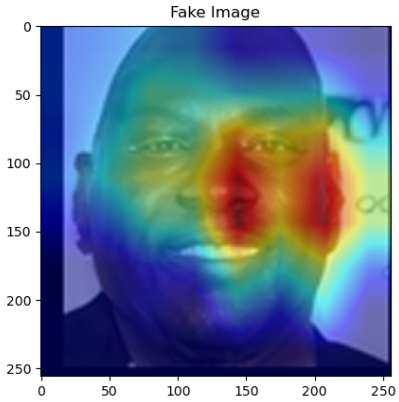
SimpleNet trained on Synthetic Faces





Xception-Based trained on Synthetic faces





Competition:

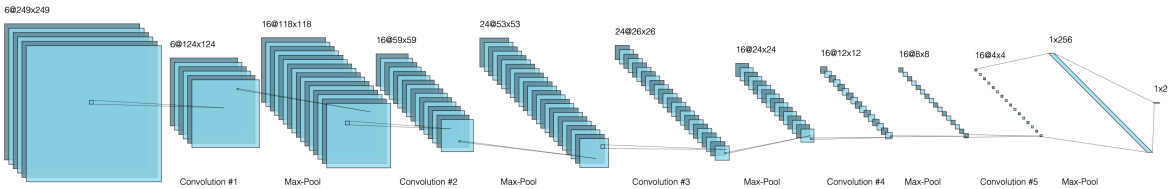
See implementation of

1. **MyNet** network class in **competition_model.py**
2. Pre-trained model in checkpoints/**competition_model.pt**
3. Accuracies and loss in out/**fakes_dataset_MyNet_Adam.json**

In order to suggest a network, we did a little research and several implementations. The implementations were included different architectures, some known (i.e., MobileNetV2) an some of them are just with trial and error.

We got the best result with network that based on similar architecture like the SimpleNet but with changes such that we finally got **AuC of 95.8%** and very low **number of parameters ~35k**. those results were gotten after we deleted the 2 fully connected (FC) (and "heavy") layers by adding 2 convolutional layers with different kernels sizes in order to get "better resolution" before we made the last FC layers for our 2-class classification. Since we tested our network a large number of epochs, we also added a dropout before the FC layer in order to reduce the probability for overfitting.

Finally, we've got the following structure.

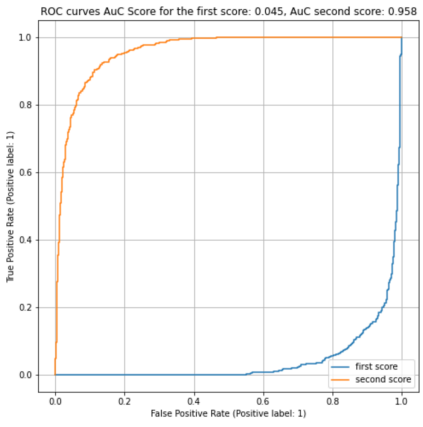


- The last 2 layers are separated because of the drawer constraint.
- The network can also be written as:
  Conv1 (6@7x7) → Max-Pool (2x2) → ReLU → Conv2 (16@7x7) → Max-Pool (2x2) → ReLU → Conv3 (24@7x7) → Max-Pool (2x2) → ReLU → Conv4 (16@3x3) → Max-Pool (2x2) → ReLU → Conv5 (16@5x5) → Max-Pool (2x2) → ReLU → Dropout (0.35) → Fully-Connected (256x2).

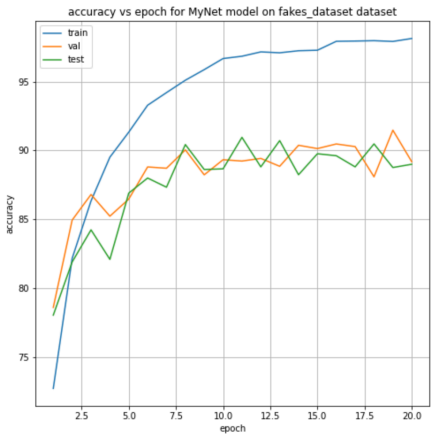
The result of 20 epochs that we ran on this network is shown below, and finally gave the following score.

$$score = \frac{1}{1+\log_{10}[34850]} \cdot 100^{10 \cdot \frac{(95.8-90)}{100}} = 2.89$$

ROC



Accuracy



Model Summary and Parameters

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 6, 249, 249]             888
         MaxPool2d-2          [-1, 6, 124, 124]               0
            Conv2d-3        [-1, 16, 118, 118]           4,720
         MaxPool2d-4          [-1, 16, 59, 59]               0
            Conv2d-5          [-1, 24, 53, 53]          18,840
         MaxPool2d-6          [-1, 24, 26, 26]               0
            Conv2d-7          [-1, 16, 24, 24]           3,472
         MaxPool2d-8          [-1, 16, 12, 12]               0
            Conv2d-9            [-1, 16, 8, 8]           6,416
        MaxPool2d-10            [-1, 16, 4, 4]               0
          Dropout-11                 [-1, 256]               0
           Linear-12                   [-1, 2]             514
================================================================
Total params: 34,850
Trainable params: 34,850
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.74
Forward/backward pass size (MB): 6.40
Params size (MB): 0.13
Estimated Total Size (MB): 7.28
----------------------------------------------------------------
```