Seminar DL: Group 2

Hannes Ehringfeld Paul Adler Hakim Guenther Damaris Gann

1. Introduction and Work Process

The objective of the first project of the seminar was to train a binary image classifier capable of distinguishing between real and fake images. Real images are any images taken with a camera, while fake images are those generated by current text-to-image models such as stable-diffusion, midjourney, or Dall-E. In the following report, we will outline our project journey, highlighting important challenges and learnings. Conceptually, the development process can be divided into three iterations, with several experiments run in parallel. At the start of each new iteration, previous findings were aggregated and used to design new experiments. The improvement that had the most significant impact on our experiment's outcome was the composition of our training dataset. Therefore, we describe it in more detail in 2. In the final experiment, we further increased the diversity in the dataset by using augmentations, which we discuss in 3. In section 4, we outline the logical flow between the iterations and how we arrived at the design of our final training run 5, which yielded the results we present in 6.

2. Dataset

From iteration 1-3, we introduced changes to our training dataset. During the first iteration, we worked with datasets 1.0 and 1.1. During the second iteration, we created datasets 2.0 and 2.1. In the third and final set of experiments, we only worked with dataset 3.0. Its composition is described in great detail in table 1. The previous datasets were gradually increased. The following table describes their composition:

Dataset 1.0 consisted of 9.4k real images from Imagenette and 10k fake images from DiffusionDB.
Dataset 1.1 expanded the collection to include 13k real images from Imagenette and 13k fake images from DiffusionDB.
Dataset 2.0 comprised 40k real images from the COCO dataset, 20k real images from the OpenImages dataset, 50k fake images from DiffusionDB, and 19k fake images from the StableDiffusion dataset.
Dataset 2.1 contained 9.4k real images from Imagenette, 20k real images from OpenImages, 9k real images from laion, 18k fake images from sd15 and sd20 train, and 15k fake images from DiffusionDB.

Dataset 3.0 contains a total of 160k images, 80k real and 80k fake. The average image size is 663x613. Prior to training, the dataset is split into training, validation, and test sets. The distribution of real and fake images is for all splits 50/50. The training set contains 128k images, while the validation and test sets each contain 16k images (80%), test (10%), and val (10%).

Real Images	images
coco (randomly sampled)	31.531
openimages_20k_test_subset	20.000
wikiart (randomly sampled*)	10.000
imagenette	9.476
laion subset	9.000
Fake Images	images
diffusiondb_2m_first_50k	50.000
diffusiondb_2m_first_51k_61k	11.000
sd15_train	9.000
sd20_train	9.000
sd15_val	500
sd20_val	500

Table 1. Composition of the final dataset 3.0

*We randomly sampled 1250 images each from the following categories: religious-painting, flower-painting, genre-painting, landscape, marina, mythological-painting, nude-painting-nu, portrait

Dataset 3.0 is available on HuggingFace.

3. Augmentation

In the first two iterations, we did not use any augmentations besides a simple resize operation to ensure the model sees images of consistent sizes. During iteration 2, we realized that due to various rectangular aspect ratios, resizing to a square image would result in distorted objects in the images. Hence, we changed the procedure to first take a fixed-size random crop with padding (which is square) and then resize the image to the desired width and height. In the final iteration, we also added various Random Augmentations shown in Table 2 to further diversify our dataset.

Augmentation	Probability
RandomCrop with size 625 and padding	100%
Resize to size 360	100%
RandomHorizontalFlip	50%
RandomVerticalFlip	50%
GaussianBlur kernelsize (5,9), sigma (0.1,5)	50%
RandomGrayscale	50%
RandomErasing with scale 0.02 to 0.2	20%

Table 2. Used augmentation techniques and their probability

4. Models

Throughout the course of the project, we performed many experiments. Here is a chronology of the major ones that influenced our decision-making. We adopt the numbering scheme from the datasets. For most experiments, we stuck to the ResNet-50 architecture and used the pretrained weights. Model 1.0 and 1.1 shared similar, very high training and testing metrics. For example, the testing loss of model 1.0 was 0.024 with a testing accuracy of 99.178. This suspiciously high performance after only 13 epochs of training seemed too good to be true, so we tested it on some arbitrary images from the internet. The model almost always predicted the image to be fake. This indicated that the model is not learning anything general about the nature of real and fake images but rather memorizes the objects depicted in them (our real images only showed 10 different objects). We also suspected that the original image resolution must be a determining factor for the model since all fake images shared a rather low resolution (e.g., 187×261), while the fake images coming from DiffusionDB were much higher (e.g., 512x512). Therefore, we made an effort to gradually increase and diversify our training dataset, which resulted in Dataset 2.0 and 2.1. Although these datasets were much bigger than the first two and the training metrics seemed plausible (Val Loss: 0.275, Accuracy: 90.769), the resulting models did not perform much better on our arbitrary images. During the training phase for models 2.0 and 2.1, we dealt with several technical challenges (Google Colab session terminations), technical errors (wrong label encodings), and explored various augmentation techniques. The natural conclusion seemed to be to combine all gained experience in aggregating a final training script, compiling a final and third dataset (see Table 1), and running the training on reliable local hardware that we got an opportunity to use for a few days. The final training pipeline and model results are discussed in sections 5 and 6.

5. Training

Besides the already described specifics of Dataset 3.0 and the augmentations, we configured our final training script with the following Hyperparameters: In our final training script for Dataset 3.0, we configured the following Hyperparameters: The number of epochs was set to a high number and stopped before reaching the end of the epochs by the regularization technique Early Stopping with a patience of 5 to prevent overfitting. We used a batch size of 24 and a learning rate of 0.001. The script utilized 4 workers for parallel processing. Images were cropped to a size of 625 and resized to 360x360 pixels. A weight decay of 0.001 was applied for regularization. The loss function was set to CrossEntropyLoss, and we used the Adam optimizer with the specified learning rate and weight decay.
We trained two models with the same pipeline and data in parallel: a ResNet-50 and a VGG, both with pretrained weights. After 37 epochs of training the ResNet-50, the EarlyStopper halted the training and returned the final model from epoch 32. We suspected that further training could be possible if we reduced the learning rate to 0.0001. Training then continued for an additional 7 epochs, before there was no more performance gain. The training results are shown in Table 3. The VGG model produced very similar results, so we can assume that the performance is not primarily dependent on the model architecture but is mostly influenced by the dataset and augmentation.

6. Results & Discussion

Metric	Epoch 32	Epoch 39
Train Loss	0.228	0.301
Train Accuracy	84.823%	86.803%
Val Loss	0.329	0.283
Val Accuracy	85.244%	87.669%
Test Loss	0.330	0.304
Test Accuracy	85.287%	86.050%
F1-score	0.853	0.872
Precision	0.853	0.807
Recall	0.853	0.947
Accuracy for Class 0	85.0%	77.9%
Accuracy for Class 1	85.7%	95.0%
Accuracy on 18 Arbitrary Images	100%	94.4%

Table 3. Comparison of model performance at Epoch 32 and Epoch 7 after learning rate adjustment

After obtaining the results shown in Table 3, we modified the architecture to include a sigmoid layer at the end, which returns the probability of an image being fake (belonging to class 1). The final prediction is determined by a probability threshold of 0.6, which we experimentally found to yield the best results, achieving an accuracy of 90.013%, F1-score of 0.901, precision of 0.896, and recall of 0.905.

Models and Training notebooks with example predicitons are available on GitHub.