Assignment 3

183.663 Deep Learning for Visual Computing (2022S)

Group 9
Yu Kinoshita (01623806), Michael Köpf (01451815)

General Remarks

For this exercise, we experimented with two different Deep Learning tasks:

- Object Detection
- Generative Adversarial Networks (GAN)

We decided on using Jupyter notebooks instead of plain Python scripts, since in our opinion they are better suited when experimenting with existing Deep Learning projects from the internet.

Further technical information on our submission is provided in the `README` file.

Object Detection

Experimental Setup

For the object detection experiments, we chose models from the `YOLOv5` family [1]. In total, we train on (fine-tuning of models pre-trained on the COCO dataset [2]) and evaluate 3 different datasets: the Oxford Pets Dataset [3] by species[1] and by breed[2], and the Aquarium Dataset[3].

We try to cover a heterogeneous and wide range of different configurations. In particular, we evaluate

- the influence of the dataset, specifically, its training set size. The Oxford Pets training set contains 2576 images of 2 classes (by species) and 37 classes (by breed). The classes of the former – cats and dogs – are also contained in the COCO dataset. However, the COCO dataset has no notion of different cats and dogs breeds. In contrast, the training set of the Aquarium Dataset only contains 448 images of 7 classes. None of these classes nor any 'super-class' (e.g., fish) are contained in the COCO dataset.
- fine-tuning on different pre-trained models, namely, `YOLOv5n` (nano) and `YOLOv5s` (small) to investigate if it is beneficial to choose one over the other.
- the influence of different optimizers.

In total, 8 different configurations were used for the training procedure (for details see Table 1 in Results). Due to the larger size of the Oxford Pets Datasets, different optimizers are only evaluated on the Aquarium Dataset. For the same reason, the models are fine-tuned for fewer epochs on the Oxford Pets Datasets. Images are resized to a resolution of 640×640 (same as used for pre-training on COCO). For hyperparameters that are not listed in Table 1, the defaults of `YOLOv5n`[4] and `YOLOv5s`[5] are used.

The mean average precision (mAP) – specifcally, mAP@0.5 and mAP@0.5:0.95 – are used as evaluation measures. Furthermore, after each training epoch, the weighted average of these measures, i.e.,

$$0.1 \cdot \text{mAP@0.5} + 0.9 \cdot \text{mAP@0.5:0.95}$$

is calculated on the validation set and the current model is saved to disk in case the weighted average has improved. The last model that is saved in this way is refered to as 'best model'.

Results

The achieved results on the validation sets using the best model of each training run, respectively, are shown in Table 1.

[1] `https://public.roboflow.com/object-detection/oxford-pets/2`, Accessed: 2022-06-29
[2] `https://public.roboflow.com/object-detection/oxford-pets/1`, Accessed: 2022-06-29
[3] `https://public.roboflow.com/object-detection/aquarium`, Accessed: 2022-06-29
[4] `https://github.com/ultralytics/yolov5/blob/master/data/hyps/hyp.scratch-low.yaml`, Accessed: 2022-06-29
[5] `https://github.com/ultralytics/yolov5/blob/master/data/hyps/hyp.scratch-high.yaml`, Accessed: 2022-06-29

Dataset	Base Model	# of Epochs	Batch Size	Optimizer	Training Duration [h]	mAP^{val}@0.5	mAP^{val}@0.5 : 0.95
Oxford Pets by Species	`YOLOv5n`	150	32	SGD	0.710	0.994	0.849
	`YOLOv5s`	150	32	SGD	1.181	**0.995**	**0.877**
Oxford Pets by Breed	`YOLOv5n`	150	32	SGD	0.737	0.895	0.732
	`YOLOv5s`	150	32	SGD	1.183	**0.933**	**0.784**
Aquarium	`YOLOv5n`	300	16	SGD	0.274	0.756	0.435
	`YOLOv5n`	300	16	AdamW	0.300	0.727	0.403
	`YOLOv5s`	300	16	SGD	0.444	**0.798**	**0.474**
	`YOLOv5s`	300	16	AdamW	0.460	0.740	0.415

Table 1: Overview of chosen hyperparameters, training duration and results on the respective validation sets.

In general, we can observe that the achieved mAP@0.5 and mAP@0.5:0.95 are significantly higher compared to the ones on the COCO validation set (`YOLOv5n` – mAP@0.5: 0.457, mAP@0.5:0.95: 0.280; `YOLOv5s` – mAP@0.5: 0.568, mAP@0.5:0.95: 0.374)[6]. One reason for this could be that the number of classes in the COCO dataset is higher and the classes are also far more heterogeneous than in our datasets.

Unsurprisingly, the mAP on the two Oxford Pets Datasets is clearly higher than on the Aquarium dataset. Presumably, the main reason for this is the small and imbalanced Aquarium training set.

Even though the pre-trained models were not aware of different breeds of cats and dogs, the achieved mAP on Oxford Pets by Breed can be considered quite good (actually better than we would have expected).

With respect to the model architecture, `YOLOv5n` offers a good trade-off between the training time and the achieved accuracy on Oxford Pets by Species. The mAP@0.5 is almost similar to the one of the `YOLOv5s` model and also the mAP@0.5:0.95 is only marginally lower. In contrast, the difference in accuracy between `YOLOv5n` and `YOLOv5s` seems to be more significant on Oxford Pets by Breed and on the Aquarium Dataset. Thus, the `YOLOv5s` model might be a better choice in these cases (although training takes longer).

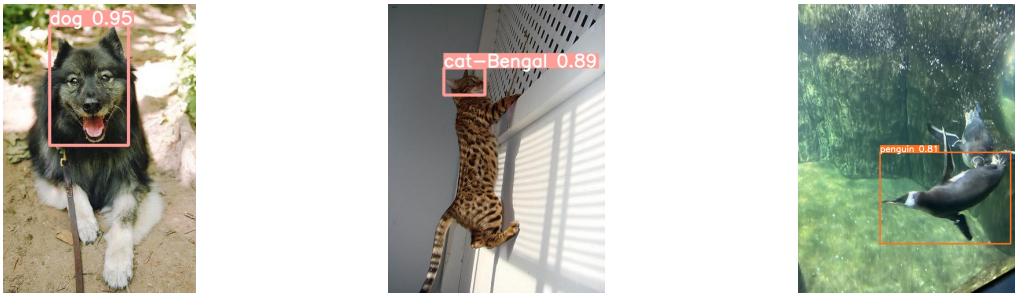
The Aquarium Dataset was evaluated using two different optimizers – SGD and AdamW. In the given configuration, the models trained with SGD achieve better results and also take less time to train.

For the final evaluation on the independent test sets, the models that achieved the best results on the validation set, respectively, are used, this is, the fine-tuned `YOLOv5s` models (optimizer SGD). Again, the images are resized to 640×640. The results are shown in Table 2. As we can see, the achieved mAP values on Oxford Pets by Breed and the Aquarium Dataset are higher than on the validation set. Hence, we can conclude that the validation sets slightly overestimates the generalization error in these cases.

Test Set	mAP@0.5	mAP@0.5 : 0.95
Oxford Pets by Species	0.994	0.879
Oxford Pets by Breed	0.940	0.803
Aquarium	0.824	0.499

Table 2: Results on the test sets.

Figure 1 exemplarily shows a selected image from each test set.



(a) (b) (c)

Figure 1: Selected images from the test sets with drawn object detection bounding boxes, predicted class labels and confidence scores: (a) Oxford Pets by Species (class dog) (b) Oxford Pets by Breed (class cat-Bengal) (c) Aquarium Dataset (class penguin).

[6] `https://github.com/ultralytics/yolov5#pretrained-checkpoints`, Accessed: 2022-06-29

Generative Adversarial Networks (GAN)

In this section image generation models, specifically generative adversarial networks (GAN) are compared using its metric on a test set and generated images. Finally there is a short demonstration of image generation from textual descriptions using CLIP (Contrastive Language-Image Pre-training).

GAN is a generative model using deep learning methods. It makes use of two networks/models. The generator model generates images and the discriminator classifies the image as either real or fake (generated). The discriminator acts similar to a loss function for the generator model. The models are trained until the discriminator is fooled about half the time.

Experimental Setup
The following models are chosen for the experiments:

- **StyleGAN-XL** [4]
  The generator architecture is redesigned using an approach from style transfer literature. The generator begins with a learned constant input and adjusts the 'style' at each convolution layer according to the latent code. This way the strength of image features are directly controlled at different scales. The input is being mapped to an intermediate latent space w, which controls the generator through AdIN (adaptive instance normalization) at each conv layer, while traditional generators feed the latent code only through the input layer. This architecture is further optimized to large divers data sets by changing the training strategy to be less restrictive.
- **Diffusion-StyleGAN2** [5]
  This model is also based on StyleGAN but instead of chaning the training strategy it employs "a Gaussian mixture distribution, defined over all the diffusion steps of a forward diffusion chain, to inject instance noise" [5].

The models were not trained on our own. Instead we used pre-trained models, since training image generation models can take between 4 to 41 days depending on the image resolution of the training set which is not feasible in private use [6]. Unfortunately, most `GitHub` repositories did not have a very detailed instruction on how to use the algorithm. But as soon as one is able to run one of the models, it gets easier to implement other models since they all have rather similar structures.

Performance Evaluation
The chosen comparison metric is the Frechet Inception Distance (FID). It calculates the distance between feature vectors calculated for real and generated images. It summarizes how similar two groups are in terms of computer vision features. A perfect is at 0.0, meaning the two image groups are identical.

CIFAR-10 was chosen as it is one of the smaller image datasets, and larger ones were not feasible to use on a private setting as they tend to have 800GB to 1TB+ in size. The models were pre-trained on the CIFAR-10 dataset and evaluated on the test set of CIFAR-10.

As expceted the current state-of-the-art model, StyleGAN-XL performs much better than the Diffusion-StyleGAN2 (see Table 3).

	FID
StyleGAN-XL	1.886
Diffusion-StyleGAN2	3.266

Table 3: Performance comparison of StyleGAN-XL vs. Diffusion-StyleGAN.

Results – Generated Images
In this section we compare the generated images of the models.

For StyleGAN-XL and Diffusion-StyleGAN2 we used the pre-trained models using `Flickr-Faces-HQ Dataset` which is a set of 70.000 high quality images of human faces with a resolution of 1024×1024.

As expected the StyleGAN-XL can generate faces with almost no visible errors (see Figure 2) while the Diffusion-StyleGAN2 performs also greatly but struggles with few errors such as slightly deformed faces (see Figure 3).

Figure 2: Generated faces using StyleGAN-XL trained on FFHQ1024 dataset.

Figure 3: Generated faces using Diffusion-StyleGAN2 trained on FFHQ1024 dataset.

Results – Image Generation using Text Input
To generate images from text prompts StyleGAN-XL and CLIP (Contrastive Language-Image Pre-training) are used. CLIP is a neural network trained on image-text pairs. Given an image it can predict the most relevant text snippet. This combined with StyleGAN-XL to generate images out of text snippets. The StyleGAN-XL was pre-trained on `ImageNet1024` dataset.

In Figure 4 we can see how the GAN generates the image in 100 steps with an text prompt input 'A red mushroom'. The image on the left most side is the initial generated image (first iteration) and the right most side image we can see the final output.

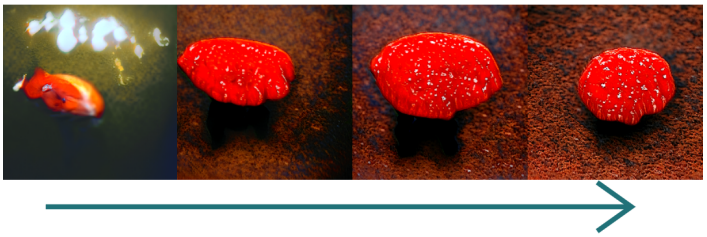


Figure 4: Generated image from text prompt: 'A red mushroom'.

References

[1] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon *et al.*, "ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference," Feb. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6222936

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[3] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[4] S. Axel, S. Katja, and G. Andreas, "StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets," 2022. [Online]. Available: https://arxiv.org/pdf/2202.00273v2.pdf

[5] W. Zhendong, Z. Huangjie, C. Weizhu, and Z. Mingyuan, "Diffusion-GAN: Training GANs with Diffusion," 2022. [Online]. Available: https://arxiv.org/pdf/2206.02262v2.pdf

[6] K. Tero, i. L. Samul, and A. Timo, "A Style-Based Generator Architecture for Generative Adversarial Networks," 2019. [Online]. Available: https://arxiv.org/pdf/1812.04948v3.pdf