Computer Vision Project: Facial Manipulation Detection

Rotem 206260150 & Ayellet 313381543

Chapter 2: Build Faces Dataset

Question 2

Figure from running plot_samples_of_faces_datasets.py:

*given code is wrong and assuming half the dataset is real images and half fake (sampling indexes based on first / second half of dataset indexes), the given output is without correcting the code. Our implementation of __getitem__ function is based on indexing all the real images and then the fake ones (if the given index is smaller than the number of real images, it will return a real image, otherwise a fake one).

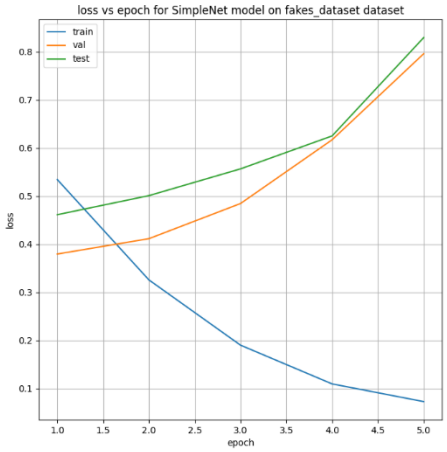
Chapter 3: Write an Abstract Trainer

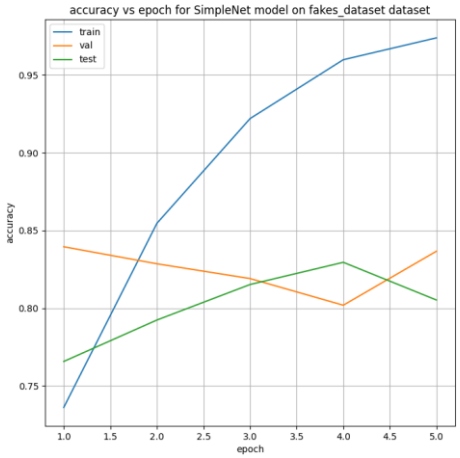
Question 6

The results in the json file are unexpected. Although the model learns, and the train loss decreases while the train accuracy increases as expected, we can see the results for the validation and test sets are different and not getting better with the training epochs.

Question 7

Plot_accuracy_and_loss.py on SimpleNet training on Deepfakes dataset:





Question 8

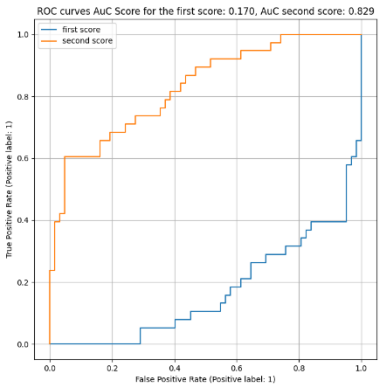
Looking at the results in the json file, we can see the highest validation accuracy received is at first epoch (83.95%), and the corresponding test accuracy is 76.57%.

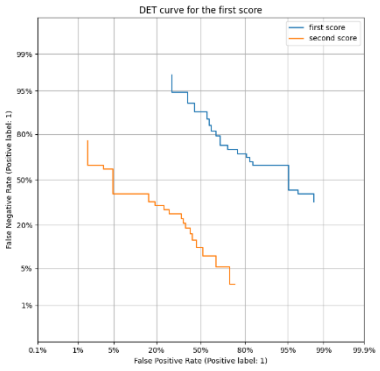
Question 9

The fakes dataset consists of 3600 fake images and 7200 real images, meaning the proportion of fake to real is 1:2.

Question 10

Output of numerical_analysis.py on SimpleNet trained on Deepfakes dataset:





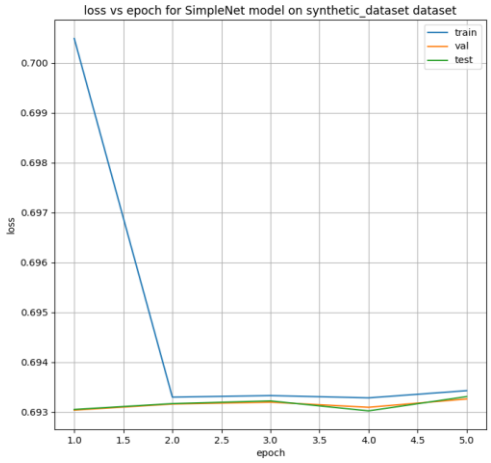
Question 11

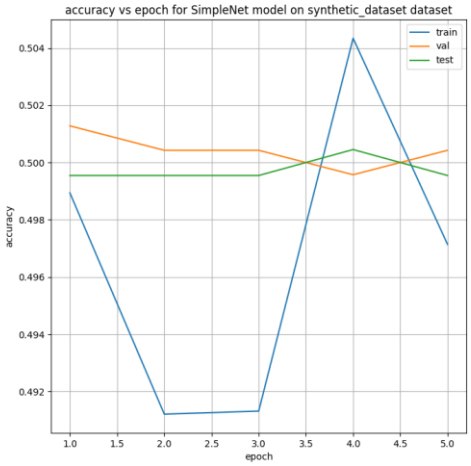
Scores of (A) or (B) show a different output – one is a prediction of how real the image is and the other one is a prediction of how fake it is. Although we expect them to complete each other (because each image is either real or fake, so we wish the model to say real with high probability and fake with low probability or vice versa), it is not exactly so: in general we do see such a negative correlation but it is important to remember that the two different scores may learn different things

by varying the weights of learned features. We do see in the graph generally complementary behaviour between those two.

Question 12+13

Loss curves of SimpleNet trained on Synthetic faces dataset:





Question 14

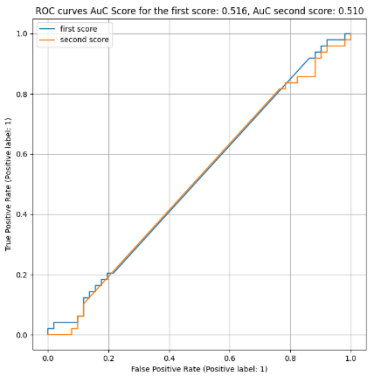
Looking at the results in the json file, we can see the highest validation accuracy received is 50.12%, and the corresponding test accuracy is 49.95%.

Question 15

The synthetic dataset test set consists of 552 fake images and 551 real images, meaning the proportion of fake to real is 1:1.

Question 16

We got a bad classifier! Kind of degenerated. It is easiest to see it by looking at the ROC curve:



Which implies that our classifier is probably kind of a constant one that doesn’t actually take into account the input image. We can see that the higher the true positive rate is, the false positive rate is pretty much the same.

Question 17

This result makes sense because the synthetic images are a lot more realistic than the fake ones since they were generated by using GANs. Therefore, it is a lot harder for our network to distinguish between the real images and the fake images that look very realistic. That is why we got better results for the first set (i.e. fake_dataset) because the fake images weren't as realistic as the synthetic ones.

Chapter 4: Fine Tuning a Pre-trained Model

Question 18

The Xception model is pre-trained on the ImageNet dataset. The ImageNet dataset is a large-scale dataset that contains millions of labeled images across thousands of categories.

Question 19

Xception model is composed of three flow structures:

- Entry flow- The initial part of the model that processes the input data and extracts features. Initial convolutional layers followed by three blocks of varying depths.
- Middle flow contains 16 blocks which are repeated eight times with residual connections.
- Exit flow contains two blocks followed by global average pooling and a fully connected layer for classification.

The basic block consists of depthwise separable convolutions, followed by ReLU activations and some pooling layers and connected by residual connections.

Question 20

The Xception model is pre-trained on the ImageNet dataset. The ImageNet dataset is a large-scale dataset that contains millions of labeled images across thousands of categories.

Question 21

The input feature dimension to the "fc" block is set in the in_features parameter. In our model the dimension is 2048.

Question 22

The number of parameters is 22855952.

Question 24

In the provided MLP architecture, we have the following connected layers (ReLU doesn't use any parameters):

- Fully-Connected 2048x1000
- Fully-Connected 1000x256
- Fully-Connected 256x64

- Fully-Connected 64x2

To calculate the number of parameters in each layer, we should use the following formula:
$$\text{nparams} = \text{current layer neurons} \cdot \text{previous layer neurons} + 1 \cdot \text{current layer neurons}$$

Where the last component in the formula refers to the bias term associated with each neuron in the layer.

- The first fully connected layer has $\text{nparams\_1} = 1000 \cdot 2048 + 1 \cdot 1000 = 2049000$
- The second fully connected layer has $\text{nparams\_2} = 256 \cdot 1000 + 1 \cdot 256 = 256256$
- The third fully connected layer has $\text{nparams\_3} = 64 \cdot 256 + 1 \cdot 64 = 16448$
- The fourth fully connected layer has $\text{nparams\_4} = 2 \cdot 64 + 1 \cdot 2 = 130$

The total number of parameters in the MLP architecture we added is the sum of the numbers from all layers above:

$$\text{nmlp\_params} = \text{nparams\_1} + \text{nparams\_2} + \text{nparams\_3} + \text{nparams\_4} = 2321834$$

As a result, we add 2321834 parameters.

Question 25

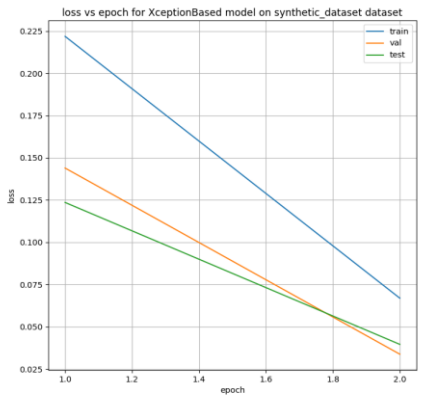
We should note that we encountered an issue with loading the model from the URL in our environment and had to make the following change in order to run the training (we downloaded the file from the URL manually and saved it locally, and loaded it from the local copy using this):

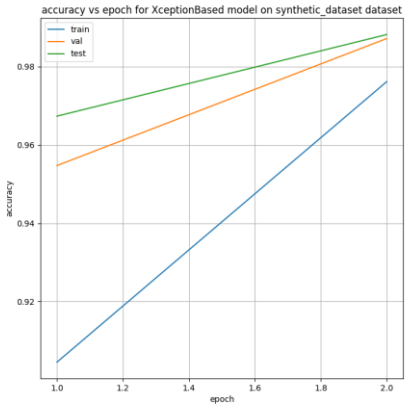
```
# model.load_state_dict(model_zoo.load_url(model_urls['xception']))

checkpoint = torch.load('xception-c0a72b38.pth.tar')

model.load_state_dict(checkpoint
```

The submitted code does not contain this change as we don't expect this problem to occur in the testing environment.

Question 26

loss curves of Xception-Based network trained on the synthetic dataset:



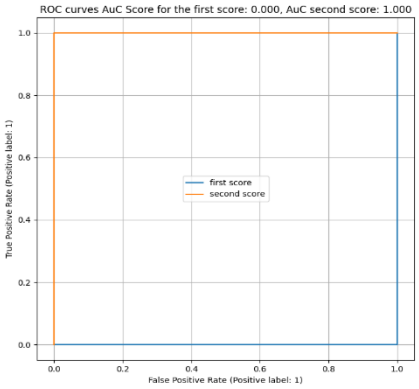


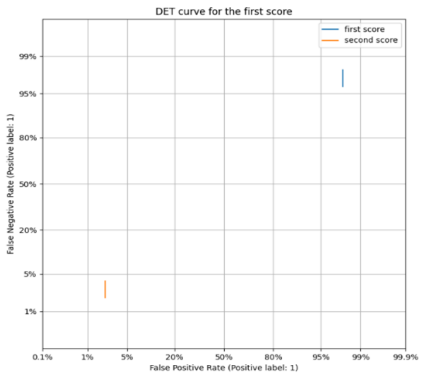
Question 27

Looking at the results in the json file, we can see the highest validation accuracy received is 98.72%, and the corresponding test accuracy is 98.82%.

Question 28

Output of numerical_analysis.py on XceptionBased trained on synthetic dataset:





We can see this model performs much better (almost perfectly) on the synthetic dataset.

Chapter 5: Saliency Maps and Grad-CAM analysis

Question 29

Image-Specific Class Saliency Visualization are techniques used in computer vision to highlight the areas of an image that contribute the most to a specific class prediction made by a neural network. In other words, it is designed to provide insights into which features or parts of an input image are crucial for the model to make a specific classification decision. The methods usually include computing the gradients of the model's output with respect to the input image and then visualizing them. The gradients emphasize how changes in the input pixel values would affect the model's prediction.

for a given image I and convolutional neural network which calculates the score ScI for class of interest c (i.e. the object, category or label that the neural network is trained to detect in images). We use the gradient to approximate the score by applying a first-order Taylor expansion since the class score is non-linear function of I:

$$\mathrm{ScI} \approx \mathrm{wTI+b}$$

Where w is the derivative of the score with respect to the image at the point (image) $\mathrm{I0}$ and b is the model's bias:

$$\mathrm{w= Sc\partial I|I0}$$

Question 30

Grad CAM which is Gradient-weighted Class Activation Mapping is a technique for explaining the decisions from Convolutional Neural Networks by visualization. The output of this technique is a heatmap that highlights the spatial regions in the input image that are considered most significant for predicting a specific class made by a Convolutional Neural Network (CNN). In other words, the

heatmap is a two-dimensional representation that aligns with the dimensions of the input image, where the intensity of the heatmap corresponds to the importance of each region.

The technique includes the following steps:

- The input image is entered forward to the CNN to obtain the final convolutional layer's output.
- The gradients of the score for the class with respect to the feature maps of the final convolutional layer are calculated (i.e. backpropagation). The score of a specific class is the model's output confidence for the specific class and reflects how strongly the model "believes" the input belongs to this class. Thus, the meaning of the gradients is how much the predicted class score would change with respect to the changes in the value of the feature map.
- Performing global average pooling over the width and height dimensions of the gradients. These values are used to obtain the neuron importance weights.
- Performing a weighted combination by multiplying feature maps with their importance weights (computed from gradients).
- Applying ReLU activation to focus on positive influences of pixels (pixels with negative influence are likely to belong to other categories in the image).

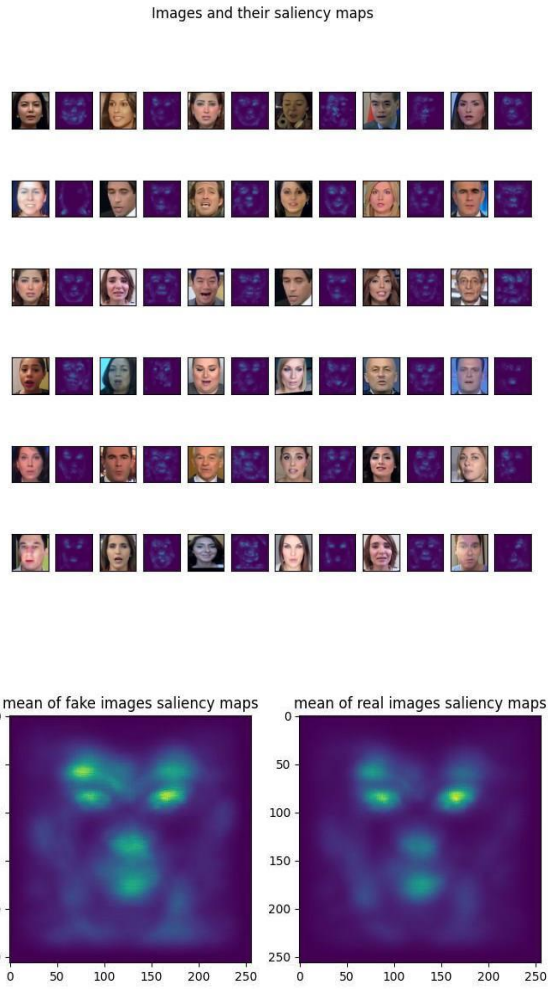
Question 32

In order to successfully run the relevant code, we added strict=False to the line:

Model.load_state_dict(torch.load(args.checkpoint_path)['model'], strict=False).

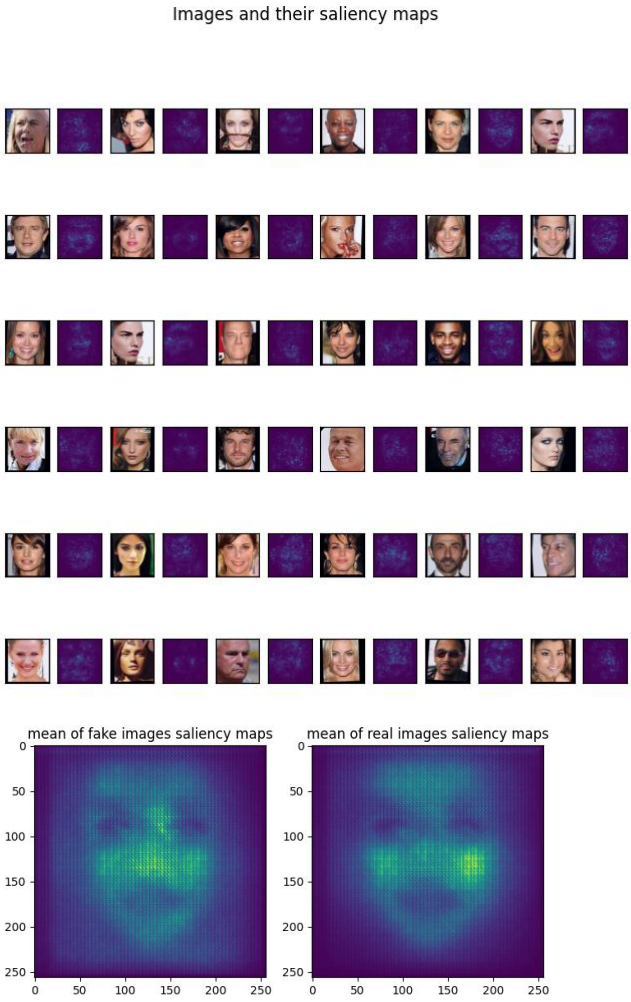
This was necessary as we were required to override the fc layer that was expected in the mode state dict.

Saliency maps for SimpleNet:



From the saliency maps we can learn that our SimpleNet model focuses mostly on the center of the face – eyes and eyebrows, nose and mouth to differentiate between real and fake images. Looking at the pairs, we see the model is also focusing on the contour of the face which defines its general shape. When looking at the mean saliency maps we only see the details of the center of the faces, as the pose and the general size and shape of the face vary, the details in the center are usually kept in the center of the image. We can also note the mean saliency maps between fake and real images are very similar, and that the areas of interest of the fake are a bit less focused (larger area), maybe because those images vary more on those parts.

Saliency maps for XceptionBased:



For the XceptionBased model the saliency maps tell a very different story. Looking at the pairs, it is actually hard to understand what the model learned to focus on, as each image has different points of interest. It is clear, however, that this model is not focused on the center of the faces. Looking at the mean saliency maps, it becomes much clearer that the model is focusing on the area of the cheeks and nose, and between the eyes for fake images. Another note we can make is that the mean saliency maps of this model seems to have higher-resolution that those of SimpleNet, which looks more blurry. This may imply that the XceptionBased model is better in learning the fine details of the images.

Question 34

We added the following imports in the grad_cam_analysis.py script according to the instructions:

```
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image
```

The imports above are used in the get_grad_cam_visualization function.

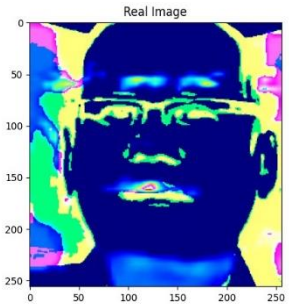
Question 35

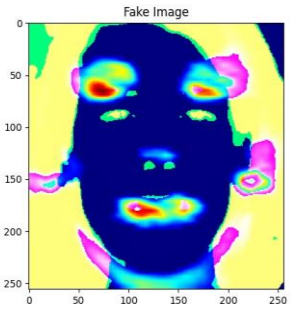
As in question 32 above, in order to successfully run the relevant code we added strict=False to the line:

Model.load_state_dict(torch.load(args.checkpoint_path)['model'], strict=False).

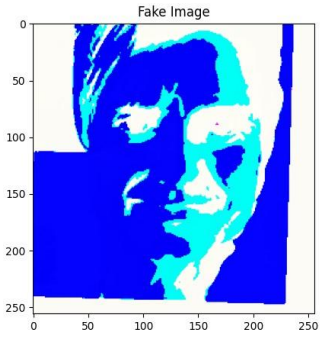
This was necessary as we were required to override the fc layer that was expected in the mode state dict.

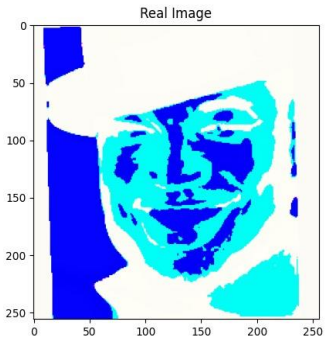
Grad cam results for real and fake images from fakes dataset for SimpleNet:





Grad cam results for real and fake images from synthetic dataset for SimpleNet:





Grad cam results for real and fake images from synthetic dataset for XceptionBased: