Synthesising Abstract Deep Computer Vision Pretraining Data

Søren Winkel Holm, s183911

December 18, 2022

Contents

1 Introduction

In the field of computer vision, large deep learning models have become the norm for solving semantic classification problems such image segmentation. These performant methods have some central problems: First of, to start training them, a larger number of images are required with corresponding manually annotated ground truth labels which are costly. Secondly, the training proces takes takes a long time and is compute expensive as the model iteratively has to reach convergence from a starting point of randomly initialized parameters. Finally, the error modes and interpretations of results are notoriously difficult.

A possible tool to mitigate these issues is the use of synthetic data. If models could learn generalizable parametrizations from cheap, computer generated images, the amount of real world data and case by case training time would be reduced. Synthetic data could also help to interpret model performance as the generated scenes can be designed to reveal flaws.

For this idea to work, the generated scenes should contain discernable objects and the learning task should require that the model converge to some representations of fundamental principles of object detection. These principles could serve as inductive biases to help the model on real-world tasks. One such principle is the modelling of object texture where the model should be able to map local visual patterns to class attributions.

In this project, I will explore an approach to generating such a synthetic dataset and perform tests of applying this dataset to a real-world use case. I explore using noise functions to give the objects variations in texture connected to class assignments.

2 Methods

All code is available with reproduction instructions on github.com/sorenmulli/imgseg-fake-balloons and the WebGL renders can be seen on student.dtu.dk/ s183911/proj/webgl-site.

2.1 Synthetic Scene Rendering

Task 1: Noise Functions and The Simple Scene In the fragment shader, two noise functions were implemented, both accepting three-dimensional vector input.

1. Improved Perlin noise (PN) using a permutation polynomial as McEwan, Sheets, Richardson, and Gustavson [McE+12]. 3D PN chooses pseudo-random gradients $\in \mathbb{R}^3$ at regular grid points in each axis, interpolating a smooth function consistent with the gradients [Gus05].

2. Sparse convolution noise (SC) as Frisvad and Wyvill [FW07]. SC targets minimal artifacts or patterns by approximates the computationally expensive white noise filtered with a Gaussian kernel using a cubic kernel instead.

Changing magnitude of the input vector – the point at which the noise landscape is queried – is equivalent to controlling the zoom level of the noise texture. The implementations were largely based on reference implementations [Fri16]: Only minor code changes and commenting were performed along with using the functions to texture the scene objects with a given noise scale r.

An initial scene was constructed by rendering a coloured sphere through 5-fold recursive subdivision of tetrahedra. This foreground object was shaded using Phong shading from a positional light behind a perspective camera. The object was textured using the noise functions evaluated at normals scaled with a free parameter called noise scale. This texturing was perform by letting the noise intensity controlling the diffuse light intensity, see Listing 1.

Behind the object, a 2D noise textured background in black and white was placed.

The resulting scene can bee seen at Figure 1 and the noise function and other variables can be user controlled on Page t1.html.

Listing 1: Fragment shader snippet related to texturing of foreground objects

```
        } else {
// Use normal vectors to get smooth noise over sphere
vec3 p = vec3(noiseScale * n.x, noiseScale * n.y, noiseScale * n.z);
float noise = do_sc ? scnoise(p) : pnoise(p);
float noiseIntensity = 0.5 * noise + 0.5;

// Phong lighting using noise intensity for diffuse radiance
float d = max(dot(normalize(n), normalize(w_i)), 0.0);
vec3 w_h = normalize(w_i + w_o);
vec3 lightSpecular = ks * Le * pow(max(dot(normalize(n), w_h), 0.0), s);
vec3 lightDiffuse = d * kd * Le * noiseIntensity;
vec3 lightAmbient = ka * Le;
```

Task 2: Random Scenes The simple scene was extended to rendering multiple objects. To generate a scene, the number of objects to generate $1 \leq N \leq 20$ and the background noise scale $r_{\text{background}} \in [1, 20]$ were randomly, uniformly chosen. For each of the N objects, the scale, (x, y)-position and a number of lighting parameters were randomly chosen; see Listing 2.

The randomly generated scenes can be seen and object variables controlled on Page t2.html.

Listing 2: Javascript code snippet with random variable generation run for each object

```javascript
// Ball scene variables are uniformly distributed
gl.ballScales.push(randomRange(0.1, 0.7));
// Placement
gl.ballX.push(randomRange(-1.5, 1.5));
gl.ballY.push(randomRange(-1.5, 1.5));
// Lighting
gl.ballKd.push(randomRange(0.25, 0.75)); // Diffuse coefficient
gl.ballKs.push(randomRange(0.0, 0.5)); // Specular coefficient
gl.ballS.push(randomRange(10.0, 100.0)); // Shininess
```

Task 3: Object Classes Three parameters controlling characteristics of each object were not set by the random scene and were instead set based on object class

- Colour Hue $h \in [0, 360]$. The emitted radiance of each object was chosen in HSL-format to be $(h, 1, 0.5)$ and converted to RGB. Thus, all objects had fixed lightness and fully saturated colours (before texturing with noise and shading by light) motivated to get a single value expressing most of the dynamics in colour.

- Object elongation $\rho \in [0, 1]$. When $\rho \neq \frac{1}{2}$, the object is subject to axis parallel scaling, converting spheres to ellipsoids. Higher ρ means a wider (x-elongated) object and the implementation can be seen in Listing 3.

- The noise scale $r > 1$ used to texture the object.

Three classes of objects where implemented and to each class was attributed an average values for each of the above characteristics, making class i be described by (h_k, ρ_k, r_k). The chosen values can be seen in Table 1 and the objects rendered with the average characteristics can be seen in Figure 2.

Class k	h_k	ρ_k	r_k
1	150	0.55	5
2	250	0.45	17.5
3	350	0.5	30

Table 1: The average value in each class prior for hue, elongation and noise scale.

Each object was randomly assigned one of the three classes with equal probability. When rendering an object, random values of h, ρ, r were chosen based on the object class according to the Gaussians

$$h \sim \mathcal{N}\left(h_k, 50^2\right), \rho \sim \mathcal{N}\left(\rho_k, 0.05^2\right), r \sim \mathcal{N}\left(r_k, 5^2\right). \tag{1}$$

After drawing these values, the sphere characteristics were set after enforcing the variable ranges; for ρ, r by clamping to borders and for h, by taking modulo 360.

These random object realisations influenced by by class prior of assigned classes can be sampled on Page t3.html with some examples shown in Appendix A.1.

Listing 3: Javascript code snippet which produces the tranformation matrix to elongate the objects according to ρ. This matrix is incorporated into the model matrix for this specific object.

```
let sx = rho;
let sy = (1 - rho);
// Make sure that rho=0.5 corresponds to no scaling
let norm = 2 ** 0.5 / (sx**2 + sy**2)**0.5;
// Only scale parallel to x or y axis to avoid more free
    parameters
let elongationTransform = scalem(sx*norm, sy*norm, 1.0);
```

Task 4: The Full Scene With Label Maps To generate training data, each image must have a corresponds target representing the ground truth of the task. For image segmentation, this ground truth should be a pixel map representing the class label of the object visible in the image at that pixel.

This was done in WebGL by adding a separate rending mode where the fragment colour had hue h_k when drawing an object with class k, bypassinng all texturing and lighting calculations. The background is rendered black, giving four unique pixel values representing four labels[1].

This feature was combined with the class assignments in Task 3 and the random scenes from Task 2 to producgtGe synthetic scenes with class assigned objects and corresponding label maps. One example of such as pair shown in 3 and another in Appendix A.2.

The ability to download the rendered scene was added, and each rendering was given a random string as output name to be able to match images and label maps. This final

[1] Pixels at the borders of objects were blended by the fragment shader giving some extra values. These were ignored when loading the dataset into Python, only accepting the object interior as corresponding to class k

interface is available on Page t4.html.

Generated Datasets To extract a dataset from the WebGL application, dynamic website scraping was implemented in `Python` using a Selenium headless Firefox web browser [Con22]. 10,000 pairs of 512 × 512 images and corresponding label maps were extracted textured with PN and another 10,000 with SC, requiring the rendering of 40,000 images, which took around five hours on a consumer CPU. The two datasets were called `perlin` and `sc` and can be downloaded through links on the GitHub repository.

2.2 Deep Image Segmentation

Baseline Learning Task The chosen benchmark task was image segmentation, also called semantic segmentation or pixel wise object detection, where the algorithm should classify each pixel according to visible object class, segmenting the image into bounded, semantic regions [SS01]. As benchmark dataset, the Common Objects in Context (COCO) 2017 Object Detection Dataset was chosen which is a collection of every day photographs with object annotations in over 80 categories [Lin+14]. A subset of 20 foreground classes including *person*, *car* and *dog* as was chosen for the learning benchmark in accordance with the PyTorch Torchvision image segmentation reference [Fal22].

Images without objects in these classes were discarded. To keep computational scope down and simulate the data limited learning case, the large dataset was subsampled to 1,000 training images and 1,000 validation images.

The machine learning model to map from images to segmentation label maps had the architecture of a Fully-Convolutional Deep Neural Network with a ResNet-50 backbone as described by Shelhamer, Long, and Darrell [SLD17].

The COCO training set was used for optimization using stochastic gradient descent for 10 epochs, evaluating generalization performance to the validation set each epoch, using direct accuracy, Intersection-over-union (IoU, also called Jaccard Index) and F1-score [Fer+17]. Note that, in this multiclass setting, all these scores have two different approaches to computation: Macro which produces the final score by a simple average of scores for each class, and micro averaging which averages class performance by class frequency. In a class imbalanced problem dominated by a single class such as image segmentation with a background class, macro average scores can be expected to be lower.

The training was performed on a Nvidia A100 GPU and took about half an hour.

Training on Synthetic Data For each of the two synthetic datasets, `perlin` and `sc`, a tenth of the data was set aside for validating performance and training of a model was performed on the 9000 images exactly as above for 30 epochs, taking 3.5 hours.

Transfer learning of Pretrained Model To test the usefulness of the synthetic data for practical image segmentation, the model was after training on synthetic data adapted to the COCO task using the paradigm of transformer learning. In this context, the synthetic data training is the pretraining step and the COCO benchmark is the downstream or finetuning task.

This was done by augmenting the training on COCO by initializing the model weights from the network converged on the synthetic dataset instead of random initialization. The final layers, the classification head, which are dependant on the number of classes and thus cannot be shared between 4-class and 21-class segmentation tasks cannot be transferred.

3 Results

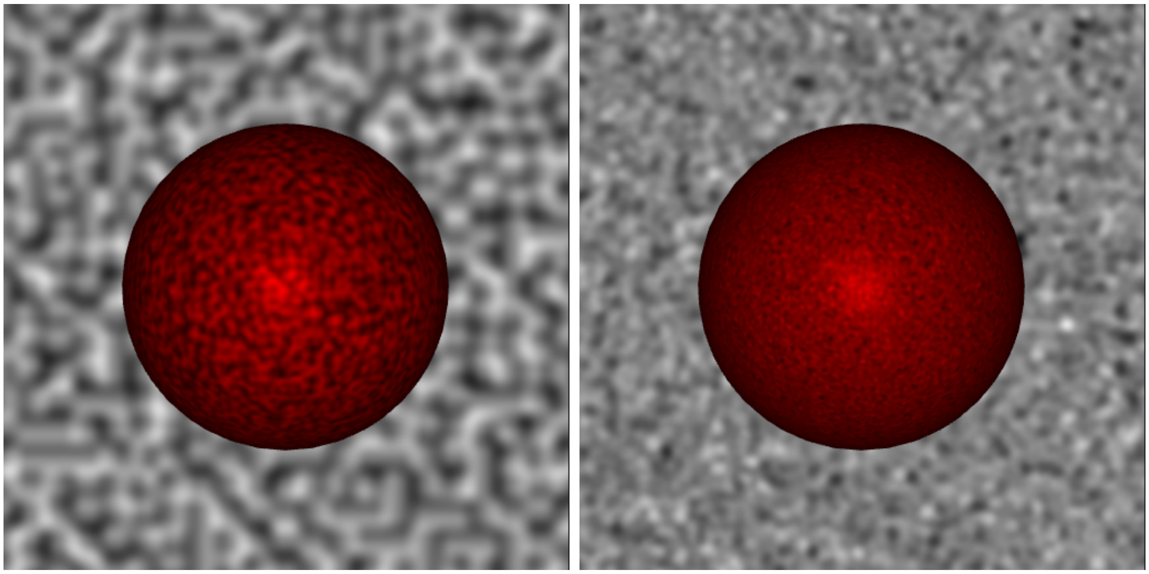

Figure 1: The Task 1 image where the scene is textured using PN (left) and SC (right); both with a noise scale of 25.

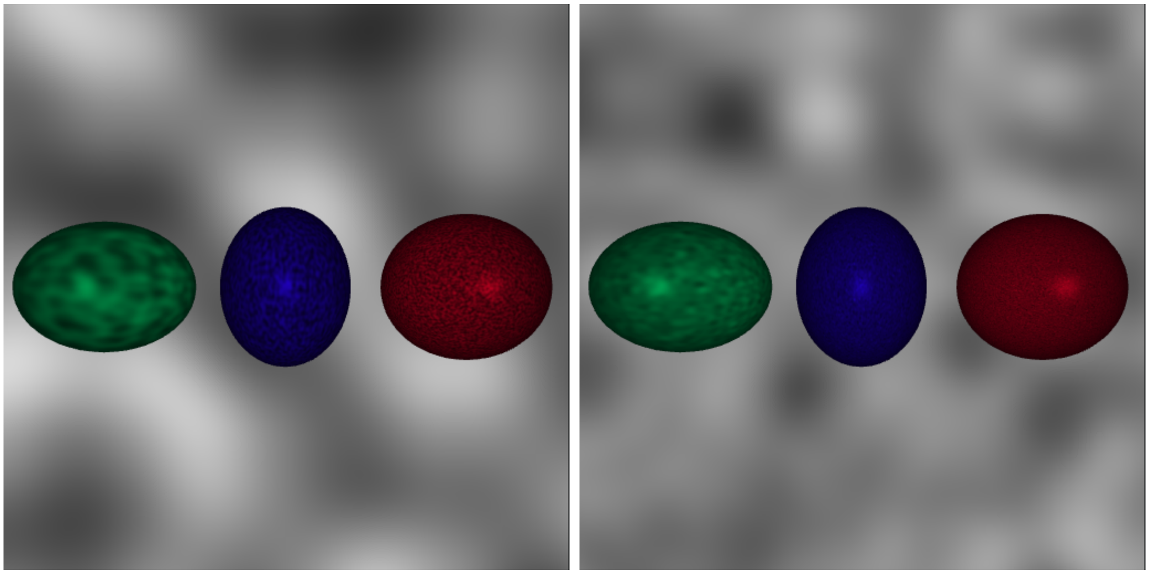

Figure 2: Task 3 images showing the class prototype for each class from left to right in both PN (left) and SC (right). These spheres thus have their elongation ρ, noise scale r and hue h set to the class averages. To see examples of variation within classes, see Appendix A.1.

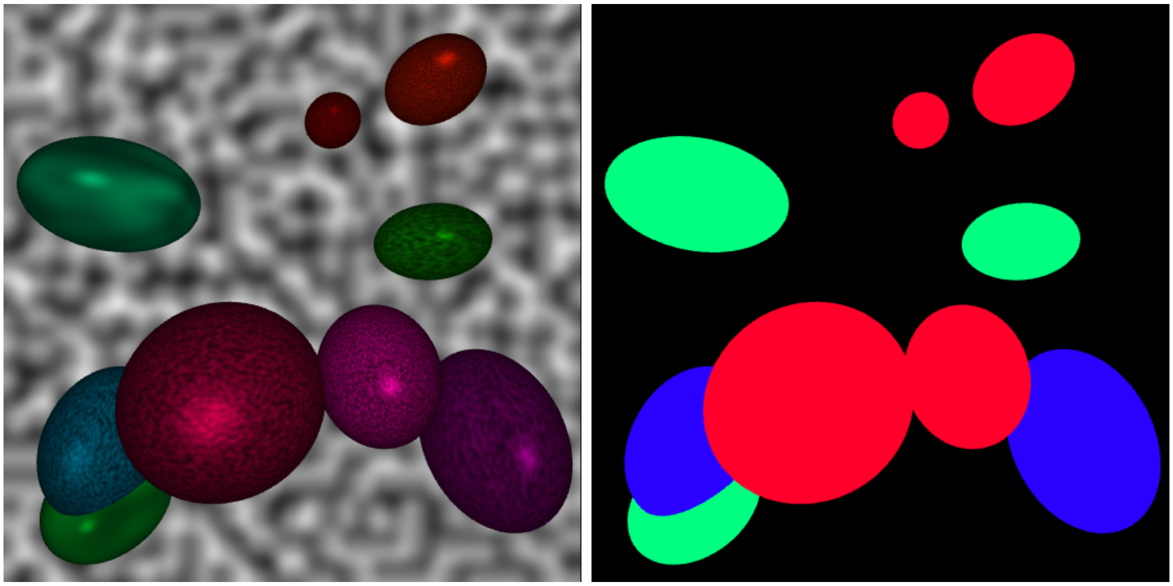

Figure 3: The final Task 4 result. Left: A random scene with class assigned objects textured with PN. Right: The corresponding ground truth class labels where green is class 1, blue class 2 and red class 3.

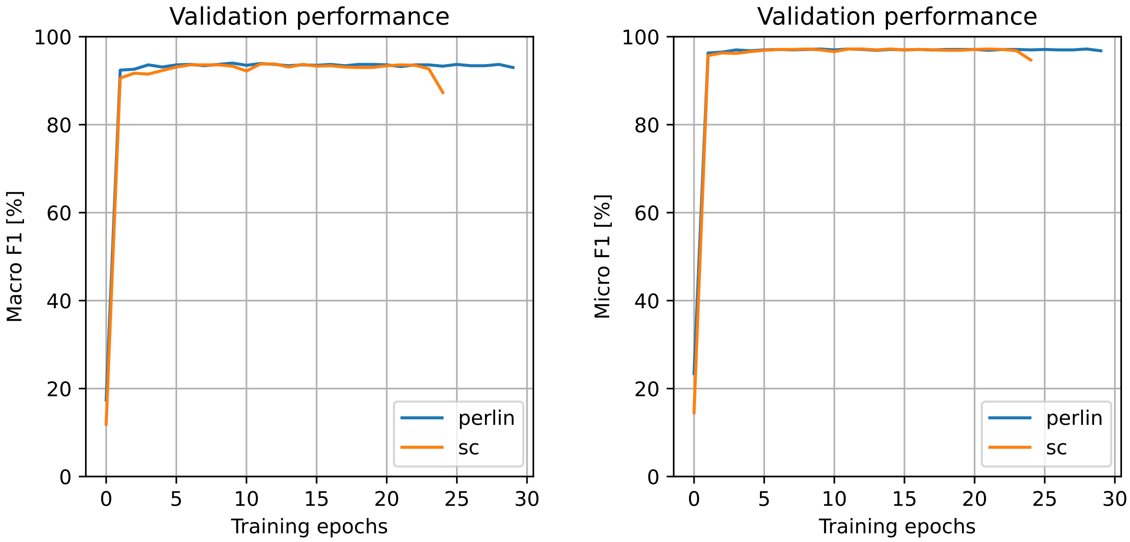

Figure 4: F1 scores on validation datasets, with macro (left) and micro (right) averaging, during training of the two synthetic datasets. See Appendix B for these curves with other measures.

Method	Micro Accuracy	Macro Accuracy	Micro IoU	Macro IoU	Micro F1	Macro F1
perlin	97 ± 1	93 ± 2	94 ± 2	88 ± 2	97 ± 1	93 ± 2
sc	97 ± 1	94 ± 2	94 ± 1	88 ± 2	97 ± 1	93 ± 2

Table 2: Validation results after the final epoch of training on the synthetic datasets with $\alpha = 5\,\%$ confidence intervals according to large sample normality assumption for proportions [Bro+22, Method 7.3].

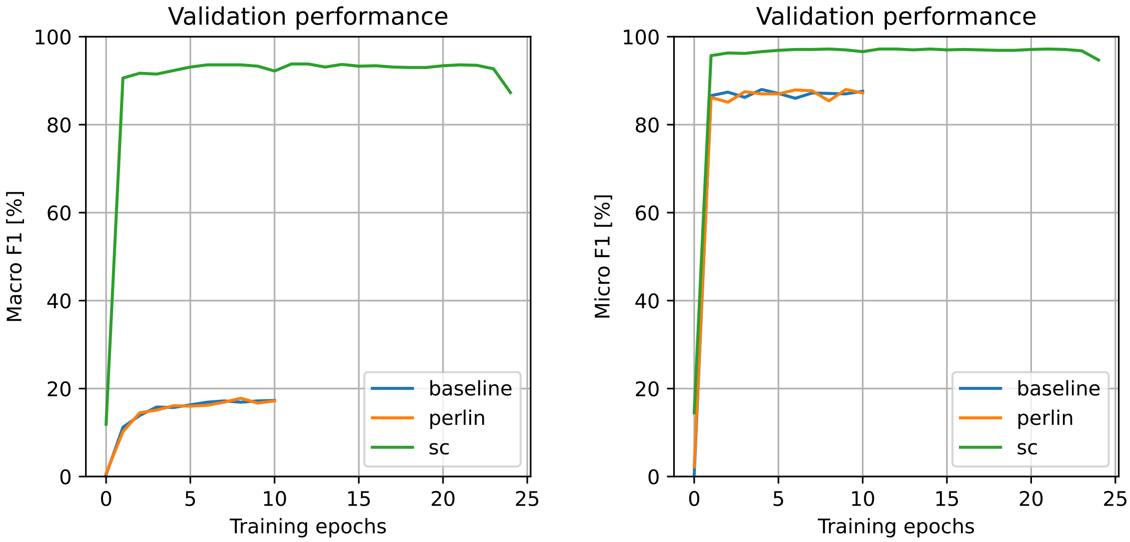

Figure 5: Macro, micro F1 scores on the COCO validation set during training of the COCO benchmark with different model initializations. Baseline corresponds to training from newly initialized models and the other two to transfer learning from models pretrained from each dataset. See Appendix B for these curves with other measures.

Method	Micro Accuracy	Macro Accuracy	Micro IoU	Macro IoU	Micro F1	Macro F1
baseline	88 ± 2	17 ± 2	78 ± 3	14 ± 2	88 ± 2	17 ± 2
perlin	87 ± 2	17 ± 2	78 ± 3	14 ± 2	87 ± 2	17 ± 2
sc	95 ± 1	88 ± 2	90 ± 2	79 ± 3	95 ± 1	87 ± 2

Table 3: COCO final validation results for three initialization approaches.

4 Discussion

Initially, based on the COCO results, I must concede that I could not show what I hoped for: That the pretraining on the synthetic dataset on different balloon-looking objects improved convergence speed or final performance. The synthetic learning task was, however, succesful: For both noise functions, the model converged to high performance. Should this be taken as a sign that different computer vision tasks follow so diverging data distributions that current models cannot extract generalizable patterns across tasks? This question deserves serious work, but I would imagine that a similar setup as this one can succeed.

Looking back, the choice of the COCO task, which appears widely used for benchmarking state of the art models, might be problematic for this project as COCO is a complex image segmentation benchmark with an incredibly variant dataset. Improved convergence through synthetic pretraining might appear if more care is taken to the machine learning part of the project but I would first try with simpler benchmarks such as medical imaging. Another way to improve the impact of pretraining is to improve the realism of the scenes. Adding shadows, adding a physical world with meaningful geometry instead of floating foreground objects and adding different objects than spheres are all within reason with the methods from the course.

Regardless on the relevance for the downstream COCO task, I still find value in the dataset because it allows for the model to converge to reliable class predictions on the synthetic dataset itself. This fact can be used to gain insights into model behaviour which would be my next step if the deadline was a week later: How would the model F1 score change if the objects are not textured using noise functions at all? What if they all have the same colour or same elongation? This approach of model interpretability using synthetic data might hold much promise for deep learning.

References

[Bro+22] Per B. Brockhoff et al. *Introduction to Statistics - eNotes*. English. 2022. URL: https://02402.compute.dtu.dk/book.

[Con22] Selenium Open Source Contributors. "The Selenium Browser Automation Project". In: *The Selenium Documentation* (2022). Acquired online 25/11 2022 at https://www.selenium.dev/documentation/.

[Fal22] Daniel Falbel. *torchvision: Models, Datasets and Transformations for Images*. https://torchvision.mlverse.org, https://github.com/mlverse/torchvision. 2022.

[Fer+17] E. Fernandez-Moral et al. *A new metric for evaluating semantic segmentation: leveraging global and contour accuracy*. HAL Id: hal-01581525. 2017. URL: https://hal.inria.fr/hal-01581525/document.

[Fri16] Jeppe Frisvad. *Exploring noise functions with WebGL*. Online. Accessed: 16/12 2022. 2016. URL: https://people.compute.dtu.dk/jerf/code/noise/.

[FW07] Jeppe Revall Frisvad and Geoff Wyvill. "Fast High-Quality Noise". In: *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*. GRAPHITE '07. Perth, Australia: Association for Computing Machinery, 2007, pp. 243–248. ISBN: 9781595939128. DOI: 10.1145/1321261.1321305. URL: https://doi.org/10.1145/1321261.1321305.

[Gus05] Stafan Gustavson. *Simplex Noise Demystified*. Online. Accessed: 16/12 2022. Sweden, 2005. URL: http://www.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf.

[Lin+14] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[McE+12] Ian McEwan, David Sheets, Mark Richardson, and Stefan Gustavson. "Efficient Computational Noise in GLSL". In: *Journal of Graphics Tools* 16.2 (2012), pp. 85–94. DOI: 10.1080/2151237X.2012.649621.

[SLD17] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651. DOI: 10.1109/TPAMI.2016.2572683.

[SS01] George Stockman and Linda G. Shapiro. *Computer Vision*. 1st. USA: Prentice Hall PTR, 2001. ISBN: 0130307963.

A Additional Example Images

A.1 Class examples

See Figure 6 for four random instances of object classes using PN and Figure 7 for the same using SC.

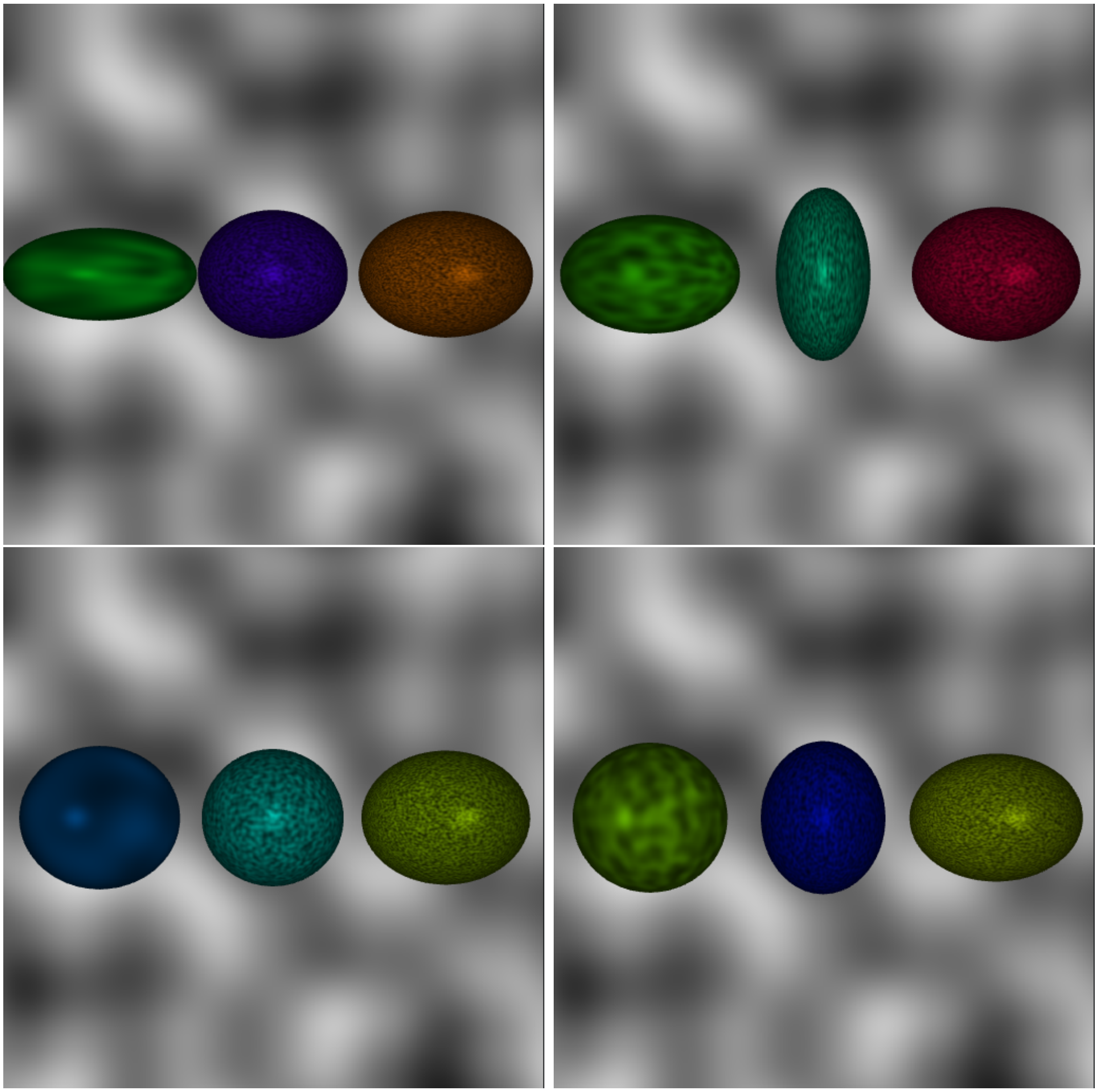

Figure 6: Random instances of classes 1, 2, 3 textured using PN.

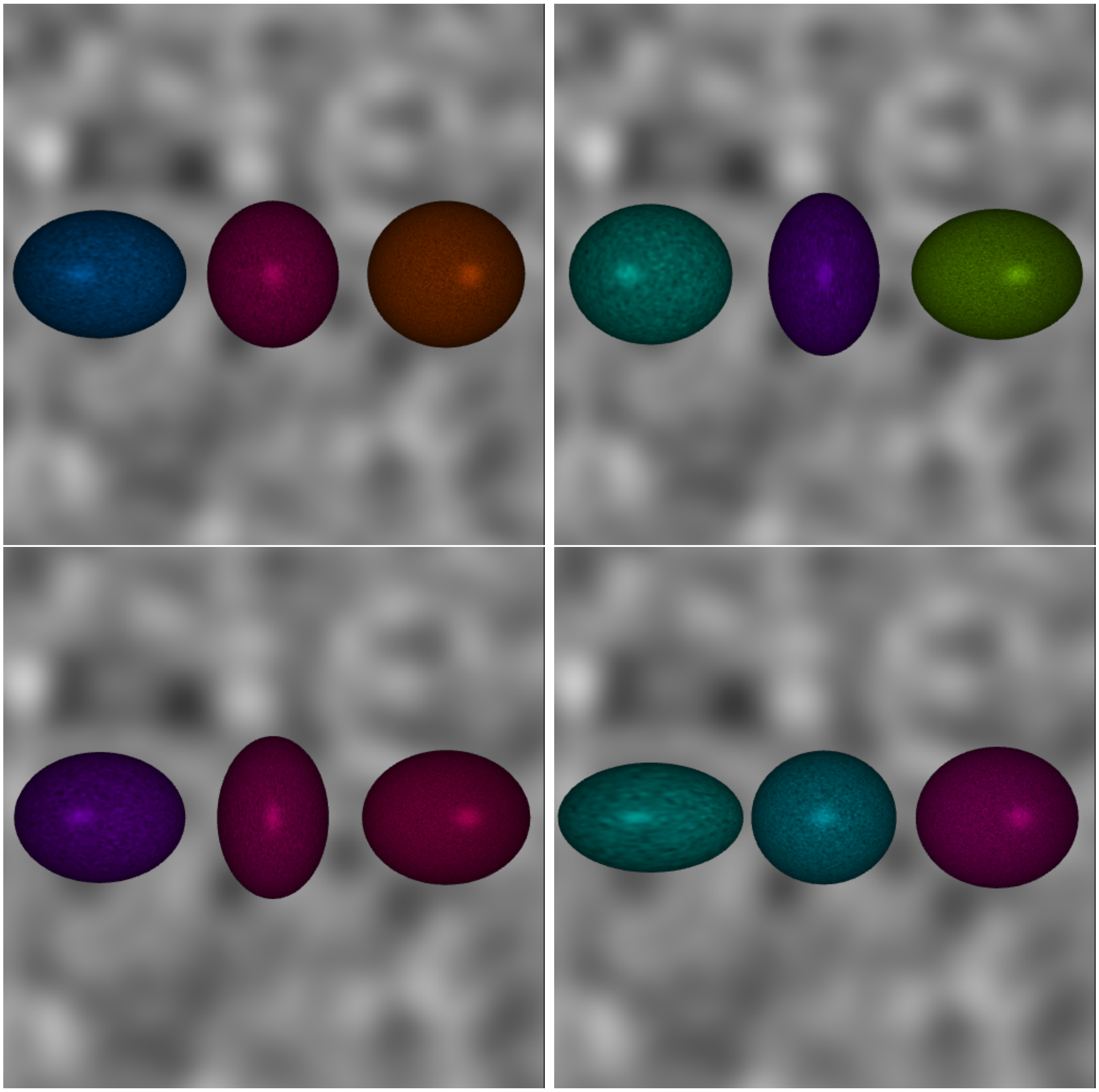

Figure 7: Random instances of classes 1, 2, 3 textured using SC.

A.2 Full Scenes in Different Noise Functions

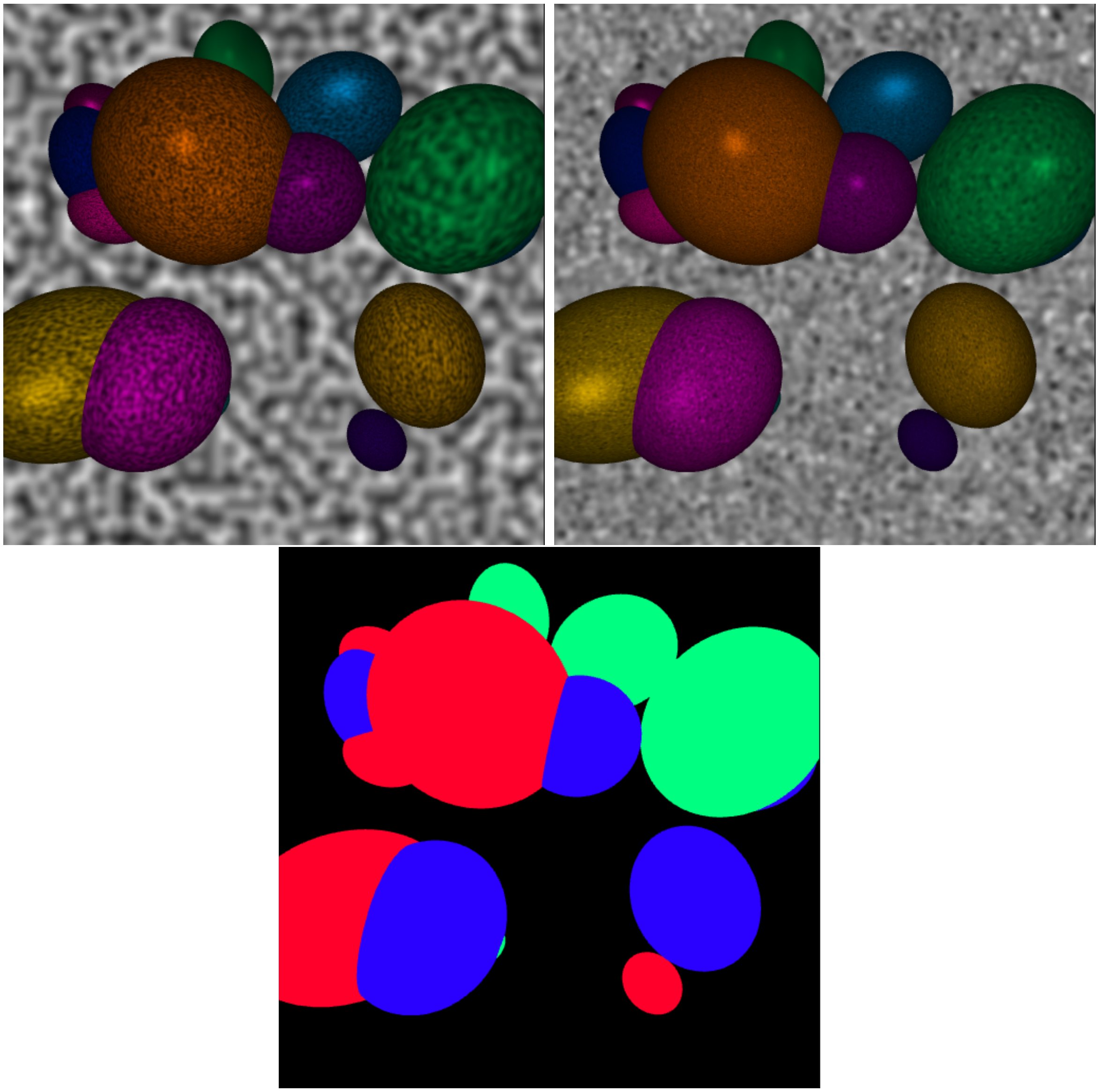

Figure 8: A random scene generated as in Task 4 with both PN (upper left) and SC (upper right).

B More Training Curves

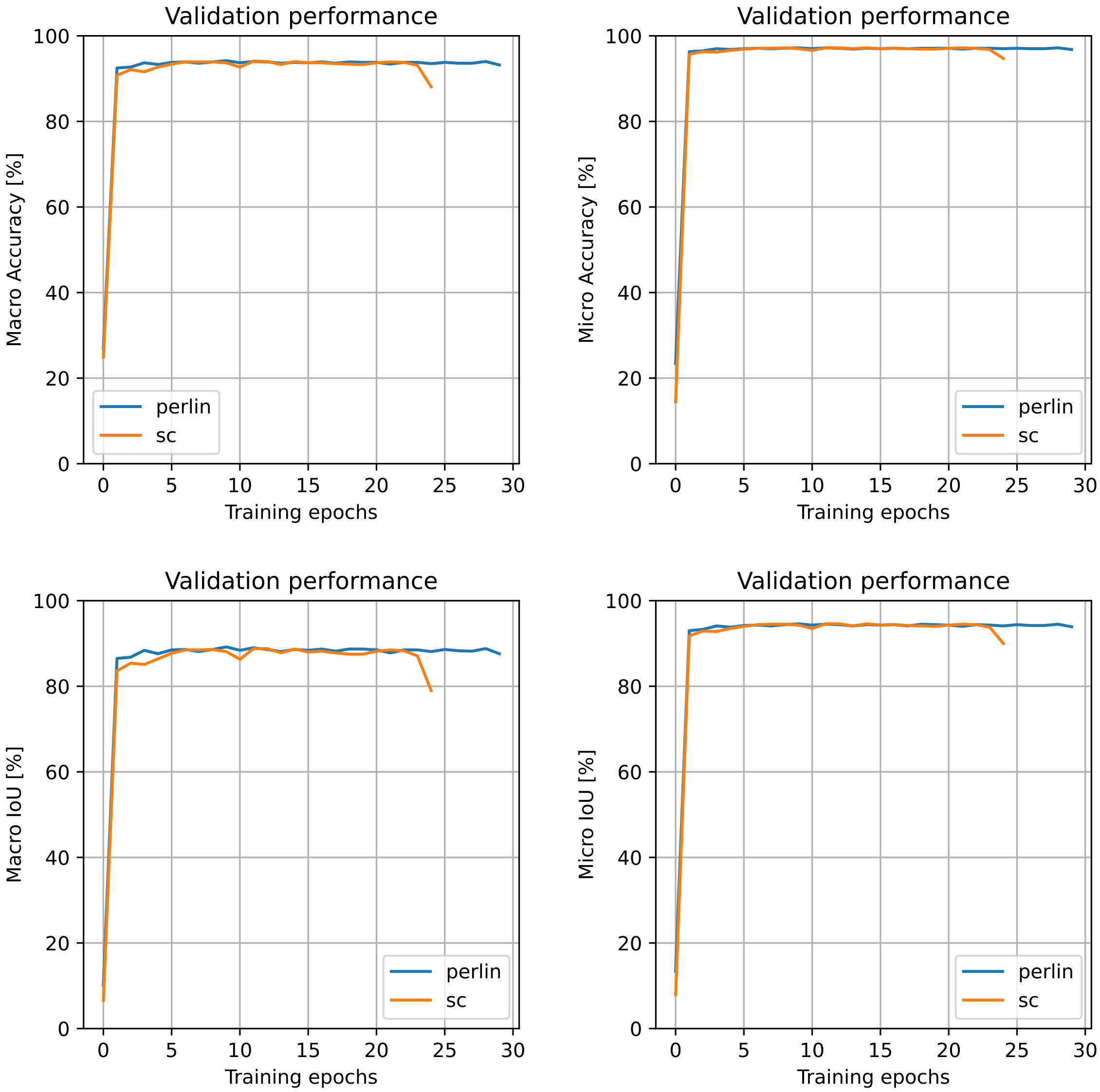

Figure 9: COCO validation scores as on figure 5 just with the other metrics which can be seen on the y-axis.

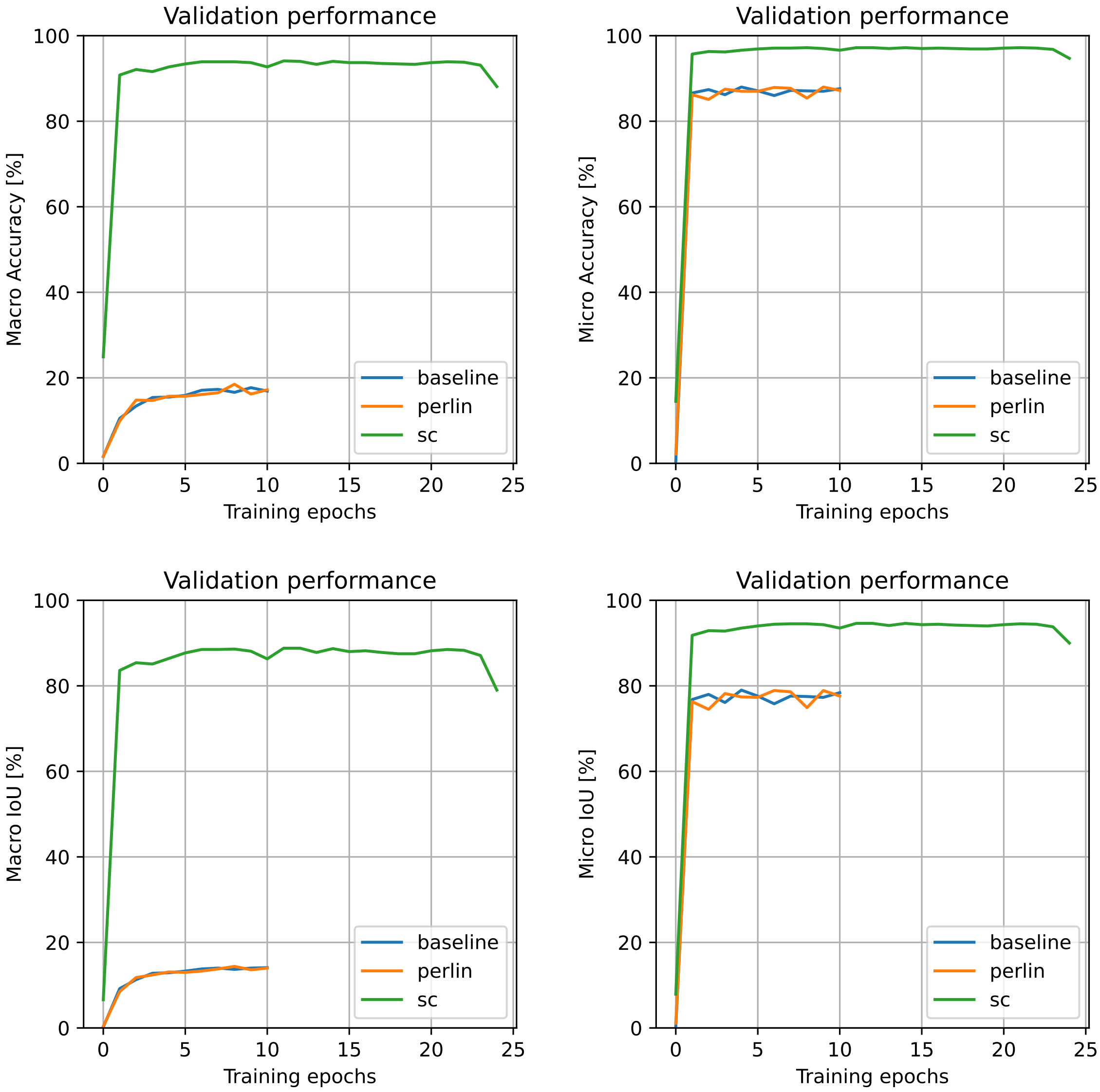

Figure 10: COCO validation scores as on figure 5 just with the other metrics which can be seen on the y-axis.